AN
INTRODUCTION
TO
MICROCOMPUTERS

VOL. II
SOME REAL
PRODUCTS

OSBORNE
A.

N.B. FAULTY BINDING

CHAPTER 3 DUPLICATED, HALF CHAPTER 4 NOT PRINTED.

lication
by any
he prior

act:

AN INTRODUCTION TO MICROCOMPUTERS

VOLUME II SOME REAL PRODUCTS

ry of Congress Catalogue Card Number 76-374891

Published by Adam Osborne and Associates, Incorporated
P.O. Box 2036, Berkeley, California 94702

nation on translations, pricing and ordering outside the U.S.A., please contact:

Osborne & Associates, Inc.
P.O. Box 2036
Berkeley, California 94702
United States of America
(415) 548-2805

Data Sheets in this book have been copied from vendor literature. The manufacture are credited as follows:

CHAPTER

1 TEXAS INSTRUMENTS, INC.
P.O. Box 1443
Houston, Texas 77001

2 FAIRCHILD SEMICONDUCTOR
464 Ellis Street
Mountain View, CA 75006

3 NATIONAL SEMICONDUCTOR CORPORATION
2900 Semiconductor Drive
Santa Clara, CA 95050

4 INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, CA 95051

NEC MICROCOMPUTERS, INC.
5 Militia Drive
Lexington, MA 02173

5 INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, CA 95051

6 INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, CA 95051

7 ZILOG, INC.
10460 Bubb Road
Cupertino, CA 95014

8 MOTOROLA, INC.
Semiconductor Products Division
3501 Ed Bluestein Boulevard
Austin, Texas 78721

9 MOS TECHNOLOGY, INC.
950 Rittenhouse Road
Norristown, PA 19401

10 SIGNETICS
811 East Arques Avenue
Sunnyvale, CA 94043

11 RCA SOLID STATE DIVISION
P.O. Box 3200
Somerville, N.J. 08876

12 INTERSIL, INC.
10900 North Tantau Avenue
Cupertino, CA 95014

13 SCIENTIFIC MICRO SYSTEMS, INC.
520 Clyde Avenue
Mountain View, CA 94043

14 NATIONAL SEMICONDUCTOR, INC.
2900 Semiconductor Drive
Santa Clara, CA 95050

15 GENERAL INSTRUMENT CORPORATION
MICROELECTRONICS
600 West John Street
Hicksville, N.Y. 11802

16 TEXAS INSTRUMENTS, INC.
P.O. Box 1443
Houston, Texas 77001

17 DATA GENERAL CORPORATION
Mail Stop 6-58
Southborough, MA 01772

18 ADVANCED MICRO DEVICES
901 Thompson Place
Sunnyvale, CA 94086

19 MOTOROLA SEMICONDUCTOR
Box 20912
Phoenix, Arizona 85036

20 HEWLETT PACKARD
Data Systems Division
11000 Wolfe Road
Cupertino, CA 95014

We would like to acknowledge the diligent work of our production staff

George M. Vrana	Art and Design
Karen deRobinson	Art
V. Mitchell	Typesetting
Penny Lawrence	Typesetting
Marcia P. McCuen	Typing
Sally Kusch	Proofreading
Mary Borchers	Coordinator

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

CHAPTER PAGE

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

CHAPTER PAGE

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

QUICK INDEX

QUICK INDEX

QUICK INDEX

QUICK INDEX

INDEX **PAGE**

QUICK INDEX

QUICK INDEX

QUICK INDEX

QUICK INDEX

QUICK INDEX

QUICK INDEX

QUICK INDEX

QUICK INDEX

QUICK INDEX

QUICK INDEX

INTRODUCTION

This is a book which is still in transition.

As the second volume in a two-volume set, the purpose of this book is to describe some real products which implement the general concepts covered in Volume I.

The problem is that we could describe some real products in many ways — and at many levels.

Since writing the first edition of "An Introduction To Microcomputers", we have come to the conclusion that what is really needed is a comprehensive and detailed description of all microcomputer LSI devices. These descriptions must stand alone; the reader must not be referred to vendor literature for any additional information. But meeting this goal is a very substantial undertaking. In the end, we will produce a universal LSI microcomputer device reference handbook. Rather than wait the year or two it would take to generate the desired end product, we have elected to achieve our goal one step at a time.

This book represents Step 2 towards our ultimate goal. Step 2 is not as big as it should have been, but the first edition of Volume II sold more rapidly than anticipated, allowing little time for production of this new book.

In this book devices of the 8080A, 8085, 8048, MC6800, Z80, 9440, MCS6500, PACE and SC/MP microcomputers are described in approximately the detail we believe to be necessary. Other devices are described at a more superficial level, and refer you to vendor literature for complete information.

These are the additions that have been made to this revision of Volume II:

1) New chapters have been added describing the 8085 and the 8048 products. The 8085 is Intel's 8080A enhancement, while the 8048 is Intel's new one-chip microcomputer. These products are described completely.

2) A brief discussion of the new Hewlett Packard MC2 microprocessor is given. This microprocessor is unlikely to be sold as a chip in the foreseeable future — a fact which is reflected in the level of the product description provided.

3) Descriptions of the National Semiconductor PACE and SC/MP, plus the Fairchild 9440, have been expanded to provide what we believe is total, stand alone documentation.

4) The 8080A coverage has been expanded to clearly identify incompatibilities in second source products. Two serial I/O devices manufactured by NEC have been added.

5) Minor product enhancements have been identified for the F8 and MC6800 product lines.

6) The new Z80 CTC and Z80 DMA devices are described in detail in the Z80 chapter

7) We have described how support devices of one microprocessor can be used with another microprocessor.

Two products have been removed from the first revision of Volume II. The EA9002 from Electronic Arrays has been withdrawn by the manufacturer. The PPS-8 from Rockwell is now selling in such low volume that the product is being manufactured by Rockwell only to fill a small number of existing orders. Chapters describing the EA9002 and the PPS-8 have therefore been eliminated.

Note that this book does not include detailed programming descriptions for any microcomputer. Instruction set summaries are given and that is all. Two additional series of books are devoted strictly to programming. The first is a "Programming For Logic Design" series aimed specifically at microprocessors being used in a digital logic environment — where programming is a simple alternative for combinatorial logic. The second is a more traditional "Assembly Language Programming" series that treats microcomputers like small general purpose computers — and pays no special attention to digital logic applications. These three books are currently available in the "Programming for Logic Design" series:

1) 8080 Programming For Logic Design
2) 6800 Programming For Logic Design
3) Z80 Programming For Logic Design

Planned for release in 1977 are Cosmac and 8048 Programming For Logic Design books.

In the traditional "Assembly Language Programming" series, "8080/8085 Assembly Language Programming" is currently available, while "6800 Assembly Language Programming" and "Z80 Assembly Language Programming" are planned for release in 1977.

We continue to request that you, the reader, send comments regarding what should be included in future revisions of this book. In particular, we would appreciate guidance from you as to whether Chapters 4, 5, 6, 7, 8 and 9 adequately describe the subject matter — and if not, why not.

Regarding this book, some conventions need to be defined.

Signals may be active high, active low or active in two states. An active high signal is one which, in the high state, causes events to occur, while in the low state has no significance. A signal that is active low causes events to occur when in the low state, but has no significance in the high state. A signal that has two active states will cause two different types of events to occur, depending upon whether the signal is high or low; this signal has no inactive state. Within this book a signal that is active low has a bar placed over the signal name. For example, $\overline{\text{WR}}$ identifies a "write strobe" signal which is pulsed low when data is ready for external logic to receive. A signal that is active high, or has two active states, has no bar over the signal name.

Every microcomputer instruction set is described with two tables. One table identifies the operations which occur when the instruction set is executed, while the second table defines object codes and instruction times.

Because of the wide differences that exist between one instruction set and another, we have elected not to use a single set of codes and symbols to describe the operations for all instructions, in all instruction sets. We believe any type of universal convention is likely to confuse rather than clarify; therefore each instruction set table is preceded by a list of symbols as used within that table alone.

A short benchmark program is given to illustrate each instruction set. Some comments regarding benchmark programs in general are, however, in order. We are not attempting to highlight strengths or weaknesses of different devices, nor does this book make any attempt at comparative analyses, since the criteria which make one microcomputer better than another are simply too dependent on the application.

COMPARATIVE ANALYSIS

Consider an application which requires relatively high speed processing. The only important criterion will be program execution speed, which may limit the choice to just one of the microcomputers we are describing.

Execution speeds of all of the microcomputers may, on the other hand, be quite adequate for a second application; in this case, price may be the only overriding factor.

In a third application, a manufacturer may have already invested in a great deal c engineering development expense, using one particular microcomputer that wa available in quantity earlier than any others; the advantages or disadvantages of usin a different microcomputer, based on minor cost of performance advantages, will likel be overwhelmed by the extra expense and time delays involved with switching in mid stream.

In Chapter 21 we will look at a logical procedure for determining which is the righ microcomputer for a new application, taking into account the various factors tha could influence an as yet unmade decision.

BENCHMARK PROGRAMS

And what about benchmark programs?

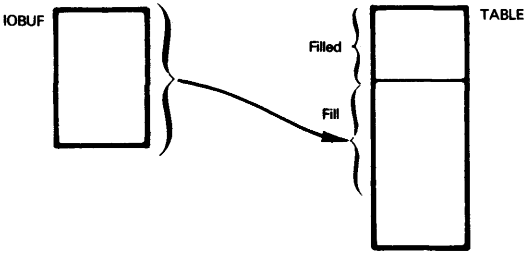
There have been a number of benchmark programs in the literature, purporting to show the strengths or weaknesses of one microcomputer versus another; individual manufacturers have added to th confusion by putting out their own competing benchmarks, aimed at showin their product to be superior to an immediate rival.

Benchmark programs are misleading, irrelevant and worthless for these reasons

1) **In a majority of microcomputer applications, program execution speed, an minor variations in program length, are simply overwhelmed by pricing considerations.**
2) **Even assuming that for some specific application, program length and execution speed are important, trivial changes in the benchmark program definitio can profoundly alter the results that are obtained. This is one point we wi demonstrate in this book, while describing individual instruction sets.**
3) **Benchmark programs are invariably written by the smartest programmers i an organization, and they take an enormous amount of time to ensure programming accuracy and excellence. This is not the level at which any use should anticipate "run of the mill" programmers working; indeed, a far mor realistic evaluation of a microcomputer's instruction set could be generate by giving an average programmer too little time in which to implement an incompletely defined benchmark. This will more closely approximate the working conditions under which real products are developed. Of course, definin the "average programmer", "too little time" and an "incomplete specification" are all sufficiently subjective that they defy resolution.**

We will demonstrate the capriciousness of benchmark programs via the following benchmark program:

Raw data has been input to a general purpose input buffer, beginning at IOBUF. This raw data is to be moved to a permanent table, which may be partially filled; the raw data is to be stored in the data table starting with the first unfilled byte. The benchmark may be illustrated as follows:



HOW THIS BOOK HAS BEEN PRINTED

Notice that text in this book has been printed in boldface type and lightface type. **This has been done to help you skip those parts of the book that cover subject matter with which you are familiar. You can be sure that lightface type only expands on information presented in the previous boldface type.** Therefore, only read boldface type until you reach a subject about which you want to know more, at which point start reading the lightface type.

Chapter 1
4-BIT MICROPROCESSORS AND THE TMS1000 SERIES MICROCOMPUTERS

The earliest microprocessors were all 4-bit devices; that is to say, data was operated o in 4-bit units, frequently referred to as "nibbles". **Early microprocessors were 4-bi devices simply because the concept of an LSI CPU was ambitious enough; starting with an 8-bit CPU would have been foolhardy.**

But LSI technology has advanced so rapidly that there is an inconsequential differenc between the cost of manufacturing an 8-bit CPU chip as against a 4-bit chip. Manufacturers attempted to maintain an artificial price differential between their 4-bit and 8-bi CPUs in order to prolong the life of the 4-bit product; but the pressure of competitio has all but extinguished these price differentials — with the result that the 4-bi microprocessor is a dying product. Price is the only advantage that 4-bi microprocessors offer when compared to the more capable 8-bit microprocessor.

Early 4-bit microcomputers included such devices as the Intel 4004 and 4040 and th National Semiconductor IMP-4. These early 4-bit microcomputers require packag counts that exceed typical 8-bit microcomputers that are now available; therefore **th economics of today dictate that the Intel 4004, the Intel 4040 and the IMP-4 offe less capability for more money.** Only the most unusual application could be more economically implemented using one of these three 4-bit microcomputers, rather than simple 8-bit device such as the F8, COSMAC,SC/MP, or one of the 28-pin MCS650 series CPUs. **We consider the Intel 4004, the Intel 4040 and the IMP-4 to be obsolete devices; therefore they are not described.**

It is interesting to note that even though these three 4-bit microcomputers are obsolete they will continue to have a significant market for many years to come, based on products that were designed around them before they became obsolete. The fact that the are obsolete simply means that were you to design a new product today, you would b better off using one of the simple 8-bit microcomputers. That does not mean it woul be economical to redesign a product that already exists, simply to take advantage c more recent microcomputer developments. The cost of re-engineering around a nev microcomputer will likely overwhelm any savings that may accrue.

The TMS1000 series microcomputer devices, manufactured by Texas Instruments, are still economically very viable — even though they are 4-bit devices This is because the TMS1000 is a one-chip microcomputer. ROM RAM CPU an I/O logic are all provided within a single package. Only Fairchild offers a equivalent, single chip 8-bit microcomputer; and it is only just becoming availabl in limited quantities. The low cost associated with the single chip TMS100 microcomputer package makes this the product of choice for a large number c simple applications that can be accommodated within the logical confines of th TMS1000.

In reality, the TMS1000 is a family of six 4-bit microcomputers whose differences are summarized in Table 1-1. The various microcomputers are sufficiently similar for us to describe them together.

Table 1-1. TMS1000 Series Microcomputer Summary

	TMS 1000	TMS 1200	TMS 1070	TMS 1270	TMS 1100	TMS 1300
Package Pin Count	28	40	28	40	28	40
ROM Program Bytes*	1024	1024	1024	1024	2048	2048
RAM Data Nibbles**	64	64	64	64	128	128
R Signal Outputs	11	13	11	13	11	16
O Data Outputs	8	8	8	10	8	8
Maximum Rated Voltage	20	20	35	35	20	20
Typical Power Dissipation	15V/ 90mW	15V/ 90mW	15V/ 90mW	15V/ 90mW	15V/ 90mW	15V/ 90mW

*A Byte is eight bits
**A Nibble is four bits

Figure 1-1 illustrates that part of our general microcomputer system logic which is implemented by the TMS1000 series microcomputers. This figure is deceptive since it would be hard to compare the primitive I/O capabilities of the TMS1000 with a device such as the 8255 Programmable Peripheral Interface device, which is described in Chapter 4. Nevertheless, Figure 1-1 does indicate the logic which is provided by a TMS1000 series microcomputer, albeit in a primitive form.

The fact that the TMS1000 series microcomputers are single chip devices has a number of secondary, nonobvious implications. Most important of all, there are no such things as support devices. The 1024 or 2048 bytes of ROM represent the exact amount of program memory which will be present; there can be neither more nor less. Similarly, the 64 or 128 nibbles of RAM cannot be expanded. Direct memory access logic is not present — and its presence would make very little sense anyway; with the small total ROM and RAM memory available, there simply is not the opportunity to transfer blocks of data long enough to warrant bypassing the CPU.

Interrupts, similarly, would be of marginal value to a TMS1000 microcomputer. Given the small amount of program memory available and the very low cost of the package, it would be hard to justify the complexities of interrupt logic, simply to have the microcomputer perform more than one task.

All devices of the TMS1000 microcomputer family are implemented using PMOS technology.

A single -15V power supply is required.

The fastest clock frequency which can drive a TMS1000 series microcomputer has a 2.5 microsecond cycle time. All instructions execute in six clock cycles, or 15 microseconds; but beware of making direct execution speed comparisons between the TMS1000 and the 8-bit microcomputers which are described next. A TMS1000 program will usually be considerably longer than the 8-bit microcomputer equivalent because the TMS1000 instruction set is more primitive; but this is not always true. It is possible for the TMS1000 instruction set to equal or surpass many 8-bit microprocessors, in terms of instruction efficiency, for certain control applications.

The manufacturer of the TMS1000 is:

TEXAS INSTRUMENTS, INC.
P. O. Box1443
Houston, Texas 77001

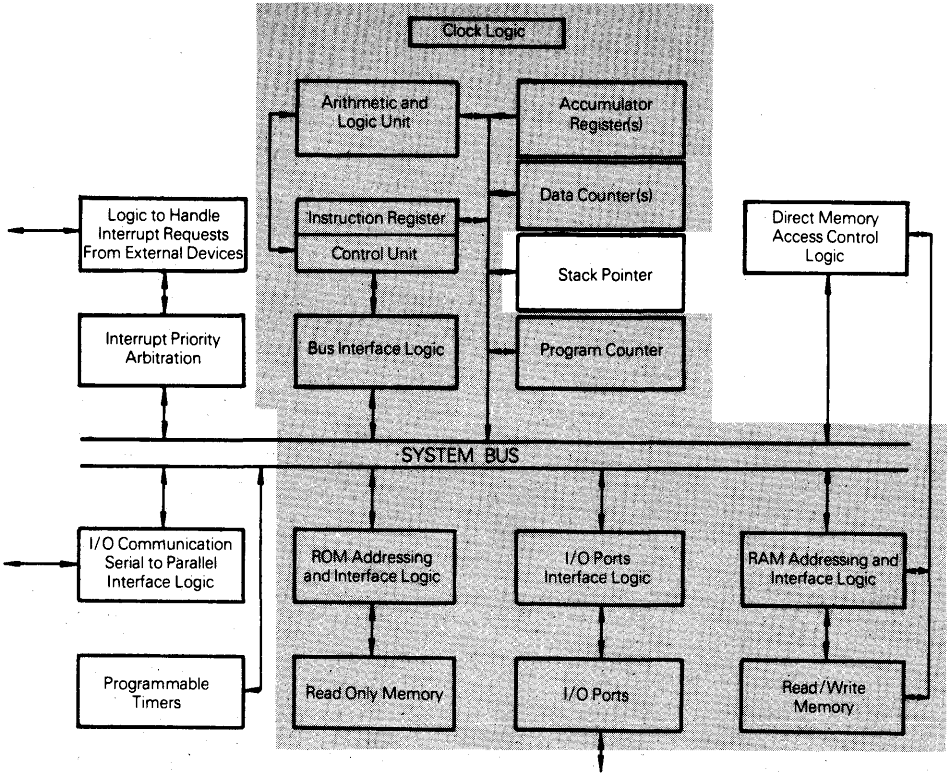
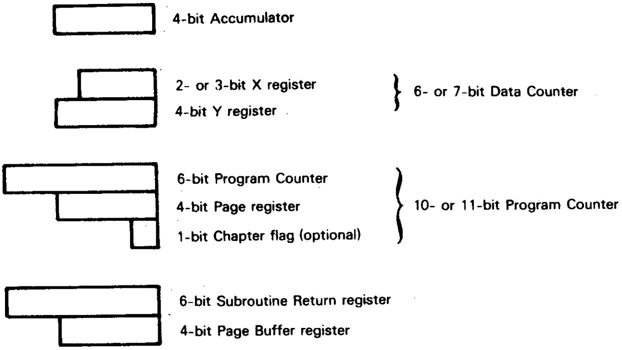

Clock Logic
Arithmetic and Logic Unit
Accumulator Register(s)
Data Counter(s)
Instruction Register
Control Unit
Stack Pointer
Bus Interface Logic
Program Counter
Logic to Handle Interrupt Requests From External Devices
Interrupt Priority Arbitration
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
Programmable Timers
ROM Addressing and Interface Logic
Read Only Memory
I/O Ports Interface Logic
I/O Ports
RAM Addressing and Interface Logic
Read/Write Memory

Figure 1-1. Logic Of The TMS1000 Series Microcomputer

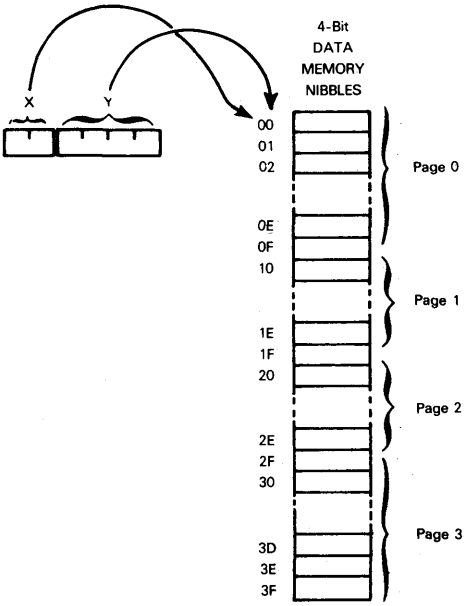
TMS1000 PROGRAMMABLE REGISTERS

TMS1000 programmable registers may be illustrated as follows:



Apart from being only four bits wide, **the Accumulator is a typical primary Accumulator.** It is the principal source and destination for data that is being operated on.

Taken together, **the X and Y registers constitute a 6- or 7-bit Data Counter** which addresses the 64 or 128 nibbles of RAM. The X register is two or three bits wide and the Y register is four bits wide. Since the X and Y registers are indeed separate and distinct registers, RAM is effectively divided into four or eight pages, each of which is 16 nibbles long. A four-page RAM may be illustrated as follows:



The Y register, in addition, serves as a secondary Accumulator and an output Address register. We will describe its use as an output Address register shortly.

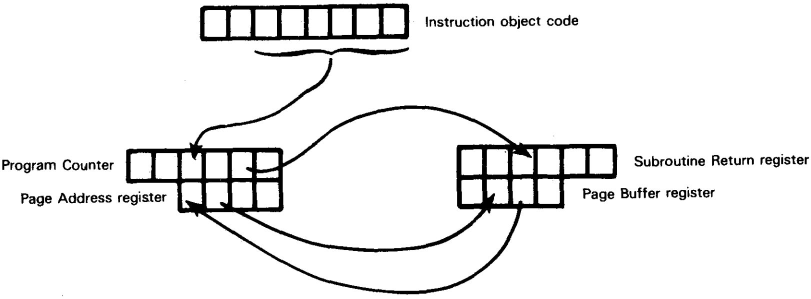
Those TMS1000 series microcomputers that provide 128 nibbles of RAM have a 3-bit X register. RAM is then divided into eight 16-nibble pages.

The Program Counter and Page Address register, taken together, constitute a 10-bit Program Counter. They are, in reality, separate and distinct registers, with the result that program memory is divided into sixteen 64-byte pages.

Those TMS1000 microcomputers that provide 2048 bytes of program memory have an additional 1-bit flag, referred to as Chapter Logic, which is used to select one of two alternate 1024-byte ROM chapters.

TMS1000 SUBROUTINES

The Subroutine Return register is simply a buffer for the Program Counter register. Similarly, the Page Buffer register is a simple buffer for the Page Address register. These two buffer registers allow the TMS1000 a single level of subroutine call logic. **When a subroutine is called,** the contents of the Page Address register and Page Buffer registers are exchanged, the Program Counter register contents are moved to the Subroutine Return register, and a new value provided by the subroutine call instruction is loaded into the Program Counter. This may be illustrated as follows:



TMS1000 MEMORY ADDRESSING MODE

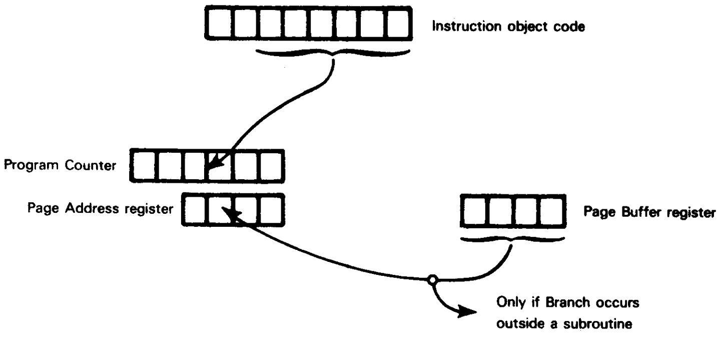
TMS1000 microcomputers have separate and distinct program and data memories. There are no instructions capable of writing into program memory and data memory cannot contain instruction object codes.

Data memory is accessed using implied addressing. The X and Y registers combine to serve as a Data Counter; we have just described this use of the X and Y registers.

Only subroutine Call instructions and Branch instructions address program memory. These instructions address program memory using variations of absolute, paged direct addressing.

We have already illustrated the addressing logic of a subroutine Call.

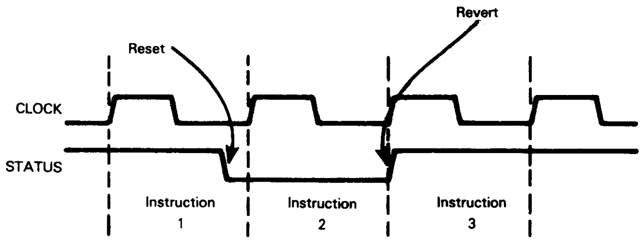
A Branch instruction loads the Program Counter from the instruction, just as a Call instruction does. If the Branch instruction occurs in a subroutine — that is, in the sequence between a subroutine Call instruction and a subroutine Return instruction — the Page Address register will not be affected. However, execution of a Branch instruction outside a subroutine will load the Page Address register from the Page Buffer register. The two types of program branch may be illustrated as follows:



TMS1000 STATUS FLAGS

The TMS1000 series microcomputers have a single status flag which combines to serve as a Carry status and a simple logic decision status. All Branch and subroutine Call instructions are conditional; the Branch or subroutine Call occurs only if the status flag is 1.

The unique feature of the status flag as compared to most status logic is that its passive level is high (1). **If an instruction causes the status flag to be reset to 0, it will revert to 1 after a single instruction cycle:**



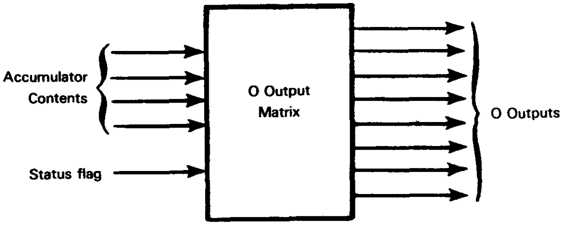
Instructions that test the condition of the status flag must directly follow the instruction which modifies the level of the status flag.

TMS1000 INPUT AND OUTPUT LOGIC

The only data input to a TMS1000 series microcomputer occurs as 4-bit nibbles, referred to in Texas Instruments literature as K inputs. Instructions that access the K inputs simply input whatever signal levels exist at the time of the access.

TMS1000 series microcomputers output data referred to as O outputs, and control signals referred to as R outputs.

There are eight data or O outputs; but they are created in an unusual way. O output logic receives, as inputs, the contents of the Accumulator, plus the status flag. These five data bits create the eight O output signals according to a matrix which you must define when you order the TMS1000 microcomputer. This may be illustrated as follows:



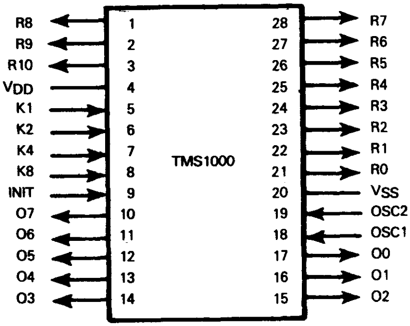
As the illustration above would imply, the five inputs select 32 of the possible 256 signal combinations which can be output via the eight O outputs.

The control R outputs are treated as 11, 13 or 16 single control signals. Refer to Table 1-1, which identifies the number of R output signals available with each of the TMS1000 series microcomputers. **You can set or reset R output signals individually. The Y register is used to identify the individual R signal which is being set or reset.**

TMS1000 SERIES MICROCOMPUTER PINS AND SIGNALS

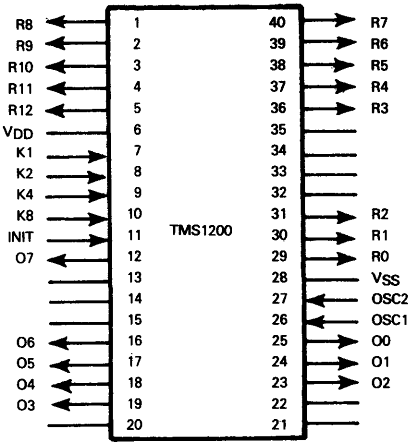
Figures 1-2 through 1-7 illustrate the pins and signals of the TMS1000 series microcomputers. Note that the TMS1000 and TMS1100 microcomputers have identical pins and signals. Since signals are consistent for the entire family of microcomputers, they will be described together.

The four data inputs are provided by K1, K2, K4 and K8. We would name these signals DI0, DI1, DI2 and DI3 to be consistent with common microcomputer terminology; however, Texas Instruments literature uses the signal names K1, K2, K4 and K8 to represent the binary level of each signal.



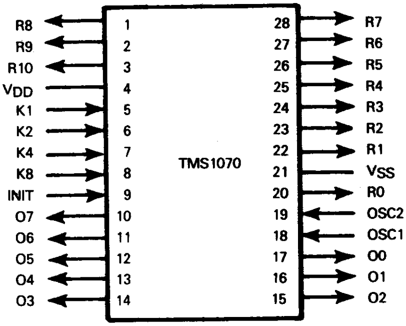
Pin Name	Description	Type
K1, K2, K4, K8	Data input	Input
O0 - O7	Data output	Output
R0 - R10	Control output	Output
OSC1, OSC2	Timing	Input
INIT	Power on reset	Input
VDD, VSS	Power and Ground	

Figure 1-2. TMS1000 Microcomputer Signals And Pin Assignments



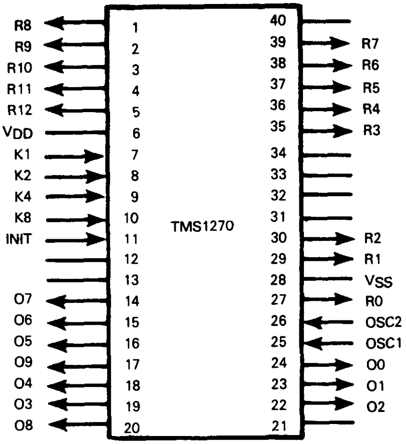
Pin Name	Description	Type
K1, K2, K4, K8	Data input	Input
O0 - O7	Data output	Output
R0 - R12	Control output	Output
OSC1, OSC2	Timing	Input
INIT	Power on reset	Input
VDD, VSS	Power and Ground	

Figure 1-3. TMS1200 Microcomputer Signals And Pin Assignments



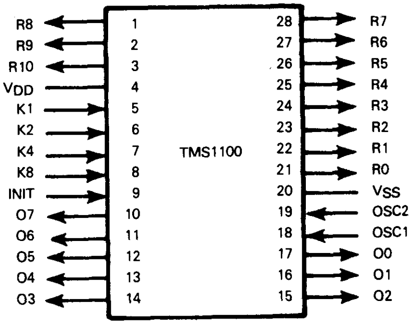
Pin Name	Description	Type
K1, K2, K4, K8	Data input	Input
O0 - O7	Data output	Output
R0 - R10	Control output	Output
OSC1, OSC2	Timing	Input
INIT	Power on reset	Input
VDD, VSS	Power and Ground	

Figure 1-4. TMS1070 Microcomputer Signals And Pin Assignments



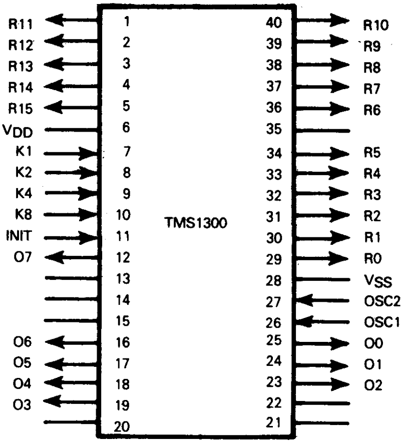
Pin Name	Description	Type
K1, K2, K4, K8	Data input	Input
O0 - O9	Data output	Output
R0 - R12	Control output	Output
OSC1, OSC2	Timing	Input
INIT	Power on reset	Input
VDD, VSS	Power and Ground	

Figure 1-5. TMS1270 Microcomputer Signals And Pin Assignments



Pin Name	Description	Type
K1, K2, K4, K8	Data input	Input
O0 - O7	Data output	Output
R0 - R10	Control output	Output
OSC1, OSC2	Timing	Input
INIT	Power on reset	Input
VDD, VSS	Power and Ground	

Figure 1-6. TMS1100 Microcomputer Signals And Pin Assignments



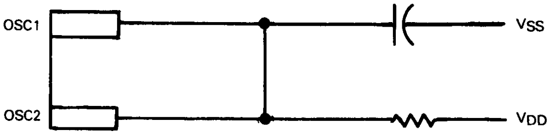
Pin Name	Description	Type
K1, K2, K4, K8	Data input	Input
O0 - O7	Data output	Output
R0 - R15	Control output	Output
OSC1, OSC2	Timing	Input
INIT	Power on reset	Input
V_{DD}, V_{SS}	Power and Ground	

Figure 1-7. TMS1300 Microcomputer Signals And Pin Assignments

The O outputs are provided by O0 - O7, or, in the case of the TMS1270, O0 - O9.

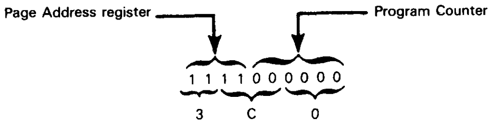
The R outputs occur at R0 - R15, or some smaller number of R outputs, depending on the microcomputer.

OSC1 and OSC2 are timing inputs and outputs. A number of timing options are provided. All TMS1000 series microcomputers contain internal clock logic which you can access in conjunction with an external RC circuit as follows:



You can also input an externally created clock signal at OSC1, in which case OSC2 must be connected to ground (V_{SS}). When you have more than one TMS1000 series microcomputer in a configuration, it is a good idea to synchronize the many microcomputers by driving them with a single clock signal.

INIT is a power on reset signal. Following power on, INIT should be input high (V_{SS}) for at least six consecutive clock cycles. The Reset operation stores binary ones in the Page Address register and the Page Buffer register. The O outputs, the R outputs and the Program Counter are all zeroed. Thus, the first instruction executed will have the hexadecimal address, $3C0_{16}$.



TMS1000 SERIES MICROCOMPUTER INSTRUCTION EXECUTION

No microcomputer described in this book has simpler instruction execution timing than the TMS1000 series. **All instructions generate one byte of object code.** There are no two or three-byte object codes. Similarly, **every instruction executes in a single machine cycle,** as timed by the system clock.

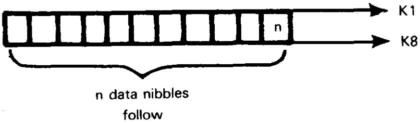
TMS1000 SERIES MICROCOMPUTER INSTRUCTION SET

There are variations in the instruction sets of the different microcomputers in the TMS1000 series. However, **the different instruction sets are similar enough for us to describe them all in Table 1-2.** As compared to similar tables for other microcomputers in this book, Table 1-2 has an additional column which identifies the instructions which are available with each of the TMS1000 series microcomputers.

Within the confines of a single chip microcomputer, the instruction set defined in Table 1-2 is both powerful and effective. It would be easy to point out instruction set features which, from a programmer's point of view, are undesirable; however, the TMS1000 series microcomputers are more oriented to digital logic than other devices described in this book (with the possible exception of the single chip F8). The TMS1000 is not a product that gets programmed, rather its instruction set is a means of defining an optional portion of the ROM mask. Within this context, the instruction set is very adequate. **Note that since you are dealing with a single chip microcomputer, there is nothing to prevent you from redefining the Control Unit, and thus creating your own instruction set.**

THE BENCHMARK PROGRAM

The benchmark program we are using throughout this book in order to exercise the various microcomputer instruction sets is essentially meaningless in any TMS1000 application. Given 64, or at most, 128 nibbles of RAM, the whole concept of moving data among tables is meaningless. We therefore simplify the problem and look upon IOBUF as external logic. Instead of reading from IOBUF, we will input K data. We will assume that each block of K data is preceded by a nibble which defines the number of data nibbles to follow:



Thus each block of data that is input must be fifteen nibbles, or less, in length.

```
        LDX     TBHI    LOAD TABLE PAGE ADDRESS
        TKA             INPUT FIRST K NIBBLE, IT EQUALS DATA NIBBLE TO
                        FOLLOW
        TAY             MOVE TO Y. XY NOW ADDRESSES END OF TABLE
LOOP    TKA             INPUT NEXT DATA NIBBLE
        TAM             SAVE IN MEMORY
        DYN             DECREMENT Y
        BR      LOOP    IF Y NOT 0, RETURN FOR NEXT NIBBLE
```

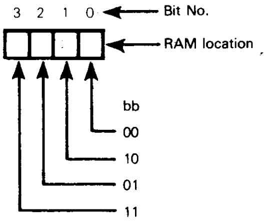
Symbols are used in Table 1-2 as follows:

Registers A — Accumulator
X,Y — Data Counter. Y also serves as an output address.
PC — Program Counter
PA — Page Address register
CF — Chapter Flag (one bit)
SR — Subroutine Return register
PB — Page Buffer

Statuses ST — The Status Flag
C — The status flag reflects a Carry. That is, it is set if there is a Carry from the most significant bit (MSB), and reset otherwise.
NE — The status flag reflects "not equal". That is, it is set if the compared bits are not equal; and reset if they are equal.

Inputs and Outputs:
K — the four input lines
O — the five-bit Output register
R — the control outputs

bb Two bits in the object code which specify one of the four bits of a RAM location:



B Operand which specifies one bit of a RAM location

DATA 2, 3, or 4 bits of immediate data

LABEL Destination of Branch instruction (6 bits of direct address in the object code)

R([Y]) The control output line specified by the contents of the Y register.

x One bit of immediate data or direct address in the object code.

[X](MSB) The most significant bit of the X register

[[X,Y]] The contents of the RAM location addressed by the contents of the Data Counter

[[X,Y]](B) The specified bit of the RAM location addressed by the contents of the Data Counter

[ ] Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If K or R is enclosed within the brackets, then the data at the inputs or control outputs is specified.

← → Data is transferred in the direction of the arrow.

←→ Data is exchanged between the two locations designated on either side of the arrow.

Where two object codes are given, the first is the code used in the TMS1000, TMS1200, TMS1070, and TMS1270, while the second is the object code used in the TMS1100 and TMS1300.

X in one of the rightmost two columns means the instruction is implemented on the designated TMS1000 device.

Table 1-2. TMS1000 Series Instruction Set Summary

TYPE	MNEMONIC	OPERAND	STATUSES		OPERATION PERFORMED	OBJECT CODE	TMS1000 TMS1200 TMS1070 TMS1270	TMS1100 TMS1300
			C	NE				
I/O	KNEZ			X	If [K] ≠ 0, ST ← 1 Set status only if data on input lines is not 0.	09 0E	X	 X
	TKA				[K]→[A] Load Accumulator with data on input lines.	08	X	X
	SETR				R([Y]) ← 1 Set R output addressed by contents of Y.	0D	X	X
	RSTR				R([Y]) ← 0 Reset R output addressed by contents of Y.	0C	X	X
	TDO				[O] ← ([A],ST) Transfer data from Accumulator and status flag to the O outputs.	0A	X	X
	CLO				[O] ← 00_{16} Clear the O Output register.	0B	X	
PRIMARY MEMORY REFERENCE	TAM				[A]→[[X,Y]] Store Accumulator to RAM location addressed by contents of XY Data Counter.	03 27	X	 X
	TMY				[[X,Y]]→[Y] Load Register Y from RAM.	22	X	X
	TMA				[[X,Y]] ← [A] Load Accumulator from RAM.	21	X	X
	XMA				[[X,Y]] ←→ [A] Exchange contents of RAM location addressed by Data Counter XY with those of Accumulator.	2E 03	X	 X
PRIMARY MEMORY REFERENCE WITH REGISTER OPERATE	TAMIY				[A]→[[X,Y]]; [Y] ← [Y] + 1 Store Accumulator to RAM and increment contents of Y register.	20	X	
	TAMIYC		X		[A]→[[X,Y]]; [Y] ← [Y] + 1; ST ← C Store Accumulator to RAM and increment contents of Y register. Set status flag only if there is a carry.	25		X
	TAMDYN		X		[A]→[[X,Y]]; [Y] ← [Y]-1; ST ← C Store Accumulator to RAM and decrement contents of Y register. St status flag only if there is no borrow.	24		X
	TAMZA				[A]→[[X,Y]]; [A] ← 0 Store Accumulator to RAM and then clear Accumulator.	04 26	X	 X

Table 1-2. TMS1000 Series Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND	STATUSES		OPERATION PERFORMED	OBJECT CODE	TMS1000 TMS1200 TMS1070 TMS1270	TMS1100 TMS1300
			C	NE				
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	AMAAC		X		[A]←[[X,Y]] + [A]; ST ← C Add contents of RAM location to those of Accumulator. Set status flag only if there is a carry.	25 06	X	 X
	SAMAN		X		[A]←[[X,Y]]-[A]; ST ← C Subtract Accumulator contents from those of RAM location. Set status flag only if there is no borrow.	27 3C	X	 X
	IMAC		X		[A]←[[X,Y]] + 1; ST ← C Load contents of RAM location to Accumulator and increment. Set status flag only if there is a carry. RAM contents are unchanged.	28 3E	X	 X
	DMAN		X		[A]←[[X,Y]]-1; ST ← C Load contents of RAM location to Accumulator and decrement. Set status flag only if there is no borrow. RAM contents are unchanged.	2A 07	X	 X
	ALEM		X		If [A] ≤ [[X,Y]], ST ← 1 Set status flag only if Accumulator contents are less than or equal to those of RAM location addressed by Data Counter XY.	29 01	X	 X
	MNEA			X	If [[X,Y]] ≠ [A], ST ← 1 Set status flag only if contents of RAM location are not equal to those of Accumulator.	00		X
	MNEZ			X	If [[X,Y]] ≠ 0, ST ← 1 Set status flag only if contents of RAM location are not equal to zero.	26 3F	X	 X
	SBIT	B			[[X,Y]](B) ← 1 Set specified bit of RAM location addressed by contents of Data Counter XY.	001100bb	X	X
	RBIT	B			[[X,Y]](B) ← 0 Reset specified bit of RAM location addressed by contents of Data Counter XY.	001101bb	X	X
	TBIT1	B		X	ST ← [[X,Y]](B) Test specified bit of RAM location and set status flag only if the bit is set.	001110bb	X	X

Table 1-2. TMS1000 Series Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND	STATUSES		OPERATION PERFORMED	OBJECT CODE	TMS1000 TMS1200 TMS1070 TMS1270	TMS1100 TMS1300
			C	NE				
IMMEDIATE	TCY	DATA			[Y] ← DATA Load Register Y immediate.	0100xxxx	X	X
	TCMIY	DATA			[[X,Y]] ← DATA; [Y] ← [Y] + 1 Load RAM location immediate and increment contents of Register Y.	0110xxxx	X	X
	LDX	DATA			[X] ← DATA Load Register X immediate.	001111xx 00101xxx	X 	 X
	LDP	DATA			[PB] ← DATA Load Page Buffer register immediate.	0001xxxx	X	X
IMMEDIATE OPERATE	ALEC	DATA	X		If [A] ≤ DATA, ST ← 1 Set status flag only if Accumulator contents are less than or equal to immediate data.	0111xxxx	X	
	YNEC	DATA		X	If [Y] ≠ DATA, ST ← 1 Set status flag only if contents of Register Y are not equal to immediate data.	0101xxxx	X	X
	A2AAC		X		[A] ← [A] + 2; ST ← C Add 2 to Accumulator contents. Set status flag only if there is a carry.	78		X
	A3AAC		X		[A] ← [A] + 3; ST ← C Add 3 to Accumulator contents. Set status flag only if there is a carry.	74		X
	A4AAC		X		[A] ← [A] + 4; ST ← C Add 4 to Accumulator contents. Set status flag only if there is a carry.	7C		X
	A5AAC		X		[A] ← [A] + 5; ST ← C Add 5 to Accumulator contents. Set status flag only if there is a carry.	72		X
	A6AAC		X		[A] ← [A] + 6; ST ← C Add 6 to Accumulator contents. Set status flag only if there is a carry.	06 7A	X 	 X
	A7AAC		X		[A] ← [A] + 7; ST ← C Add 7 to Accumulator contents. Set status flag only if there is a carry.	76		X
	A8AAC		X		[A] ← [A] + 8; ST ← C Add 8 to Accumulator contents. Set status flag only if there is a carry.	01 7E	X 	 X
	A9AAC		X		[A] ← [A] + 9; ST ← C Add 9 to Accumulator contents. Set status flag only if there is a carry.	71		X
	A10AAC		X		[A] ← [A] + 10; ST ← C Add 10 to Accumulator contents. Set status flag only if there is a carry.	05 79	X 	 X

Table 1-2. TMS1000 Series Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND	STATUSES		OPERATION PERFORMED	OBJECT CODE	TMS1000 TMS1200 TMS1070 TMS1270	TMS1100 TMS1300
			C	NE				
IMMEDIATE OPERATE (CONTINUED)	A11AAC		X		[A]←[A]+11; ST ← C Add 11 to Accumulator contents. Set status flag only if there is a carry.	75		X
	A12AAC		X		[A]←[A]+12; ST ← C Add 12 to Accumulator contents. Set status flag only if there is a carry.	7D		X
	A13AAC		X		[A]←[A]+13; ST ← C Add 13 to Accumulator contents. Set status flag only if there is a carry.	73		X
	A14AAC		X		[A]←[A]+14; ST ← C Add 14 to Accumulator contents. Set status flag only if there is a carry.	7B		X
JUMP	RETN				[PC]←[SR], [PA]←[PB] Return from subroutine.	0F	X	X
BRANCH ON CONDITION	BR	LABEL			If ST = 1, then [PC]← LABEL; outside subroutine, [PA]←[PB] Branch if status flag is set.	10xxxxxx	X	X
	CALL	LABEL			If ST = 1, then [SR]←[PC] + 1, [PB]⟶[PA], [PC]← LABEL Call subroutine if status flag is set. A subroutine call within a subroutine will act as a branch, and load the Page Buffer from the Page Address register: [PC]← LABEL [PB]←[PA]	11xxxxxx	X	X
REGISTER-REGISTER MOVE	TAY				[A]→[Y] Transfer Accumulator contents to Register Y.	24 20	X	 X
	TYA				[Y]→[A] Transfer Register Y contents to Accumulator.	23	X	X
REGISTER-REGISTER OPERATE	YNEA			X	If [Y] ≠ [A], ST ← 1 Set status flag only if contents of Y register are not equal to those of Accumulator.	02	X	X

Table 1-2. TMS1000 Series Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND	STATUSES C	STATUSES NE	OPERATION PERFORMED	OBJECT CODE	TMS1000 TMS1200 TMS1070 TMS1270	TMS1100 TMS1300
REGISTER OPERATE	CLA				[A]←0 Clear Accumulator.	2F 7F	X 	 X
	IA				[A]←[A]+1 Increment Accumulator. No status affected.	0E	X	
	IAC		X		[A]←[A]+1; ST←C Increment Accumulator. Set status flag only if there is a carry.	70		X
	DAN		X		[A]←[A]-1; ST←C Decrement Accumulator. Set status flag only if there is no borrow.	07 77	X 	 X
	IYC		X		[Y]←[Y]+1; ST←C Increment Register Y. Set status flag only if there is a carry.	2B 05	X 	 X
	DYN		X		[Y]←[Y]-1; ST←C Decrement Register Y. Set status flag only if there is no borrow.	2C 04	X 	 X
	CPAIZ		X		[A]←[A]+1; if [A]=0, ST←1 Negate Accumulator contents (twos complement). Set status only if result is zero.	2D 3D	X 	 X
	COMX				[X]←[X] Complement contents of X register (ones complement).	00	X	
	COMX				[X](MSB)←[X](MSB) Complement most significant bit of X register.	09		X
	COMC				CF←CF Complement Chapter flag.	0B		X

DATA SHEETS

TMS 1000/1200 AND TMS 1100/1300 ELECTRICAL AND MECHANICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

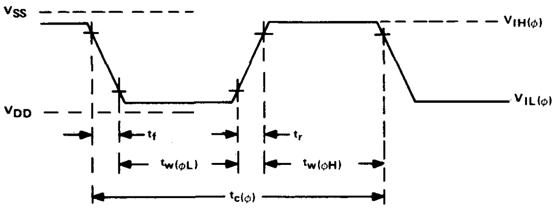
Voltage applied to any device terminal (see Note 1)		−20 V
Supply voltage, V_{DD}		−20 V to 0.3 V
Data input voltage		−20 V to 0.3 V
Clock input voltage		−20 V to 0.3 V
Average output current (see Note 2):	O outputs	−24 mA
	R outputs	−14 mA
Peak output current:	O outputs	−48 mA
	R outputs	−28 mA
Continuous power dissipation:	TMS 1000/1100 NL	400 mW
	TMS 1200/1300 NL	600 mW
Operating free-air temperature range		0°C to 70°C
Storage temperature range		−55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

RECOMMENDED OPERATING CONDITIONS

PARAMETER		MIN	NOM	MAX	UNIT
Supply voltage, V_{DD} (see Note 3)		−14	−15	−17.5	V
High-level input voltage, V_{IH} (see Note 4)	K	−1.3	−1	0.3	V
	INIT or Clock	−1.3	−1	0.3	
Low-level input voltage, V_{IL} (see Note 4)	K	V_{DD}		−4	V
	INIT or Clock	V_{DD}	−15	−8	
Clock cycle time, $t_{c(\phi)}$		2.5	3	10	μs
Instruction cycle time, t_c		15		60	μs
Pulse width, clock high, $t_{w(\phi H)}$		1			μs
Pulse width, clock low, $t_{w(\phi L)}$		1			μs
Sum of rise time and pulse width, clock high, $t_r + t_{w(\phi H)}$		1.25			μs
Sum of fall time and pulse width, clock low, $t_f + t_{w(\phi L)}$		1.25			μs
Oscillator frequency, f_{osc}		100		400	kHz
Operating free-air temperature, T_A		0		70	°C

NOTES: 1. Unless otherwise noted, all voltages are with respect to V_{SS}.
2. These average values apply for any 100-ms period.
3. Ripple must not exceed 0.2 volts peak-to-peak in the operating frequency range.
4. The algebraic convention where the most-positive (least-negative) limit is designated as maximum is used in this specification for logic voltage levels only.



NOTE: Timing points are 90% (high) and 10% (low).

EXTERNALLY DRIVEN CLOCK INPUT WAVEFORM

ELECTRICAL CHARACTERISTICS OVER RECOMMENDED OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS		MIN	TYP†	MAX	UNIT
I_I	Input current, K inputs		$V_I = 0$ V		50	300	500	μA
V_{OH}	High-level output voltage (see Note 1)	O outputs	$I_O = -10$ mA		−1.1‡	−0.6‡		V
		R outputs	$I_O = -2$ mA		−0.75	−0.4		
I_{OL}	Low-level output current		$V_{OL} = V_{DD}$				−100	μA
$I_{DD(av)}$	Average supply current from V_{DD} TMS 1000/1200 (see Note 2)		All outputs open			−6	−10	mA
$I_{DD(av)}$	Average supply current from V_{DD} TMS1100/1300 (see Note 2)		All outputs open			−7	−11	mA
$P_{(AV)}$	Average power dissipation TMS 1000/1200 (see Note 2		All outputs open			90	175	mW
$P_{(AV)}$	Average power dissipation TMS1100/1300 (see Note 2)		All outputs open			105	193	mW
f_{osc}	Internal oscillator frequency		$R_{ext} = 50$ kΩ,	$C_{ext} = 47$ pF	250	300	350	kHz
C_i	Small-signal input capacitance, K inputs		$V_I = 0$,	f = 1 kHz		10		pF
$C_{i(\phi)}$	Input capacitance, clock input		$V_I = 0$,	f = 100 kHz		25		pF

†All typical values are at $V_{DD} = -15$ V, $T_A = 25°$C.

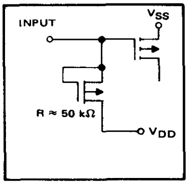
‡Parts with V_{OH} of −2 V minimum, −1.3 V typical, are available if requested.

NOTES: 1. The algebraic convention where the most-positive (least-negative) limit is designated as maximum is used in this specification for logic voltage levels only.

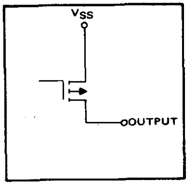
2. Values are given for the open-drain O and R output configurations. Pull-down resistors are optionally available on all outputs and increase I_{DD}.

SCHEMATICS OF INPUTS AND OUTPUTS

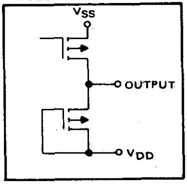
TYPICAL OF ALL K INPUTS



TYPICAL OF ALL O AND R OPEN-DRAIN OUTPUTS



TYPICAL OF ALL O AND R OUTPUTS WITH OPTIONAL PULL-DOWN RESISTORS



The O outputs have nominally 60 Ω on-state impedance; however, upon request a 130-Ω buffer can be mask programmed,

The value of the pull-down resistors is mask alterable and provides the following nominal short-circuit output currents (outputs shorted to V_{SS}):

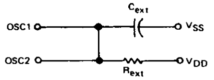
O outputs: 100, 200, 300, 500, or 900 μA

R outputs: 100, 150, or 200 μA.

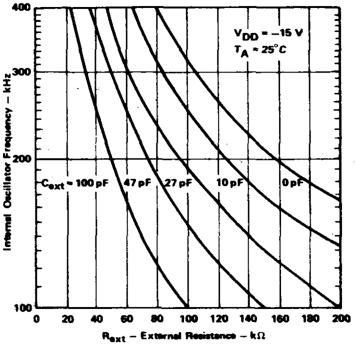
NTERNAL OR EXTERNAL CLOCK

If the internal oscillator is used, the OSC1 and OSC2 terminals are shorted together and tied to an external resistor to V_{DD} and a capacitor to V_{SS}. If an external clock is desired, the clock source may be connected to OSC1 and OSC2 shorted to V_{SS}.

CONNECTION FOR INTERNAL OSCILLATOR

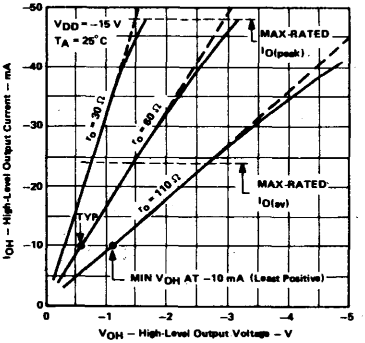


TYPICAL INTERNAL OSCILLATOR FREQUENCY
vs
EXTERNAL RESISTANCE

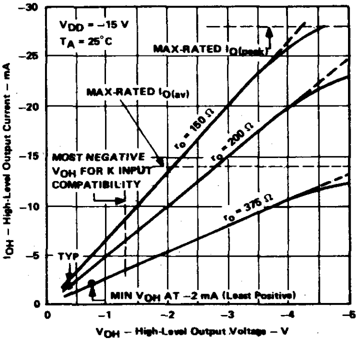


TYPICAL BUFFER CHARACTERISTICS

O OUTPUTS
HIGH-LEVEL OUTPUT CURRENT
vs
HIGH-LEVEL OUTPUT VOLTAGE



R OUTPUTS
HIGH-LEVEL OUTPUT CURRENT
vs
HIGH-LEVEL OUTPUT VOLTAGE



ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Voltage applied to any device terminal (see Note 1)		−20 V
Supply voltage, V_{DD}		−20 V to 0.3 V
Data input and output voltage with V_{DD} applied (see Note 2)		−35 V to 0.3 V
Clock input and INIT input voltage		−20 V to 0.3 V
Average output current (see Note 3):	O outputs	−2.5 mA
	R outputs	−12 mA
Peak output current:	O outputs	−5 mA
	R outputs	−24 mA
Continuous power dissipation:	TMS 1070 NL	400 mW
	TMS 1270 NL	600 mW
Operating free-air temperature range		0°C to 70°C
Storage temperature range		−55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

RECOMMENDED OPERATING CONDITIONS

PARAMETER		MIN	NOM	MAX	UNIT
Supply voltage, V_{DD} (see Note 4)		−14	−15	−17.5	V
High-level input voltage, V_{IH} (see Note 5)	K	−6		0.3	V
	INIT or Clock	−1.3	−1	0.3	
Low-level input voltage, V_{IL} (see Note 5)	K (See Note 2)	−35		−8	V
	INIT or Clock	V_{DD}	−15	−8	
Clock cycle time, $t_{c(\phi)}$		2.5	3	10	μs
Instruction cycle time, t_c		15		60	μs
Pulse width, clock high, $t_{w(\phi H)}$		1			μs
Pulse width, clock low, $t_{w(\phi L)}$		1			μs
Sum of rise time and pulse width, clock high, $t_r + t_{w(\phi H)}$		1.25			μs
Sum of fall time and pulse width, clock low, $t_f + t_{w(\phi L)}$		1.25			μs
Oscillator frequency, f_{osc}		100		400	kHz
Operating free-air temperature, T_A		0		70	°C

NOTES:
1. Unless otherwise noted, all voltages are with respect to V_{SS}.
2. V_{DD} must be within the recommended operating conditions specified in 5.4.
3. These average values apply for any 100-ms period.
4. Ripple must not exceed 0.2 volts peak-to-peak in the operating frequency range.
5. The algebraic convention where the most-positive (least-negative) limit is designated as maximum is used in this specification for logic voltage levels only.

ELECTRICAL CHARACTERISTICS OVER RECOMMENDED OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS		MIN	TYP†	MAX	UNIT
I_I	Input current, K inputs		$V_I = 0$ V		40	100	300	μA
V_{OH}	High-level output voltage (see Note 1)	O outputs	$I_O = -1$ mA		−1	−0.5		V
		R outputs	$I_O = -10$ mA		−4.5	−2.25		
I_{OL}	Low-level output current		$V_{OL} = V_{DD}$				−100	μA
$I_{DD(av)}$	Average supply current from V_{DD}		All outputs open			−6	−10	mA
$P_{(AV)}$	Average power dissipation		All outputs open			90	175	mW
f_{osc}	Internal oscillator frequency		$R_{ext} = 50$ kΩ,	$C_{ext} = 47$ pF	250	300	350	kHz
C_i	Small-signal input capacitance, K inputs		$V_I = 0$ V,	f = 1 kHz		10		pF
$C_{i(\phi)}$	Input capacitance, clock input		$V_I = 0$ V,	f = 100 kHz		25		pF

†All typical values are at $V_{DD} = -15$ V, $T_A = 25°C$.

NOTE 1: The algebraic convention where the most-positive (least-negative) limit is designated as maximum is used in this specification for logic voltage levels only.

Chapter 2
THE FAIRCHILD F8

The F8 has had a profound impact on the microcomputer industry. When it first appeared, the F8 was dismissed as an off-beat product with a strange set of chips and a ridiculous instruction set. The chip set was strange because logic was organized with the goal of minimizing chip counts; in contrast, microprocessors such as the 8080 and 6800 were designed with logic distributed functionally on chips — one traditional CPU logic function per chip. The F8 instruction set is indeed strange, and in some cases quite limiting, but it reflects the simple chip design of the F8 CPU.

The vast majority of microprocessors are now going into consumer products. In this market place the two-chip F8 system provided by a 3850 CPU and 8351 PSU quickly came to dominate; any other microcomputer system requires 7 or more chips to provide the same variety of logic functions as the two-chip F8 system. The economics of consumer product volumes render the inefficiencies of the F8 nstruction set inconsequential, with the result that in 1977 the F8 will be the world's leading microprocessor — in terms of CPU sales. The 8080A still has the argest number of individual users.

n recognition of the F8 success story, Intel has created the 8048 — a one-chip microcomputer described in Chapter 6 of this book.

F8 devices described in this chapter:

THE FAIRCHILD F8 DEVICE SET

The 3850 CPU
The 3851 Programmable Storage Unit (PSU), which provides Read-Only Memory, plus various additional logic functions.
The 3852 Dynamic Memory Interface (DMI), which primarily provides interface logic for dynamic or static read-write memory.
The 3853 Static Memory Interface (SMI), which primarily provides interface logic for static read/write memory.
The 3854 Direct Memory Access (DMA), which, in conjunction with the 3852 DMI, implements Direct Memory Access logic.
The 3856 and 3857 16K Programmable Storage Units (PSU 16), which are variations of the 3851 PSU, but provide more Read-Only Memory.

We are also going to look at the 3859 and 3870 single chip microcomputers, which are really an alternative to, rather than a member of the 3850 device set.

Figure 2-1 illustrates a 3850 CPU based F8 microcomputer system.

All devices of the F8 family require +5V and +12V power supplies.

Using a 500 ns clock, instruction cycle time is 2 μsec. Instruction execution times range from 1 to 6.5 instruction cycles, or 2 to 13 μsec.

N-channel isoplanar MOS technology is used for the F8.

The principal manufacturer for the F8 is:

FAIRCHILD SEMICONDUCTOR
464 Ellis Street
Mountain View, California 94040

The second source is:

MOSTEK, INC.
P.O. Box 169
Carrollton, Texas 75006

THE 3850 CPU

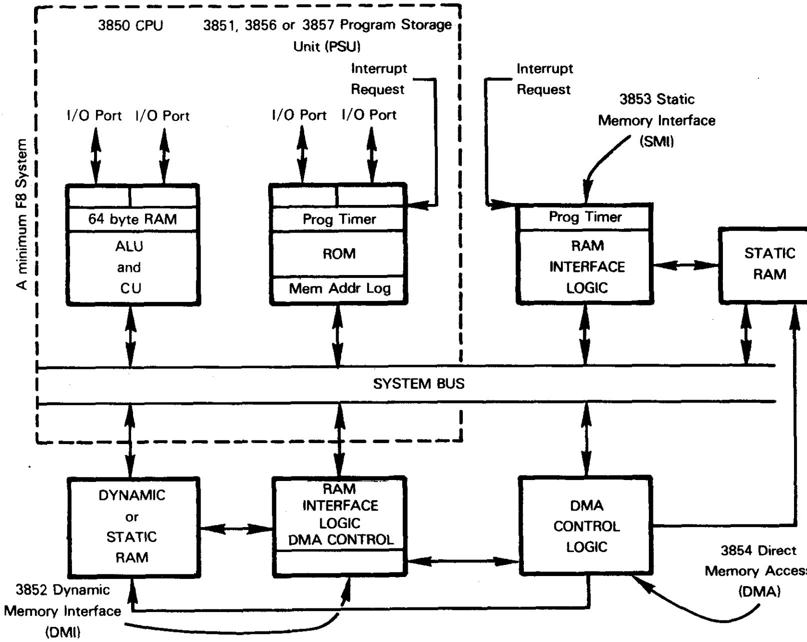
Functions implemented on the 3850 CPU are illustrated in Figure 2-2.

These are the functions which one would expect to find on a CPU chip, and whic are on the 3850 CPU:

- The Arithmetic and Logic Unit
- The Control Unit and Instruction Register
- Logic needed to interface the System Bus with the control signals which are input an output by the CPU.
- Accumulator register

There is no memory addressing logic, and there are no memory addressin registers on the 3850 CPU. Stack Pointer, Program Counter and Data Counte registers are all maintained on memory chips, and memory interface chips.

With the Fairchild scheme, memory addressing logic will be duplicated if mor than one memory device is present in an F8 microcomputer system. We will discuss shortly how potential address contention problems are resolved under these circumstances.



A maximum of 65,536 bytes of memory may be present in an F8 microcomputer system.

Figure 2-1. A Fairchild F8 Microcomputer System

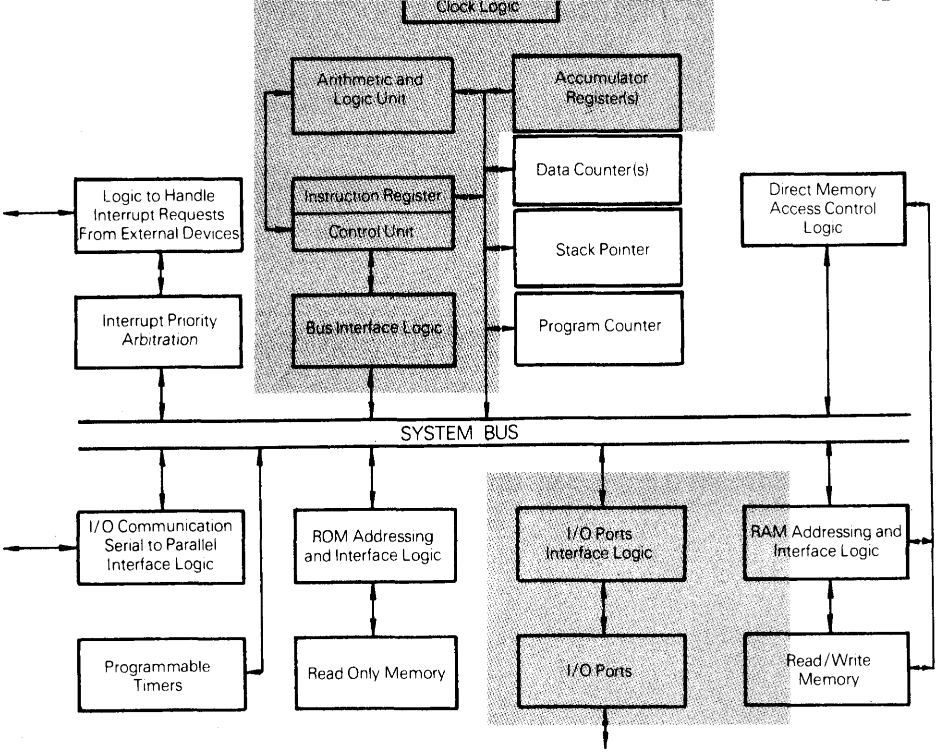

Clock Logic
Arithmetic and Logic Unit
Accumulator Register(s)
Data Counter(s)
Logic to Handle Interrupt Requests From External Devices
Instruction Register
Control Unit
Direct Memory Access Control Logic
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 2-2. Logic Of The Fairchild F8 3850 CPU

Two advantages accrue from having no memory address logic on the CPU chip:

1) No address lines are needed on the System Bus, so neither the CPU nor connectin devices need 16 address pins. These 16 pins are used instead for two 8-bit I/ ports at each device.
2) The real estate on the CPU chip which would have been used by Address registe and memory addressing logic is available for other purposes; it is used to implement 64 bytes of read-write memory.

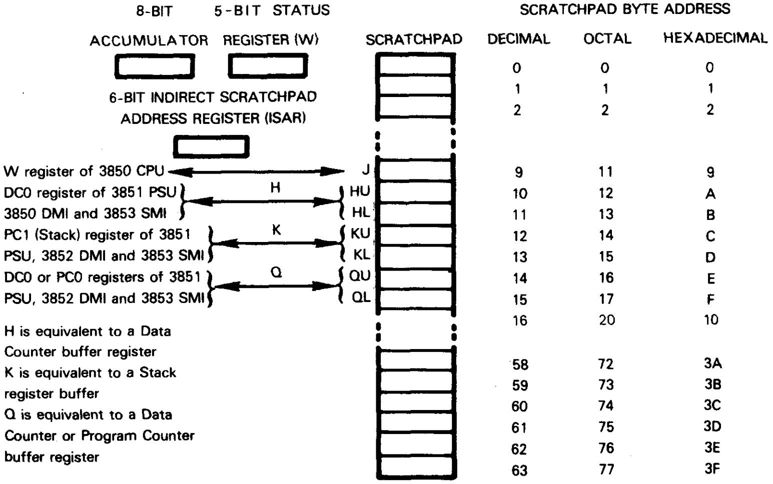
Having I/O ports and read-write memory on the CPU chip paves the way for som very low-cost small microcomputer configurations; for example, the 8350 CPU an the 3851 PSU form a two-device microcomputer system, with all of the necessary prerequisites for reasonable performance. The 3859 and 3870 single chip microcompute are simply a combination of 3850 CPU and 3851 PSU — with minor enhancements.

The disadvantage of removing memory addressing logic from the CPU chip is tha memories can no longer connect directly to the System Bus; this bus has no address lines; therefore, separate logic devices must create the interface needed by standard memories. In the F8 system this is done by the 3852 DMI and the 3853 SM devices.

Clock signal generation logic is also part of the 3850 CPU. This is now standar among microcomputers. Depending on the signal timing accuracy demanded by an application, either a crystal or a resistor-capacitor network can be connected at 3850 input pins; clock logic generates the timing signals required by other devices within th F8 microcomputer system. There is also the option of slaving a 3850 CPU by receivin timing signals from some other source, bypassing 3850 CPU clock logic.

F8 PROGRAMMABLE REGISTERS

These are the programmable registers of the F8 CPU:


8-BIT ACCUMULATOR
5-BIT STATUS REGISTER (W)
6-BIT INDIRECT SCRATCHPAD ADDRESS REGISTER (ISAR)
SCRATCHPAD
W register of 3850 CPU
J
DC0 register of 3851 PSU 3850 DMI and 3853 SMI
H
HU
HL
PC1 (Stack) register of 3851 PSU, 3852 DMI and 3853 SMI
K
KU
KL
DC0 or PC0 registers of 3851 PSU, 3852 DMI and 3853 SMI
Q
QU
QL
H is equivalent to a Data Counter buffer register
K is equivalent to a Stack register buffer
Q is equivalent to a Data Counter or Program Counter buffer register

SCRATCHPAD BYTE ADDRESS		
DECIMAL	OCTAL	HEXADECIMAL
0	0	0
1	1	1
2	2	2
9	11	9
10	12	A
11	13	B
12	14	C
13	15	D
14	16	E
15	17	F
16	20	10
58	72	3A
59	73	3B
60	74	3C
61	75	3D
62	76	3E
63	77	3F

F8 ACCUMULATOR

There is one 8-bit Accumulator, which may be likened to the Primary Accumulator (A0) of our hypothetical microcomputer. Wherever there is a choice, this Accumulator is the usual source or destination for data operations associated with any instruction's execution.

F8 SCRATCHPAD

The 64-byte scratchpad may be viewed either as a small read-write memory, or as 64, 8-bit secondary Accumulators. The first 11 scratchpad bytes may be accessed directly, as though they were secondary Accumulators. Remaining RAM bytes can only be accessed using a form of implied memory addressing, where a 6-bit register (identified as the ISAR register) must provide the address of the byte being accessed. The ISAR register is in every way identical to a 6-bit Data Counter.

There is no Program Counter, and there are no Data Counters on the 3850 CPU. However, as illustrated above, 6 of the 64 scratchpad bytes communicate directly with Program Counter and Data Counter registers on other devices of the F8 microcomputer system. What this means is that the F8 provides register-to-register Move instructions that move data between scratchpad bytes and Data Counters or Program Counters on other devices.

These Address registers are implemented on the 3851 PSU, 3852 DMI and 3853 SMI devices:

16 Bits	Data Counter DC0
16 Bits	Data Counter buffer DC1
16 Bits	Program Counter PC0
16 Bits	Stack register PC1

F8 DATA COUNTERS

Data Counter DC0 is an implied addressing register, as described for our hypothetical microcomputer.

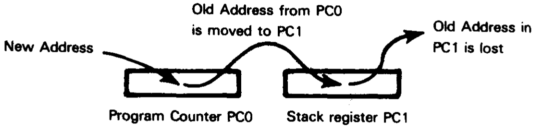
Data Counter DC1 is simply a buffer for the contents of Data Counter DC0. Implied addressing via Data Counter DC1 is not allowed. The only instruction that accesses Data Counter DC1 is an instruction which will exchange the contents of Data Counters DC0 and DC1. Data Counter DC1 is not present in the 3851 PSU.

F8 PROGRAM COUNTER

Program Counter PC0 serves the same function in an F8 system as it does in our hypothetical microcomputer.

F8 STACK REGISTER

The Stack register (PC1) is, in reality, a buffer for Program Counter PC0; the Stack register does not address an area in read-write memory, and there are no Push or Pop instructions, as described in Volume I, Chapter 6. Interrupts and Jump-to-Subroutine instructions save the contents of Program Counter PC0 in Stack register PC1, before loading a new address into Program Counter PC0:



The classical Stack can be implemented in an F8 system, but a short program needs to be written to do this.

F8 ADDRESS REGISTERS DUPLICATION

Since address registers are present on every PSU, DMI or SMI device in an F8 microcomputer system, these registers will be duplicated in any F8 system that contains more than a minimum amount of memory. So long as the microcomputer system has been correctly configured, this presents no problem. Every memory device contains identical connections to the common System Bus, and instructions that modify the contents of any address register, do so identically for all memory devices. For example, if there are three memory devices, and therefore three Program Counters in an F8 system, every Program Counter is incremented identically after every byte of object code is fetched. This being the case, address registers on different memory devices will always contain identical address ir formation.

When there are two or more memory devices in an F8 system, which will respond to a memor reference instruction?

Every memory device has its own permanently defined address space. An address space is range of memory addresses to which one, and only one device will respond. So long as the ac dress spaces of memory devices within the F8 system do not overlap (and if they do that is a logi design error) there is no chance of memory contentions arising, either for read or write operation

But there is one exception. Since the 3851 PSU does not contain a DC1 register, it does not res pond to an instruction that exchanges the contents of DC0 and DC1 registers. If such an instruc tion is executed, it is possible for the DC0 registers of the 3851 PSUs to contain different addres information from the DC0 registers of 3852 and/or 3853 devices. Each of the different addresse can be within the address space of the 3851 PSU, or the DMI/SMI devices, respectively. Shoul this situation arise, both devices will attempt to respond to a memory reference instructior Eliminating this contention is simply a question of ensuring that instructions that exchange Dat Counter contents are executed in pairs.

F8 MEMORY ADDRESSING MODES

Different memory addressing options are provided for scratchpad memory, and fo memory external to the 3850 CPU.

It is important to note that scratchpad memory byte addresses have no connec tion with external memory byte addresses.

As you will see in Table 2-2, there are different instructions to reference scratchpad bytes and external memory.

A scratchpad reference instruction must identify one scratchpad byte via an address in the range 0 through 63.

A memory reference instruction identifies an external memory byte via an address in the range o 0 through 65,535.

A scratchpad reference instruction cannot reference memory, and a memory referencing instruc tion cannot reference a scratchpad byte; therefore, the fact that memory bytes can have ad dresses in the range 0 through 63 will not cause confusion. If this point confuses you, just im agine the Scratchpad as 64 Accumulators; the hypothetical microcomputer described in Volume I, Chapter 7 has two Accumulators, but it can still address memory locations 0000 and 0001; the F8 has 64 scratchpad Accumulators which does not prevent it from addressing memory location 0000 through $003F_{16}$.

EXTERNAL MEMORY ADDRESSING

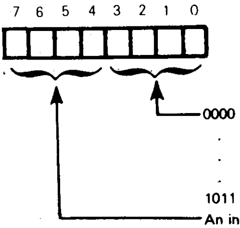
External memory is always addressed using implied addressing, with auto-increment, via Data Counter DC0. No other external memory addressing modes are provided.

There are a number of instructions which load immediate data into Data Counter DC0; data may also be transferred between Data Counter DC0 and scratchpad bytes, an it is possible to add the contents of the Accumulator to Data Counter DC0.

SCRATCHPAD MEMORY ADDRESSING

In order to understand scratchpad addressing, one has to view it as neither representing 64 Accumulators, nor 64 bytes of read-write memory, but rather something in between the two.

There are a number of register-register instructions that operate on the Accumulator, and one of the first 12 scratchpad bytes using object code as follows:



One scratchpad byte from bytes 0 through 11 is specified

An instruction that accesses the Accumulator and one of the scratchpad bytes, is specified

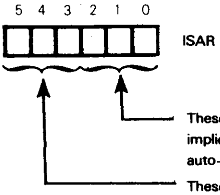
DIRECT SCRATCHPAD ADDRESSING

This type of object code treats the first 12 scratchpad bytes as secondary Accumulators.

Any scratchpad byte may be addressed via the ISAR register, using implied addressing; that is to say, the 6-bit number in the ISAR (which can have a value in the range 0 through 63) identifies the one scratchpad byte which will be accessed by the next scratchpad referencing instruction.

IMPLIED SCRATCHPAD ADDRESSING

The ISAR register provides implied addressing, and implied addressing with auto-increment or auto-decrement; however, only the low order three bits of the ISAR register are involved in the auto-increment or auto-decrement operation:



These three bits may be incremented or decremented by an implied addressing scratchpad memory reference with auto-increment/decrement.

These three bits are not affected by an auto-increment or an auto-decrement operation.

F8 scratchpad bytes may therefore be accessed, as contiguous, 8-byte buffers, with wraparound auto-increment or auto-decrement within each 8-byte buffer.

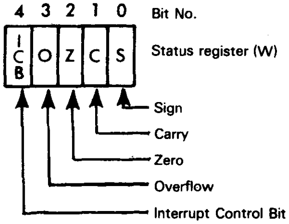
r SCRATCHPAD ADDRESSING

F8 instructions, as shown in Table 2-2, use the symbol r in the operand to represent scratchpad addressing. This is what the symbol r represents:

- If r is a number between 0 and 11, one of scratchpad bytes 0 through 11 is addressed directly.
- If r is S, implied addressing via ISAR is specified.
- If r is I, implied addressing via ISAR, with auto-increment of the low order three implied address bits is specified.
- If r is D, implied addressing via ISAR, with auto-decrement of the low order three address bits is specified.

FAIRCHILD F8 STATUS FLAGS

There is a Status register, also called the W register, which holds five status flags as follows:



The O, Z, C and S status flags are identical to the flags with equivalent symbols, as described in Volume I, Chapter 6 for our hypothetical microcomputer.

The Interrupt Control Bit is treated as a fifth status; this status will not be modified by arithmetic or logic operations but it will be transferred, as a unit with the other four status flags, to or from Scratchpad byte 0.

F8 CPU PINS AND SIGNALS

3850 CPU pins and signals are illustrated in Figure 2-3. A description of these signals is useful as a guide to the way in which the F8 microcomputer system works.

The Data Bus lines (DB0 - DB7) and the control lines (ROMC0 - ROMC4) provide the heart of all data and control information flow.

The Data Bus lines are common, bidirectional lines, and are the only conduit for data to be transmitted between devices of an F8 microcomputer system.

A lack of address lines on the System Bus usually means that data and addresses must be multiplexed on a single set of eight lines — which slows down all memory reference operations; they must now proceed in three serial increments, rather than in one parallel increment. In the F8 System Bus, multiplexing is rarely needed, since addresses originate within memory devices, or memory interface devices, whence they are transmitted directly to memory. In other words, the only time addresses are ever transmitted on the Data Bus is when they are being transmitted as data.

Refer to Figure 2-1. Suppose a memory reference instruction needs to access a byte of dynamic RAM. ROMC control signals (described in the next paragraph) specify that the memory byte whose address is implied by the Data Counters (DC0) is to be referenced. Every memory device receives the ROMC control signals, but only the 3852 DMI finds that its address space includes the Data Counter implied address; therefore, only the 3852 DMI will respond to the memory reference instruction. The 3852 DMI then outputs an address directly to dynamic RAM; this address is not transmitted via the System Bus. If the memory reference instruction requires data to be input to, or output from dynamic RAM, the data transfer occurs directly between the System Bus and Dynamic RAM, bypassing the 3852 DMI entirely.

ROMC STATE

Since the 3851 PSU, the 3852 DMI and the 3853 SMI devices all contain address registers and address generation logic, they also contain rudimentary arithmetic and logic units and control units equivalent to very primitive CPUs. **Every instruction's execution causes the 3850 CPU to output a sequence of signals on the five control lines, ROMC0 - ROMC4.** Each five-bit combination of ROMC signal states identifies one of 32 possible operations which the memory devices may have to perform to accomplish one step of an instruction's execution. For example, ROMC state 00000 causes the contents of the memory byte addressed by the Program Counter to be transmitted to the CPU; this is the "instruction fetch" ROMC state. Table 2-1 summarizes the interpretation of ROMC states.

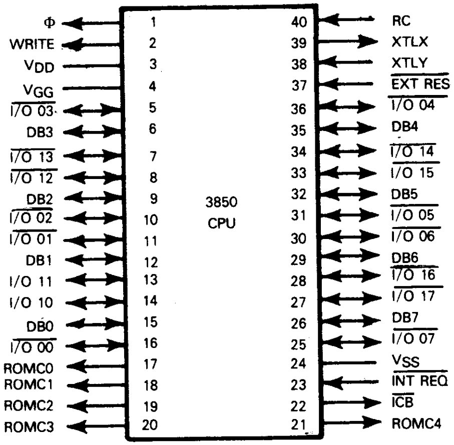
Φ and WRITE are two timing signals output by the 3850 CPU to synchronize events within the rest of the F8 system.

The $\overline{\textbf{EXT RES}}$ line disables interrupts and loads a 0 address into the Program Counters, causing program execution to restart with the instruction code stored in external memory byte 0.

$\overline{\textbf{INT REQ}}$ and $\overline{\textbf{ICB}}$ are signals used for overall interrupt control. $\overline{\text{INT REQ}}$ is the master line on which all interrupt requests are transmitted to the 3850 CPU. $\overline{\text{ICB}}$ is output low by the CPU if interrupts are enabled, and it is output high by the CPU if interrupts are disabled.

RC, XTLX and XTLY are the three pins used for clock inputs. These three pins may be connected to an external resistor-capacitor network, an external crystal or to a single external timing signal.

The two I/O ports which are part of the 3850 CPU device use pins I/O00 - I/O07 and I/O10 - I/O17, respectively.



Pin Name	Description	Type
*DB0 - DB7	Data Bus Lines	Bidirectional
*Φ, WRITE	Clock Lines	Output
$\overline{\text{I/O 00 - I/O 07}}$	I/O Port Zero	Bidirectional
$\overline{\text{I/O 10 - I/O 17}}$	I/O Port One	Bidirectional
*ROMC0 - ROMC4	Control Lines	Output
*$\overline{\text{EXT RES}}$	External Reset	Input
*$\overline{\text{INT REQ}}$	Interrupt Request	Input
*$\overline{\text{ICB}}$	Interrupt Control Bit	Output
RC	Clock Oscillator	Input
XTLX	Crystal Clock Line	Output
XTLY	External Clock Line	Input
VSS, VDD, VGG	Power Lines	

*These signals connect to the System Bus.

Figure 2-3. Fairchild 3850 CPU Sgnals And Pin Assignments

Table 2-1. ROMC Signals And What They Imply

ROMC						CYCLE	FUNCTION
4	3	2	1	0	HEX	LENGTH	
0	0	0	0	0	00	S,L	Instruction Fetch. The device whose address space includes the contents of the PC0 register must place on the Data Bus the op code addressed by PC0. Then all devices increment the contents of PC0.
0	0	0	0	1	01	L	The device whose address space includes the contents of the PC0 register must place on the Data Bus the contents of the memory location addressed by PC0. Then all devices add the 8-bit value on the Data Bus, as a signed binary number, to PC0.
0	0	0	1	0	02	L	The device whose DC0 addresses a memory word within the address space of that device must place on the Data Bus the contents of the memory location addressed by DC0. Then all devices increment DC0.
0	0	0	1	1	03	L,S	Similar to 00, except that it is used for Immediate Operand fetches (using PC0) instead of instruction fetches.
0	0	1	0	0	04	S	Copy the contents of PC1 into PC0.
0	0	1	0	1	05	L	Store the Data Bus contents or write bus contents into the memory location pointed to by DC0. Increment DC0.
0	0	1	1	0	06	L	Place the high order byte of DC0 on the Data Bus.
0	0	1	1	1	07	L	Place the high order byte of PC1 on the Data Bus.
0	1	0	0	0	08	L	All devices copy the contents of PC0 into PC1. The CPU outputs zero on the Data Bus in this ROMC state. Load the Data Bus into both halves of PC0 thus clearing the register.
0	1	0	0	1	09	L	The device whose address space includes the contents of the DC0 register must place the low order byte of DC0 onto the Data Bus.
0	1	0	1	0	0A	L	All devices add the 8-bit value on the Data Bus, treated as a signed binary number, to the Data Counter.
0	1	0	1	1	0B	L	The device whose address space includes the value in PC1 must place the low order byte of PC1 on the Data Bus.
0	1	1	0	0	0C	L	The device whose address space includes the contents of the PC0 register must place the contents of the memory word addressed by PC0 onto the Data Bus. Then all devices move the value which has just been placed on the Data Bus into the low order byte of PC0.
0	1	1	0	1	0D	S	All devices store in PC1 the current contents of PC0, incremented by 1. PC0 is unaltered.
0	1	1	1	0	0E	L	The device whose address space includes the contents of PC0 must place the contents of the word addressed by PC0 onto the Data Bus. The value on the Data Bus is then moved to the low order byte of DC0 by all devices.
0	1	1	1	1	0F	L	The interrupting device with highest priority must place the low order byte of the interrupt vector on the Data Bus. All devices must copy the contents of PC0 into PC1. All devices must move the contents of the Data Bus into the low order byte of PC0.
1	0	0	0	0	10	L	Inhibit any modification to the interrupt priority logic.
1	0	0	0	1	11	L	The device whose memory space includes the contents of PC0 must place the contents of the addressed memory word on the Data Bus. All devices must then move the contents of the Data Bus to the upper byte of DC0.
1	0	0	1	0	12	L	All devices copy the contents of PC0 into PC1. All devices then move the contents of the Data Bus into the low order byte of PC0.

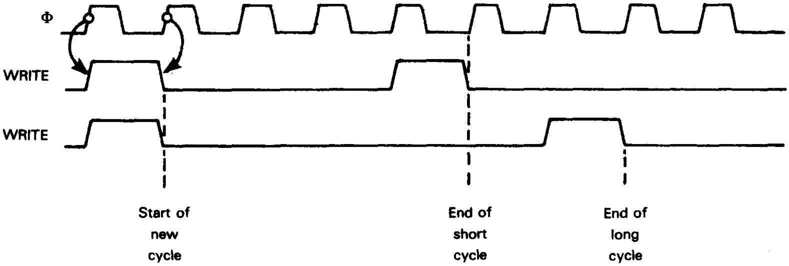
Table 2-1. ROMC Signals And What They Imply (Continued)

ROMC					HEX	CYCLE LENGTH	FUNCTION
4	3	2	1	0			
1	0	0	1	1	13	L	The interrupting device with highest priority must move the high order half of the interrupt vector onto the Data Bus. All devices must move the contents of the Data Bus into the high order byte of PC0. The interrupting device will reset its interrupt circuitry (so that it is no longer requesting CPU servicing and can respond to another interrupt).
1	0	1	0	0	14	L	All devices move the contents of the Data Bus into the high order byte of PC0.
1	0	1	0	1	15	L	All devices move the contents of the Data Bus into the high order byte of PC1.
1	0	1	1	0	16	L	All devices move the contents of the Data Bus into the high order byte of DC0.
1	0	1	1	1	17	L	All devices move the contents of the Data Bus into the low order byte of PC0.
1	1	0	0	0	18	L	All devices move the contents of the Data Bus into the low order byte of PC1.
1	1	0	0	1	19	L	All devices move the contents of the Data Bus into the low order byte of DC0.
1	1	0	1	0	1A	L	During the prior cycle an I/O port timer or interrupt control register was addressed. The device containing the addressed port must move the current contents of the Data Bus into the addressed port.
1	1	0	1	1	1B	L	During the prior cycle the Data Bus specified the address of an I/O port. The device containing the addressed I/O port must place the contents of the I/O port on the Data Bus. (Note that the contents of timer and interrupt control registers cannot be read back onto the Data Bus.)
1	1	1	0	0	1C	L or S	None.
1	1	1	0	1	1D	S	Devices with DC0 and DC1 registers must switch registers. Devices without a DC1 register perform no operation.
1	1	1	1	0	1E	L	The device whose address space includes the contents of PC0 must place the low order byte of PC0 onto the Data Bus.
1	1	1	1	1	1F	L	The device whose address space includes the contents of PC0 must place the high order byte of PC0 on the Data Bus.

F8 TIMING AND INSTRUCTION EXECUTION

All instructions are executed in cycles, which are timed by the trailing edge of WRITE.

There are two types of instruction cycle, the short cycle which is four Φ clock periods long, **and the long cycle** which is six Φ clock periods long. The long cycle is sometimes referred to as 1.5 cycles. WRITE high appears only at the end of an instruction cycle. Timing may be illustrated as follows:



The simplest instructions of the F8 instruction set execute in one short cycle. The most complex instruction (PI) requires two short cycles plus three long cycles.

Table 2-3 summarizes the sequence in which short (S) and long (L) machine cycles are executed for each F8 instruction. ROMC states defining operations performed during each machine cycle are summarized in Table 2-1.

A SUMMARY OF F8 INTERRUPT PROCESSING

The interrupt handling capabilities of the F8 system are described with the 3851 PSU and 3853 SMI devices. Although many different interrupt priority arbitration schemes could be implemented, the simplest scheme would be to daisy chain 3851 PSUs, terminating the daisy chain with a 3853 SMI if present.

As soon as an interrupt is acknowledged, the contents of Program Counters (PC0) are saved in Stack registers (PC1); then an interrupt vector address is loaded into the Program Counters. This address is a permanent mask option for PSUs, with the exception of bit 7, which discriminates between timer interrupts and external interrupts. The interrupt address vector is completely programmable for the 3853 SMI, again with the exception of bit 7, which discriminates between timer interrupts and external device interrupts.

Post-interrupt housekeeping operations must be handled via an appropriate program. Defining just what this program consists of is not simple; an F8 system has only the Accumulator and Status register which must be saved, but at the other extreme, it has the entire scratchpad which could be saved.

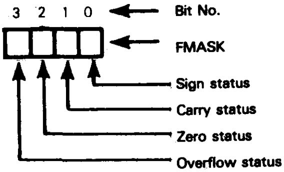
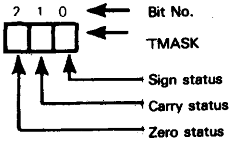
THE F8 INSTRUCTION SET

Table 2-2 summarizes the F8 instruction set; instructions are grouped into categories that conform with our hypothetical microcomputer instruction set, as described in Volume I, Chapter 7.

With reference to Table 2-2, refer to the F8 addressing modes description for an explanation of "r", which occurs in the operand column to represent some of the scratchpad addressing options.

One of the more confusing aspects of F8 programming is to understand the ways in which data may be moved between different registers; this information is therefore summarized in Figure 2-4.

The following symbols are used in Table 2-2:

A	The Accumulator
ADDR	A 16-bit memory address
C	Carry status
DATA3	A 3-bit binary data unit
DATA4	A 4-bit binary data unit
DATA5	A 5-bit binary data unit
DC0	Data Counter register
DC1	Data Counter buffer
DPCHR	Scratchpad Data or Program Counter Half Registers. These are KU (Register 12), KL (Register13), QU (Register14) and QL (Register15).
DISP	An 8-bit signed binary address displacement.
FMASK	A 4-bit mask composed of a portion of the Status register (W):  3 2 1 0 Bit No. FMASK Sign status Carry status Zero status Overflow status
H	Scratchpad Data Counter Register H (Registers 10 and 11).
I	The Interrupt Control Bit in the Status register (W).
ISAR	Indirect Scratchpad Address Register
J	Scratchpad Register 9
K	Scratchpad Registers 12 and 13
O	Overflow status
P4	A 4-bit I/O port number
P8	An 8-bit I/O port number
PC0	Program Counter
PC1	Stack register
Q	Scratchpad Registers 14 and 15
r	Any of the following operands and Scratchpad addressing modes: R direct address of bytes 0 through 11 S implied addressing via ISAR I implied addressing via ISAR, with auto-increment of the low order three ISAR bits D implied addressing via ISAR, with auto-decrement of the low order three ISAR bits
R	A Scratchpad register number in the range 0 - 11
S	Sign status
SR	The register specified by the r argument
TMASK	A 3-bit mask composed of a portion of the Status register (W):  2 1 0 Bit No. TMASK Sign status Carry status Zero status

W The CPU Status register

Z Zero status

x<y,z> Bits y through z of the quantity x. For example, A<3,0> represents the low order fou bits of the Accumulator; ADDR<15,8> represents the high order 8 bits of a 16-bi memory address.

[ ] Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If an I/O por number is enclosed within the brackets, then the I/O port contents are specified. If a memory address is enclosed within the brackets, then the contents of the addresse memory location are specified.

[[ ]] Implied memory addressing; the contents of the memory location or register designate by the contents of a register.

Λ Logical AND .

V Logical OR

∀ Logical Exclusive OR

← Data is transferred in the direction of the arrow.

←→ Data is exchanged between the two locations designated on either side of the arrow.

Under the heading of STATUSES in Table 2-2, an X indicates statuses which are modified in the course of the instructions' execution. If there is no X, it means that the status maintains the value it had before the instruction was executed. A 0 or 1 means the status is cleared or set, respectively.

Table 2-2. F8 Instruction Set Summary

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES	OPERATION PERFORMED
				C Z S O	
I/O	INS	P4	1		[A]←[P4] Input to Accumulator from I/O port.
	IN	.P8	2		[A]←[P8] Input to Accumulator from I/O port.
	OUTS	P4	1		[P4]←[A] Output to I/O port from Accumulator.
	OUT	P8	2		[P8]←[A] Output to I/O port from Accumulator.
PRIMARY MEMORY REFERENCE	LM		1		[A]←[[DC0]], [DC0]←[DC0] + 1 Load the Accumulator via DC0 and auto-increment DC0.
	ST		1		[[DC0]]←[A], [DC0]←[DC0 + 1] Store the Accumulator via DC0 and auto-increment DC0.
	LR	A,r	1		[A]←[SR] Load the contents of the specified register, SR, into the Accumulator. Increment or decrement ISAR if specified by r.
	LR	A,DPCHR	1		[A]←[DPCHR] Load Accumulator with the contents of the specified DPCHR.
	LR	r,A			[SR]←[A] Load the contents of the Accumulator into the specified register. Increment or decrement ISAR if specified by r.
	LR	DPCHR,A	1		[DPCHR]←[A] Load the contents of the Accumulator into the specified DPCHR.
	LR	H,DC0	1		[DC0]←[H] Load the contents of Scratchpad registers 10 and 11 into DC0.
	LR	Q,DC0	1		[DC0]←[Q] Load the contents of Scratchpad registers 14 and 15 into DC0.
	LR	DC0,H	1		[H]←[DC0] Load the contents of DC0 into Scratchpad registers 10 and 11.

Table 2-2. F8 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES	OPERATION PERFORMED
				C Z S O	
PRIMARY MEMORY REFERENCE (CONTINUED)	LR	DC0,Q	1		[Q]←[DC0] Load the contents of DC0 into Scratchpad registers 14 and 15.
	LR	K,PC1	1		[PC1]←[K] Load the contents of Register K into the Stack register.
	LR	PC1,K	1		[K]←[PC1] Load the contents of the Stack register into Register K.
	LR	PC0,Q	1		[PC0]←[Q] Load the contents of Register Q into the Program Counter.
	PK		1		[PC1]←[PC0], [PC0]←[Q] Save the contents of the Program Counter in the Stack register, then load the contents of Register Q into the Program Counter.
SECONDARY MEMORY REFERENCE (SCRATCHPAD OPERATE)	AS	r	1	X X X X	[A]←[A] + [SR] Add binary the contents of the specified register to the contents of the Accumulator. Increment or decrement ISAR if specified by r.
	ASD	r	1	X X X X	[A]←[A] + [SR] Add decimal the contents of the specified register to the contents of the Accumulator; that is, both numbers are assumed to be BCD digits. Increment or decrement ISAR if specified by r.
	NS	r	1	0 X X 0	[A]←[A] ∧ [SR] AND the contents of the specified register with the contents of the Accumulator. Increment or decrement ISAR if specified by r.
	XS	r		0 X X 0	[A]←[A] ⊻ [SR] Exclusive-OR the contents of the specified register with the contents of the Accumulator. Increment or decrement the ISAR if specified by r.
	DS	r	1	X X X X	[SR]←[SR] - 1 Decrement the specified register. Increment or decrement ISAR if specified by r.

Table 2-2. F8 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES	OPERATION PERFORMED
				C Z S O	
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	AM		1	X X X X	[A]←[A] + [[DC0]], [DC0]←[DC0] + 1 Add Accumulator contents to the contents of the memory location addressed by DC0. Increment DC0.
	AMD		1	X X X X	[A]←[A] + [[DC0]], [DC0]←[DC0] + 1 Decimal add Accumulator contents to the contents of the memory location addressed by DC0. Increment DC0.
	NM		1	0 X X 0	[A]←[A] ∧ [[DC0]], [DC0]←[DC0] + 1 AND Accumulator contents with the contents of the memory location addressed by DC0. Increment DC0.
	OM		1	0 X X 0	[A]←[A] V [[DC0]], [DC0]←[DC0] + 1 OR Accumulator contents with the contents of the memory location addressed by DC0. Increment DC0.
	XM		1	0 X X 0	[A]←[A]⊻[[DC0]], [DC0]←[DC0] + 1 Exclusive-OR Accumulator contents with the contents of the memory location addressed by DC0. Increment DC0.
	CM		1	X X X X	[[DC0]] - [A], [DC0]←[DC0] + 1 Subtract the contents of the Accumulator from the contents of the memory location addressed by DC0. Only the status flags are affected. Increment DC0.
IMMEDIATE	LISU	DATA3	1		[ISAR <5,3 >]←DATA3 Load immediate into the upper three bits of the ISAR.
	LISL	DATA3	1		[ISAR <2,0 >]←DATA3 Load immediate into the lower three bits of the ISAR.
	DCI	ADDR	3		[DC0]← ADDR Load immediate data into the DC0.
	LIS	DATA4	1		[A <3,0 >]← DATA4 Load immediate data into the lower four bits of the Accumulator. Clear the high four bits of the Accumulator.
	LI	DATA8	2		[A]← DATA8 Load immediate data into Accumulator.

Table 2-2. F8 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES	OPERATION PERFORMED
				C Z S O	
IMMEDIATE OPERATE	AI	DATA8	2	X X X X	$[A] \leftarrow [A] + \text{DATA8}$ Add immediate to Accumulator.
	NI	DATA8	2	0 X X 0	$[A] \leftarrow [A] \wedge \text{DATA8}$ AND immediate with Accumulator.
	OI	DATA8	2	0 X X 0	$[A] \leftarrow [A] \vee \text{DATA8}$ OR immediate with Accumulator.
	XI	DATA8	2	0 X X 0	$[A] \leftarrow [A] \veebar \text{DATA8}$ Exclusive-OR immediate with Accumulator.
	CI	DATA8	2	X X X X	$\text{DATA8} + \overline{[A]} + 1$ Compare immediate: subtract Accumulator contents from immediate data, but only the status flags are affected.
JUMP	PI	ADDR	3		$[PC1] \leftarrow [PC0]$, $[PC0] \leftarrow \text{ADDR}$ Save Program Counter in Stack register, then load immediate address into Program Counter.
	BR	DISP	2		$[PC0] \leftarrow [PC0] + \text{DISP}$ Add immediate displacement to contents of Program Counter.
	JMP	ADDR	3		$[PC0] \leftarrow \text{ADDR}$, $[A] \leftarrow \text{ADDR} < 15,8 >$ Load immediate address into Program Counter. Load the high order byte of the address into the Accumulator.
BRANCH ON CONDITION	BT	DATA3,DISP	2		If $\text{DATA3} \vee \text{TMASK} \neq 0$ then $[PC0] \leftarrow [PC0] + \text{DISP}$ OR the 3 bits of immediate data with the current TMASK. If any resulting bit is a 1, add the displacement to PC0.
	BF	DATA4,DISP	2		If $\text{DATA4} = \text{FMASK}$, then $[PC0] \leftarrow [PC0] + \text{DISP}$ If the 4 bits of immediate data are equal to FMASK, add the displacement to PC0.
	BP	DISP	2		If $[S] = 1$ then $[PC0] \leftarrow [PC0] + \text{DISP}$ Branch relative if the Sign bit is set.
	BC	DISP	2		If $[C] = 1$ then $[PC0] \leftarrow [PC0] + \text{DISP}$ Branch relative if the Carry bit is set.
	BZ	DISP	2		If $[Z] = 1$ then $[PC0] \leftarrow [PC0] + \text{DISP}$ Branch relative if the Zero bit is set.

Table 2-2. F8 Instruction Set Summary (Continued)

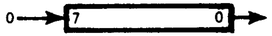
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S O	OPERATION PERFORMED
BRANCH ON CONDITION (CONTINUED)	BM	DISP	2		If [S] = 0 then [PC0]←[PC0] + DISP Branch relative if the Sign bit is reset.
	BNC	DISP	2		If [C] = 0 then [PC0]←[PC0] + DISP Branch relative if the Carry bit is reset.
	BNZ	DISP	2		If [Z] = 0 then [PC0]←[PC0] + DISP Branch relative if the Zero bit is reset.
	BNO	DISP	2		If [O] = 0 then [PC0]←[PC0] + DISP Branch relative if the Overflow bit is reset.
	BR7	DISP	2		If [ISAR <2,0 >] = 7 then [PC0]←[PC0] + DISP If the low three bits of the ISAR are not all 1s, branch relative.
REGISTER-REGISTER MOVE	XDC		1		[DC0]⟷[DC1] Exchange the contents of DC0 with the contents of DC1.
	LR	A,IS	1		[A]←[ISAR] Load the contents of ISAR into the Accumulator.
	LR	IS,A	1		[ISAR]←[A] Load the contents of the Accumulator into the ISAR.
	POP		1		[PC0]←[PC1] Load the contents of the Stack register into the Program Counter.
REGISTER-REGISTER OPERATE	ADC		1	0 X 1 0	[DC0]←[DC0] + [A] Add the contents of DC0 to the contents of the Accumulator, which is treated as a signed binary number. Store the result in DC0.
REGISTER OPERATE	SR	1	1		 Shift the contents of the Accumulator right one bit. The most significant bit becomes a 0.

Table 2-2. F8 Instruction Set Summary (Continued)

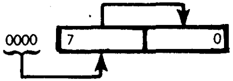
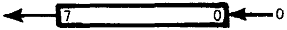
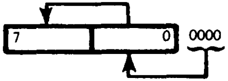
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S O	OPERATION PERFORMED
REGISTER OPERATE (CONTINUED)	SR	4	1	0 X 1 0	 Shift the contents of the Accumulator right four bits. The most significant four bits become 0s.
	SL	1	1	0 X X 0	 Shift the contents of the Accumulator left one bit. The least significant bit becomes a 0.
	SL	4	1	0 X X 0	 Shift the contents of the Accumulator left four bits. The least significant four bits become 0s.
	COM		1	0 X X 0	[A]←[A] Complement Accumulator contents.
	LNK		1	X X X X	[A]←[A] + C Add the Carry to the contents of the Accumulator.
	INC		1	X X X X	[A]←[A] + 1 Increment the contents of the Accumulator.
	CLR		1		[A]←0 Clear the Accumulator.
INTERRUPT	DI		1		[I]←0 Set the interrupt enable bit in the Status register, W, to 0.
	EI		1		[I]←1 Set the interrupt enable bit in the Status register, W, to 1.

Table 2-2. F8 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES	OPERATION PERFORMED
				C Z S O	
STATUS	LR	W,J	1		[W]←[J] Move the contents of Scratchpad register 9 into the Status register, W.
	LR	J,W	1		[J]←[W] Move the contents of the Status register, W, into Scratchpad register 9.
	NOP		1		No operation is performed. This is not a Halt.

THE BENCHMARK PROGRAM

Now consider our benchmark program; for the F8 it looks like this:

```
        DCI     TABLE   LOAD TABLE BASE ADDRESS
        LM              LOAD DISPLACEMENT TO FIRST FREE BYTE
        ADC             ADD TO BASE ADDRESS
        XDC             SAVE THIS ADDRESS IN DC1
        DCI     IOBUF   LOAD I/O BUFFER BASE ADDRESS
LOOP    LM              LOAD NEXT BYTE FROM I/O BUFFER
        XDC             SWITCH ADDRESSES
        ST              STORE IN NEXT BYTE OF TABLE
        XDC             SWITCH ADDRESSES
        DS      0       DECREMENT I/O BUFFER LENGTH
        BNZ     LOOP    RETURN IF NOT END
        LR      H,DC    IF END, STORE SECOND BYTE OF CURRENT
        LR      A,HL    TABLE ADDRESS AS DISPLACEMENT TO
        DCI     TABLE   FIRST FREE BYTE
        ST
```

The benchmark program above makes the following assumptions:

1) The I/O buffer can be located anywhere in read-write memory.

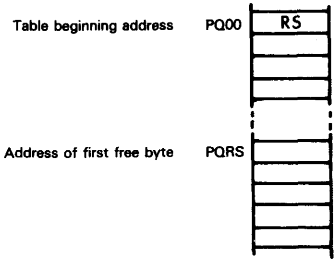
2) The number of occupied bytes in the I/O buffer is maintained in scratchpad byte 0. Thus decrementing scratchpad byte 0 to zero provides the I/O buffer length.

3) The permanent data table beginning memory address has all 0s for the low order eight bits:

$$\underbrace{\text{XXXXXXXX00000000}}$$

Binary representation of permanent table beginning memory address
X may be 0 or 1

The table is not more than 256 bytes long, and the displacement to the first free byte is stored in the first byte of the table. Since the table beginning address has 0s in the low order eight bits, the displacement to the first free byte also becomes the low order eight bits of the first free byte address:



PQ and RS are hexadecimal digits

All of the above assumptions are valid — and depending upon the application, may also be realistic. Removing any of the above assumptions will make the F8 program longer, by removing one of the inherent strengths of the F8 instruction set.

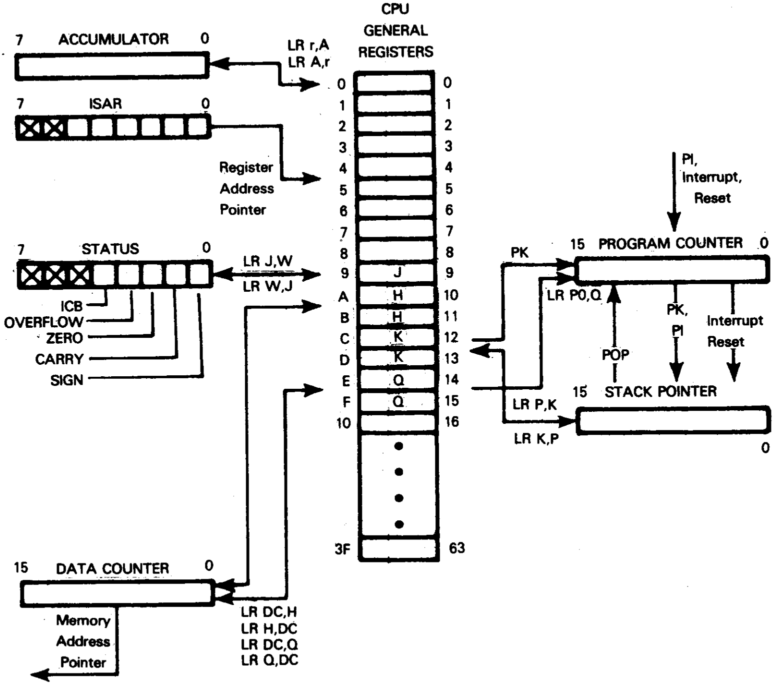


Figure 2-4. Instructions That Move Data Between The Scratchpad And Various Registers

Table 2-3. Timing And ROMC States For F8 Instruction Set

MNEMONIC	OPERAND(S)	CYCLE	ROMC STATE
ADC		L	A
		S	0
AI	DATA8	L	3
		S	0
AM		L	2
		S	0
AMD		L	2
		S	0
AS	r	S	0
ASD	r	S	1C
		S	0
BF	DATA4,DISP	S	1C
Branch		L	1
		S	0
		S	1C
No		S	3
Branch		S	0
BR7	DISP	S	3
No Branch		S	0
		L	1
Branch		S	0
BT	DATA3,DISP	S	1C
No		S	3
Branch		S	0
		S	1C
Branch		L	1
		S	0
CI	DATA8	L	3
		S	0
CM		L	2
		S	0
COM		S	0
DCI	ADDR	L	11
		S	3
		L	E
		S	3
		S	0
DI		S	1C
		S	0
DS	r	L	0
EI		S	1C
		S	0
IN	P8	L	3
		L	1B
		S	0
INC		S	0
INS	0 or 1	S	1C
		S	0
INS	2	L	1C
	through	L	1B
	15	S	0
(INTERRUPT)		L	1C
		L	08
		L	13
		S	0
JMP	ADDR	L	3
		L	C
		L	14
		S	0
LI	DATA8	L	3
		S	0
LIS	DATA4	S	0
LISL	DATA3	S	0
LISU	DATA3	S	0
LM		L	2
		S	0
LNK		S	0
LR	A,IS	S	0
LR	A,KL	S	0
LR	A,KU	S	0
LR	A,QL	S	0
LR	A,QU	S	0
LR	A,r	S	0
LR	DC0,H	L	16
		L	19
		S	0
LR	DC0,Q	L	16
		L	19
		S	0
LR	H,DC0	L	6
		L	9
		S	0
LR	IS,A	S	0
LR	J,W	S	0
LR	K,P	L	7
		L	B
		S	0
LR	KL,A	S	0
LR	KU,A	S	0
LR	P,K	L	15
		L	18
		S	0
LR	PC0,Q	L	
		L	
		S	
LR	Q,DC0	L	6
		L	9
		S	0
LR	QL,A	S	0
LR	QU,A	S	0
LR	r,A	S	0
LR	W,J	S	1C
		S	0
NI	DATA8	L	3
		S	0
NM		L	2
		S	0
NS	r	S	0
OI	DATA8	L	3
		S	0

Table 2-3. Timing And ROMC States For F8 Instruction Set (Continued)

MNEMONIC	OPERAND(S)	CYCLE	ROMC STATE
OM		L	2
		S	0
OUT	P8	L	3
		L	1A
		S	0
OUTS	0 or 1	S	1C
		S	0
OUTS	2	L	1C
	through	L	1A
	15	S	0
PI	ADDR	L	3
		S	D
		L	C
		L	14
		S	0
PK		L	12
		L	14
		S	0
POP		S	4
		S	0
(RESET)		S	1C
		L	8
		S	0
SL	1	S	0
SL	4	S	0
SR	1	S	0
SR	4	S	0
ST		L	5
		S	0
XI	DATA8	L	3
		S	0
XM		L	2
		S	0
XS	r	S	0

The following symbols are used in Table 2-4:

aaaa	Four bits choosing the register addressing mode:	
	0000-1011	Registers 0 - B directly addressed
	1100	ISAR addresses the register
	1101	ISAR addresses the register. Increment low three bits of ISAR.
	1110	ISAR addresses the register. Decrement low three bits of ISAR.
	1111	NOP. No operation is performed if aaaa = F_{16}.
cc	Two bits choosing a Scratchpad register:	
	00--KU	Scratchpad Register 12
	01--KL	Scratchpad Register 13
	10--QU	Scratchpad Register 14
	11--QL	Scratchpad Register 15
d	One bit of immediate data.	
eeee	A 4-bit port number.	
QQQQ	A 16-bit address.	

RR An 8-bit, signed displacement.

SS An 8-bit port number.

When two numbers are given in the "Machine Cycles" column (for example 3/3.5), the first is the execution time if no branch is taken, and the second is execution time if the branch is taken.

Table 2-4. F8 Instruction Set Object Code

INSTRUCTION	OBJECT CODE	BYTES	MACHINE CYCLES
ADC	8E	1	2.5
AI DATA8	24 YY	2	2.5
AM	88	1	2.5
AMD	89	1	2.5
AS r	1100aaaa	1	1
ASD r	1101aaaa	1	2
BC DISP	82 RR	2	3/3.5
BF DATA4,DISP	1001dddd RR	2	3/3.5
BM DISP	91 RR	2	3/3.5
BNC DISP	92 RR	2	3/3.5
BNO DISP	98 RR	2	3/3.5
BNZ DISP	94 RR	2	3/3.5
BP DISP	81 RR	2	3/3.5
BR DISP	90 RR	2	3.5
BR7 DISP	8F RR	2	3/3.5
BT DATA3,DISP	10000ddd RR	2	3/3.5
BZ DISP	84 RR	2	3/3.5
CI DATA8	25 YY	2	2.5
CLR	70	1	1
CM	8D	1	2.5
COM	18	1	1
DCI ADDR	2A QQQQ	3	6
DI	1A	1	2
DS r	0011aaaa	1	1.5
EI	1B	1	2
IN P8	26 SS	2	4
INC	1F	1	1
INS P4	1010eeee	1	4
JMP ADDR	29 QQQQ	3	5.5
LI DATA8	20 YY	2	2.5
LIS DATA4	0111dddd	1	1
LISL DATA3	01101ddd	1	1
LISU DATA3	01100ddd	1	1
LM	16	1	2.5
LNK	19	1	1
LR A,DPCHR	000000cc	1	1
LR A,IS	0A	1	1
LR A,r	0100aaaa	1	1
LR DC,H	10	1	4
LR DC,Q	0F	1	4
LR DPCHR,A	000001cc	1	1
LR H,DC	11	1	4
LR IS,A	0B	1	1
LR J,W	1E	1	1
LR K,PC1	08	1	4
LR PC0,Q	0D	1	4
LR PC1,K	09	1	4
LR Q,DC	0E	1	4
LR r,A	0101aaaa	1	1
LR W,J	1D	1	2
NI DATA8	21 YY	2	2.5
NM	8A	1	2.5
NOP	2B	1	1
NS r	1111aaaa	1	1
OI DATA8	22 YY	2	2.5
OM	8B	1	2.5
OUT P8	27 SS	2	4
OUTS P4	1011eeee	1	4
PI ADDR	28 QQQQ	3	6.5
PK	0C	1	4
POP	1C	1	2
SL 1	13	1	1
SL 4	15	1	1
SR 1	12	1	1
SR 4	14	1	1
ST	17	1	2.5
XDC	2C	1	2
XI DATA8	23 YY	2	2.5
XM	8C	1	2.5
XS r	1110aaaa	1	1

THE 3851 PROGRAM STORAGE UNIT (PSU)

This has been the principal read-only program storage device in small F8 microcomputer systems.

Figure 2-5 illustrates functions provided by the 3851 PSU. Device pins and signals are given in Figure 2-6.

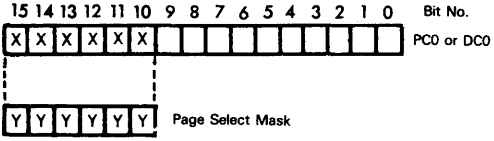
In the future, the 3851 PSU is likely to be rendered obsolete by the 3856 and 3857 16K Programmable Storage Units. An erasable, programmable read-only memory (EPROM) version of the 3851 PSU is also likely to be made available.

As shown in Figure 2-5, the 3851 PSU is, in fact, more than a ROM unit. However, we will begin the description of this device by describing its program storage capabilities.

PSU ADDRESS SPACE

Every 3851 PSU has 1024 bytes of read-only memory, the contents of which must be defined at the time the chip is created. There is also a 6-bit page select mask, which must be specified when the chip is created; the page select represents the high order six bits of the memory address for all ROM bytes of the PSU. As such, the page select defines the PSU's address space.

Recall that every 3851 PSU contains its own memory addressing logic, which accesses a Data Counter (DC0), a Program Counter (PC0) and a Stack register (PC1). When a ROMC state output by the 3850 CPU, and received by the 3851 PSU, identifies a memory reference operation, the ROMC state also identifies whether the memory address is to be found in PC0 or in DC0. In response to this ROMC state, PSU memory addressing logic will compare its 6-bit page select mask with the high order six bits of the specified address register's contents:



If there is coincidence, the 3851 PSU will respond to the memory reference operation; if there is no coincidence, the 3851 PSU addressing logic modifies the contents of address registers, as might be required by the ROMC state, but it does not respond to the actual memory reference instruction.

In addition to having read-only memory and memory addressing logic, the 3851 PSU has two I/O ports, interrupt request and priority handling logic and a programmable timer.

Two I/O ports are available on the 3851 PSU for much the same reason they are available on the 3850 CPU. The 3851 PSU needs no address pins since the source and destination for address information flow is entirely within the 3851 PSU; from the Data Counter, or the Program Counter, to read-only memory on the PSU chip. The 16 pins that would normally be used to input or output addresses instead provide two I/O ports.

PROGRAMMABLE TIMER

Interrupt logic and the programmable timer logic are connected.

The basic time unit of the programmable timer is 31 times the microcomputer system clock frequency. The timer can be programmed to time-out after any number of time intervals in the range 0 to 254. The programmable timer can also be stopped by loading it with FF_{16}. A time-out is marked by a timer interrupt request.

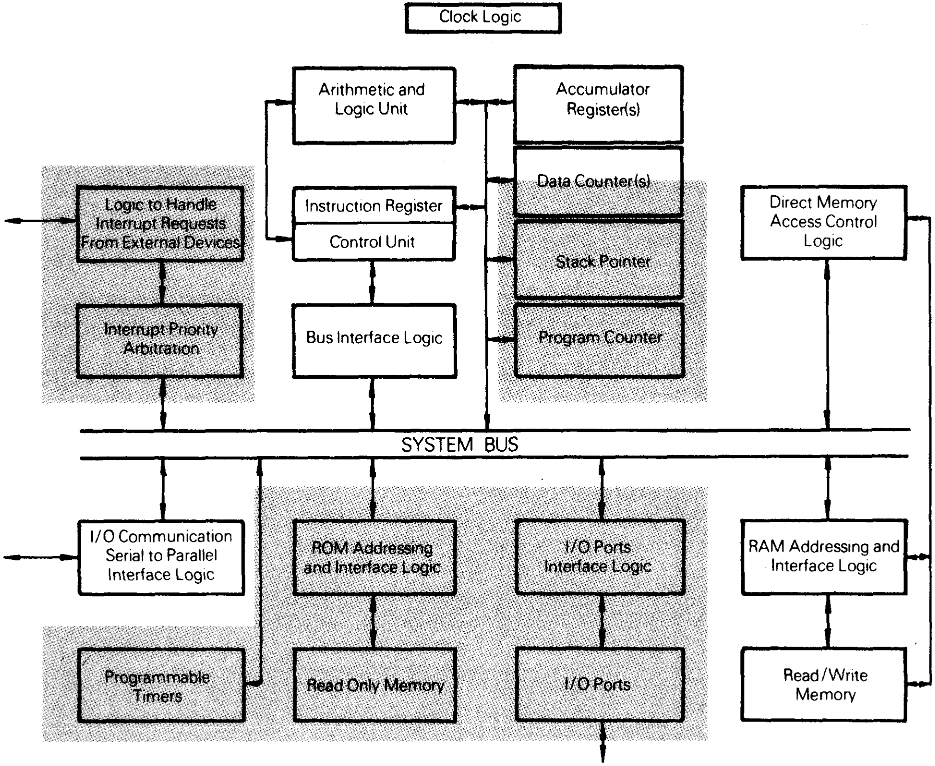
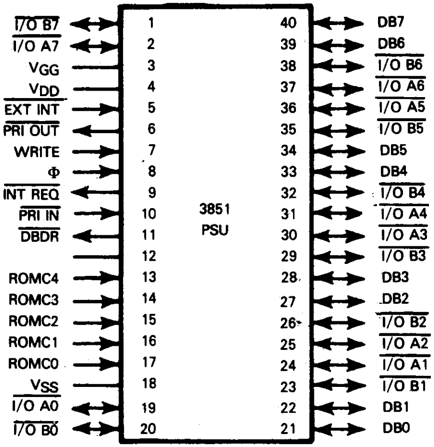

Clock Logic
Arithmetic and Logic Unit
Accumulator Register(s)
Data Counter(s)
Logic to Handle Interrupt Requests From External Devices
Instruction Register
Control Unit
Direct Memory Access Control Logic
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 2-5. Logic Of The Fairchild F8 3851, 3856 And 3857 Programmable Storage Unit



Pin Name	Description	Type
I/O A0 - I/O A7	I/O Port A	Input/Output
I/O B0 - I/O B7	I/O Port B	Input/Output
DB0 - DB7	Data Bus	Tristate, Bidirectional
ROMC0 - ROMC4	Control Lines	Input
Φ, WRITE	Clock Lines	Input
EXT INT	External Interrupt	Input
PRI IN	Priority In	Input
PRI OUT	Priority Out	Output
INT REQ	Interrupt Request	Output
DBDR	Data Bus Drive	Output
VSS, VDD, VGG	Power Supply Lines	Input

Figure 2-6. 3851 PSU Signals And Pin Assignments

Interrupt request and priority arbitration logic works as follows:

F8 PSU INTERRUPT LOGIC

Interrupt requests may arrive from two sources: the programmable timer, as described above, or from an external device. Under program control you select which of the two interrupts (if either) is going to be acknowledged.

The 3851 PSU has a "priority in" and a "priority out" signal. Typically the "priority in" signal will be the ICB signal from the 3850 CPU, or it will be the "priority out" signal from another device in an F8 system.

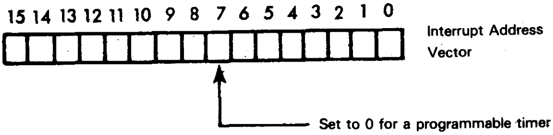
If the "priority in" signal is true, and an interrupt request which can be acknowledged is true, then "priority out" is output false, and an interrupt is transmitted to the 3850 CPU.

If the "priority in" is true, but a valid interrupt request is not pending, then "priority out" is passed on true.

If "priority in" is false, then no interrupt request can be acknowledged, and "priority out" is passed on as false.

Another mask option of the 3851 PSU is an interrupt address vector. Once an interrupt has been acknowledged by the CPU, the address in the interrupt vector is transmitted to the Program Counter of the 3851 PSU, and the Program Counters of any other memory devices in the F8 system. This becomes the address at which execution continues; that is, it is the beginning address of the interrupt service routine.dedicated to this individual 3851 PSU.

One bit of the interrupt address vector (it is bit 7) is set or reset to identify the interrupt request as external, or as coming from the PSU's programmable timer:



Set to 0 for a programmable timer interrupt, set to 1 otherwise

All other bits are unalterable mask options

THE 3856 AND 3857 16K PROGRAMMABLE STORAGE UNITS (16K PSU)

These two devices are enhancements of, and replacements for the 3851 PSU which we have just described.

An interesting characteristic of microcomputer devices is that residual markets remain for products long after enhancements have appeared on the market. This is unlikely to be true for the 3851 PSU. The 3856 and 3857 16K PSUs offer advantages over the 3851 PSU that are significant enough to render the 3851 PSU obsolete.

Superficially, Figure 2-5 represents the logic implemented on all three PSUs — the 3851, 3856 and 3857. **Table 2-5 summarizes the differences between the devices. These are the most significant features of the 3856 and 3857 PSUs:**

1) RESET sets all I/O port pins and address lines to zero. In the 3851 PSU RESET leaves I/O port pins indeterminate — and this has caused problems in many applications.

2) The Interval Timers of the 3856 and 3857 PSUs are binary decrementers rather than polynomial shifters — with the result that you can read timer contents directly and determine lapsed times. Also a programmable option allows you to measure pulse widths being input to the PSU.

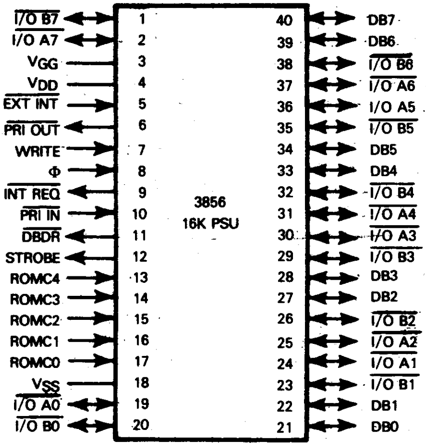
3) The 3857 PSU uses the 16 pins of the two 8-bit I/O ports for 16 address lines, so that additional ROM or RAM can be interfaced directly to a 3857 PSU — without requiring a 3852 DMI or 3853 SMI, as was the case with the 3851 PSU.

4) The 3856 and 3857 PSUs both provide 2K bytes of ROM for program storage; this is twice the program memory available on the 3851 PSU. This significantly increases the scope of two-device F8 microcomputer systems.

THE 3852 DYNAMIC MEMORY INTERFACE (DMI)

Primarily this device contains the necessary address generation and memory refresh logic needed to include dynamic read-write memory in an F8 system.

Because of the way in which the F8 microcomputer system is organized, however, memory refresh and direct memory access logic are closely related. That is why, in Figure 2-9, a small part of the direct memory access control logic is shown as being implemented on the 3852 DMI chip.



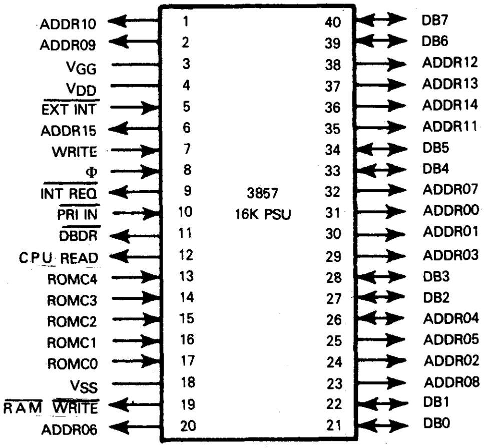
Pin Name	Description	Type
$\overline{\text{I/O A0}}$ - $\overline{\text{I/O A7}}$	I/O Port A	Input/Output
$\overline{\text{I/O B0}}$ - $\overline{\text{I/OB7}}$	I/O Port B	Input/Output
STROBE	STROBE for I/O Port A	Output
DB0 - DB7	Data Bus	Tristate, Bidirectional
ROMC0 - ROMC4	Control Lines	Input
Φ, WRITE	Clock Lines	Input
$\overline{\text{EXT INT}}$	External Interrupt	Input
$\overline{\text{PRI IN}}$	Priority In	Input
$\overline{\text{PRI OUT}}$	Priority Out	Output
$\overline{\text{INT REQ}}$	Interrupt Request	Output
$\overline{\text{DBDR}}$	Data Bus Drive	Output
VSS, VDD, VGG	Power Supply Lines	

Figure 2-7. 3856 PSU Signals And Pin Assignments

igures 2-7 and 2-8 illustrate the pins and signals of the 3856 and 3857 16K PSUs respectively.

Table 2-5. A Summary Of Differences Between 3851, 3856 And 3857 PSUs

FUNCTION	3851 PSU	3856 PSU	3857 PSU
ROM	1024 bytes	2048 bytes	2048 bytes
I/O Ports	2 x 8 bits	2 x 8 bits	None
Address lines	None	None	16
Interrupt signals	Priority in and Priority out	Priority in and Priority out	Priority in only. Must be end of daisy chain.
Interrupt options	Enable timer or external, but not both	Enable timer and/or external	Enable timer and/or external
Timer register	8-bit Polynomial	8-bit Count down	8-bit Count down
Timer decrement interval	31 clock cycles	2, 8, 32 or 128 clock cycles	2, 8, 32 or 128 clock cycles
Timer stop/start control	No	Yes	Yes
Timer readback	No	Yes	Yes
Timer read pulse width?	No	Yes	Yes
RESET zero I/O ports?	No	Yes	No I/O ports



Pin Name	Description	Type
ADDR00 - ADDR15	Address Lines	Output
$\overline{\text{CPU READ}}$	Memory Read Enable	Output
$\overline{\text{RAM WRITE}}$	Memory Write Signal	Output
DB0 - DB7	Data Bus	Tristate, Bidirectional
ROMC0 - ROMC4	Control Lines	Input
Φ, WRITE	Clock Lines	Input
$\overline{\text{EXT INT}}$	External Interrupt	Input
$\overline{\text{PRI IN}}$	Priority In	Input
$\overline{\text{INT REQ}}$	Interrupt Request	Output
$\overline{\text{DBDR}}$	Data Bus Drive	Output
VSS, VDD, VGG	Power Supply Lines	

Figure 2-8. 3857 PSU Signals And Pin Assignments

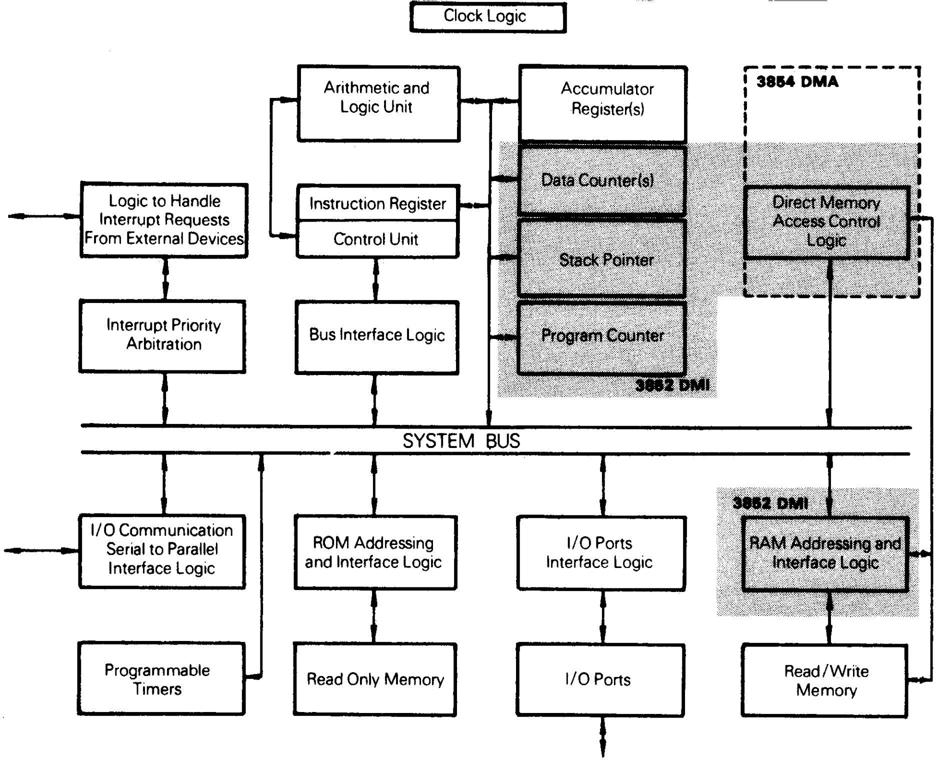

Clock Logic
Arithmetic and Logic Unit
Accumulator Register(s)
3854 DMA
Logic to Handle Interrupt Requests From External Devices
Instruction Register
Control Unit
Data Counter(s)
Direct Memory Access Control Logic
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
3852 DMI
SYSTEM BUS
3852 DMI
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 2-9. Logic Of The Fairchild F8 3852 Dynamic Memory Interface (DMI), And For The 3854 Direct Memory Access (DMA) Devices

Figure 2-10 illustrates pins and signals of the 3852 DMI.

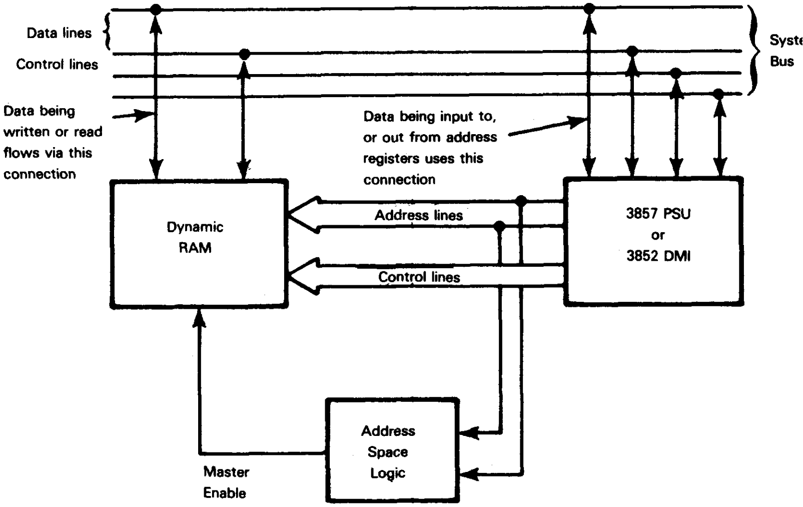
Conceptually, memory addressing logic of the 3852 DMI is very similar to 385 PSU memory addressing logic; there are, however, some differences between th 3852 DMI memory addressing and the 3851 or 3856 PSU:

1) The 3852 DMI contains two Data Counters, DC0 and DC1. The presence of the auxiliary Da Counter (DC1) has no immediate impact on memory addressing logic within the 3852 DM However, as we discussed earlier, its presence in an F8 system that also includes a 3851 PS calls for programming caution.

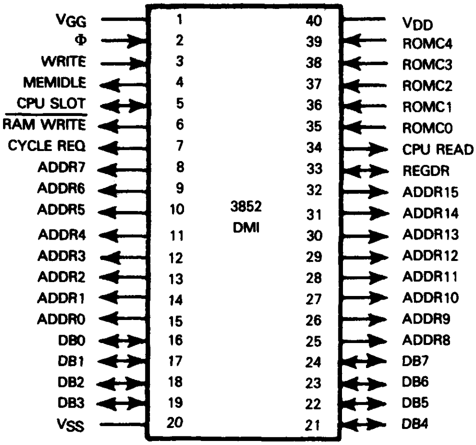
2) Data and address flows surrounding a 3852 DMI are totally unlike the 3851 or 3856 PSU. the case of these PSUs, addresses are transmitted entirely within the logic of the PSU; th only communication needed between a PSU and the CPU is via the eight Data Bus lines o the System Bus. The DMI, on the other hand, generates a 16-bit address, which it outpu directly to the read-write memory which it is controlling.

   These address pins are equivalent to 3857 PSU address pins — that is, the address pir which a CPU would have, if the CPU contained memory addressing logic for the microcom puter system. In other words, the 3852 DMI creates the address lines and control signal which so far as the read-write memory is concerned, are lacking on the F8 System Bus. Th F8 System Bus does, however, contain data lines needed by the read-write memory to ac tually transmit data to, or from the CPU.

   Data and address flows around the 3852 DMI may be illustrated as follows:


Data lines
Control lines
Syste
Bus
Data being written or read flows via this connection
Data being input to, or out from address registers uses this connection
Address lines
Control lines
Dynamic RAM
3857 PSU or 3852 DMI
Address Space Logic
Master Enable

3) Unlike the 3851, 3856 or 3857 PSU, the 3852 DMI has no on-chip logic to determine addres space for read-write memory which the DMI is controlling. Address space determination i made by logic in between the DMI and the read-write memory. Typically, selected high orde address lines output by the DMI are gated through elementary Boolean logic components t create the master enable signal used to strobe attached read-write memory. This is illustrate above.



Pin Name	Description	Type
DB0 - DB7	Data Bus Lines	Tristate, Bidirectional
ADDR0 - ADDR15	Address Lines	Tristate, Output
Φ, WRITE	Clock Lines	Input
MEMIDLE	DMA Timing Line	Output
CYCLE REQ	RAM Timing Line	Output
CPU SLOT	Timing Line	Input/Output
CPU READ	RAM Timing Line	Output
REGDR	Register Drive Line	Input/Output
$\overline{\text{RAM WRITE}}$	Write Line	Tristate, Output
ROMC0 - ROMC4	Control Lines	Input
VSS, VDD, VGG	Power Lines	Input

Figure 2-10. 3852 DMI Signals And Pin Assignments

he process of refreshing dynamic memory and implementing direct memory acess, are integrally related in an F8 system.

F8 DMI MEMORY REFRESH

he presence of a separate DMI interface device means that here can be a limited overlap between a memory reference peration which was initiated by the CPU, and a memory eference operation that is not initiated by the CPU.

wo types of memory reference operations are not initiated by the CPU: memory efresh and direct memory access.

et us consider how a direct memory access may follow a CPU-initiated memory ead operation. These are the events which occur:

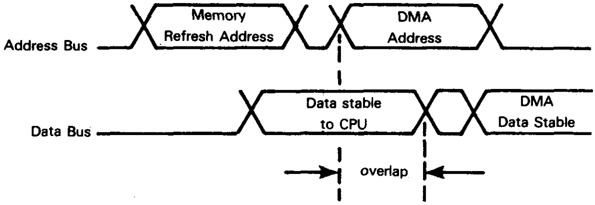
Upon receiving an appropriate ROMC state from the CPU, the 3852 DMI outputs a 16-bit memory address, together with a read strobe; these outputs from the 3852 DMI are received by read-write memory.

Read-write memory responds by placing data directly on the Data Bus. The data must remain stable on the Data Bus until the CPU has had time to read the data.

F8 DIRECT MEMORY ACCESS

While data is stable on the Data Bus, DMA logic may apply a new memory address to read-write memory. Following the arrival of address and control signals at read-write memory, there is a fixed time delay before read-write memory responds by placing data on the

Data Bus. This time delay can overlap with time when prior data must be stable on the Dat Bus. This may be illustrated as follows:



DMI logic outputs control signals which identify the way in which each memor access period is being used; there are three possibilities:

1) Memory is communicating with the F8 System Bus.

2) Memory is not communicating with the System Bus, but since it is dynamic memory it i being refreshed.

3) Memory is not communicating with the System Bus and is available for external access.

Cases 2 or 3 above may follow case 1 in separate memory access periods of the same instructio cycle.

THE 3854 DIRECT MEMORY ACCESS (DMA) DEVICE

This device receives memory access period identification signals output by th 3852 DMI. Based on the direct memory access requirements specified by the cur rently executing program, the DMA device accesses read-write memory durin available memory access periods, as defined by the 3852 DMI. Figure 2-11 illustr ates 3854 DMA pins and signals.

These are the variables which must be specified for a direct memory acces operation:

1) The beginning address for the memory buffer into which data must be written, or out c which data must be read.

2) The length of the buffer.

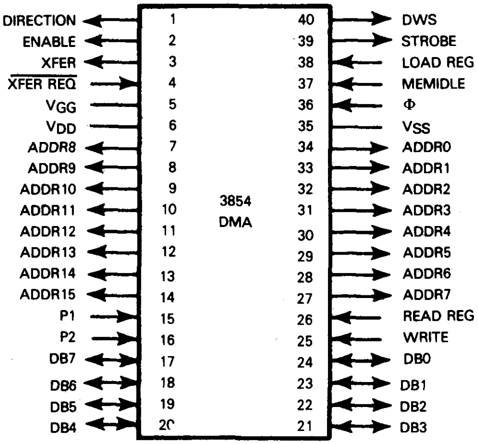
3) Whether data is to be written or read out of the buffer.

Once a direct memory access operation has been initiated, it proceeds in paralle with other events occurring within the F8 microcomputer system, using memor access periods which are defined by the 3852 DMI as available for direct memor access. In other words, direct memory access operations in no way slow down program execu tion that may be occurring in parallel.

DMA data transfer may be high speed or low speed. Low speed DMA transfer mean that each DMA access is enabled by a signal from the external device stating that it is ready t transmit or receive data. High speed access assumes that the external device will always be read to transmit or receive data; therefore every single available memory access period is utilized.

As a direct memory access operation proceeds, after each access, the memory address is incre mented, and the buffer length is decremented. Memory address, buffer length and DMA control are stored in buffers which the CPU accesses as though they were I/O ports. The contents c these I/O ports may be written into, or read at any time. **This means that the F8 DM system allows total flexibility for every type of programmable DMA operatior** ıese include such things as stopping a DMA operation temporarily, or interrogating a DMA ɔeration, to determine how far it has progressed.

ıdefinite DMA transfer may also be specified. In this case, no buffer length is given; ıther, the DMA operation will proceed until stopped.



Pin Name	Description	Type
DB0 - DB7	Data Bus Lines	Tristate, Bidirectional
ADDR0 - ADDR15	Address Lines	Tristate, Output
Φ, WRITE	Clock Lines	Input
LOAD REG/READ REG	Registers Load/Read Line	Input
P1, P2	Port Address Select	Input
MEMIDLE	Memory Idle Line	Input
XFER REQ	Transfer Request Line	Input
ENABLE, DIRECTION	Control Status Lines	Output
DWS, XFER	DMA Write Slot, Transfer	Output
STROBE	Output Strobe Line	Output
VSS, VDD, VGG	Power Lines	Input

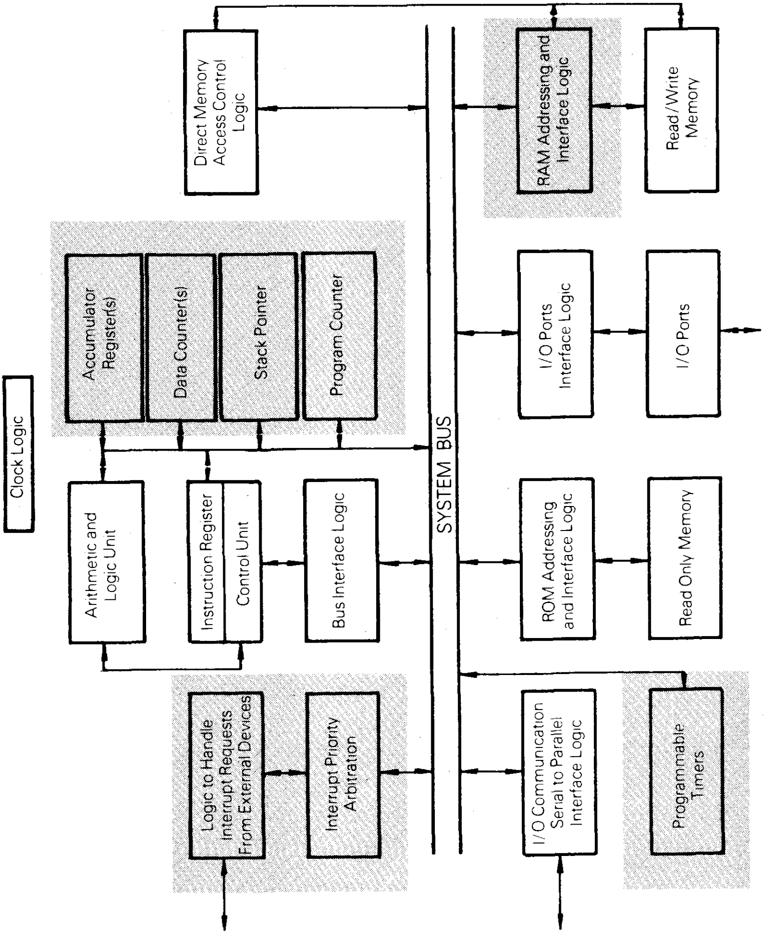
Figure 2-11. 3854 DMA Signals And Pin Assignments

THE 3853 STATIC MEMORY INTERFACE (SMI)

he 3853 SMI provides interface logic for static read-write memory, that is, for ıemory which does not need to be refreshed. Logic implemented on this device is lustrated in Figure 2-12, and is a simple combination of functions which have lready been described for the 3851 PSU and for the 3852 DMI. Figure 2-13 il- ıstrates 3853 SMI pins and signals.

he description of memory interface logic which was given for the 3852 DMI ap- lies also for the 3853 SMI. The 3853 SMI, however, does not identify memory ccess periods, and cannot be used to implement direct memory access.

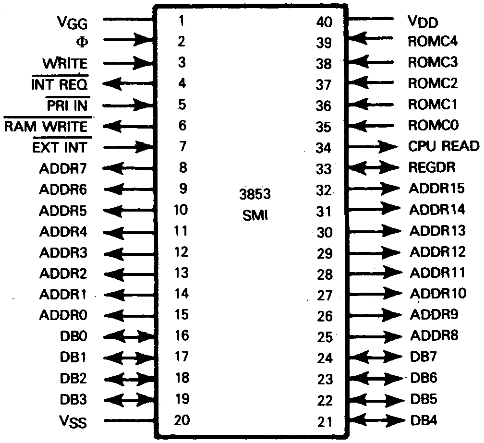
ecause the 3853 SMI does not have memory refresh or direct memory access support logic, ere is unused real estate on the SMI chip. The real estate is used to implement a programmable


Clock Logic
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Direct Memory Access Control Logic
Arithmetic and Logic Unit
Instruction Register
Control Unit
Bus Interface Logic
Logic to Handle Interrupt Requests From External Devices
Interrupt Priority Arbitration
SYSTEM BUS
RAM Addressing and Interface Logic
Read/Write Memory
I/O Ports Interface Logic
I/O Ports
ROM Addressing and Interface Logic
Read Only Memory
I/O Communication Serial to Parallel Interface Logic
Programmable Timers

timer, and interrupt processing logic, as described for the 3851 PSU. There are, however, two small differences between interrupt logic as implemented on the PSU and the SMI devices; they are:

1) The 3853 SMI interrupt address vector is not a permanent mask option, as it is on the PSU, rather it is programmable.

2) The 3853 SMI has no priority output line, which means that in a daisy chain interrupt configuration it must have lowest priority; that is, it must come at the end of the daisy chain.



Pin Name	Description	Type
DB0 - DB7	Data Bus Lines	Bidirectional
ADDR0 - ADDR15	Address Lines	Output
Φ, WRITE	Clock Lines	Input
$\overline{\text{INT REQ}}$	Interrupt Request	Output
$\overline{\text{PRI IN}}$	Priority In Line	Input
$\overline{\text{RAM WRITE}}$	Write Line	Output
$\overline{\text{EXT INT}}$	External Interrupt Line	Input
REGDR	Register Drive Line	Input/Output
CPU READ	CPU Read Line	Output
ROMC0 - ROMC4	Control Lines	Input
VSS, VDD, VGG	Power Supply Lines	Input

Figure 2-13. 3853 SMI Signals And Pin Assignments

THE 3859 AND 3870 F8 MICROCOMPUTERS

These were the first single chip, 8-bit microcomputers to be available commercially.

The 3859 is manufactured by Fairchild and is simply a combination of the 3850 CPU and the 3851 PSU, implemented on a single chip and subject to certain enhancements and changes.

The 3870 is manufactured by Mostek; it is equivalent to a 3850 CPU plus a 3856 PSU. When you compare the 3870 and 3859, these are the principal advantages of the 3870:

1) The 3870 has 2048 bytes of read-only memory wher,eas the 3859 has 1024 bytes of read-only memory.
2) The chip manufacturing technique used by Mostek for the 3870 allows the ROM mask to be defined during the final chip creation step. This has enabled Mostek to offer ROM masking for a $1000.00 charge. In contrast the 3859 or any other typical ROM masking charge will vary between $10,000 and $15,000.
3) The 3870 uses a single +5V power supply. The 3859 uses two power supplies: +5V and +12V.

Figure 2-14 illustrates that part of our general microcomputer system's logic which is provided by the 3859 and 3870 microcomputers.

If you look again at Figure 2-1, the broken line which defines the bounds of a minimum F8 system also defines the bounds of the 3859 and 3870 microcomputers. Thus, **the 3859 and 3870 microcomputers offer the following logic:**

1) **1024 bytes of ROM** for program storage for the 3859. This ROM must, of course, be defined when you order a 3859 microcomputer. **The 3870 has 2048 bytes of ROM.**
2) **64 bytes of read/write memory.**
3) **The standard CPU of the 3850.**
4) **The standard memory addressing logic of the 3851 PSU.** Since the 3859 has a total addressable space of 1024 ROM bytes, it only needs ten address lines. The address registers are therefore all 10 bits wide, rather than being 16 bits wide as was the case with the 3851 PSU. The 3870, having 2048 ROM bytes, requires 11-bit address registers.
5) **Four I/O ports** are provided, corresponding to the two 3850 and two 3851 I/O ports. However, only 31 of the 32 I/O port pins support bidirectional data. The 32nd pin is an output only line.
6) **Clock logic** is internal to the 3859 and the 3870 as is the case with the 3850 CPU.
7) **A programmable timer** is provided. It is the 3851 PSU type of timer, rather than the 3856/3857 variety.
8) **Interrupts** may be requested by an interval timer timeout, or by external logic. Under program control you can enable one or the other interrupt. You cannot enable both interrupts simultaneously.

Note that the external RESET signal which is input to the 3859 microcomputer operates as described for the 3850 CPU and 3851 PSU; the RESET enhancements of the 3856/3857 PSUs are not available. Following a Reset, therefore, I/O port pins of the 3859 microcomputer will be undefined.

The 3859 microcomputer is designed to operate with a 1.5 microsecond cycle time. The device is manufactured using N-channel Isoplanar MOS technology and it is packaged as a 40-pin DIP.

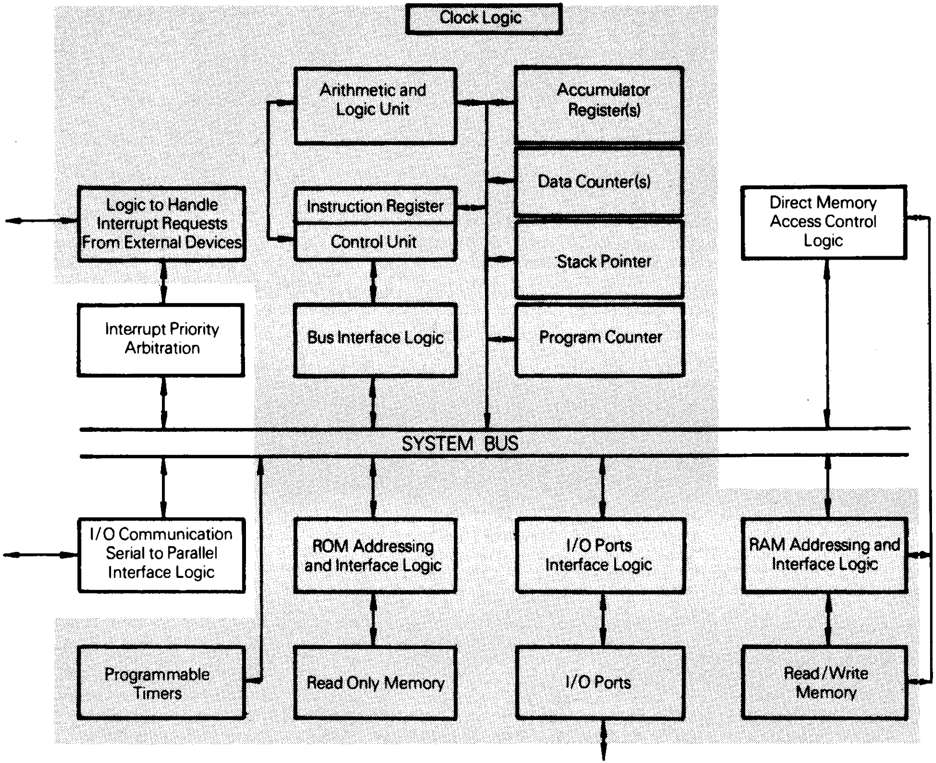

Clock Logic
Arithmetic and Logic Unit
Accumulator Register(s)
Data Counter(s)
Logic to Handle Interrupt Requests From External Devices
Instruction Register
Control Unit
Stack Pointer
Direct Memory Access Control Logic
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 2-14. Logic Of The 3859 And 3870 Microcomputers

DATA SHEETS

This section contains electrical data and timing specifications for the following F8 devices discussed in this chapter:

- 3850 CPU
- 3851 Programmable Storage Unit
- 3852 Dynamic Memory Interface
- 3853 Static Memory Interface
- 3854 Direct Memory Access Controller

ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS (above which useful life may be impaired)

V_{GG}	+15V to -.3V
V_{DD}	+ 7V to -.3V
RC, XTLX and XTLY	+15V to -.3V (RC with 5KΩ series resistor)
All other Inputs	+ 7V to -.3V
Storage Temperature	-55°C to +150°C
Operating Temperature	0°C to 70°C

NOTE: All voltages with respect to V_{SS}.

DC CHARACTERISTICS: V_{SS} = 0V, V_{DD} = 5V±5%, V_{GG} = 12V±5%, T_A = 0 to 70°C

SUPPLY CURRENTS

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
I_{DD}	V_{DD} Current		30	80	mA	f=2MHz, Outputs unloaded
I_{GG}	V_{GG} Current		15	25	mA	f-2MHz, Outputs unloaded

A Summary of 3850 CPU Signal DC Characteristics

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
Φ,	V_{OH}	Output High Voltage	4.4	V_{DD}	Volts	$I_{OH} = -10\ \mu A$
WRITE	V_{OL}	Output Low Voltage	V_{SS}	0.4	Volts	$I_{OL} = 1.6$ mA
	V_{OH}	Output High Voltage	2.9		Volts	$I_{OH} = -100\ \mu A$
	I_L	Leakage Current		1	μA	$V_{IN} = 6V$
XTLY	V_{IH}	Input High Voltage	4.5	V_{GG}	Volts	
	V_{IL}	Input Low Voltage	V_{SS}	0.8	Volts	
	I_{IH}	Input High Current	5	50	μA	$V_{IN} = V_{DD}$
	I_{IL}	Input Low Current	-10	-80	μA	$V_{IN} = V_{SS}$
ROMC0	V_{OH}	Output High Voltage	3.5	V_{DD}	Volts	$I_{OH} = -100\ \mu A$
⋮	V_{OL}	Output Low Voltage	V_{SS}	0.4	Volts	$I_{OL} = 1.6$ mA
ROMC4	I_L	Leakage Current		1	μA	$V_{IN} = 6V$
DB0	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	
⋮	V_{IL}	Input Low Voltage	V_{SS}	0.8	Volts	
DB7	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	$I_{OH} = -100\ \mu A$
	V_{OL}	Output Low Voltage	V_{SS}	0.4	Volts	$I_{OL} = 1.6$ mA
	I_{IH}	Input High Current		1	μA	$V_{IN} = 6V$, 3-State mode
	I_{OL}	Input Low Current		-1	μA	$V_{IN} = V_{SS}$, 3-State mode

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
I/O 0	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -30µA
.	V_{OH}	Output High Voltage	2.9	V_{DD}	Volts	I_{OH} = -100µA
. .	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 2mA
I/O 17	V_{IH}	Input High Voltage (1)	2.9	V_{DD}	Volts	Internal pull-up to V_{DD}
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_L	Leakage Current		1	µA	V_{IN} = 6V
	I_{IL}	Input Low Current		-1.6	mA	V_{IN} = .4V (2)
	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	Internal pull-up to V_{DD}
$\overline{\text{EXT RES}}$	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_{IL}	Input Low Current	-.1	-1.0	mA	V_{IN} = V_{SS}
	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	Internal pull-up to V_{DD}
$\overline{\text{INT REQ}}$	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_{IL}	Input Low Current	-.1	-1.0	mA	V_{IN} = V_{SS}
	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -100µA
$\overline{\text{ICB}}$	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 100µA
	I_L	Leakage Current		1	µA	V_{IN} = 6V

(1) Hysteresis input circuit provides additional .3V noise immunity while internal pull-up provides TTL compatability.

(2) Measured while F8 port is outputting a high level.

NOTE: Positive Current is defined as conventional current flowing into the pin referenced.

A Summary of 3850 CPU Signal AC Characteristics

AC CHARACTERISTICS: V_{SS} = 0V, V_{DD} = 5V ± 5%, V_{GG} = 12V ± 5%, T_A = 0°C to +70°C

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
P_x*	External Input Period	0.5		10	µS	
PW_x*	External Pulse Width	200		P_x-200	nS	t_r, $t_f \leq$ 30 nS
tx_1	Ext. to Φ - to - Delay	20		110	nS	
tx_2	Ext. to Φ + to + Delay	20		125	nS	
PΦ	Φ Period	0.5		10	µS	
PW_1	Φ Pulse Width	180		PΦ-180	nS	t_r,t_f=50nS;C_L=100pf
td_1	Φ to WRITE + Delay	60	150	250	nS	C_L = 100 pf
td_2	Φ to WRITE - Delay	60	150	225	nS	C_L = 100 pf
PW_2	WRITE Pulse Width	PΦ-100		PΦ	nS	t_r,t_f=50nS typ;C_L=100pf
PW_S	WRITE Period; Short		4PΦ			
PW_L	WRITE Period; Long		6PΦ			
td_3	WRITE to ROMC Delay	80	300	550	nS	C_L = 100 pf
td_4*	WRITE to $\overline{\text{ICB}}$ Delay			550	nS	C_L = 50 pf
td_5	WRITE to $\overline{\text{INT REQ}}$ Delay			430	nS	C_L = 100 pf
t_{sx}*	$\overline{\text{EXT RES}}$ set-up time	1.0			µS	C_L = 20 pf
t_{su}*	I/O set-up time	300			nS	
t_h*	I/O hold time	50			nS	
t_o*	I/O Output Delay			2.5	µS	C_L = 50 pf
tdb_0*	WRITE to data bus High Impedance		250	500	nS	
tdb_1*	WRITE to DB Stable		0.6	1.3	µS	C_L = 100 pf
tdb_2	WRITE to DB Stable	2PΦ		2PΦ+1.0	µS	C_L = 100 pf
tdb_3*	DB Set-up	200			nS	
tdb_4*	DB Set-up	350			nS	
tdb_5	DB Set-up	500			nS	
tdb_6*	DB Set-up	350			nS	

* The parameters which are starred in the table above represent those which are most frequently of importance when interfacing to an F8 system. These encompass I/O timing, external timing generation and possible external RAM timing. Unshaded parameters are typically those that are only relevant between F8 chips and not normally of concern to the user.

Input and output capacitance is 3 to 5 pf typical on all pins except V_{DD}, V_{GG}, and V_{SS}.

ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS (Above which useful life may be impaired)

V_{GG}	+15V to -0.3V
V_{DD}	+ 7V to -0.3V
I/O Port Open Drain Option	+15V to -0.3V
External Interrupt Input	-600μA to 225μA
All other Inputs & Outputs	+ 7V to -0.3V
Storage Temperature	-55 C to +150 C
Operating Temperature	0°C to +70°C

NOTE: All voltages with respect to V_{SS}.

DC CHARACTERISTICS: V_{SS} = 0.0V, V_{DD} = +5V±5%, V_{GG} = +12V±5%,
T_A = 0°C to 70°C

SUPPLY CURRENTS

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
I_{DD}	V_{DD} Current		30	70	mA	f=2MHz, Outputs Unloaded
I_{GG}	V_{GG} Current		10	18	mA	f=2MHz, Outputs Unloaded

A Summary of 3851 PSU Signal Characteristics

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
DATA BUS (DBØ - DB7)·	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -100µA
	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 1.6mA
	I_{IH}	Input High Current		1	µA	V_{IN} = 6V,3-State mode
	I_{OL}	Input Low Current		-1	µA	V_{IN} = V_{SS},3-State mode
CLOCK LINES (ϕ, WRITE)	V_{IH}	Input High Voltage	4.Ø	V_{DD}	Volts	
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_L	Leakage Current		1	µA	V_{IN} = 6V
PRIORITY IN AND CONTROL LINES ($\overline{\text{PRI IN}}$, ROMCØ - ROMC4)	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_L	Leakage Current		1	µA	V_{IN} = 6V
PRIORITY OUT ($\overline{\text{PRI OUT}}$)	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -1ØØµA
	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 1ØØµA
INTERRUPT REQUEST ($\overline{\text{INT REQ}}$)	V_{OH}	Output High Voltage			Volts	Open Drain Output [1]
	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 1mA
	I_L	Leakage Current		1	µA	V_{IN} = 6V

NOTES: 1. Pullup resistor to V_{DD} on CPU.

2. Positive current is defined as conventional current flowing into the pin referenced.

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
DATA BUS	V_{OH}	Output High Voltage				External Pullup
DRIVE	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 2mA
($\overline{\text{DBDR}}$)	I_L	Leakage Current		1	µA	V_{IN} = 6V
EXTERNAL	V_{IH}	Input High Voltage	3.5		Volts	
INTERRUPT	V_{IL}	Input Low Voltage		1.2	Volts	
($\overline{\text{EXT INT}}$)	V_{IC}	Input Clamp Voltage		15	Volts	I_{IH} = 185µA
	I_{IH}	Input High Current		1Ø	µA	V_{IN} = V_{DD}
	I_{IL}	Input Low Current		-225	µA	V_{IN} = 2V
	I_{IL}	Input Low Current	-15Ø	-5ØØ	µA	V_{IN} = V_{SS}
I/O PORT	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -3ØµA
OPTION A	V_{OH}	Output High Voltage	2.9	V_{DD}	Volts	I_{OH} = -1ØØµA
(STANDARD	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 2mA
PULLUP)	V_{IH}	Input High Voltage	2.9	V_{DD}	Volts	Internal Pullup to V_{DD} [3]
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_L	Leakage Current		1	µA	V_{IN} = 6V
	I_{IL}	Input Low Current		-1.6	mA	V_{IN} = .4V [4]

Notes: 3. Hysteresis input circuit provides additional .3V noise immunity while internal/external pullup provides TTL compatibility.

4. Measured while I/O port is outputting a high level.

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
I/O PORT	V_{OH}	Output High Voltage				External Pullup
OPTION B	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 2mA
(OPEN DRAIN)	V_{IH}	Input High Voltage	2.9	V_{DD}	Volts	[3]
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_L	Leakage Current		2	µA	$V_{IN} = V_{GG}$
I/O PORT	V_{OH}	Output High Voltage	3.75	V_{DD}	Volts	I_{OH} = -1mA
OPTION C	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 2mA
(DRIVER	I_L	Leakage Current		1	µA	V_{IN} = 6V
PULLUP)						

A Summary of 3851 PSU Signal AC Characteristics

AC CHARACTERISTICS: V_{SS} = 0V, V_{DD} = 5V 5%, V_{GG} = 12V 5%, T_A = 0°C to +70°C

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
PΦ	Φ Period	.5		10	µS	
PW_1	Φ Pulse Width h	180		PΦ-180	nS	t_r,t_f=50nS typ.
td_1	Φ to WRITE + Delay	60		250	nS	C_L=100pf
td_2	Φ to WRITE - Delay	60		225	nS	C_L=100pf
td_4	WRITE to DB Input Delay			2PΦ+1.0	µS	
PW_2	WRITE Pulse Width	Pφ-100		PΦ	nS	t_r,t_f=50nS typ.
PW_S	WRITE Period; Short		4PΦ			
PW_L	WRITE Period; Long		6PΦ			
td_3	WRITE to ROMC Delay			550	nS	
td_7	WRITE to DB Output Delay WRITE to $\overline{\text{DBDR}}$ - Delay	$2PΦ+100-td_2$	2PΦ+200	$2PΦ+850-td_2$	nS	C_L=100pf
td_8	WRITE to $\overline{\text{DBDR}}$ + Delay		200		nS	Open Drain
tr_1	WRITE to $\overline{\text{INT REQ}}$ - Delay			430	nS	C_L=100pf [1]
tr_2	WRITE to $\overline{\text{INT REQ}}$ + Delay			430	nS	C_L=100pf [3]
tpr_1	$\overline{\text{PRI IN}}$ to $\overline{\text{INT REQ}}$-Delay			240	nS	C_L=100pf [2]
tpr_2	$\overline{\text{PRI IN}}$ to $\overline{\text{INT REQ}}$+Delay			430	nS	C_L=100pf
tpd_1	$\overline{\text{PRI IN}}$ to $\overline{\text{PRI OUT}}$-Delay			300	nS	C_L=50pf
tpd_2	$\overline{\text{PRI IN}}$ to $\overline{\text{PRI OUT}}$+Delay			365	nS	C_L=50pf
tpd_3	WRITE to $\overline{\text{PRI OUT}}$+Delay			700	nS	C_L=50pf
tpd_4	WRITE to $\overline{\text{PRI OUT}}$-Delay			640	nS	C_L=50pf
t_{sp}	WRITE to Output Stable			2.5	µS	C_L=50pf,Stan. Pullup
t_{od}	Write to Output Stable			2.5	µS	C_L=50pf,R_L=12.5KΩ Open Drain
t_{dp}	Write to Output Stable		200	400	nS	C_L=50pf,Driver Pullup
t_{su}	I/O Setup Time	1.3			µS	
t_h	I/O Hold Time	0			nS	
t_{ex}	$\overline{\text{EXT INT}}$ Setup Time	400			nS	

1. Assume Priority In was enabled ($\overline{\text{PRI IN}}$ = 0) in previous F8 cycle before interrupt is detected in the PSU.
2. PSU has interrupt pending before priority in is enabled.
3. Assume pin tied to $\overline{\text{INT REQ}}$ input of the 3850 CPU.
4. The parameters which are shaded in the table above represent those which are most frequently of importance when interfacing to an F8 system. Unshaded parameters are typically those that are relevant only between F8 chips and not normally of concern to the user.
5. Input and output capacitance is 3 to 5 pF typical on all pins except V_{DD}, V_{GG}, and V_{SS}.

DC ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS (Above which useful life may be impaired).

V_{GG}	+15V to -.3V
V_{DD}	+7V to -.3V
All other Inputs & Outputs	+7V to -.3V
Storage Temperature	$-55^{\circ}C$ to $150^{\circ}C$
Operating Temperature	$0^{\circ}C$ to $70^{\circ}C$

Note: All voltages with respect to V_{SS}

DC CHARACTERISTICS: $V_{SS}=0V$, $V_{DD}=5V\pm5\%$, $V_{GG}=12V\pm5\%$, $T_A=0$ to $70^{\circ}C$

SUPPLY CURRENTS

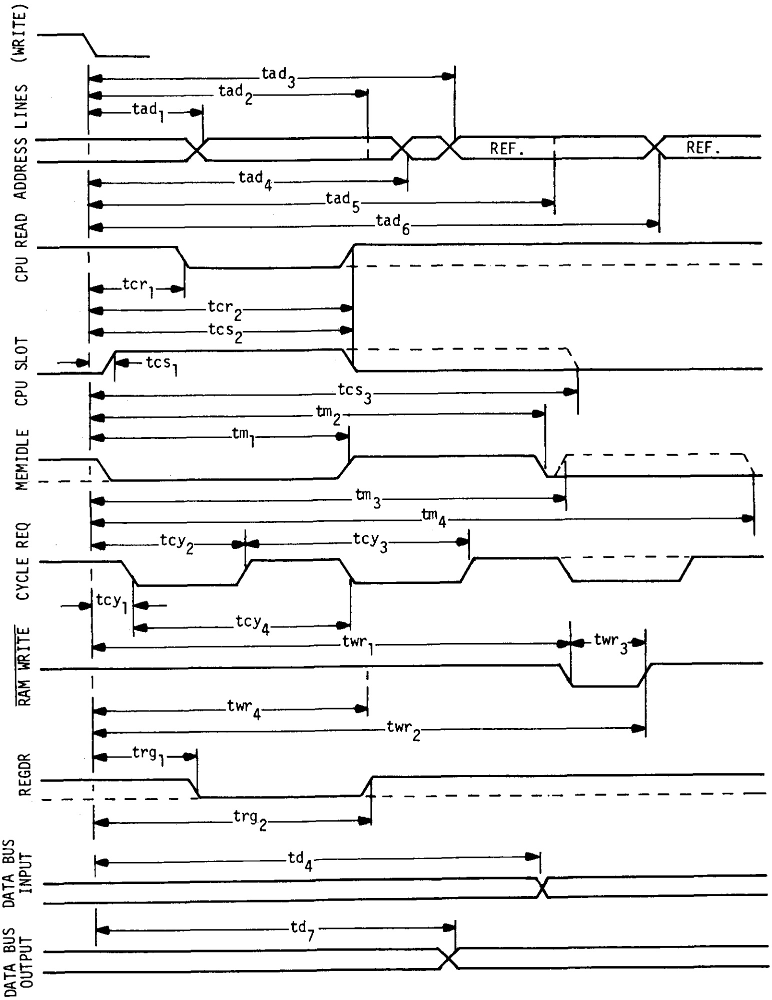
SYMBOL	PARAMETER	MIN	TYP.	MAX	UNITS	TEST CONDITIONS
I_{DD}	V_{DD} Current		35	70	mA	f=2 MHz, outputs unloaded
I_{GG}	V_{GG} Current		13	30	mA	f=2 MHz, outputs unloaded

Summary of 3852 DMI Signal Characteristics

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
DATA BUS	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	
(DBØ - DB7)	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -1ØØμA
	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 1.6mA
	I_{IH}	Input High Current		1	μA	V_{IN} = 6V,3-State mode
	I_{IL}	Input Low Current		-1	μA	V_{IN} = V_{SS},3-State mode
ADDRESS LINES	V_{OH}	Output High Voltage	4.0	V_{DD}	Volts	I_{OH} = -1mA
(ADDRØ - ADDR15)	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 3.2mA
AND	I_L	Leakage Current		1	μA	V = 6V,3-State mode
$\overline{\text{RAM WRITE}}$	I_L	Leakage Current		-1	μA	V = V_{SS},3-State mode
CLOCK	V_{IH}	Input High Voltage	4.0	V_{DD}	Volts	
(ϕ, WRITE)	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_L	Leakage Current		1	μA	V_{IN} = 6V
MEMIDLE,	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -1mA
CYCLE REQ,	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 2mA
CPU READ	I_L	Leakage Current		1	μA	V_{IN} = 6V

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
CONTROL LINES (ROMCØ - ROMC4)	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_L	Leakage Current		1	µA	V_{IN} = 6V
REGDR, CPU SLOT	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -300µA
	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 2mA
	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	Internal Pullup
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_{IL}	Input Low Current (REGDR)	-3.5	-14.0	mA	V_{IN} = .4V & Device outputting a logic "1"
	I_{IL}	Input Low Current (CPU SLOT)	-4.5	-16.0	mA	
	I_L	Leakage Current		1	µA	V_{IN} = 6V

Note: Positive current is defined as conventional current flowing into the pin referenced.



Timing Characteristics for 3852 DMI Output Signals

3852 DMI Output Signals Timing Summary

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	NOTES
$P\Phi$	Φ clock period	--		--		Fig 2.9
td_2	Φ to WRITE - Delay	--		--		Fig 2.9
tad_1	Address delay if PC0	50	300	500	nS	3
tad_2	Address delay to high Z (short cycle with DMA on)	tcs_2+50		tcs_2+200	nS	3
tad_3	Address delay to refresh (short cycle with REF on)	tcs_2+50		tcs_2+400	nS	3
tad_4	Address delay if DC	$2P\Phi-td_2+50$		$2P\Phi+400-td_2$	nS	3
tad_5	Address delay to high Z (long cycle with DMA on)	tcs_3+50		tcs_3+200	nS	3
tad_6	Address delay to refresh (long cycle with REF on)	tcs_3+50		tcs_3+400	nS	3
tcr_1	CPU READ - Delay	50	250	450	nS	1
tcr_2	CPU READ + Delay	$2P\Phi+50-6d2$		$2P\Phi+400-td_2$	nS	1
tcs_1	CPU SLOT + Delay	$80-td_2$		$320-td_2$	nS	1
tcs_2	CPU SLOT - Delay (PC0 access)	$2P\Phi+60-td_2$		$2P\Phi+420-td_2$	nS	1
tcs_3	CPU SLOT - Delay (DC access)	$4P\Phi+60-td_2$		$2P\Phi+420-td_2$	nS	1
tm_1	MEMIDLE + Delay (PC0 access)	$2P\Phi+50-td_2$		$4P\Phi+400-td_2$	nS	1
tm_2	MEMIDLE - Delay (PC0 access)	$4P\Phi+50-td_2$		$4P\Phi+350-td_2$	nS	1
tm_3	MEMIDLE + Delay (DC access)	$4P\Phi+50-td_2$		$4P\Phi+400-td_2$	nS	1
tm_4	MEMIDLE - Delay (DC access)	$6P\Phi+50-td_2$		$6P\Phi+350-td_2$	nS	1

3852 DMI Output Signals Timing Summary (Continued)

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	NOTES
tcy_1	WRITE to CYCLE REQ - Delay	$80-td_2$		$400-td_2$	nS	1, 4
tcy_2	WRITE to CYCLE REQ + Delay	$P\Phi+80-td_2$		$P\Phi+400-td_2$	nS	1, 4
tcy_3	CYCLE REQ + to + Edge Delay		$2P\Phi$			1,4
tcy_4	CYCLE REQ - to - Edge Delay		$2P\Phi$			1, 4
twr_1	$\overline{\text{RAM WRITE}}$ - Delay	$4P\Phi+50-td_2$		$4P\Phi+450-td_2$	nS	3
twr_2	$\overline{\text{RAM WRITE}}$ + Delay	$5P\Phi+50-td_2$		$5P\Phi+300-td_2$	nS	3
twr_3	$\overline{\text{RAM WRITE}}$ Pulse Width	350		$P\Phi$	nS	3
twr_4	$\overline{\text{RAM WRITE}}$ to High Z Delay	tcs_2+40		tcs_2+200	nS	3
trg_1	REGDR - Delay	70	300	500	nS	1
trg_2	REGDR + Delay	$2P\Phi+80-td_2$		$2P\Phi+500-td_2$	nS	1
td_4	WRITE to Data Bus Input Delay			$2P\Phi+1000$	nS	
td_7	WRITE to Data Bus Output Delay	$2P\Phi+100-td_2$		$2P\Phi+850-td_2$	nS	2

Notes:

1. $C_L = 50$ pf
2. $C_L = 100$ pf
3. $C_L = 500$ pf
4. CYCLE REQ is a divide by 2 of Φ for all instructions except the STORE instruction.
5. On a given chip, the timing for all signals will tend to track. For example, if CPU SLOT for a particular chip is fairly slow and its timing falls out near the MAX delay value specified, then the timing for all signals on that chip will tend to be out near the MAX delay values. Likewise for a fast chip whose signals fall near the MIN values. This is a result of the fact that processing parameters (which affect device speed) are quite uniform over small physical areas on the surface of a wafer.
6. Input and output capacitance is 3 to 5 pf typical on all pins except V_{DD}, V_{GG}, and V_{SS}.

DC ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS (Above which useful life may be impaired).

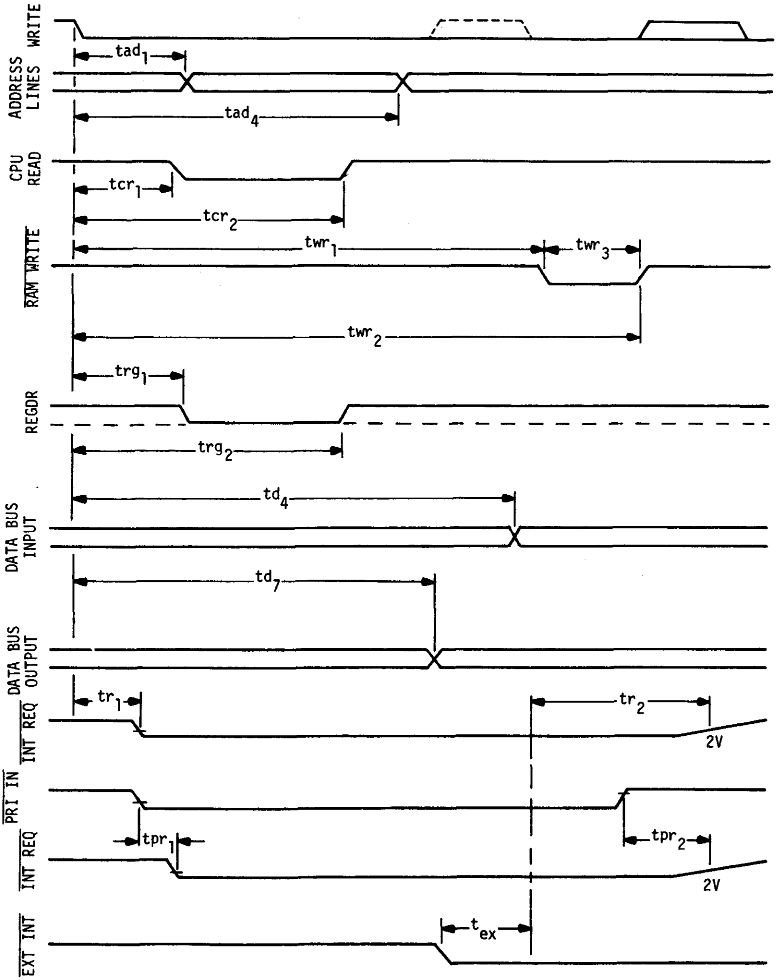
V_{GG}	+15V to -.3V
V_{DD}	+7V to -.3V
All other Inputs & Outputs	+7V to -.3V
Storage Temperature	-55°C to 150°C
Operating Temperature	0°C to 70°C

Note: All voltages with respect to V_{SS}

DC CHARACTERISTICS: V_{SS}=0V, V_{DD}=5V±5%, V_{GG}=12V±5%, T_A=0 to 70°C

SUPPLY CURRENTS

SYMBOL	PARAMETER	MIN	TYP.	MAX	UNITS	TEST CONDITIONS
I_{DD}	V_{DD} Current		35	70	mA	f=2 MHz, outputs unloaded
I_{GG}	V_{GG} Current		13	30	mA	f=2 MHz, outputs unloaded



3853 Signal Timing

3853 SMI Output Signals Timing Summary

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNITS	NOTES
$P\Phi$	Φ clock period	--		--		Fig 2-9
td_2	Φ to WRITE - Delay	--	-	--		Fig 2-9
tad_1	Address delay if PCO	50	300	500	nS	3
tad_4	Address delay if DCO	$2P\Phi-td_2+50$		$2P\Phi+400-td_2$	nS	3
tcr_1	CPU READ - Delay	50	250	450	nS	1
tcr_2	CPU READ + Delay	$2P\Phi+50-td_2$		$2P\Phi+400-td_2$	nS	1
twr_1	$\overline{\text{RAM WRITE}}$ - Delay	$4P\Phi+50-td_2$		$4P\Phi+450-td_2$	nS	3
twr_2	$\overline{\text{RAM WRITE}}$ + Delay	$5P\Phi+50-td_2$		$5P\Phi+300-td_2$	nS	3
twr_3	$\overline{\text{RAM WRITE}}$ pulse	350		$P\Phi$	nS	3
trg_1	REGDR - Delay	70	300	500	nS	1
trg_2	REGDR + Delay	$2P\Phi+80-td_2$		$2P\Phi+500-td_2$	nS	1
td_4	WRITE to Data Bus Input Delay			$2P\Phi+1000$	nS	
td_7	WRITE to Data Bus Output Delay	$2P\Phi+100-td_2$		$2P\Phi+850-td_2$	nS	2
tr_1	WRITE to $\overline{\text{INT REQ}}$ - Delay			430	nS	6
tr_2	WRITE to $\overline{\text{INT REQ}}$ + Delay			430	nS	8
tpr_1	$\overline{\text{PRI IN}}$ to $\overline{\text{INT REQ}}$ - Delay			240	nS	7
tpr_2	$\overline{\text{PRI IN}}$ to $\overline{\text{INT REQ}}$ + Delay			240	nS	2
t_{ex}	$\overline{\text{EXT INT}}$ set-up time	400			nS	-

Notes:

1. C_L = 50 pf
2. C_L = 100 pf
3. C_L = 500 pf

4. On a given chip, the timing for all signals will tend to track. For example, if CPU SLOT for a particular chip is fairly slow and its timing falls out near the MAX delay value specified, then the timing for all signals on that chip will tend to be out near the MAX delay values. Likewise for a fast chip whose signals fall near the MIN values. This is a result of the fact that processing parameters (which affect device speed) are quite uniform.

5. Input and Output capacitance is 3 to 5 pf typical on all pins except V_{DD}, V_{GG}, and V_{SS}.

6. Assume Priority In was enabled ($\overline{\text{PRI IN}}$ = 0) in previous F8 cycle before interrupt is detected in the PSU.

7. PSU has interrupt pending before priority in is enabled.

8. Assume pin tied to $\overline{\text{INT REQ}}$ input of the 3850 CPU.

Summary of 3854 DMA Signal Characteristics

ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS (Above which useful life may be impaired)

V_{GG}	+15V to -.3V
V_{DD}	+ 7V to -.3V
All other Inputs & Outputs	+ 7V to -.3V
Storage Temperature	-55°C to 150°C
Operating Temperature	0°C to 70°C

Note: All voltages with respect to V_{SS}

DC CHARACTERISTICS: V_{SS} = 0V, V_{DD} = +5V±5%, V_{GG} = +12V±5%, T_A = 0 to 70°C

SUPPLY CURRENTS

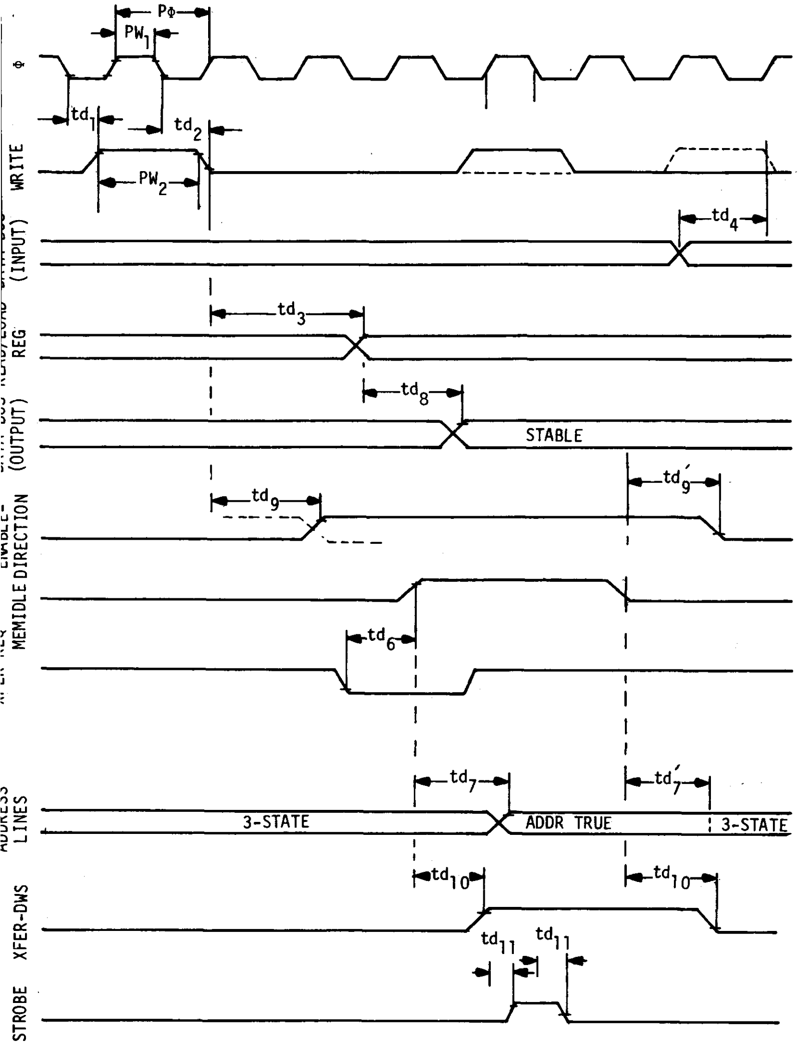
SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
I_{DD}	V_{DD} Current		20	40	mA	f=2MHz, Outputs Unloaded
I_{GG}	V_{GG} Current		15	28	mA	f=2MHz, Outputs Unloaded

Summary of 3854 DMA Signal Characteristics (Continued)

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
DATA BUS (DBØ - DB7)	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -100μA
	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 1.6mA
	I_{IH}	Input High Current		1	μA	V_{IN} = 6V, 3-State mode
	I_{IL}	Input Low Current		-1	μA	V_{IN} = V_{SS},3-State mode
ADDRESS LINES (ADDRØ - ADDR15)	V_{OH}	Output High Voltage	4.0	V_{DD}	Volts	I_{OH} = -1mA
	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 3.2mA
	I_L	Leakage Current		1	μA	V_{IN} = 6V,3-State mode
ENABLE, DIRECTION DWS (DMA WRITE SLOT), XFER, STROBE	V_{OH}	Output High Voltage	3.9	V_{DD}	Volts	I_{OH} = -100μA
	V_{OL}	Output Low Voltage	V_{SS}	.4	Volts	I_{OL} = 2mA
	I_L	Leakage Current		1	μA	V_{IN} = 6V
MEMIDLE, $\overline{\text{XFER REQ}}$	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_L	Leakage Current		1	μA	V_{IN} = 6V

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
LOAD REG,	V_{IH}	Input High Voltage	3.5	V_{DD}	Volts	
READ REG,	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
P1, P2	I_L	Leakage Current	0	1	µA	$V_{IN} = 6V$
WRITE, φ	V_{IH}	Input High Voltage	4.0	V_{DD}	Volts	
	V_{IL}	Input Low Voltage	V_{SS}	.8	Volts	
	I_L	Leakage Current	0	1	µA	$V_{IN} = 6V$

NOTE: Positive current is defined as conventional current flowing into the pin referenced.



3854 DMA Device Signals and Timing

contents of a selected register onto the DATA BUS only while READ REG is high, if there was a similar address match during the prior cycle. I/O address assignment is made using pins P1 and P2

3854 DMA Device Signals Summary

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNITS	NOTES
$P\Phi$	Φ Clock Period	.5		10	μS	Note 1
PW_1	Φ Pulse Width	180		$P\Phi$-180	nS	t_r, t_f = 50 nS typ
td_1	Φ to WRITE + Delay	60		300	nS	Note 1
td_2	Φ to WRITE - Delay	60		250	nS	Note 1
PW_2	WRITE Pulse Width	$P\Phi$-100		$P\Phi$	nS	t_r, t_f = 50 nS typ
td_3	WRITE to READ/LOAD REG Delay			600	nS	
td_4	DB Input Set-up Time			300	nS	
td_6	$\overline{\text{XFER REQ}}$ to MEMIDLE Set-up	200			nS	
td_7	MEMIDLE to ADDR True	50	200	500	nS	C_L = 500 pf
td_7'	MEMIDLE to ADDR 3-State	30		250	nS	C_L = 500 pf
td_8	READ REG to DB Output	40		300	nS	C_L = 100 pf
td_9	WRITE to ENABLE & DIRECTION + Delay			450	nS	C_L = 50 pf
td_9'	MEMIDLE to ENABLE - Delay			400	nS	C_L = 50 pf
td_{10}	MEMIDLE to XFER & DWS + Delay			300	nS	C_L = 50 pf
td_{10}	MEMIDLE to XFER & DWS - Delay			300	nS	C_L = 50 pf
td_{11}	Φ to STROBE + Delay	30		200	nS	C_L = 50 pf
td_{11}	Φ to STROBE - Delay	30		200	nS	C_L = 50 pf

Notes:

1. These specifications are those of Φ and WRITE as supplied by the 3850 CPU.
2. Input and output capacitance is 3 to 5 pf typical on all pins except V_{DD}, V_{GG}, and V_{SS}.

Chapter 3
THE NATIONAL SEMICONDUCTOR SC/MP

escribing this microprocessor immediately after the F8 is a good idea because he two products contrast well. The F8 differs more markedly from minicomputers han any other 8-bit microprocessor described in this book. SC/MP, by way of ontrast, is one of the more minicomputer-like products.

Vhen we say that SC/MP is a more minicomputer-like product what we mean is that its ogic distribution and instruction set are more traditional.

he SC/MP is a single device, not a family of devices. The single device provides rithmetic and Logic Unit, Control Unit, registers and memory addressing logic, exactly hat you would expect to find in any minicomputer Central Processing Unit. There are o additional devices forming an SC/MP "family", comparable to the F8 PSU DMI or MI. Additional logic that will support SC/MP consists of standard off-the-shelf buffers, idirectional drivers, ROM and RAM. This approach is the same as that taken with PACE, ational Semiconductor's other single-chip microprocessor which is described later in is book among the 16-bit microprocessors. Both SC/MP and PACE provide a wealth of ontrol signals and thus limit the need for special-purpose support chips.

he only current manufacturer for SC/MP is:

NATIONAL SEMICONDUCTOR INC
2900 Semiconductor Drive
Santa Clara, CA 95050

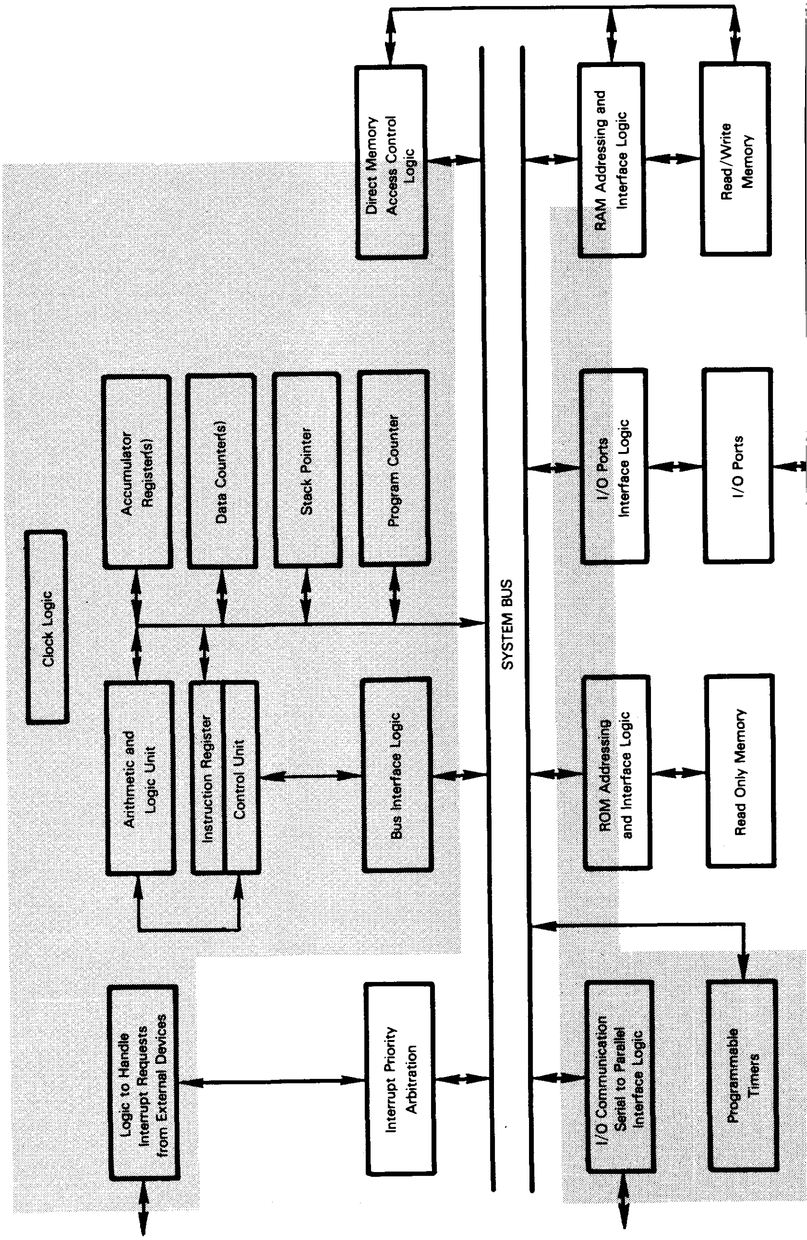
lthough there is an agreement between Rockwell International and National Semiconuctor to exchange microcomputer technical information and produce each other's roducts, at the present time Rockwell International has not elected to second source C/MP.

igure 3-1 conceptually illustrates the logic functions which are implemented on e SC/MP chip. One of the weaknesses of Figure 3-1, and the equivalent figures r the other microcomputers, is that the way in which logic functions are impleented cannot be identified. SC/MP, for example, implements non-CPU logic at a ery elementary level, well suited for simple applications only.

SC/MP
SERIAL I/O

lonetheless, Figure 3-1 does reveal a few of the rather nusual capabilities provided by SC/MP. Notice that Serial-to-arallel Interface Logic is shown as implemented by the C/MP chip. SC/MP has two serial I/O device pins, one for serial binary input data, the ther for serial binary output data. The assembly and disassembly of serial-to-parallel ata is accomplished by one SC/MP instruction.

igure 3-1 also shows Programmable Timer logic as being implemented by the C/MP chip. This is barely justifiable — the SC/MP instruction set includes a Delay struction that is used to generate timed durations ranging from 13 to 131,593 icrocycles. Note, however, that during this delay interval the CPU can be performing o other actions: the CPU is, in effect, operating solely as a programmable timer. This is


Clock Logic
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Arithmetic and Logic Unit
Instruction Register
Control Unit
Bus Interface Logic
Logic to Handle Interrupt Requests from External Devices
Interrupt Priority Arbitration
Direct Memory Access Control Logic
SYSTEM BUS
RAM Addressing and Interface Logic
Read/Write Memory
I/O Ports Interface Logic
I/O Ports
ROM Addressing and Interface Logic
Read Only Memory
I/O Communication Serial to Parallel Interface Logic
Programmable Timers

obviously quite different from having a separate logic device that performs this timer function within a system. Once again, this points out the weakness of a generalized representation such as Figure 3-1.

SC/MP DMA AND MULTIPROCESSOR LOGIC

One other area of non-CPU logic shown as being implemented by SC/MP further illustrates this point. **A portion of the Direct Memory Access (DMA) logic is provided by SC/MP using a few signals to control bus access.** A significant amount of external logic would still be required to obtain an operational DMA system. Therefore, Figure 3-1 can be misleading because it cannot indicate the way in which the CPU implements a particular function. In this particular case there is also a significant area of non-CPU logic provided by SC/MP that is nowhere indicated by Figure 3-1: **The signals that can be used for DMA are primarily intended to simplify the design of multiprocessor systems.** This is a very unusual logic function for a CPU to provide and therefore is not even suggested in Figure 3-1. But for SC/MP, the inclusion of this multiprocessor-oriented logic makes a lot of sense: its low cost and modest performance makes it a likely candidate for multiprocessor systems.

SC/MP AND SC/MP-II

There are two versions of the SC/MP CPU: the original version uses P-channel silicon-gate MOS/LSI technology and its part number is ISP-8A/500; the new version (SC/MP-II) uses N-channel technology and its part number is ISP-8A/600. The two versions are functionally equivalent and fully compatible in terms of object code and pin configuration. (A few minor signal level conversions are required for complete signal compatibility: see Figure 3-3.) **The SC/MP-II provides some significant advantages over the original version — *it is twice as fast and uses only one-fourth the power of the original P-channel version.* Additionally, while SC/MP requires two power sources (a +5 volt and a -7 volt supply), SC/MP-II needs only a single +5 volt supply.** Throughout this chapter, we will simply refer to the CPU as SC/MP: all the descriptions apply to both versions of the CPU unless we specifically mention SC/MP-II.

SC/MP INSTRUCTION EXECUTION SPEED

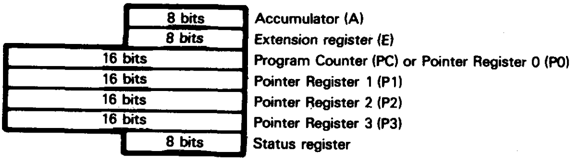
Both versions of the SC/MP CPU have *an on-chip clock oscillator* and can use a capacitor, crystal, or TTL clock input to drive the clock. The P-channel SC/MP can run at a maximum frequency of 1 megahertz which results in instruction execution times in the range of 10 to 50 microseconds. SC/MP-II can operate at frequencies up to 4 megahertz with resulting instruction execution times in the range of 5 to 25 microseconds. Notice, that although the input frequency for SC/MP-II can be four times that of SC/MP, the instruction execution time for SC/MP-II is twice as fast (not four times as fast): This is because of internal differences in the way the on-chip clock oscillator uses the timing inputs.

SC/MP LOGIC LEVEL

Both versions of SC/MP provide TTL-compatible input and output signals.

SC/MP PROGRAMMABLE REGISTERS

SC/MP has an 8-bit Accumulator, an 8-bit Extension register, a 16-bit Program Counter, three 16-bit Pointer registers, and an 8-bit Status register. These programmable registers are illustrated as follows:



The Accumulator is a single, primary Accumulator, as described for our hypothetical microcomputer.

The Extension register is used to assemble or disassemble serial-to-parallel dat for serial data input and output. This register is also used as a buffer for the Accumulator.

SC/MP MEMORY PAGES

The Program Counter is 16 bits wide; therefore up to 65,536 bytes of memory may be addressed in the normal course of events. The four high-order bits of the Program Counter represent page select bits; therefore the memory of a SC/MP system is divided into 16 pages of 4096 words each.

Notice that the Program Counter is shown as Pointer Register 0; this is done becaus some instructions move data between Pointer registers including the Program Counte There is one other unusual fact about the SC/MP Program Counter: **the four most significant bits (the page select bits) of the Program Counter are never incremente during the instruction fetch sequence. Instead, when the last address of a page i reached, the Program Counter "wraps-around" to the first address of the curren page.** For example, if the Program Counter contains $2FFF_{16}$, when it is incremente the new contents of the Program Counter will be 2000_{16} instead of 3000_{16}. The pag select bits of the Program Counter can only be changed by executing an instructio that loads a new value into the most significant bits of the Program Counter.

Note that the four high-order address bits are not output on separate address pins; instead they are output on the data lines at the beginning of an input/output cycle an must be demultiplexed by external logic in order to generate page select signals.

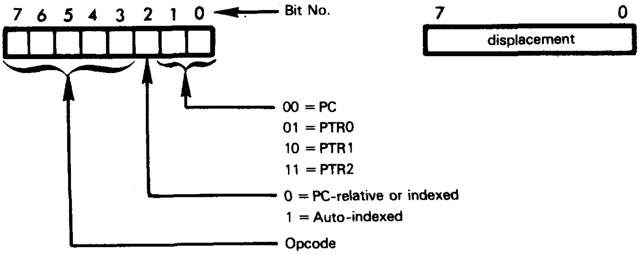
The three Pointer registers are Data Counters that can also be used as Inde registers, Page Pointers, or Stack Pointers. Typically, you would assign a specifi function to each register. For example, the following assignments might be used:

P1	-	ROM Pointer
P2	-	Stack Pointer
P3	-	Subroutine Pointer

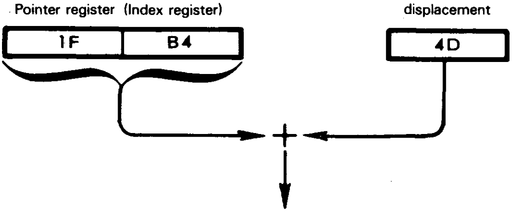
These arbitrary assignments also reveal several interesting facts about the architectur of SC/MP. First, the SC/MP CPU does not provide an on-chip stack; instead, a stack ca be maintained in memory using one of the Pointer registers as a Stack Pointer. Secondly, the SC/MP instruction set does not include a Jump-to-Subroutine instruction: one of the Pointer registers must be used to hold subroutine addresses which ca then be swapped with the Program Counter. We will discuss this in detail when we describe the SC/MP instruction set.

ADDRESSING MODES

The SC/MP memory reference instructions use program-relative direct addressing, indexed addressing, and auto-indexed addressing. All memory reference instructions are two-byte instructions and have the following object code forma



’rogram relative and indexed addressing are as described in Volume I, Chapter 6. Ve will just re-emphasize here that all addressing in SC/MP is paged and uses the vrap-around technique — that is, there is no carry from the low order 12 bits of an ad-ress into the most significant 4 bits of an address. We mentioned this earlier when we iscussed the Program Counter and it also applies to indexed addressing. Thus, if the um of the Index register (that is, one of the Pointer registers) and the second object ode byte contents (displacement) is more than FFF_{16}, the Carry bit will be discarded. 'his may be illustrated as follows:



Effective Address = 1FB4 + 4D

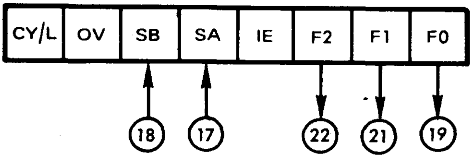
1 F B 4
+ 4 D
Expected result = 2 0 0 1
Discard Carry
Actual Result is 1001_{16}

emember, *all arithmetic operations during address formation, regardless of the* ddressing mode, obey this wrap-around technique: there is never a carry from it 11 into bit 12.

he auto-indexing mode of addressing provided by SC/MP instructions is actually n auto-increment/auto-decrement operation. When auto-indexing is specified, the isplacement, as a signed binary number, is added to the contents of a Pointer register ı order to compute an effective address. If the displacement is less than zero, the ɔinter register is decremented by the displacement before the memory access. If the isplacement is equal to or greater than zero, then the contents of the Pointer register is ıe effective address and the Pointer register contents are incremented by the displace-ıent after the memory access. **This method of auto-increment and auto-decrement ddressing is the same as that described in Volume I with one significant ifference: *SC/MP allows an address to be incremented or decremented by any* alue in the range 0 - 127 instead of just by a value of one.**

C/MP STATUS REGISTER

C/MP has a programmable 8-bit Status register which may be illustrated as ɔllows:



rcled numbers represent device pin numbers to which bits of the Status register are ɔnnected.

The Carry (CY), Link (L) and Overflow (OV) status bits are typical microcompute status bits as were described in Volume I, Chapter 7.

The two sense bits, SB and SA, are tied to SC/MP device pins. These two bit directly reflect the state of the logic signals applied to the device pins and thu can be used to detect external events. Although there are no SC/MP instruction that allow you to directly jump or branch on the condition of one of these bits, a sequence of masking and testing instructions can be used to accomplish the same effec albeit somewhat slower. **The SA and SB bits are read-only bits. Instructions ma read the status of these two bits, but only incoming signals may change their condition.** For example, an instruction that moves the contents of the Accumulator to th Status register may modify any of the other status bits, but bits 4 and 5 will not change **The SA bit serves a dual function. If the Interrupt Enable (IE) bit is set to one, th SA input serves as the interrupt input.** We will discuss interrupt processing later i this chapter.

F0, F1 and F2 are control flags that are tied to SC/MP device pins. The state o these three flags may be changed under program control and may be used to control external devices. When the state of any of these flags is changed, it is immediately reflected by a change in the signal level at the associated device pin.

SC/MP CPU SIGNALS AND PIN ASSIGNMENTS

Figure 3-2 illustrates the SC/MP pins and signals. A description of these signals i useful as a guide to the way in which an SC/MP microcomputer system works.

The 12 address lines AD00 - AD11 output memory and I/O device addresses These are tristate lines, and may be floated, giving external logic control of th Address Bus. The four most significant address bits (AD12 - AD15) are tim multiplexed on the data lines.

The eight Data Bus lines DB0 - DB7 are multiplexed, bidirectional data lines vi which 8-bit data units are input and output, and on which statuses and addres bits are output at the beginning of any input/output cycle. Statuses on the Dat Bus identify the type or purpose of the input/output cycle. The address bits on th Data Bus are the four most significant address bits (AD12 - AD15) which can b used to generate page select signals for memory or peripheral devices. Table 3- describes the status and address information that is output on the Data Bus. Lik the address lines, the data lines are tristate.

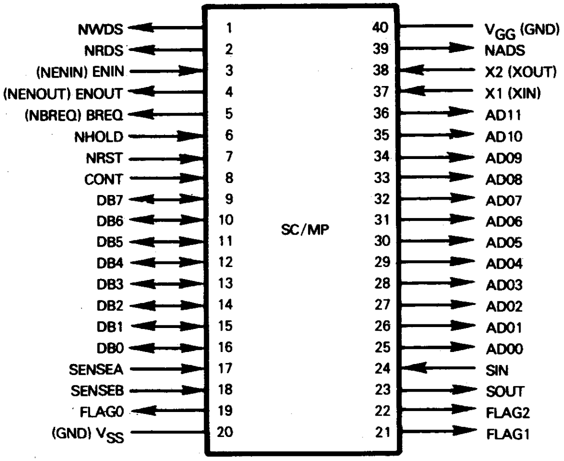
SENSEA SENSEB FLAG0, 1, and 2 are pin connections for the similarly name Status register bits described earlier.

SIN and SOUT are used in combination with the SIO instruction for serial input c data to the Extension register and serial output of data from the Extensio register.

The remaining signals (excluding clock, power and ground) may be divided into bus access, Data Bus definition, and timing control signals.

You will notice that some of the SC/MP pins in Figure 3-2 have two sets of signal names: the names enclosed in parentheses reflect the nomenclature used with SC/MP-II. Aside from the clock and power signals which we shall discuss separately, the only difference between SC/MP and SC/MP-II is in the polarity of bus access signals: Bus Request (BREQ/NBREQ), Enable In (ENIN/NENIN), and Enable Out (ENOUT/NENOUT). The "N" prefix to each of the SC/MP-II signals indicates that thes signals are negative-true — as opposed to the positive- (or logic "1") true signals for th P-channel SC/MP. **In the descriptions that follow, we will use P-channel SC/M nomenclature.** If you are using the N-channel SC/MP-II version, you must simply inve these signals.

SIGNAL DIFFERENCES BETWEEN SC/MP (P-CHANNEL) AND SC/MP-II (N-CHANNEL)



PIN NAME†	DESCRIPTION	TYPE
X1,X2	Crystal/Capacitor Connections	Input
*DB0 - DB7	Data Bus	Bidirectional, Tristate
*AD00 - AD11	Address Lines	Output, Tristate
*SENSEA,SENSEB	External Status Input	Input
*FLAG0,1,2	Flags	Output
*NRST	Reset	Input
*CONT	Halt/Continue	Input
*BREQ (NBREQ)	Bus Request/Busy	Bidirectional
*ENIN (NENIN)	Data Bus Enable	Input
*ENOUT (NENOUT)	CPU Bus Access Status	Output
*NADS	Address on Data Bus	Output
*NRDS	Data Input Strobe	Output, Tristate
*NWDS	Data Output Strobe	Output, Tristate
*NHOLD	Clock Delay	Input
SIN	Serial Data In	Input
SOUT	Serial Data Out	Output
V_{GG},V_{SS}(V_{CC},GND)	Power and Ground	

*These signals connect to the System Bus.

†Signals in parenthesis are SC/MP-II signal names.

Figure 3-2. SC/MP CPU Signals And Pin Assignments

SC/MP BUS ACCESS CONTROL SIGNALS

Before the SC/MP CPU can begin any input/output operation, it must gain access to the System Busses. This approach reflects the design philosophy behind SC/MP. It is a relatively low-cost, low-performance CPU and the designers anticipated that it would frequently be used in multiprocessor systems or in systems utilizing Direct Memory Access. Accordingly, three signals are provided to control access to the System Busses.

BREQ is used as a bus busy input indicating that some other device is using the System Busses, and as a bus request output when the System Busses are free and SC/MP requires access to the busses.

ENIN is a control signal which is input to the CPU by external logic. When ENIN is low, the CPU is denied access to the System Busses and the SC/MP address and data lines are held in tristate mode.

ENOUT is the CPU's output response to ENIN. When output high, ENOUT indicates that ENIN is high; therefore, the CPU can gain access to the System Busses, but it has not done so. If ENOUT is low, it indicates either that ENIN is low, therefore the CPU is being denied access to the System Busses or, if ENIN is high, then it indicates that the CPU is using the System Busses.

SC/MP DATA BUS DEFINITION SIGNALS

When the CPU has gained access to the System Busses, three signals identify the way in which the CPU is using the Data Bus.

NADS is output to indicate that a valid address has been output on the address lines and that the low order four bits of the Data Bus contain the high order four bits of a 16-bit address. NADS also indicates that status information is being output on the high order four bits of the Data Bus.

NRDS, when output by the CPU, indicates that the CPU wishes to receive data on the Data Bus.

NWDS when output by the CPU, indicates that data is being output by the CPU on the Data Bus. NWDS may be used by external logic as a write strobe.

SC/MP TIMING CONTROL SIGNALS

There are three signals which control CPU timing.

NRST is a system reset signal. When input low, it aborts any in-process operations. When returned high, all programmable registers are cleared, and program execution begins with the instruction fetched from memory location 0001_{16}.

CONT may be input to stop the CPU between instructions. When CONT is input low, all CPU operations are halted after the current instruction execution has been completed. The CPU remains halted until CONT goes high.

NHOLD is an input signal used during input/output operations to lengthen the allowed time interval for devices to respond to CPU access requests.

SC/MP TIMING AND INSTRUCTION EXECUTION

The SC/MP timing for instruction execution is very simple. Instruction execution times are expressed in terms of microcycles. A typical instruction is executed in 10 microcycles and one or more of these microcycles is an input/output cycle. The length of a microcycle depends on the frequency of the clock inputs to the CPU: with the P-channel SC/MP, the minimum microcycle length is 2 microseconds; for SC/MP-II, the N-channel version, minimum microcycle length is 1 microsecond. Thus, typical instruction execution time is 20 microseconds for the P-channel SC/MP, and 10 microseconds for SC/MP-II. **All microcycles, whether internal machine cycles or input/output cycles, are of the same length: the only variance occurs when the NHOLD signal is used to stretch an input or output cycle.**

There are basically only three types of SC/MP machine (or micro) cycles: data input (read) cycles, data output (write) cycles, and internal microcycles. The execution of each instruction is merely a concatenation of these three types of microcycles.

SC/MP I/O CYCLE STATUS INFORMATION

SC/MP does, however, output some status information at the beginning of every input or output cycle which provides a more precise definition of the purpose of that microcycle. Table 3-1 lists the information which may be output on the data line at the beginning of an I/O cycle (when NADS is low). Table 3-2 defines the way in which the status information may be interpreted to identify the various possible types of microcycles.

Table 3-1. Status And Address Output Via The Data Lines During The Beginning Of An I/O Cycle

SYMBOLS	DATA BUS BIT	DEFINITION
H-Flag	7	Indicates that a Halt instruction has been executed.
D-Flag	6	Indicates that a Delay instruction has been executed and that a delay cycle is starting.
I-Flag	5	Indicates that the CPU is in the fetch cycle for the first byte of an instruction.
R-Flag	4	When high, indicates that the I/O cycle is a read cycle and that input data should be placed on the Data Bus when NRDS is active. When low, indicates that the I/O cycle is a write cycle and that the Data Bus will contain output data when NWDS is active.
AD15	3	The four most significant bits of a 16-bit address. Can be used as page select signals.
AD14	2	
AD13	1	
AD12	0	

Table 3-2. Statuses Output On The Data Bus For Various Types Of Machine Cycles

Status Information	Data Bus Bit	TYPE OF MACHINE CYCLE				
		Instruction Fetch	Halt Instruction	Delay Instruction	Data Input (Read)	Data Output (Write)
H-Flag	7	0	1	0	0	0
D-Flag	6	0	0	1	0	0
I-Flag	5	1	1	0	0	0
R-Flag	4	1	1	1	1	0

SC/MP BUS ACCESS LOGIC

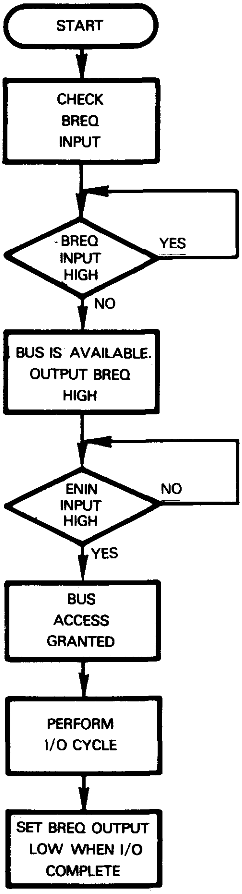
Since the SC/MP CPU must gain access to the System Busses before it can perform an input or output cycle, we will describe the bus access logic before discussing input/output cycles.

Figure 3-3 illustrates the bus access logic processing sequence that occurs whenever the SC/MP CPU is going to perform an input/output cycle.

First, the bidirectional BREQ line is tested. If the BREQ input is high, it indicates that the System Bus is currently in use: the CPU holds the outputs of the address and data lines, and the NRDS and NWDS signals in the high-impedance (tristate) mode.

When the BREQ input signal is low (or goes low) it indicates that the System Bus is free, and the CPU then outputs a logic "1" on the BREQ line. This informs external devices (for example, other SC/MP CPUs or a DMA controller) that a request for bus access has been initiated.

The CPU next tests the state of the ENIN input line. ENIN is essentially the "bus grant" signal: if it is low, it indicates the Bus Request (BREQ) is denied and the CPU remains in an idle state with its output held in the high impedance mode. When the ENIN input is high (or goes high) it indicates that the CPU's bus request has been granted and the I/O cycle can now be initiated.



Initiate an I/O cycle. (Instruction fetch, data input, or data output)

If BREQ input high, bus is busy. Address and data lines, and NRDS, NWDS held in tristate mode.

Request bus access.

If ENIN low, bus access is denied until ENIN goes high.

Figure 3-3. SC/MP Bus Access Logic Processing Sequence

When the I/O cycle has been completed, the CPU sets the BREQ output low to indicate that it has finished using the System Bus and that its outputs are once again in the high impedance mode.

SUSPENSION OF A SC/MP I/O CYCLE

There are a couple of aspects of the bus access sequence which are not revealed by Figure 3-3.

First, the SC/MP CPU has the rather unusual capability of suspending an I/O operation after it has already begun. If the ENIN input line goes low while the CPU has access to the bus, the SC/MP address and data lines will go to the high impedance state, thus relinquishing access to the System Busses. The BREQ output signal will remain high and, when the ENIN input line subsequently goes high once more, the input/output cycle which had been suspended will begin again.

This ability to suspend an I/O cycle might be quite useful in a system where bus access is granted on a priority basis. In such a system, it is conceivable that one or more of the system devices (another CPU, for example) might have overriding priorities and require immediate access to the System Busses. The SC/MP bus access logic we've just described allows this to be accomplished with no difficulty whatsoever. **There is, however, one gray area in this I/O-suspend function.** If a SC/MP I/O cycle is nearly complete, it would seem to be more efficient to go ahead and complete the cycle rather than suspending it and then restarting the entire cycle later. This is precisely what SC/MP does. Unfortunately, the SC/MP literature does not tell us where this "point-of-no-return" lies within an I/O cycle. One would assume, or at least hope that this point is prior to the time when NRDS or NWDS is sent out. These signals are the read and write strobe signals; if they were repeated when an I/O cycle was restarted, the same data might be read or written twice — a potentially vexing situation. However, you are at least assured that if ENIN goes low while SC/MP is performing an I/O cycle, the cycle will be performed — either by continuing to completion or by being restarted when the System Busses are again available.

SC/MP ENOUT SIGNAL USED TO ESTABLISH ACCESS PRIORITIES

If you refer back to Figure 3-3 once again, you will notice that there is no mention of the third SC/MP bus access control signal — ENOUT. This is not an oversight — it is simply due to the fact that the ENOUT signal performs a rather specialized function which is not necessary to an understanding of the SC/MP bus access logic. **The primary function of the ENOUT output signal is as an enabling signal in systems where a "daisy chain" technique is used to establish priorities for bus access.** We will defer a discussion of this use of ENOUT until later in this chapter when we discuss the use of SC/MP in multiprocessor and DMA systems.

SC/MP I/O WITH BUS ACCESS LOGIC CONTINUOUSLY ENABLED

If the SC/MP CPU is used in a single-processor, non-DMA system then there is no need for the built-in bus access logic. In these cases, which may in fact be in the majority, the bus access signals should be connected so that the SC/MP CPU is always guaranteed immediate access to the System Busses. This is easily accomplished by making the following connections:

	SIGNAL	CONNECT TO
SC/MP	BREQ ENIN ENOUT	VGG through a pull-down resistor. VSS Leave unterminated
SC/MP-II	NBREQ NENIN NENOUT	VCC via external resistor Ground Leave unterminated

n the descriptions of SC/MP input/output operations that follow, we will always assume that the SC/MP CPU has already been granted access to the System Busses, and that this access is not interrupted (or suspended).

SC/MP INPUT/OUTPUT OPERATIONS

Once the SC/MP CPU has control of the System Busses, an actual input or output cycle can begin. As we mentioned earlier in this chapter, the execution of any SC/MP instruction includes some combinations of input/output cycles and internal machine cycles. **Figure 3-4 illustrates the bus utilization required for each of the SC/MP instructions, and also reveals an interesting, non-obvious fact about SC/MP input/output operations.** Observe that each bus utilization interval is shown as being two microcycles in duration. This is true because **each input/output operation effectively requires two microcycles.** The CPU spends a portion of the first microcycle gaining access to the System Bus and placing address and status information on the address and data lines. The actual data transfer (read or write) occurs during the second microcycle. This can be confusing if you are designing a DMA or multiprocessor system: the actual time that the bus is available is a great deal less than you would expect if you based your computations solely on the number of read and write cycles required for each instruction. To make this more clear, refer to Table 3-3 which lists the read cycles, write cycles, and total microcycles required for execution of each SC/MP instruction. If you total up each of the columns from this table, you come up with the following figures:

Total Read Cycles	=	79
Total Write Cycles	=	3
Total Input/Output Cycles	=	82
Total Microcycles	=	466

Based on these figures, it would appear that bus utilization is less than 20% (82/466). However, since the CPU maintains control of the bus for approximately two microcycles each time a read or write cycle is performed, the actual bus utilization is quite a bit greater than you would have expected. For precise timing parameters refer to the data sheets at the end of this chapter. Keep in mind that bus utilization computations should be based not only on these data sheets, but also on the actual program being used since bus utilization is directly related to the composition of instructions which comprise your program — these calculations can differ significantly from any theoretical calculations based solely on a CPU's complete instruction set.

Now, having discussed those areas of SC/MP bus access and utilization which might be confusing, let us proceed to examine the actual data input/output operations — we will find that these SC/MP operations are quite straightforward.

SC/MP DATA INPUT CYCLE

Figure 3-5 illustrates the timing for a standard SC/MP data input cycle. This timing applies regardless of whether the input cycle is to access data from memory or peripheral devices and also applies to instruction fetch operations.

Once the CPU has gained access to the System Busses, **the input cycle begins by presenting address and statuses on the address and data lines.** When the NADS signal is sent out, the least significant 12 bits of address data are valid on the SC/MP address lines, and the SC/MP data lines are outputting status information and the most significant 4 bits of address information. Table 3-1 defines the information that is output on the data lines while NADS is true. When these address bits and/or status bits need to be latched, either the leading or trailing edge of NADS can be used as a clock signal.

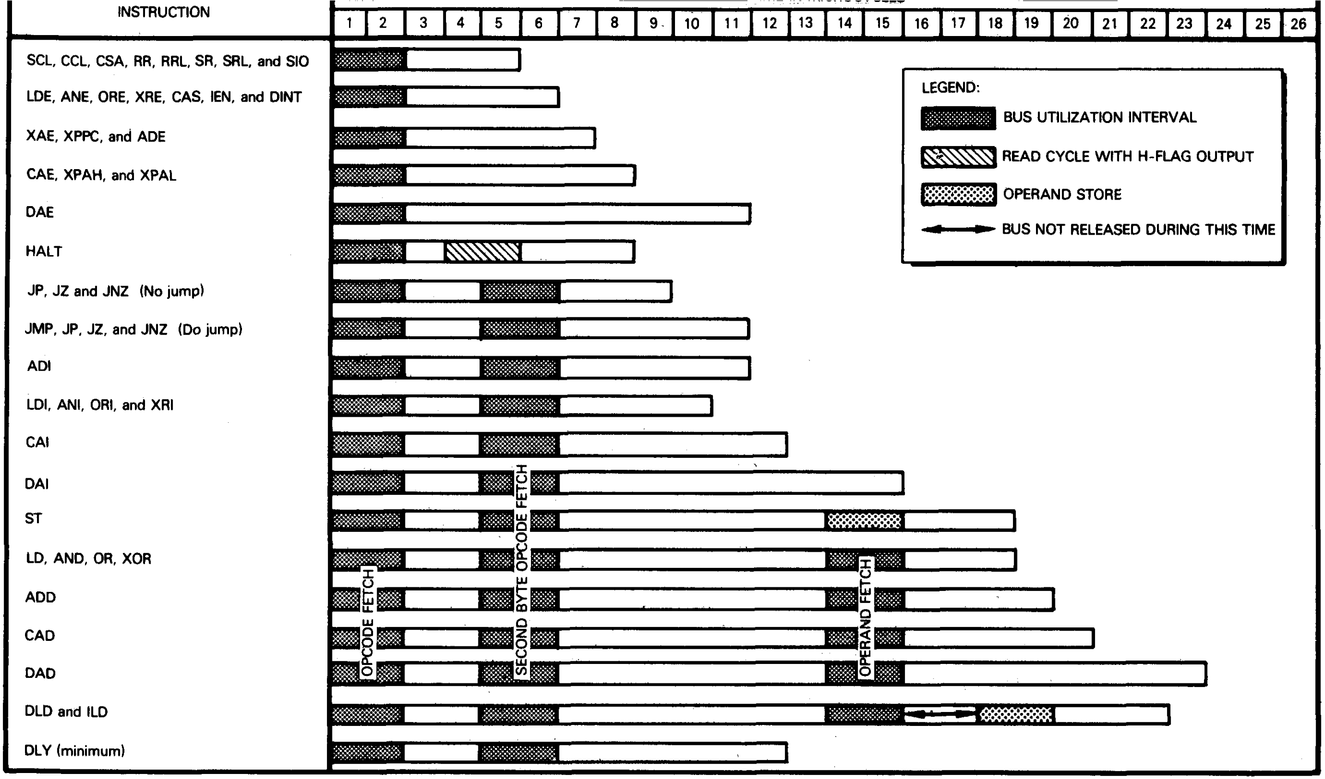


Figure 3-4. Bus Utilization Of Each SC/MP Instruction

Table 3-3. SC/MP Instruction Execution Times

INSTRUCTION	READ CYCLES	WRITE CYCLES	TOTAL MICROCYCLES
ADD	3	0	19
ADE	1	0	7
ADI	2	0	11
AND	3	0	18
ANE	1	0	6
ANI	2	0	10
CAD	3	0	20
CAE	1	0	8
CAI	2	0	12
CAS	1	0	6
CCL	1	0	5
CSA	1	0	5
DAD	3	0	23
DAE	1	0	11
DAI	2	0	15
DINT	1	0	6
DLD	3	1	22
DLY	2	0	13 - 131593
HALT	2	0	8
IEN	1	0	6
ILD	3	1	22
JMP	2	0	11
JNZ	2	0	9, 11 for Jump
JP	2	0	9, 11 for Jump
JZ	2	0	9, 11 for Jump
LD	3	0	18
LDE	1	0	6
LDI	2	0	10
NOP	1	0	5
OR	3	0	18
ORE	1	0	6
ORI	2	0	10
RR	1	0	5
RRL	1	0	5
SCL	1	0	5
SIO	1	0	5
SR	1	0	5
SRL	1	0	5
ST	2	1	18
XAE	1	0	7
XOR	3	0	18
XPAH	1	0	8
XPAL	1	0	8
XPPC	1	0	7
XRE	1	0	6
XRI	2	0	10

Note: If slow memory is being used, the appropriate delay should be added for each read or write cycle.

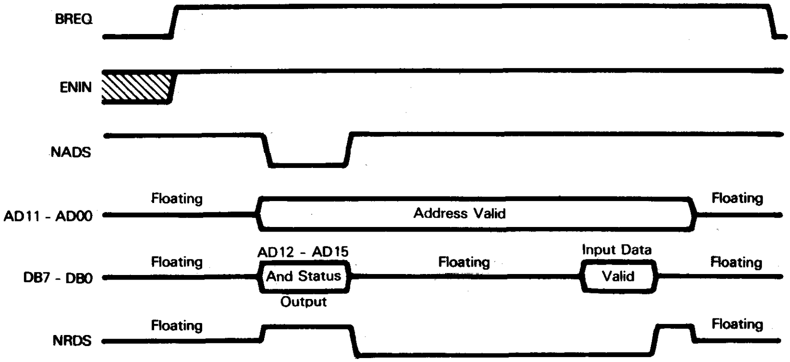


Figure 3-5. SC/MP Data Input Cycle

Shortly after the trailing edge of NADS, the Data Bus is floated and the Read Data Strobe (NRDS) signal is output. Valid input data is expected prior to the trailing edge of NRDS

SC/MP DATA OUTPUT CYCLE

The SC/MP data output cycle begins in the same way as the data input cycle. The only difference is that immediately after the status/address information is output on the data lines, the write or output data is placed on the data lines. As shown in Figure 3-6, the NWDS signal is sent out to indicate when valid output data is prese Either the leading or trailing edge of NWDS could be used to latch the output data i external data latches.

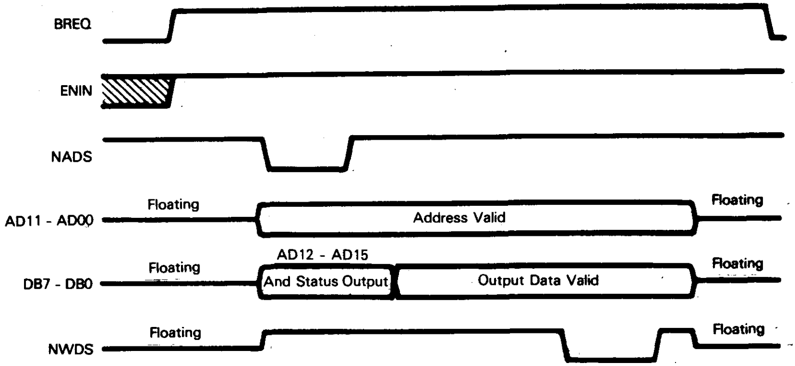


Figure 3-6. SC/MP Data Output Cycle

SC/MP NHOLD SIGNAL FOR SLOW I/O OPERATIONS

The data input/output cycles just described allow approximately one microcycle for external logic to respond. If additional access time is required, the NHOLD input signal to the CPU can be used to lengthen an input/output cycle. The NHOLD signal can be set low any time prior to the trailing edge of NRDS or NWDS as shown in Figure 3-7; this causes the trailing edge of NRDS or NWDS to be delayed until after NHOLD has been returned high. On data input cycles, the time until valid input data must be presented is simply delayed. On data output cycles, the valid output data is maintained on the data lines by the CPU until the delayed trailing edge of NWDS

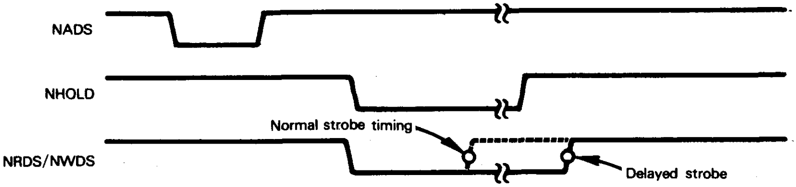


Figure 3-7. NHOLD Signal Used To Lengthen SC/MP I/O Operation

The NHOLD signal causes the I/O cycle to be lengthened in increments of 1/2 microcycle. There is no limit on the duration of the NHOLD signal.

THE SC/MP HALT STATE

The SC/MP Halt state differs from those described for other microprocessors in this book in one significant and unusual way — execution of the SC/MP Halt instruction does not cause the CPU to enter the Halt state. Instead, when SC/MP executes a Halt instruction, it simply outputs the H-Flag status on data line 7 (DB7) when NADS is true.

In order to actually place the CPU in the Halt state the CONT input signal to the CPU must be forced low.

You can use external logic to force CONTIN low either in response to the H-Flag or completely asynchronously: whenever a low is applied to the CONTIN input, the CPU enters the Halt state upon completion of the current instruction. Figure 3-8 shows a circuit that can be used to force the CPU into the Halt state when a Halt instruction is executed. When DB7 is output high while NADS is true, it indicates the Halt instruction has been executed: this combination of events is used to generate a low-going pulse (NHALT) which is applied to the clear (CLR) input of a D flip-flop. The Q output of the flip-flop is applied to the CONT input signal to the CPU. Thus, whenever a Halt instruction is executed, the CPU will be forced into the Halt mode. CPU operation is resumed when the start switch S1 is momentarily closed to the NO contacts. This causes a positive-going clock pulse that sets the D flip-flop and returns the CONT input to the CPU high.

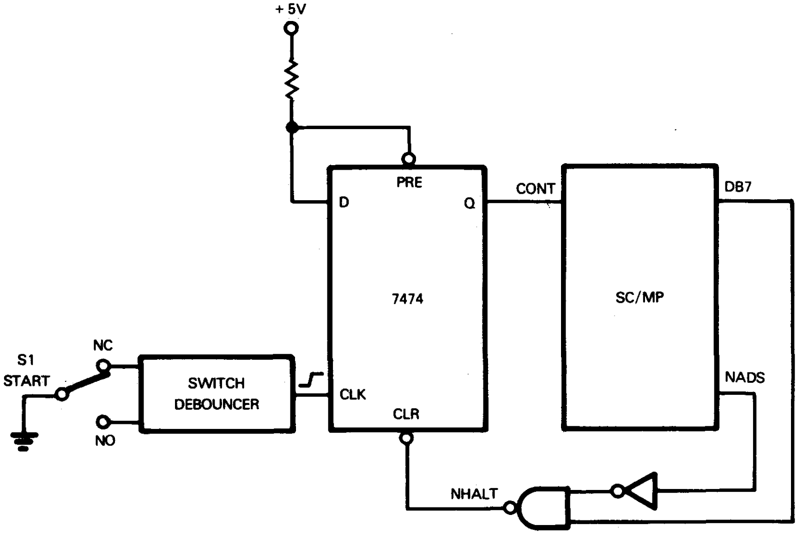


Figure 3-8. Circuit To Cause Programmed Halt For SC/MP CPU

While the SC/MP CPU is in the Halt state, the address and data lines are floated. The CPU remains in the Halt state until the CONT input is returned high. There is one exception to this rule: if an interrupt request is detected while in the Halt state, the CPU responds to the interrupt by executing a single instruction. Thus, you could use the first instruction of your interrupt service routine to reset the external CONT input signal, and thereby terminate the Halt state.

SC/MP INTERRUPT PROCESSING

The SENSEA input signal to the SC/MP CPU serves as the interrupt request line if bit 3 of the CPU's Status register is set to "1". Bit 3 of the Status register is the Interrupt Enable (IE) flag and can be set using the Interrupt Enable (IEN) instruction.

When interrupts are enabled, the SENSEA input line is tested at the beginning of every instruction fetch operation as shown in Figure 3-9. If SENSEA is high, the IE flag is reset, and the contents of the Program Counter are exchanged with the contents of Pointer Register 3. In other words, Pointer Register 3 must contain the beginning address of your interrupt service routine. The return address, that is, the address at which program execution must continue after the interrupt request has been serviced, is now held in Pointer Register 3. Thus, the return-from-interrupt sequence would be to set the IE flag high and then once again exchange the contents of the Program Counter and Pointer Register 3 to resume the main program.

Let us examine some of the special requirements and limitations of this interrupt processing sequence. First, before enabling interrupts you must load Pointer Register 3 (P3) with the beginning address of your interrupt service routine. Notice that the contents of P3 should actually be one less than the beginning address of the first instruction since the new contents of the Program Counter will be incremented prior to fetching the instruction.

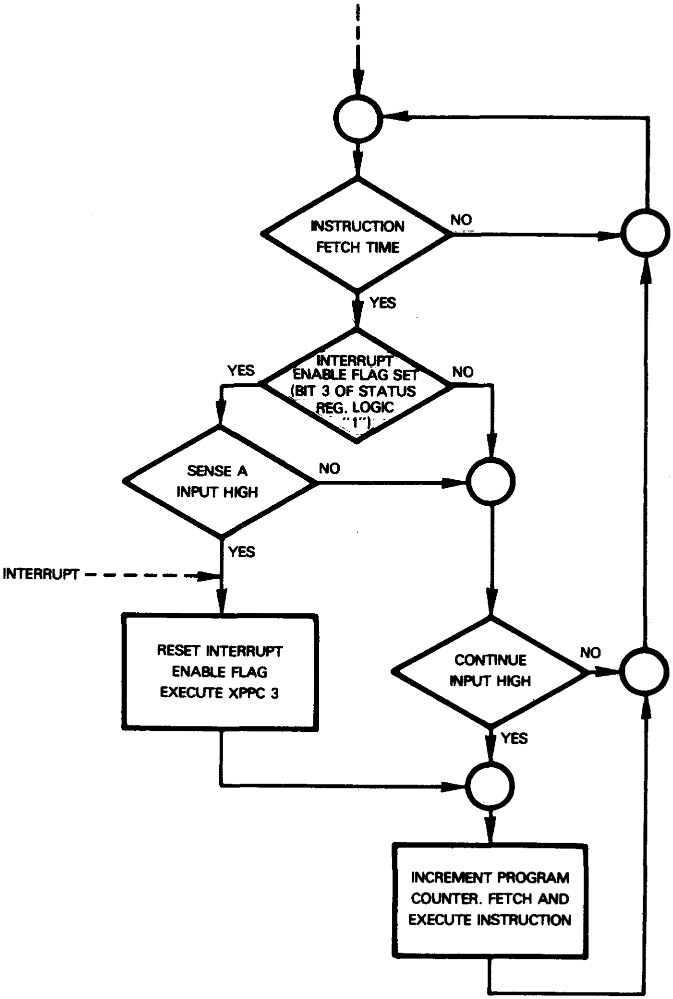

INSTRUCTION FETCH TIME
NO
YES
INTERRUPT ENABLE FLAG SET (BIT 3 OF STATUS REG. LOGIC "1")
YES
NO
SENSE A INPUT HIGH
NO
YES
INTERRUPT
RESET INTERRUPT ENABLE FLAG EXECUTE XPPC 3
CONTINUE INPUT HIGH
NO
YES
INCREMENT PROGRAM COUNTER. FETCH AND EXECUTE INSTRUCTION

Figure 3-9. SC/MP Interrupt Instruction Fetch Process

Next, if you compare the interrupt response of SC/MP to those of most other microcomputers or to our hypothetical microcomputer described in Volume I, you will notice that the following two steps are missing:

1) **There is no interrupt acknowledge signal.**
2) **None of the SC/MP register contents are saved.**

In a SC/MP system, both of these functions are left up to your interrupt service routine. For example, you might provide an interrupt acknowledge indication using one of the CPU Flag outputs or by outputting a specially defined address. If it is necessary to save the contents of the SC/MP registers, this must also be done by your program using a software stack or a predefined area of read/write memory. You must also provide the instructions necessary to restore the contents of any "saved" registers since, as we shall discuss next, the return-from-interrupt sequence used by SC/MP is also quite primitive.

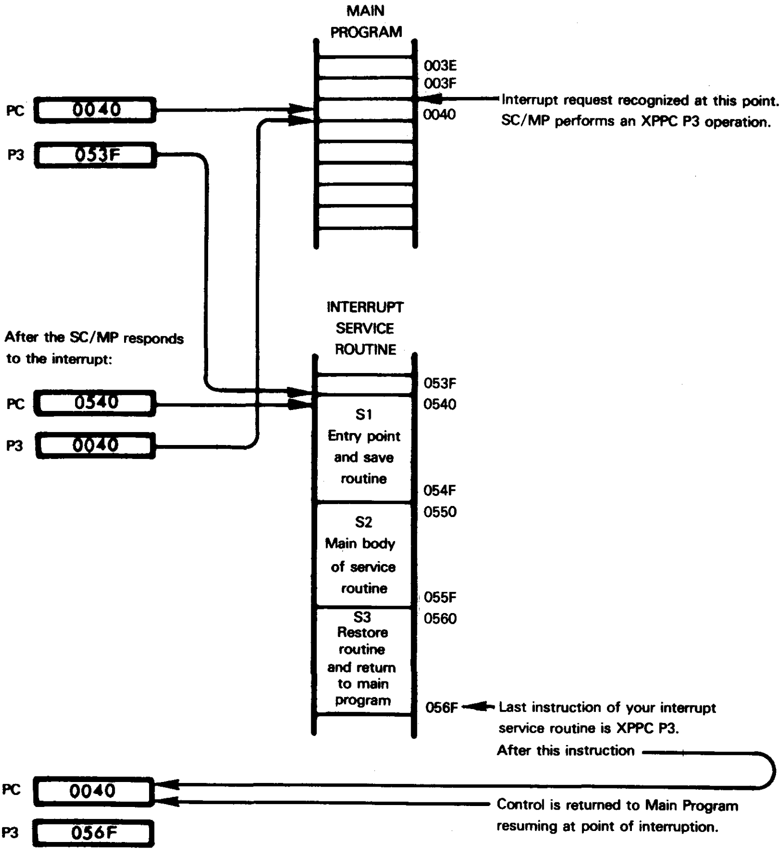
SC/MP RETURN-FROM-INTERRUPT TECHNIQUE

The final unusual aspect of the SC/MP interrupt system is that there is no Return-From-Interrupt instruction. Instead, as we mentioned earlier, the last instruction of your interrupt service routine must be an XPPC P3 instruction which restores the original contents of the Program Counter by exchanging the contents of PC and P3. This might seem quite straightforward, but it will require some special programming considerations.

The XPPC P3 instruction, which we just mentioned, restores the correct value to the Program Counter — but what about P3? Remember that P3 is always supposed to point to the beginning address (minus 1) of your interrupt service routine (if interrupts are enabled). Yet, the interrupt response sequence we just described loaded the contents of P3 into the Program Counter (PC) and then incremented the PC. And, as our interrupt service routine is executed, the contents of PC will be incremented each time an instruction is executed. Thus, when we complete the interrupt service routine and again exchange the contents of PC and P3, we will be loading P3 (our service routine pointer) with a value that has been altered. So, the problem is — how do we perform an interrupt service routine and ensure that P3 will contain the correct pointer value upon completion of the service routine.

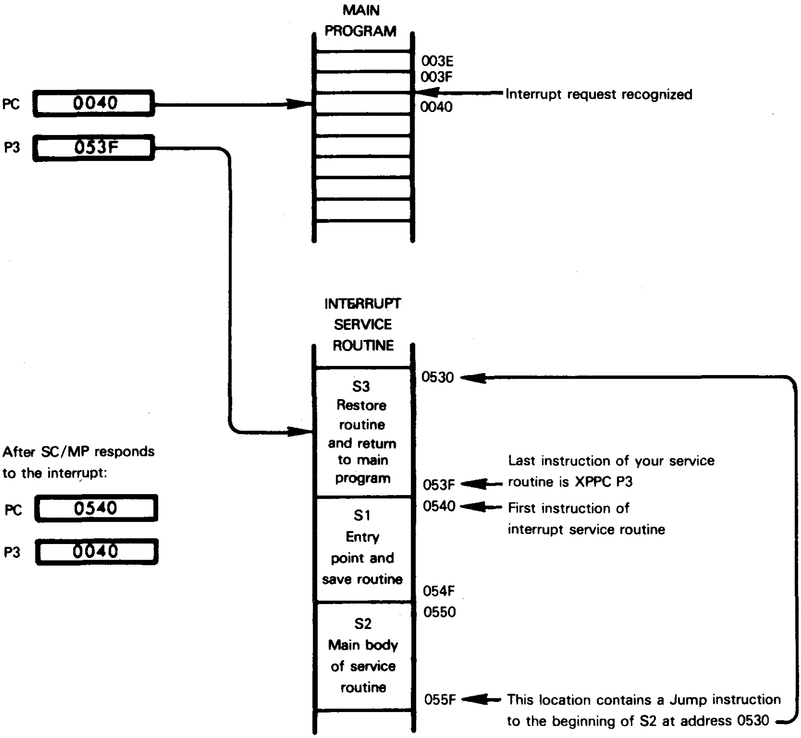
The solution to this quandary requires a closer examination of interrupt service routines. A typical interrupt service routine might consist of three primary segments. One segment would be the entry point to the routine and would include such things as register save operations: let us call this segment "S1". The second segment would be the instruction sequence which actually services the device which requested the interrupt: we will call this segment "S2". The final segment would restore registers and other system elements to their 'pre-interrupt' values, and then return control to the main (interrupted) program: we will call this segment "S3". Thus, the entire interrupt recognition/response/return sequence might be represented as follows:

(We will use arbitrary addresses to simplify our discussion.)

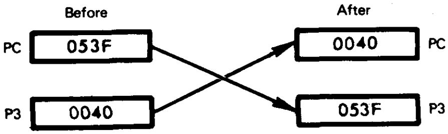


MAIN PROGRAM
003E
003F
0040
Interrupt request recognized at this point. SC/MP performs an XPPC P3 operation.
PC 0040
P3 053F
After the SC/MP responds to the interrupt:
PC 0540
P3 0040
INTERRUPT SERVICE ROUTINE
053F
0540
S1 Entry point and save routine
054F
0550
S2 Main body of service routine
055F
0560
S3 Restore routine and return to main program
056F
Last instruction of your interrupt service routine is XPPC P3.
After this instruction
Control is returned to Main Program resuming at point of interruption.
PC 0040
P3 056F

This sequence causes a proper return to the interrupted program but, as we have discussed, does not leave us with our desired pointer value (053F in this example) in P3. The solution requires us to rearrange the segments of our interrupt service routine as follows:


MAIN PROGRAM
PC 0040
P3 053F
003E
003F
Interrupt request recognized
0040
INTERRUPT SERVICE ROUTINE
0530
S3 Restore routine and return to main program
After SC/MP responds to the interrupt:
PC 0540
P3 0040
053F
Last instruction of your service routine is XPPC P3
0540
First instruction of interrupt service routine
S1 Entry point and save routine
054F
0550
S2 Main body of service routine
055F
This location contains a Jump instruction to the beginning of S2 at address 0530

Now, our entry point for the interrupt service routine is still 0540, so we load P3 with a pointer of 053F as before. However, by rearranging the segments and adding a Jump instruction at the end of the second segment (S2), we can have the last instruction of our interrupt service routine located at 053F. When this instruction (XPPC P3) is executed the following operation occurs:



We have now returned control to the main program and we have also restored the contents of P3 to the required pointer value to allow servicing of subsequent interrupts.

One final point: the CPU's interrupt processing sequence resets the Interrupt Enable (IE) flag to zero. To allow subsequent interrupts to be serviced, your service routine must set the IE flag to "1". This would typically be the next to last instruction of your interrupt service routine. So the sequence of instructions would be:

```
          -
          -
          -
          IEN                     SET IE FLAG TO 1
          XPPC      P3            RETURN TO MAIN PROGRAM
                                  FIRST INSTRUCTION OF SERVICE ROUTINE
```

SC/MP DMA AND MULTIPROCESSOR OPERATIONS

Because the SC/MP CPU is a low-cost, low-performance microprocessor, its designers anticipated that it would frequently be used in systems which include other devices of equal or greater intelligence and processing power. Accordingly, **logic is provided on the CPU which provides a simple yet effective method of operating in systems where the System Busses are shared.** The logic required to implement a shared-bus system is essentially the same regardless of whether the purpose is to allow another device (such as a high-speed peripheral) to perform a DMA operation or if it is required because there is more than one CPU operating in the system. There are a few rather subtle differences between the techniques used and we shall point these out as we proceed with our discussion.

SC/MP BUS-SHARING CONTROL SIGNALS

As we have already described, **three SC/MP signals are dedicated to bus-sharing activities; BREQ is an input/output signal which serves both as a bus-request and bus-busy signal, ENIN is effectively a bus-grant input signal, and ENOUT is an output signal that can be used to establish priorities in daisy chained configurations.** Let us begin by seeing how SC/MP might operate in a system which includes a DMA controller.

The DMA logic provided by the SC/MP CPU is nearly the inverse of that provided by other microcomputers in this book. Most CPUs assume that they always have control of the System Busses. If another system device requires access to the System Busses, it makes a request to a DMA controller which, in turn, inputs a signal to the CPU requesting that the CPU yield control of the busses. When the CPU has no need for the bus, it outputs an acknowledgement signal to the DMA controller which then sends a bus-grant signal to the requesting device. **The SC/MP CPU, however, competes for the System Busses just as any other system device:** it never assumes that it has control of the busses. Thus, there are really no special considerations that need be accounted for when designing DMA logic for systems that include the SC/MP CPU. The DMA controller can treat the CPU as simply another device (no different from a peripheral device, albeit the CPU might be assigned to a higher priority) that requires access to the System Busses. Therefore, a typical DMA application would only require the use of the SC/MP BREQ and ENIN signals as shown in Figure 3-10.

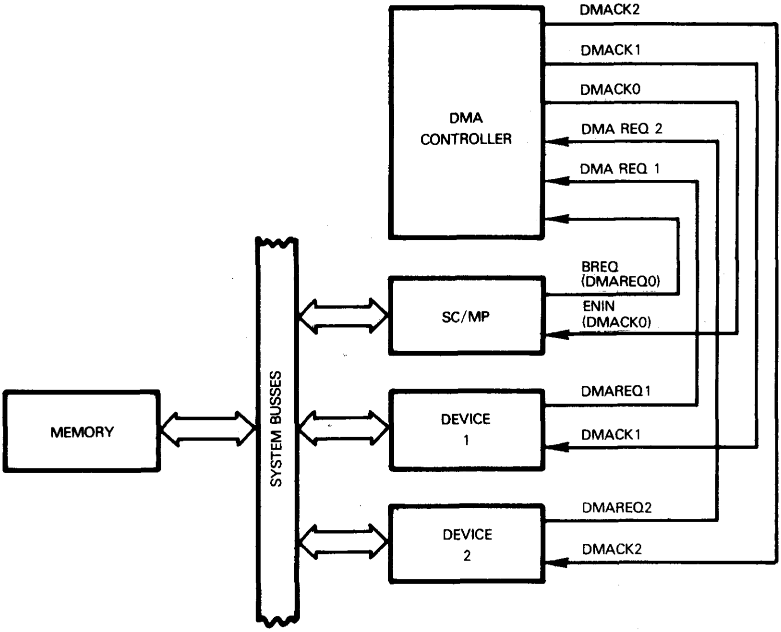


Figure 3-10. Using SC/MP In A System With Direct Memory Access

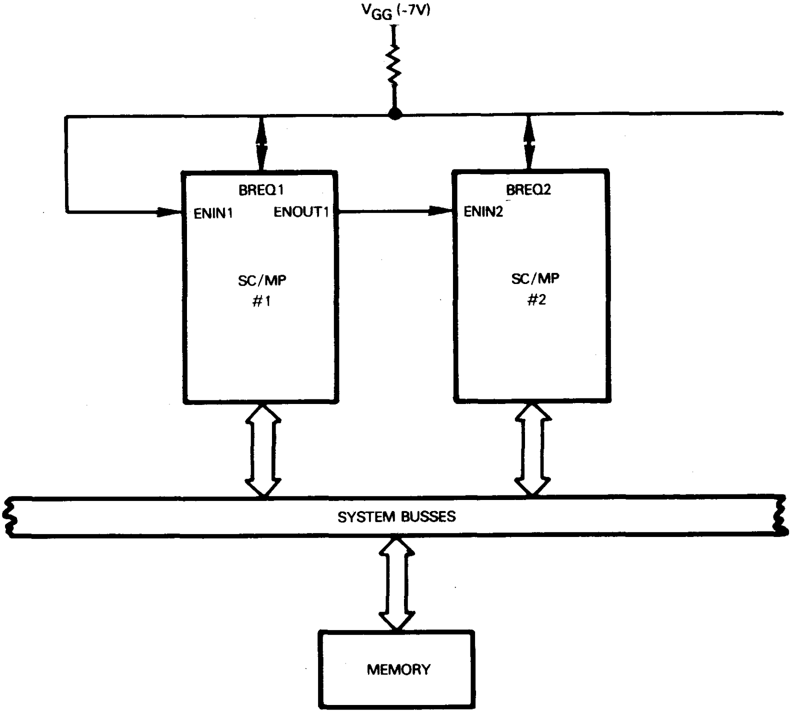
SC/MP IN MULTIPROCESSOR SYSTEMS

Now let us look at how the SC/MP bus-sharing logic might be used in a multiprocessor system. It is in such a system that the CPU's bus-sharing logic can be most appreciated. First, let us restate the rules which govern the conditions of the SC/MP ENOUT output signal.

1) ENOUT is always low while SC/MP is actually using the System Busses; that is, while the ENIN input and BREQ output are both high.

2) When SC/MP is not using the System Busses (either BREQ output or ENIN input low), ENOUT is held in the same state as the ENIN input.

The effect of these rules may not be immediately obvious. To see how they function to simplify bus-sharing, let us construct a simple multiprocessor system consisting of two SC/MP CPUs and some memory.



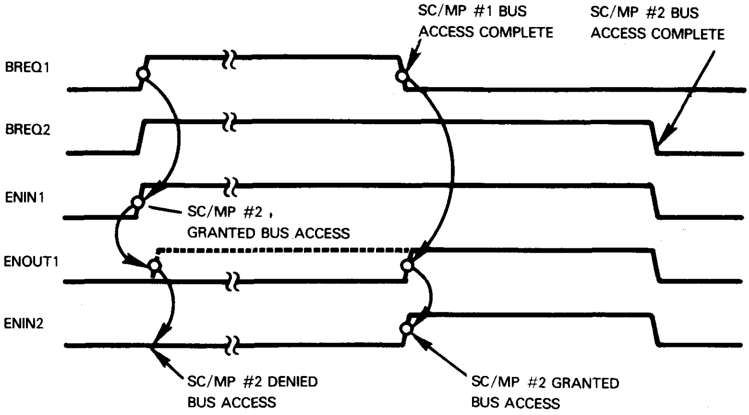
There are three possible situations that can exist with this configuration.

1) If one of the CPUs is currently using the bus, it is outputting a high on the BREQ line. This automatically prevents the other CPU from vying for the bus until the BREQ line goes low upon completion of the bus access by the first CPU.

2) If neither CPU is currently using the bus, the BREQ line is low. If one of the CPUs requires bus access, it can now output a high on the BREQ line. Once again, this will prevent the other CPU from subsequently vying for the bus.

Thus far there would seem to be no need for any control signals except the bidirectional BREQ line. However, it is when the third possible situation is encountered that the ENIN and ENOUT signals are needed.

3) If both CPUs require bus access at the same time, each will test the BREQ line and, finding it low, will output a high on BREQ. This simultaneous occurrence of requests for bus access is resolved by using the ENIN and ENOUT signals. The operation of these bus access signals to resolve this situation can be illustrated as follows:


SC/MP #1 BUS ACCESS COMPLETE
SC/MP #2 BUS ACCESS COMPLETE
BREQ1
BREQ2
ENIN1
SC/MP #2 , GRANTED BUS ACCESS
ENOUT1
ENIN2
SC/MP #2 DENIED BUS ACCESS
SC/MP #2 GRANTED BUS ACCESS

When the BREQ line goes high it applies a high input to the ENIN1 input of SC/MP #1. Since BREQ1 is also high at this time, SC/MP #1 now has access to the bus and it outputs a low on ENOUT1. This is applied to the ENIN2 input to SC/MP #2 and thus denies bus access by SC/MP #2. Notice that SC/MP #2 holds its BREQ2 output signal high even though its request has not yet been granted. When SC/MP #1 has finished its bus access, the BREQ1 output returns low. However, since the BREQ2 output is still high, ENIN1 remains high. This condition of BREQ1 low and ENIN1 high causes the ENOUT1 signal to go high thus enabling SC/MP #2.

This arrangement allows the first CPU in a daisy-chain string to have the highest priority for bus access and also automatically allows any other CPU to gain immediate access to the busses whenever they become available.

SC/MP CONTROL TECHNIQUES IN MULTIPROCESSOR APPLICATIONS

Now that we have described the way in which the bus-sharing logic of the SC/MP CPU can be used in a multiprocessor system, let us continue just a bit further and describe a few more common considerations that you must deal with if you are designing a multiprocessor system. We will limit this discussion primarily to hardware and control considerations since programming in a multiprocessor system can become quite complex and is beyond the scope of this book. However, the techniques we will describe here are the first step towards simplifying the programming for such a system.

The first operation that you must deal with in any microcomputer system is initialization of the system. This operation requires some additional thought when designing a multiprocessor system. Typically, one CPU will be the primary or controlling CPU: how do you ensure that this CPU has control of the system when power is first applied?

Figure 3-11 illustrates an easy method of establishing system control upon initialization. The system reset signal (NRST), which is generated at power-up, is applied to SC/MP #1. The Flag1 output from SC/MP #1 is then applied to the NRST input of SC/MP #2. Since the Flag 1 line is connected to a bit in the CPU's Status register which is set to zero on power-up, SC/MP #2 will be held in a reset condition until SC/MP #1 executes an instruction which sets that bit (and thus, the Flag 1 output line) high.

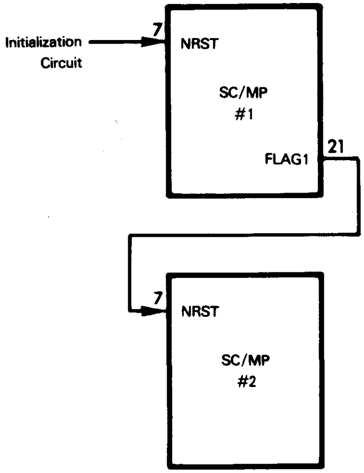


Figure 3-11. One Method Of Initializing A SC/MP Multiprocessor System

Of course, this method requires the FLAG 1 output from SC/MP #1 to be dedicated to this initialization operation. If this is a problem, you could use two separate initialization circuits with, for example, the RC time constant for the SC/MP #2 circuitry being greater than that of the circuitry for SC/MP #1. This approach, however, does not provide the positive control of the first method we described.

Once the multiprocessor system has been initialized and is running, the bus-sharing logic that we've already described will resolve contentions between the CPUs as far as access to System Busses is concerned. However, **there might be situations where we want to assure that one of the CPUs will be guaranteed immediate and extended access to the System Busses. This can also be accomplished quite easily with SC/MP as illustrated in Figure 3-12.**

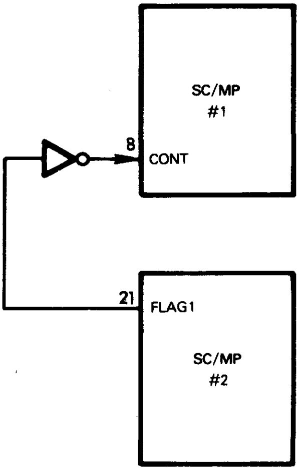


Figure 3-12. Forcing The Halt State In A SC/MP Multiprocessor System

In this illustration the FLAG 1 output of SC/MP #2 is inverted and applied to the CONT input of SC/MP #1. Now, if the F1 bit in the Status register of SC/MP #2 is set to "1", SC/MP #1 will be forced into the Halt state and is effectively removed from the system until the F1 bit is reset under program control.

THE SC/MP RESET OPERATION

A NRST low signal input to the SC/MP CPU initializes the microprocessor. While NRST is low, any in-process operations are automatically aborted and the CPU's strobes and address and data lines are floated. NRST must be held low for a minimum of two microcycles. After NRST goes high again, this is what happens:

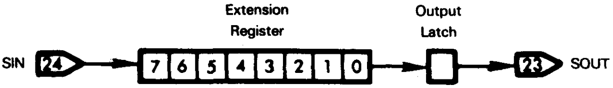
1) All of the programmable registers are cleared.
2) The first instruction is fetched from memory location 0001_{16}.
3) The Bus Request (BREQ) for this first input/output operation occurs within 6-1/2 microcycles after NRST goes high.

The NRST signal can be used at any time to reset the CPU, and must be used following power-up since SC/MP may power up in a random condition. After power has first been applied to the CPU, you should allow approximately 100 milliseconds for the oscillator and internal clocks to stabilize before applying the NRST signal.

SC/MP SERIAL INPUT/OUTPUT OPERATIONS

The SC/MP CPU not only has two of its 40 pins designated primarily for serial input/output operations, it also dedicates one instruction from its rather limited instruction set solely to serial I/O. Allocation of this amount of a CPU's resources for this purpose would seem unwarranted with most microprocessors: however, keep in mind that SC/MP is a very low cost device and intended primarily for use in slow-speed applications. It is quite likely that SC/MP will frequently be used to transfer data serially, so it is therefore not only reasonable but advantageous to provide straightforward methods of performing these operations. Let us look now at how this is done with SC/MP.

In our description of SC/MP's programmable registers, we described the Extension (E) register as an 8-bit register. **When the E register is used for serial I/O, it is actually a 9-bit register with connections to two of the device pins as shown in the figure below.**



When the SC/MP SIO (Serial Input/Output) instruction is executed, the contents of the Extension register are shifted right one bit position: the previous contents of bit 0 are loaded into the output latch and output on the SOUT pin, and the level (1 or 0) present at the SIN pin is loaded into bit 7 of the Extension register. The Extension register can be loaded from, and its contents can be transferred to the Accumulator. A typical serial output operation would thus consist of:

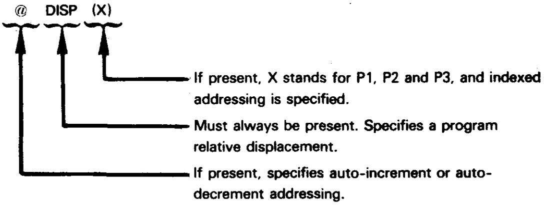
1) Loading the Accumulator with the data byte that is to be transmitted.
2) Transferring the contents of the Accumulator into the Extension register.
3) Performing eight SIO instructions to shift the contents of the Extension register into the output latch and out onto the SOUT pin.

Of course, this sequence does not cover all the programming requirements for serial data transfers. For example, your program must provide some method of timing the bit transmission. This is easily accomplished with SC/MP by using the Delay (DLY) instruction which can generate variable time delays ranging from 13 to 131,593 microcycles. For asynchronous operations, one of the SC/MP Flags which are connected to device pins can be pulsed each time a new bit is shifted out (or in) and one of the sense conditions inputs (SENSEA or SENSEB) can be tested to detect bit received/ready.

THE SC/MP INSTRUCTION SET

Table 3-4 lists the SC/MP instruction set.

Memory reference instructions are shown as having either full or limited addressing capability. Full addressing capability is identified in the operand as follows:



If present, X stands for P1, P2 and P3, and indexed addressing is specified.

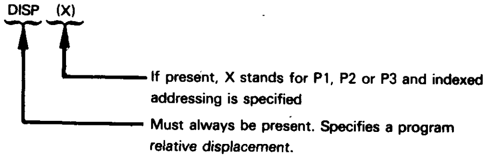
Must always be present. Specifies a program relative displacement.

If present, specifies auto-increment or auto-decrement addressing.

Thus, the real options associated with full addressing capability are:

DISP	Direct, program relative addressing
DISP(X)	Direct, indexed addressing
@DISP(X)	Auto-increment or auto-decrement addressing

Limited addressing capabilities do not include the auto-increment and auto-decrement feature. The operand field for instructions with limited addressing capability is shown as follows:



If present, X stands for P1, P2 or P3 and indexed addressing is specified

Must always be present. Specifies a program relative displacement.

The serial I/O instruction inputs serial data via the high order bit of the Extension register, and/or outputs serial data via the low order bit of the Extension register.

The serial I/O instruction works as a one-bit right shift of the Extension register contents, with bit 0 being shifted to the SOUT pin and the SIN pin being shifted into bit 7. This has been illustrated along with the logic description.

It is worth noting that SC/MP has no Jump-to-Subroutine instruction; rather the XPPC instruction is used to exchange the contents of the Program Counter with the contents of a Pointer register. In very simple applications (and those are the applications for which SC/MP is intended) this is a very effective scheme. Providing subroutines are not nested, a subroutine's beginning address may be stored in a Pointer register, then execution of XPPC moves the subroutine's starting address to the Program Counter, thereby executing the subroutine — but at the same time, the Program Counter contents are stored in the Pointer register, thus preserving the return address. At the conclusion of the subroutine, execution of another XPPC instruction is all that is needed to return from the subroutine. The only penalty paid is that one Pointer register is out of service while the subroutine is being executed. If all Pointer registers are needed by the subroutine, or if subroutines are nested, then the return address which is stored in the Pointer register must be saved in memory. In these more complicated applications, one of the Pointer registers will probably be used as a Stack Pointer, and addresses will be saved on the Stack.

This type of subroutine access, while it may appear primitive to a minicomputer programmer, is very effective in simple microcomputer applications.

The following symbols are used in Table 3-4.

AC	Accumulator
C	Carry status
DATA	An 8-bit binary data unit
DISP	An 8-bit signed binary displacement
E	The Extension register
EA	Effective address, determined by the instruction. Options are: DISP EA is [PC] + DISP DISP(X) EA is [X] + DISP @DISP(X) EA is [X] if DISP ≥ 0, EA is [X] + DISP if DISP < 0; in both cases [X]←[X] + DISP after EA is calculated.
E<i>	The ith bit of the Extension register
IE	Interrupt Enable
O	Overflow status
PC	Program Counter
X	One of the three Pointer registers
SIN	Serial Input pin
SOUT	Serial Output pin
SR	Status register
Z	Zero status
@	Auto-increment flag
X<y,z>	Bits y through z of a Pointer register. For example, X<7,0> represents the low order byte of one of the Pointer registers.
@DISP(X)	This designates the available addressing modes for the SC/MP, as described above. In all three of the addressing modes, if -128 is specified for DISP, the contents of the Extension register are used instead of DISP.
[ ]	Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.
[[ ]]	Implied memory addressing; the contents of the memory location designated by the contents of a register.
∧	Logical AND
∨	Logical OR
∀	Logical Exclusive-OR
←	Data is transferred in the direction of the arrow.
⟷	Data is exchanged between the two locations designated on either side of the arrow.

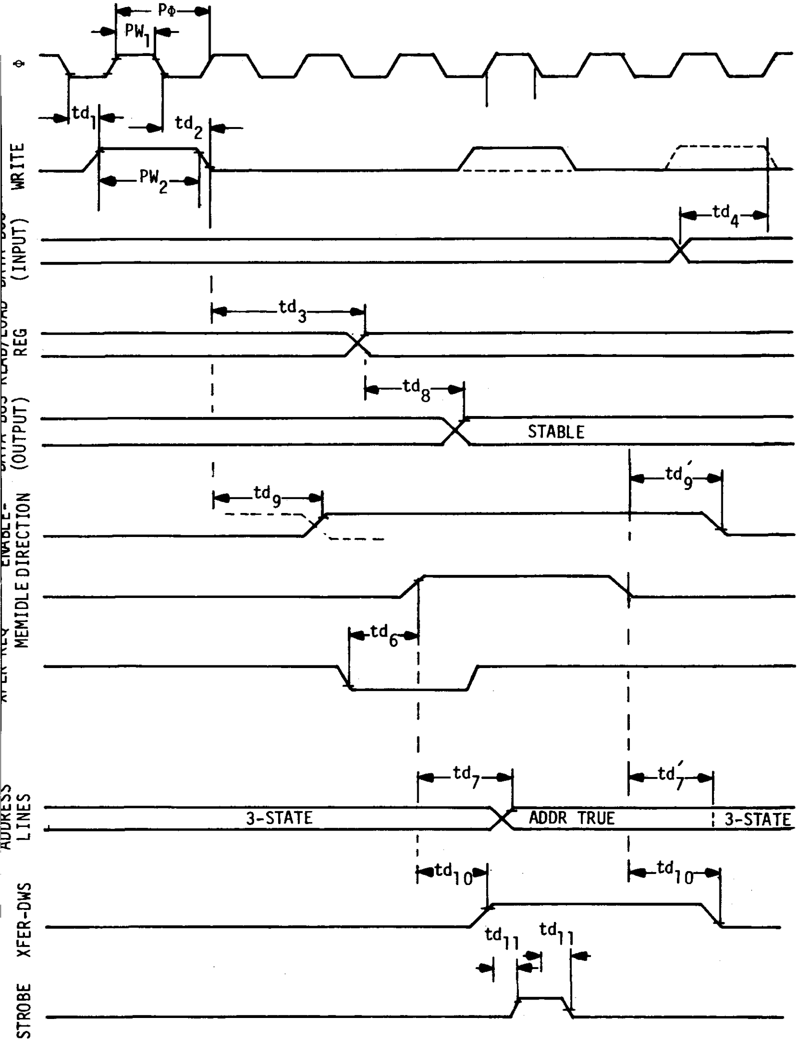
Under the heading of STATUSES in Table 3-4, an X indicates statuses which are modified in the course of the instructions execution. If there is no X, it means that the status maintains the value it had before the instruction was executed.

Table 3-4. SC/MP Instruction Set Summary

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	O	
I/O	SIO		1			[E<I-1>]←[E<I>] SOUT ← [E0] [E7] ← SIN Shift the Extension register right one bit. Shift bit 0 of the Extension register to the output pin SOUT. Shift the data at input pin SIN into bit 7 of the Extension register.
PRIMARY MEMORY REF AND I/O	LD	@ DISP(X)	2			[AC]←[EA] Load Accumulator from addressed memory location.
	ST	@ DISP(X)	2			[EA]←[AC] Store Accumulator contents in addressed memory location.
SECONDARY MEMORY REFERENCE AND MEMORY OPERATE	ADD	@ DISP(X)	2	X	X	[AC]←[AC] + [EA] + [C] Add binary to Accumulator the addressed memory location's contents with Carry.
	DAD	@ DISP(X)	2	X		[AC]←[AC] + [EA] + [C] Add decimal to Accumulator the addressed memory location's contents with Carry.
	CAD	@ DISP(X)	2	X	X	[AC]←[AC] + [EA] + [C] Add complement of addressed memory location's contents with Carry to Accumulator.
	AND	@ DISP(X)	2			[AC]←[AC] ∧ [EA] AND Accumulator with addressed memory location's contents.
	OR	@ DISP(X)	2			[AC]←[AC] V [EA] OR Accumulator with addressed memory location's contents.
	XOR	@ DISP(X)	2			[AC]←[AC] ⊻ [EA] Exclusive-OR Accumulator with addressed memory location's contents.
	ILD	@ DISP(X)	2			[EA]←[EA] + 1; [AC]←[EA] Increment addressed memory location's contents then load into Accumulator.
	DLD	@ DISP(X)	2			[EA]←[EA] - 1; [AC]←[EA] Decrement addressed memory location's contents, then load into Accumulator.

Table 3-4. SC/MP Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	O	
IMMEDIATE	LDI	DATA	2			[AC] ← DATA Load immediate into Accumulator.
IMMEDIATE OPERATE	ADI	DATA	2	X	X	[AC] ← [AC] + DATA + [C] Add binary immediate. Add Carry to result.
	DAI	DATA	2	X		[AC] ← [AC] + DATA + [C] Decimal add immediate. Add Carry to result.
	CAI	DATA	2	X	X	[AC] ← [AC] + DATA + [C]' Add the contents of the Accumulator to the complement of the immediate data value. Add Carry to result.
	ANI	DATA	2			[AC] ← [AC] ∧ DATA AND immediate.
	ORI	DATA	2			[AC] ← [AC] V DATA OR immediate.
	XRI	DATA	2			[AC] ← [AC] ⊻ DATA Exclusive-OR immediate.
JUMP	JMP	DISP(X)	2			[PC] ← EA Unconditional jump to effective address.
JUMP ON CONDITION	JP	DISP(X)	2			If [AC] ≥ 0; [PC] ← EA If the Accumulator contents are greater than 0, jump to effective address.
	JZ	DISP(X)	2			If [AC] = 0; [PC] ← EA If the Accumulator contents equal 0, jump to effective address.
	JNZ	DISP(X)	2			If [AC] = 0; [PC] ← EA If the Accumulator contents are not 0, jump to effective address.



3854 DMA Device Signals and Timing

contents of a selected register onto the DATA BUS only while READ REG is high, if there was a similar address match during the prior cycle. I/O address assignment is made using pins P1 and P2

3854 DMA Device Signals Summary

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNITS	NOTES
$P\Phi$	Φ Clock Period	.5		10	μS	Note 1
PW_1	Φ Pulse Width	180		$P\Phi$-180	nS	t_r, t_f = 50 nS typ
td_1	Φ to WRITE + Delay	60		300	nS	Note 1
td_2	Φ to WRITE - Delay	60		250	nS	Note 1
PW_2	WRITE Pulse Width	$P\Phi$-100		$P\Phi$	nS	t_r, t_f = 50 nS typ
td_3	WRITE to READ/LOAD REG Delay			600	nS	
td_4	DB Input Set-up Time			300	nS	
td_6	$\overline{\text{XFER REQ}}$ to MEMIDLE Set-up	200			nS	
td_7	MEMIDLE to ADDR True	50	200	500	nS	C_L = 500 pf
td'_7	MEMIDLE to ADDR 3-State	30		250	nS	C_L = 500 pf
td_8	READ REG to DB Output	40		300	nS	C_L = 100 pf
td_9	WRITE to ENABLE & DIRECTION + Delay			450	nS	C_L = 50 pf
td'_9	MEMIDLE to ENABLE - Delay			400	nS	C_L = 50 pf
td_{10}	MEMIDLE to XFER & DWS + Delay			300	nS	C_L = 50 pf
td_{10}	MEMIDLE to XFER & DWS - Delay			300	nS	C_L = 50 pf
td_{11}	Φ to STROBE + Delay	30		200	nS	C_L = 50 pf
td_{11}	Φ to STROBE - Delay	30		200	nS	C_L = 50 pf

Notes:

1. These specifications are those of Φ and WRITE as supplied by the 3850 CPU.
2. Input and output capacitance is 3 to 5 pf typical on all pins except V_{DD}, V_{GG}, and V_{SS}.

Chapter 3
THE NATIONAL SEMICONDUCTOR SC/MP

Describing this microprocessor immediately after the F8 is a good idea because the two products contrast well. The F8 differs more markedly from minicomputers than any other 8-bit microprocessor described in this book. SC/MP, by way of contrast, is one of the more minicomputer-like products.

When we say that SC/MP is a more minicomputer-like product what we mean is that its logic distribution and instruction set are more traditional.

The SC/MP is a single device, not a family of devices. The single device provides Arithmetic and Logic Unit, Control Unit, registers and memory addressing logic, exactly what you would expect to find in any minicomputer Central Processing Unit. There are no additional devices forming an SC/MP "family", comparable to the F8 PSU DMI or SMI. Additional logic that will support SC/MP consists of standard off-the-shelf buffers, bidirectional drivers, ROM and RAM. This approach is the same as that taken with PACE, National Semiconductor's other single-chip microprocessor which is described later in this book among the 16-bit microprocessors. Both SC/MP and PACE provide a wealth of control signals and thus limit the need for special-purpose support chips.

The only current manufacturer for SC/MP is:

NATIONAL SEMICONDUCTOR INC
2900 Semiconductor Drive
Santa Clara, CA 95050

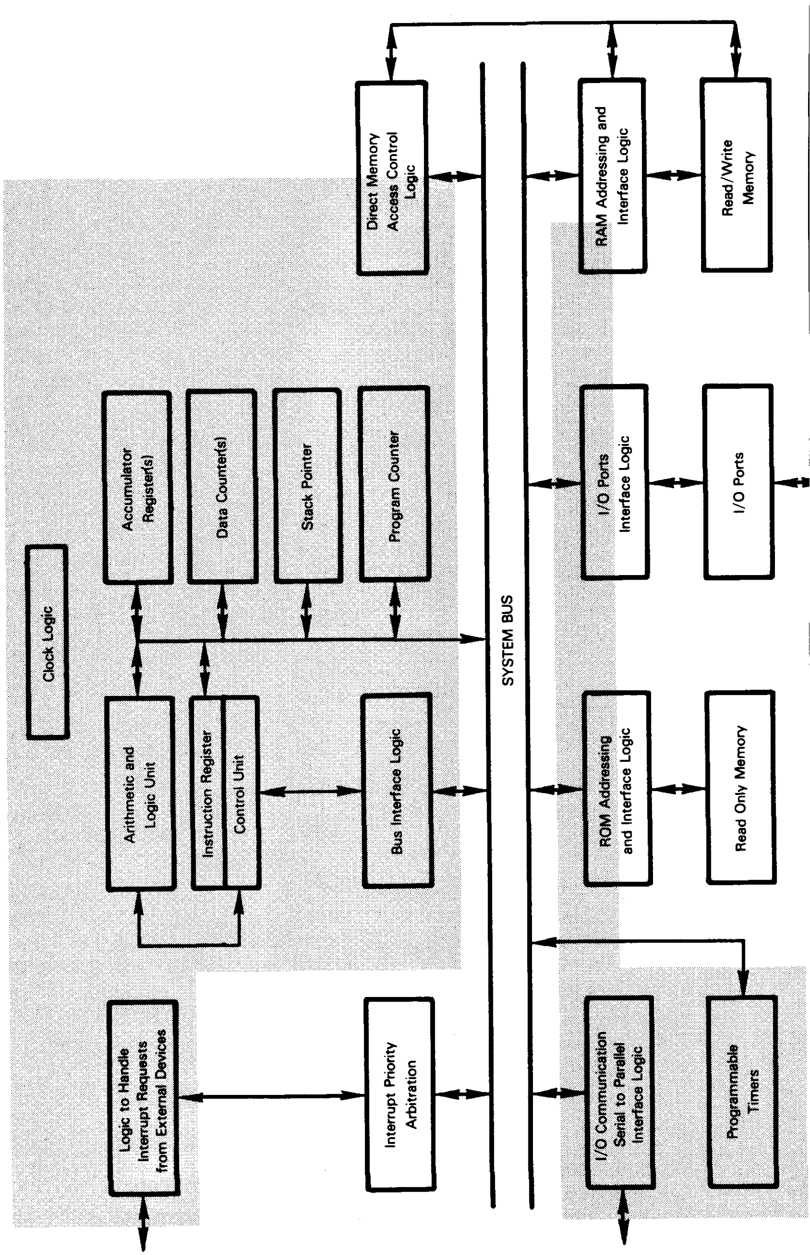
Although there is an agreement between Rockwell International and National Semiconductor to exchange microcomputer technical information and produce each other's products, at the present time Rockwell International has not elected to second source SC/MP.

Figure 3-1 conceptually illustrates the logic functions which are implemented on the SC/MP chip. One of the weaknesses of Figure 3-1, and the equivalent figures for the other microcomputers, is that the way in which logic functions are implemented cannot be identified. SC/MP, for example, implements non-CPU logic at a very elementary level, well suited for simple applications only.

SC/MP SERIAL I/O

Nonetheless, Figure 3-1 does reveal a few of the rather unusual capabilities provided by SC/MP. Notice that Serial-to-Parallel Interface Logic is shown as implemented by the SC/MP chip. SC/MP has two serial I/O device pins, one for serial binary input data, the other for serial binary output data. The assembly and disassembly of serial-to-parallel data is accomplished by one SC/MP instruction.

Figure 3-1 also shows Programmable Timer logic as being implemented by the SC/MP chip. This is barely justifiable — the SC/MP instruction set includes a Delay instruction that is used to generate timed durations ranging from 13 to 131,593 microcycles. Note, however, that during this delay interval the CPU can be performing no other actions: the CPU is, in effect, operating solely as a programmable timer. This is


Direct Memory Access Control Logic
RAM Addressing and Interface Logic
Read/Write Memory
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
I/O Ports Interface Logic
I/O Ports
Clock Logic
SYSTEM BUS
Arithmetic and Logic Unit
Instruction Register
Control Unit
Bus Interface Logic
ROM Addressing and Interface Logic
Read Only Memory
Logic to Handle Interrupt Requests from External Devices
Interrupt Priority Arbitration
I/O Communication Serial to Parallel Interface Logic
Programmable Timers

obviously quite different from having a separate logic device that performs this timer function within a system. Once again, this points out the weakness of a generalized representation such as Figure 3-1.

SC/MP DMA AND MULTIPROCESSOR LOGIC

One other area of non-CPU logic shown as being implemented by SC/MP further illustrates this point. **A portion of the Direct Memory Access (DMA) logic is provided by SC/MP using a few signals to control bus access.** A significant amount of external logic would still be required to obtain an operational DMA system. Therefore, Figure 3-1 can be misleading because it cannot indicate the way in which the CPU implements a particular function. In this particular case there is also a significant area of non-CPU logic provided by SC/MP that is nowhere indicated by Figure 3-1: **The signals that can be used for DMA are primarily intended to simplify the design of multiprocessor systems.** This is a very unusual logic function for a CPU to provide and therefore is not even suggested in Figure 3-1. But for SC/MP, the inclusion of this multiprocessor-oriented logic makes a lot of sense: its low cost and modest performance makes it a likely candidate for multiprocessor systems.

SC/MP AND SC/MP-II

There are two versions of the SC/MP CPU: the original version uses P-channel silicon-gate MOS/LSI technology and its part number is ISP-8A/500; the new version (SC/MP-II) uses N-channel technology and its part number is ISP-8A/600. The two versions are functionally equivalent and fully compatible in terms of object code and pin configuration. (A few minor signal level conversions are required for complete signal compatibility: see Figure 3-3.) **The SC/MP-II provides some significant advantages over the original version — it is twice as fast and uses only one-fourth the power of the original P-channel version. Additionally, while SC/MP requires two power sources (a +5 volt and a -7 volt supply), SC/MP-II needs only a single +5 volt supply.** Throughout this chapter, we will simply refer to the CPU as SC/MP: all the descriptions apply to both versions of the CPU unless we specifically mention SC/MP-II.

SC/MP INSTRUCTION EXECUTION SPEED

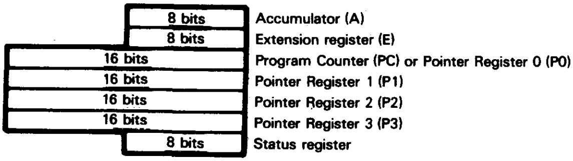
Both versions of the SC/MP CPU have an on-chip clock oscillator and can use a capacitor, crystal, or TTL clock input to drive the clock. The P-channel SC/MP can run at a maximum frequency of 1 megahertz which results in instruction execution times in the range of 10 to 50 microseconds. SC/MP-II can operate at frequencies up to 4 megahertz with resulting instruction execution times in the range of 5 to 25 microseconds. Notice, that although the input frequency for SC/MP-II can be four times that of SC/MP, the instruction execution time for SC/MP-II is twice as fast (not four times as fast): This is because of internal differences in the way the on-chip clock oscillator uses the timing inputs.

SC/MP LOGIC LEVEL

Both versions of SC/MP provide TTL-compatible input and output signals.

SC/MP PROGRAMMABLE REGISTERS

SC/MP has an 8-bit Accumulator, an 8-bit Extension register, a 16-bit Program Counter, three 16-bit Pointer registers, and an 8-bit Status register. These programmable registers are illustrated as follows:



The Accumulator is a single, primary Accumulator, as described for our hypothetical microcomputer.

The Extension register is used to assemble or disassemble serial-to-parallel dat for serial data input and output. This register is also used as a buffer for the Accumulator.

SC/MP MEMORY PAGES

The Program Counter is 16 bits wide; therefore up to 65,536 bytes of memory may be addressed in the normal course of events. The four high-order bits of the Program Counter represent page select bits; therefore the memory of a SC/MP system is divided into 16 pages of 4096 words each.

Notice that the Program Counter is shown as Pointer Register 0; this is done becaus some instructions move data between Pointer registers including the Program Counter There is one other unusual fact about the SC/MP Program Counter: **the four most significant bits (the page select bits) of the Program Counter are never incremente during the instruction fetch sequence. Instead, when the last address of a page i reached, the Program Counter "wraps-around" to the first address of the curren page.** For example, if the Program Counter contains $2FFF_{16}$, when it is incremente the new contents of the Program Counter will be 2000_{16} instead of 3000_{16}. The pag select bits of the Program Counter can only be changed by executing an instructio that loads a new value into the most significant bits of the Program Counter.

Note that the four high-order address bits are not output on separate address pins; instead they are output on the data lines at the beginning of an input/output cycle an must be demultiplexed by external logic in order to generate page select signals.

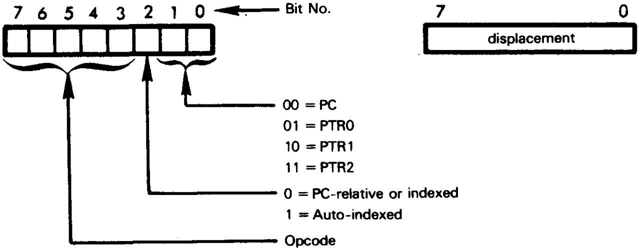
The three Pointer registers are Data Counters that can also be used as Inde registers, Page Pointers, or Stack Pointers. Typically, you would assign a specifi function to each register. For example, the following assignments might be used:

P1	-	ROM Pointer
P2	-	Stack Pointer
P3	-	Subroutine Pointer

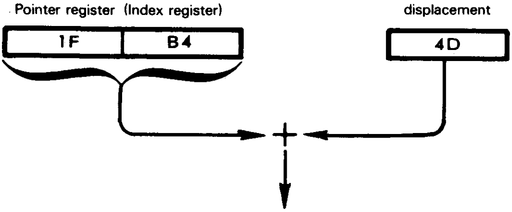
These arbitrary assignments also reveal several interesting facts about the architectur of SC/MP. First, the SC/MP CPU does not provide an on-chip stack; instead, a stack ca be maintained in memory using one of the Pointer registers as a Stack Pointer. Secondly, the SC/MP instruction set does not include a Jump-to-Subroutine instruction: one of the Pointer registers must be used to hold subroutine addresses which ca then be swapped with the Program Counter. We will discuss this in detail when we describe the SC/MP instruction set.

ADDRESSING MODES

The SC/MP memory reference instructions use program-relative direct addressing, indexed addressing, and auto-indexed addressing. All memory reference instructions are two-byte instructions and have the following object code format



rogram relative and indexed addressing are as described in Volume I, Chapter 6. Ve will just re-emphasize here that all addressing in SC/MP is paged and uses the vrap-around technique — that is, there is no carry from the low order 12 bits of an adress into the most significant 4 bits of an address. We mentioned this earlier when we iscussed the Program Counter and it also applies to indexed addressing. Thus, if the um of the Index register (that is, one of the Pointer registers) and the second object ode byte contents (displacement) is more than FFF_{16}, the Carry bit will be discarded. his may be illustrated as follows:



Effective Address = 1FB4 + 4D

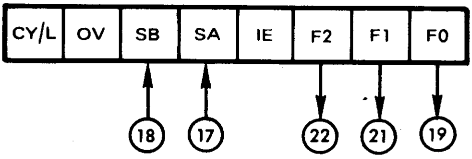
```
                    1 F B 4
                  +   4 D
Expected result = 2 1 0 0 1
  Discard Carry            Actual Result is 1001₁₆
```

emember, all arithmetic operations during address formation, regardless of the ddressing mode, obey this wrap-around technique: there is never a carry from it 11 into bit 12.

he auto-indexing mode of addressing provided by SC/MP instructions is actually n auto-increment/auto-decrement operation. When auto-indexing is specified, the isplacement, as a signed binary number, is added to the contents of a Pointer register ı order to compute an effective address. If the displacement is less than zero, the ointer register is decremented by the displacement before the memory access. If the isplacement is equal to or greater than zero, then the contents of the Pointer register is ıe effective address and the Pointer register contents are incremented by the displaceıent after the memory access. **This method of auto-increment and auto-decrement ddressing is the same as that described in Volume I with one significant ifference: SC/MP allows an address to be incremented or decremented by any alue in the range 0 - 127 instead of just by a value of one.**

C/MP STATUS REGISTER

C/MP has a programmable 8-bit Status register which may be illustrated as ollows:



rcled numbers represent device pin numbers to which bits of the Status register are ›nnected.

The Carry (CY), Link (L) and Overflow (OV) status bits are typical microcompute status bits as were described in Volume I, Chapter 7.

The two sense bits, SB and SA, are tied to SC/MP device pins. These two bit: directly reflect the state of the logic signals applied to the device pins and thu: can be used to detect external events. Although there are no SC/MP instruction: that allow you to directly jump or branch on the condition of one of these bits, a se quence of masking and testing instructions can be used to accomplish the same effect albeit somewhat slower. **The SA and SB bits are read-only bits. Instructions ma read the status of these two bits, but only incoming signals may change their con dition.** For example, an instruction that moves the contents of the Accumulator to th Status register may modify any of the other status bits, but bits 4 and 5 will not change **The SA bit serves a dual function. If the Interrupt Enable (IE) bit is set to one, th SA input serves as the interrupt input.** We will discuss interrupt processing later i this chapter.

F0, F1 and F2 are control flags that are tied to SC/MP device pins. The state o these three flags may be changed under program control and may be used to con trol external devices. When the state of any of these flags is changed, it is im mediately reflected by a change in the signal level at the associated device pin.

SC/MP CPU SIGNALS AND PIN ASSIGNMENTS

Figure 3-2 illustrates the SC/MP pins and signals. A description of these signals i useful as a guide to the way in which an SC/MP microcomputer system works.

The 12 address lines AD00 - AD11 output memory and I/O device addresse: These are tristate lines, and may be floated, giving external logic control of th Address Bus. The four most significant address bits (AD12 - AD15) are tim multiplexed on the data lines.

The eight Data Bus lines DB0 - DB7 are multiplexed, bidirectional data lines vi which 8-bit data units are input and output, and on which statuses and addres bits are output at the beginning of any input/output cycle. Statuses on the Dat Bus identify the type or purpose of the input/output cycle. The address bits on th Data Bus are the four most significant address bits (AD12 - AD15) which can b used to generate page select signals for memory or peripheral devices. Table 3- describes the status and address information that is output on the Data Bus. Lik the address lines, the data lines are tristate.

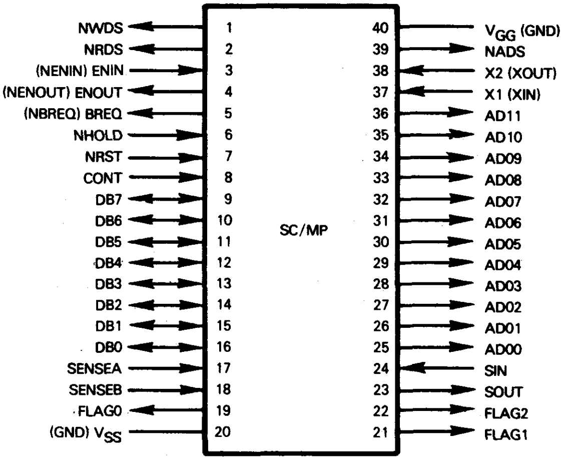
SENSEA SENSEB FLAG0, 1, and 2 are pin connections for the similarly name Status register bits described earlier.

SIN and SOUT are used in combination with the SIO instruction for serial input c data to the Extension register and serial output of data from the Extensio register.

The remaining signals (excluding clock, power and ground) may be divided into bus ac cess, Data Bus definition, and timing control signals.

SIGNAL DIFFERENCES BETWEEN SC/MP (P-CHANNEL) AND SC/MP-II (N-CHANNEL)

You will notice that some of the SC/MP pins in Figure 3-2 have two sets of signal names: the names enclosed in parentheses reflect the nomenclature used with SC/MP-II. Aside from the clock and power signals which we shall discuss separately, the only difference between SC/MP and SC/MP-II is in the polarity of bus access signals: Bus Request (BREQ/NBREQ), Enable In (ENIN/NENIN), and Enable Out (ENOUT/NENOUT). The "N" prefix to each of the SC/MP-II signals indicates that thes signals are negative-true — as opposed to the positive- (or logic "1") true signals for tl P-channel SC/MP. **In the descriptions that follow, we will use P-channel SC/M nomenclature.** If you are using the N-channel SC/MP-II version, you must simply inve these signals.



PIN NAME†	DESCRIPTION	TYPE
X1,X2	Crystal/Capacitor Connections	Input
*DB0 - DB7	Data Bus	Bidirectional, Tristate
*AD00 - AD11	Address Lines	Output, Tristate
*SENSEA,SENSEB	External Status Input	Input
*FLAG0,1,2	Flags	Output
*NRST	Reset	Input
*CONT	Halt/Continue	Input
*BREQ (NBREQ)	Bus Request/Busy	Bidirectional
*ENIN (NENIN)	Data Bus Enable	Input
*ENOUT (NENOUT)	CPU Bus Access Status	Output
*NADS	Address on Data Bus	Output
*NRDS	Data Input Strobe	Output, Tristate
*NWDS	Data Output Strobe	Output, Tristate
*NHOLD	Clock Delay	Input
SIN	Serial Data In	Input
SOUT	Serial Data Out	Output
V_{GG},V_{SS}(V_{CC},GND)	Power and Ground	

*These signals connect to the System Bus.

† Signals in parenthesis are SC/MP-II signal names.

Figure 3-2. SC/MP CPU Signals And Pin Assignments

SC/MP BUS ACCESS CONTROL SIGNALS

Before the SC/MP CPU can begin any input/output operation, it must gain access to the System Busses. This approach reflects the design philosophy behind SC/MP. It is a relatively low-cost, low-performance CPU and the designers anticipated that it would frequently be used in multiprocessor systems or in systems utilizing Direct Memory Access. Accordingly, three signals are provided to control access to the System Busses.

BREQ is used as a bus busy input indicating that some other device is using the System Busses, and as a bus request output when the System Busses are free and SC/MP requires access to the busses.

ENIN is a control signal which is input to the CPU by external logic. When ENIN is low, the CPU is denied access to the System Busses and the SC/MP address and data lines are held in tristate mode.

ENOUT is the CPU's output response to ENIN. When output high, ENOUT indicates that ENIN is high; therefore, the CPU can gain access to the System Busses, but it has not done so. If ENOUT is low, it indicates either that ENIN is low, therefore the CPU is being denied access to the System Busses or, if ENIN is high, then it indicates that the CPU is using the System Busses.

When the CPU has gained access to the System Busses, three *signals* identify the way *in* which the CPU is using the Data Bus.

SC/MP DATA BUS DEFINITION SIGNALS

NADS is output to indicate that a valid address has been output on the address lines and that the low order four bits of the Data Bus contain the high order four bits of a 16-bit address. NADS also indicates that status information is being output on the high order four bits of the Data Bus.

NRDS, when output by the CPU, indicates that the CPU wishes to receive data on the Data Bus.

NWDS when output by the CPU, indicates that data is being output by the CPU on the Data Bus. NWDS may be used by external logic as a write strobe.

SC/MP TIMING CONTROL SIGNALS

There are three *signals* which *control* CPU timing.

NRST is a system reset signal. When input low, it aborts any in-process operations. When returned high, all programmable registers are cleared, and program execution begins with the instruction fetched from memory location 0001_{16}.

CONT *may be input to stop* the CPU between instructions. When CONT is input low, all CPU operations are halted after the current instruction execution has been completed. The CPU remains halted until CONT goes high.

NHOLD is an input signal used during input/output operations to lengthen the allowed time interval for devices to respond to CPU access requests.

SC/MP TIMING AND INSTRUCTION EXECUTION

The SC/MP timing for instruction execution is very simple. Instruction execution times are expressed in terms of microcycles. A typical instruction is executed in 10 microcycles and one or more of these microcycles is an input/output cycle. The length of a microcycle depends on the frequency of the clock inputs to the CPU: with the P-channel SC/MP, the minimum microcycle length is 2 microseconds; for SC/MP-II, the N-channel version, minimum microcycle length is 1 microsecond. Thus, typical instruction execution time is 20 microseconds for the P-channel SC/MP, and 10 microseconds for SC/MP-II. **All microcycles, whether internal machine cycles or input/output cycles, are of the same length: the only *variance* occurs when the NHOLD signal is used to stretch an input or output cycle.**

There are basically only three types of SC/MP machine (or micro) cycles: data input (read) cycles, data output (write) cycles, and internal microcycles. The execution of each instruction is merely a concatenation of these three types of microcycles.

SC/MP I/O CYCLE STATUS INFORMATION

SC/MP does, however, output some status information at the beginning of every input or output cycle which provides a more precise definition of the purpose of that microcycle. Table 3-1 lists the information which may be output on the data line at the beginning of an I/O cycle (when NADS is low). Table 3-2 defines the way in which the status information may be interpreted to identify the various possible types of microcycles.

Table 3-1. Status And Address Output Via The Data Lines During The Beginning Of An I/O Cycle

SYMBOLS	DATA BUS BIT	DEFINITION
H-Flag	7	Indicates that a Halt instruction has been executed.
D-Flag	6	Indicates that a Delay instruction has been executed and that a delay cycle is starting.
I-Flag	5	Indicates that the CPU is in the fetch cycle for the first byte of an instruction.
R-Flag	4	When high, indicates that the I/O cycle is a read cycle and that input data should be placed on the Data Bus when NRDS is active. When low, indicates that the I/O cycle is a write cycle and that the Data Bus will contain output data when NWDS is active.
AD15	3	The four most significant bits of a 16-bit address. Can be used as page select signals.
AD14	2	
AD13	1	
AD12	0	

Table 3-2. Statuses Output On The Data Bus For Various Types Of Machine Cycles

Status Information	Data Bus Bit	TYPE OF MACHINE CYCLE				
		Instruction Fetch	Halt Instruction	Delay Instruction	Data Input (Read)	Data Output (Write)
H-Flag	7	0	1	0	0	0
D-Flag	6	0	0	1	0	0
I-Flag	5	1	1	0	0	0
R-Flag	4	1	1	1	1	0

SC/MP BUS ACCESS LOGIC

Since the SC/MP CPU must gain access to the System Busses before it can perform an input or output cycle, we will describe the bus access logic before discussing input/output cycles.

Figure 3-3 illustrates the bus access logic processing sequence that occurs whenever the SC/MP CPU is going to perform an input/output cycle.

First, the bidirectional BREQ line is tested. If the BREQ input is high, it indicates that the System Bus is currently in use: the CPU holds the outputs of the address and data lines, and the NRDS and NWDS signals in the high-impedance (tristate) mode.

When the BREQ input signal is low (or goes low) it indicates that the System Bus is free, and the CPU then outputs a logic "1" on the BREQ line. This informs external devices (for example, other SC/MP CPUs or a DMA controller) that a request for bus access has been initiated.

The CPU next tests the state of the ENIN input line. ENIN is essentially the "bus grant" signal: if it is low, it indicates the Bus Request (BREQ) is denied and the CPU remains in an idle state with its output held in the high impedance mode. When the ENIN input is high (or goes high) it indicates that the CPU's bus request has been granted and the I/O cycle can now be initiated.

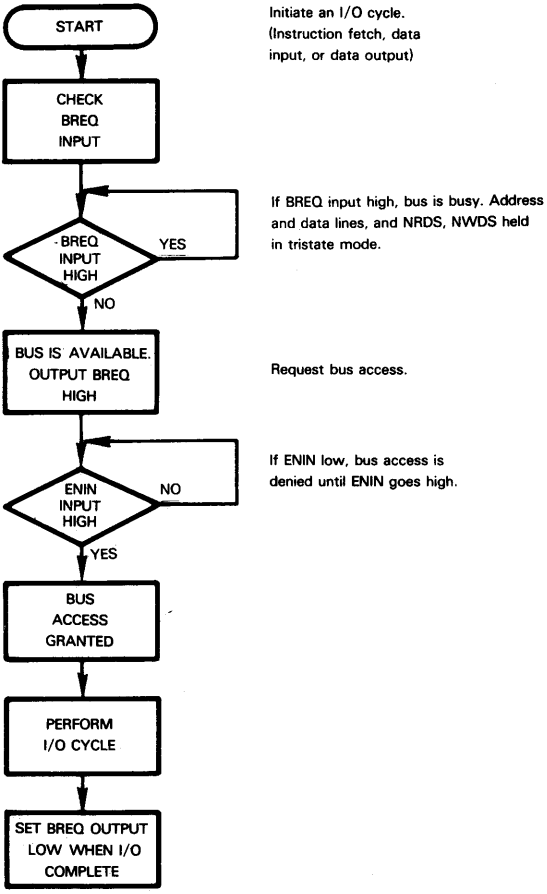

START
CHECK BREQ INPUT
BREQ INPUT HIGH
YES
NO
BUS IS AVAILABLE. OUTPUT BREQ HIGH
ENIN INPUT HIGH
NO
YES
BUS ACCESS GRANTED
PERFORM I/O CYCLE
SET BREQ OUTPUT LOW WHEN I/O COMPLETE
Initiate an I/O cycle. (Instruction fetch, data input, or data output)
If BREQ input high, bus is busy. Address and data lines, and NRDS, NWDS held in tristate mode.
Request bus access.
If ENIN low, bus access is denied until ENIN goes high.

Figure 3-3. SC/MP Bus Access Logic Processing Sequence

When the I/O cycle has been completed, the CPU sets the BREQ output low to indicate that it has finished using the System Bus and that its outputs are once again in the high impedance mode.

There are a couple of aspects of the bus access sequence which are not revealed by Figure 3-3.

SUSPENSION OF A SC/MP I/O CYCLE

First, the SC/MP CPU has the rather unusual capability of suspending an I/O operation after it has already begun. If the ENIN input line goes low while the CPU has access to the bus, the SC/MP address and data lines will go to the high impedance state, thus relinquishing access to the System Busses. The BREQ output signal will remain high and, when the ENIN input line subsequently goes high once more, the input/output cycle which had been suspended will begin again.

This ability to suspend an I/O cycle might be quite useful in a system where bus access is granted on a priority basis. In such a system, it is conceivable that one or more of the system devices (another CPU, for example) might have overriding priorities and require immediate access to the System Busses. The SC/MP bus access logic we've just described allows this to be accomplished with no difficulty whatsoever. **There is, however, one gray area in this I/O-suspend function.** If a SC/MP I/O cycle is nearly complete, it would seem to be more efficient to go ahead and complete the cycle rather than suspending it and then restarting the entire cycle later. This is precisely what SC/MP does. Unfortunately, the SC/MP literature does not tell us where this "point-of-no-return" lies within an I/O cycle. One would assume, or at least hope that this point is prior to the time when NRDS or NWDS is sent out. These signals are the read and write strobe signals; if they were repeated when an I/O cycle was restarted, the same data might be read or written twice — a potentially vexing situation. However, you are at least assured that if ENIN goes low while SC/MP is performing an I/O cycle, the cycle will be performed — either by continuing to completion or by being restarted when the System Busses are again available.

SC/MP ENOUT SIGNAL USED TO ESTABLISH ACCESS PRIORITIES

If you refer back to Figure 3-3 once again, you will notice that there is no mention of the third SC/MP bus access control signal — ENOUT. This is not an oversight — it is simply due to the fact that the ENOUT signal performs a rather specialized function which is not necessary to an understanding of the SC/MP bus access logic. **The primary function of the ENOUT output signal is as an enabling signal in systems where a "daisy chain" technique is used to establish priorities for bus access.** We will defer a discussion of this use of ENOUT until later in this chapter when we discuss the use of SC/MP in multiprocessor and DMA systems.

SC/MP I/O WITH BUS ACCESS LOGIC CONTINUOUSLY ENABLED

If the SC/MP CPU is used in a single-processor, non-DMA system then there is no need for the built-in bus access logic. In these cases, which may in fact be in the majority, the bus access signals should be connected so that the SC/MP CPU is always guaranteed immediate access to the System Busses. This is easily accomplished by making the following connections:

	SIGNAL	CONNECT TO
SC/MP	BREQ ENIN ENOUT	VGG through a pull-down resistor. VSS Leave unterminated
SC/MP-II	NBREQ NENIN NENOUT	VCC via external resistor Ground Leave unterminated

In the descriptions of SC/MP input/output operations that follow, we will always assume that the SC/MP CPU has already been granted access to the System Busses, and that this access is not interrupted (or suspended).

SC/MP INPUT/OUTPUT OPERATIONS

Once the SC/MP CPU has control of the System Busses, an actual input or output cycle can begin. As we mentioned earlier in this chapter, the execution of any SC/MP instruction includes some combinations of input/output cycles and internal machine cycles. **Figure 3-4 illustrates the bus utilization required for each of the SC/MP instructions, and also reveals an interesting, non-obvious fact about SC/MP input/output operations.** Observe that each bus utilization interval is shown as being two microcycles in duration. This is true because **each input/output operation effectively requires two microcycles.** The CPU spends a portion of the first microcycle gaining access to the System Bus and placing address and status information on the address and data lines. The actual data transfer (read or write) occurs during the second microcycle. This can be confusing if you are designing a DMA or multiprocessor system: the actual time that the bus is available is a great deal less than you would expect if you based your computations solely on the number of read and write cycles required for each instruction. To make this more clear, refer to Table 3-3 which lists the read cycles, write cycles, and total microcycles required for execution of each SC/MP instruction. If you total up each of the columns from this table, you come up with the following figures:

Total Read Cycles	= 79
Total Write Cycles	= 3
Total Input/Output Cycles	= 82
Total Microcycles	= 466

Based on these figures, it would appear that bus utilization is less than 20% (82/466). However, since the CPU maintains control of the bus for approximately two microcycles each time a read or write cycle is performed, the actual bus utilization is quite a bit greater than you would have expected. For precise timing parameters refer to the data sheets at the end of this chapter. Keep in mind that bus utilization computations should be based not only on these data sheets, but also on the actual program being used since bus utilization is directly related to the composition of instructions which comprise your program — these calculations can differ significantly from any theoretical calculations based solely on a CPU's complete instruction set.

Now, having discussed those areas of SC/MP bus access and utilization which might be confusing, let us proceed to examine the actual data input/output operations — we will find that these SC/MP operations are quite straightforward.

SC/MP DATA INPUT CYCLE

Figure 3-5 illustrates the timing for a standard SC/MP data input cycle. This timing applies regardless of whether the input cycle is to access data from memory or peripheral devices and also applies to instruction fetch operations.

Once the CPU has gained access to the System Busses, **the input cycle begins by presenting address and statuses on the address and data lines.** When the NADS signal is sent out, the least significant 12 bits of address data are valid on the SC/MP address lines, and the SC/MP data lines are outputting status information and the most significant 4 bits of address information. Table 3-1 defines the information that is output on the data lines while NADS is true. When these address bits and/or status bits need to be latched, either the leading or trailing edge of NADS can be used as a clock signal.

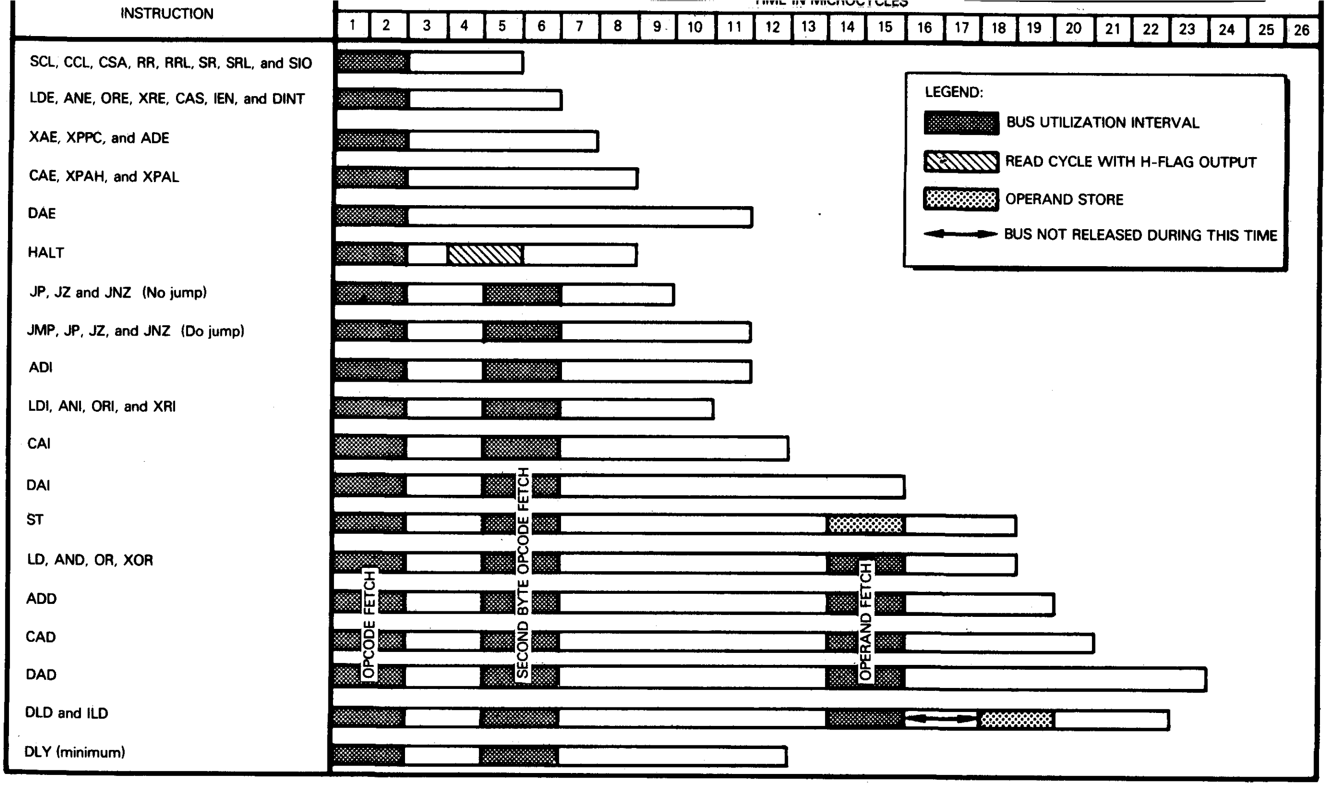


Figure 3-4. Bus Utilization Of Each SC/MP Instruction

Table 3-3. SC/MP Instruction Execution Times

INSTRUCTION	READ CYCLES	WRITE CYCLES	TOTAL MICROCYCLES
ADD	3	0	19
ADE	1	0	7
ADI	2	0	11
AND	3	0	18
ANE	1	0	6
ANI	2	0	10
CAD	3	0	20
CAE	1	0	8
CAI	2	0	12
CAS	1	0	6
CCL	1	0	5
CSA	1	0	5
DAD	3	0	23
DAE	1	0	11
DAI	2	0	15
DINT	1	0	6
DLD	3	1	22
DLY	2	0	13 - 131593
HALT	2	0	8
IEN	1	0	6
ILD	3	1	22
JMP	2	0	11
JNZ	2	0	9, 11 for Jump
JP	2	0	9, 11 for Jump
JZ	2	0	9, 11 for Jump
LD	3	0	18
LDE	1	0	6
LDI	2	0	10
NOP	1	0	5
OR	3	0	18
ORE	1	0	6
ORI	2	0	10
RR	1	0	5
RRL	1	0	5
SCL	1	0	5
SIO	1	0	5
SR	1	0	5
SRL	1	0	5
ST	2	1	18
XAE	1	0	7
XOR	3	0	18
XPAH	1	0	8
XPAL	1	0	8
XPPC	1	0	7
XRE	1	0	6
XRI	2	0	10

Note: If slow memory is being used, the appropriate delay should be added for each read or write cycle.

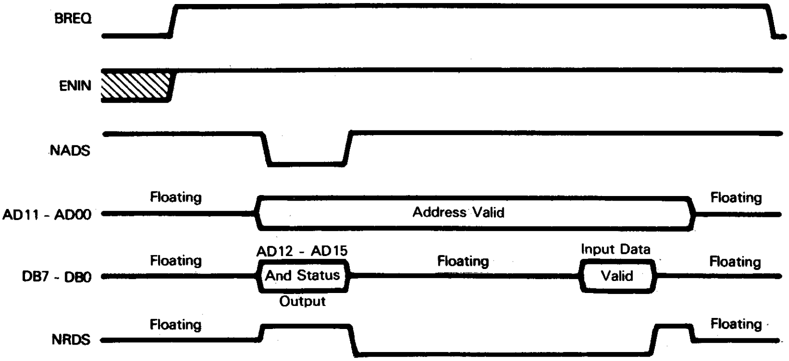


Figure 3-5. SC/MP Data Input Cycle

Shortly after the trailing edge of NADS, the Data Bus is floated and the Read Data Strobe (NRDS) signal is output. Valid input data is expected prior to the trailing edge of NRDS

SC/MP DATA OUTPUT CYCLE

The SC/MP data output cycle begins in the same way as the data input cycle. The only difference is that immediately after the status/address information is output on the data lines, the write or output data is placed on the data lines. As shown in Figure 3-6, the NWDS signal is sent out to indicate when valid output data is preser Either the leading or trailing edge of NWDS could be used to latch the output data in external data latches.

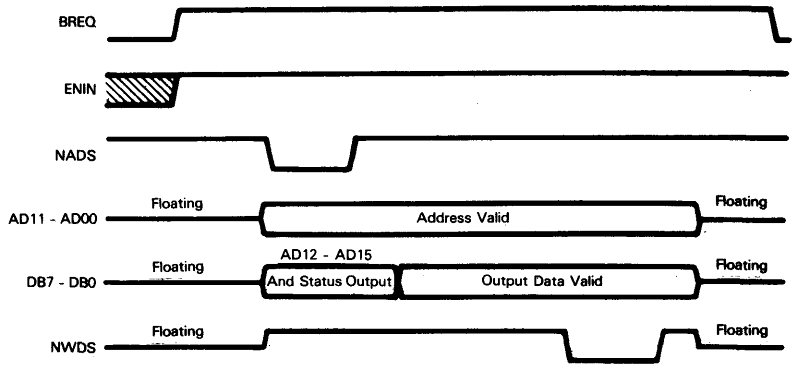


Figure 3-6. SC/MP Data Output Cycle

SC/MP NHOLD SIGNAL FOR SLOW I/O OPERATIONS

The data input/output cycles just described allow approximately one microcycle for external logic to respond. If additional access time is required, the NHOLD input signal to the CPU can be used to lengthen an input/output cycle. The NHOLD signal can be set low any time prior to the trailing edge of NRDS or NWDS as shown in Figure 3-7; this causes the trailing edge of NRDS or NWDS to be delayed until after NHOLD has been returned high. On data input cycles, the time until valid input data must be presented is simply delayed. On data output cycles, the valid output data is maintained on the data lines by the CPU until the delayed trailing edge of NWDS

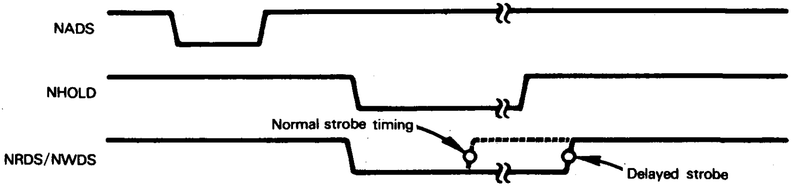


Figure 3-7. NHOLD Signal Used To Lengthen SC/MP I/O Operation

The NHOLD signal causes the I/O cycle to be lengthened in increments of 1/2 microcycle. There is no limit on the duration of the NHOLD signal.

THE SC/MP HALT STATE

The SC/MP Halt state differs from those described for other microprocessors in this book in one significant and unusual way — execution of the SC/MP Halt instruction does not cause the CPU to enter the Halt state. Instead, when SC/MP executes a Halt instruction, it simply outputs the H-Flag status on data line 7 (DB7) when NADS is true.

In order to actually place the CPU in the Halt state the CONT input signal to the CPU must be forced low.

You can use external logic to force CONTIN low either in response to the H-Flag or completely asynchronously: whenever a low is applied to the CONTIN input, the CPU enters the Halt state upon completion of the current instruction. Figure 3-8 shows a circuit that can be used to force the CPU into the Halt state when a Halt instruction is executed. When DB7 is output high while NADS is true, it indicates the Halt instruction has been executed: this combination of events is used to generate a low-going pulse (NHALT) which is applied to the clear (CLR) input of a D flip-flop. The Q output of the flip-flop is applied to the CONT input signal to the CPU. Thus, whenever a Halt instruction is executed, the CPU will be forced into the Halt mode. CPU operation is resumed when the start switch S1 is momentarily closed to the NO contacts. This causes a positive-going clock pulse that sets the D flip-flop and returns the CONT input to the CPU high.

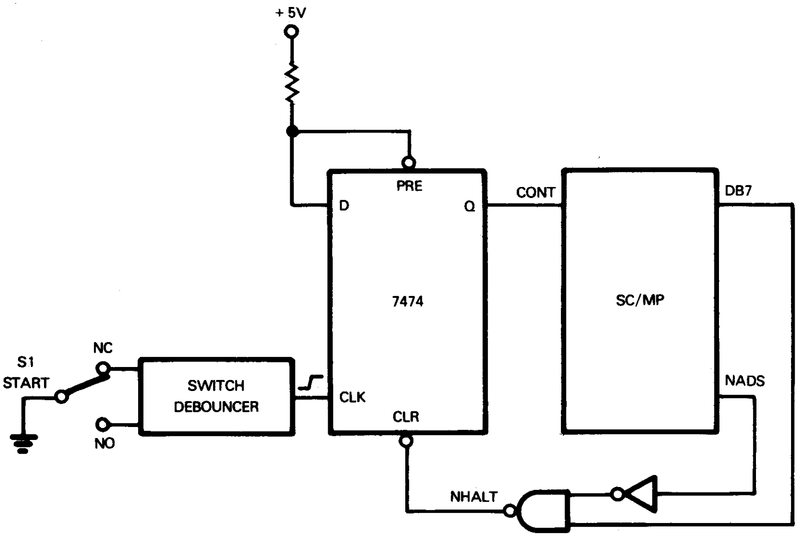


Figure 3-8. Circuit To Cause Programmed Halt For SC/MP CPU

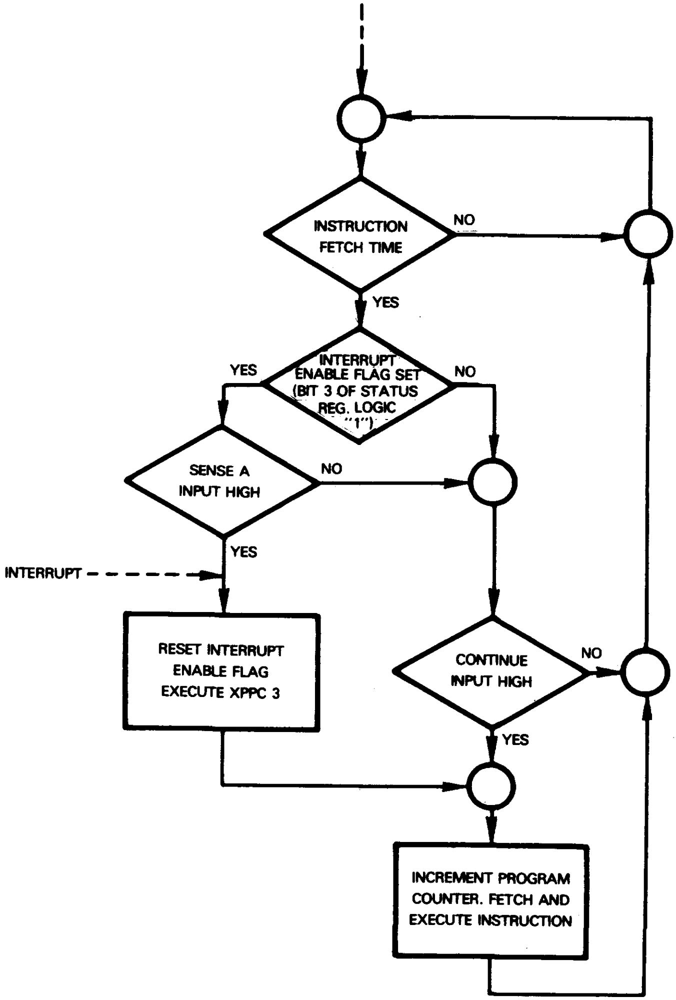
While the SC/MP CPU is in the Halt state, the address and data lines are floated. The CPU remains in the Halt state until the CONT input is returned high. There is one exception to this rule: if an interrupt request is detected while in the Halt state, the CPU responds to the interrupt by executing a single instruction. Thus, you could use the first instruction of your interrupt service routine to reset the external CONT input signal, and thereby terminate the Halt state.

SC/MP INTERRUPT PROCESSING

The SENSEA input signal to the SC/MP CPU serves as the interrupt request line if bit 3 of the CPU's Status register is set to "1". Bit 3 of the Status register is the Interrupt Enable (IE) flag and can be set using the Interrupt Enable (IEN) instruction.

When interrupts are enabled, the SENSEA input line is tested at the beginning of every instruction fetch operation as shown in Figure 3-9. If SENSEA is high, the IE flag is reset, and the contents of the Program Counter are exchanged with the contents of Pointer Register 3. In other words, Pointer Register 3 must contain the beginning address of your interrupt service routine. The return address, that is, the address at which program execution must continue after the interrupt request has been serviced, is now held in Pointer Register 3. Thus, the return-from-interrupt sequence would be to set the IE flag high and then once again exchange the contents of the Program Counter and Pointer Register 3 to resume the main program.

Let us examine some of the special requirements and limitations of this interrupt processing sequence. First, before enabling interrupts you must load Pointer Register 3 (P3) with the beginning address of your interrupt service routine. Notice that the contents of P3 should actually be one less than the beginning address of the first instruction since the new contents of the Program Counter will be incremented prior to fetching the instruction.



INSTRUCTION FETCH TIME
NO
YES
INTERRUPT ENABLE FLAG SET (BIT 3 OF STATUS REG. LOGIC "1")
YES
NO
SENSE A INPUT HIGH
NO
YES
INTERRUPT
RESET INTERRUPT ENABLE FLAG EXECUTE XPPC 3
CONTINUE INPUT HIGH
NO
YES
INCREMENT PROGRAM COUNTER. FETCH AND EXECUTE INSTRUCTION

Figure 3-9. SC/MP Interrupt Instruction Fetch Process

Next, if you compare the interrupt response of SC/MP to those of most other microcomputers or to our hypothetical microcomputer described in Volume I, you will notice that the following two steps are missing:

1) **There is no interrupt acknowledge signal.**
2) **None of the SC/MP register contents are saved.**

In a SC/MP system, both of these functions are left up to your interrupt service routine. For example, you might provide an interrupt acknowledge indication using one of the CPU Flag outputs or by outputting a specially defined address. If it is necessary to save the contents of the SC/MP registers, this must also be done by your program using a software stack or a predefined area of read/write memory. You must also provide the instructions necessary to restore the contents of any "saved" registers since, as we shall discuss next, the return-from-interrupt sequence used by SC/MP is also quite primitive.

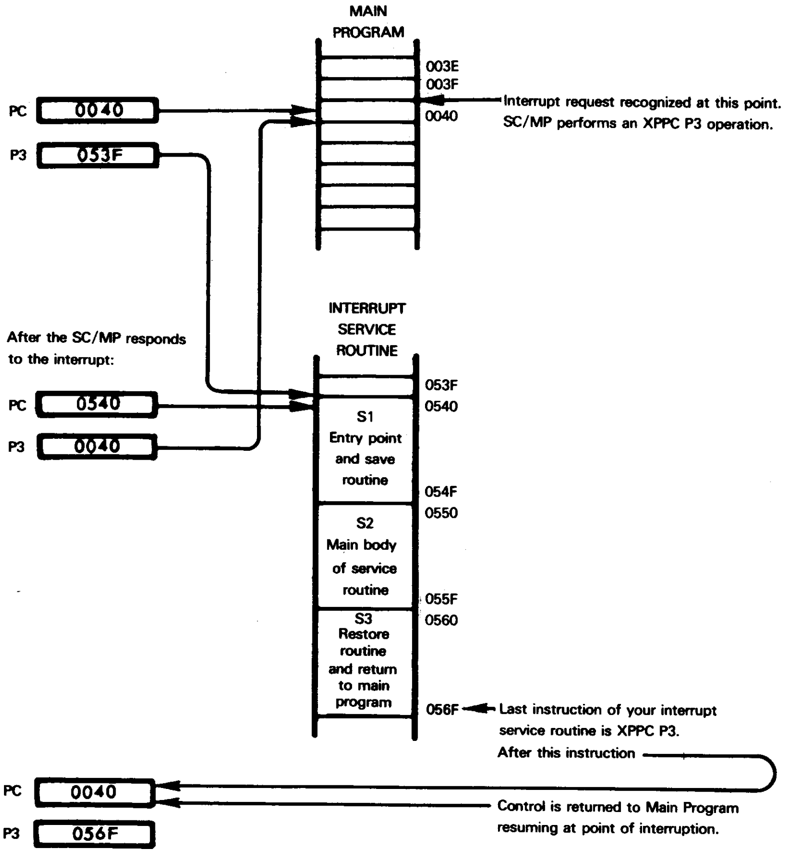
SC/MP RETURN-FROM-INTERRUPT TECHNIQUE

The final unusual aspect of the SC/MP interrupt system is that there is no Return-From-Interrupt instruction. Instead, as we mentioned earlier, the last instruction of your interrupt service routine must be an XPPC P3 instruction which restores the original contents of the Program Counter by exchanging the contents of PC and P3. This might seem quite straightforward, but it will require some special programming considerations.

The XPPC P3 instruction, which we just mentioned, restores the correct value to the Program Counter — but what about P3? Remember that P3 is always supposed to point to the beginning address (minus 1) of your interrupt service routine (if interrupts are enabled). Yet, the interrupt response sequence we just described loaded the contents of P3 into the Program Counter (PC) and then incremented the PC. And, as our interrupt service routine is executed, the contents of PC will be incremented each time an instruction is executed. Thus, when we complete the interrupt service routine and again exchange the contents of PC and P3, we will be loading P3 (our service routine pointer) with a value that has been altered. So, the problem is — how do we perform an interrupt service routine and ensure that P3 will contain the correct pointer value upon completion of the service routine.

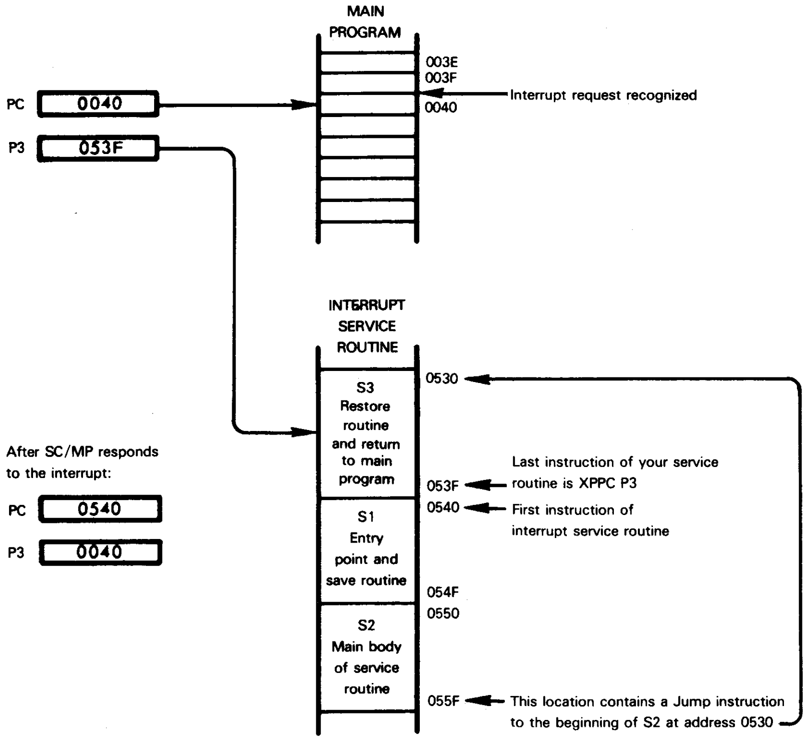
The solution to this quandary requires a closer examination of interrupt service routines. A typical interrupt service routine might consist of three primary segments. One segment would be the entry point to the routine and would include such things as register save operations: let us call this segment "S1". The second segment would be the instruction sequence which actually services the device which requested the interrupt: we will call this segment "S2". The final segment would restore registers and other system elements to their 'pre-interrupt' values, and then return control to the main (interrupted) program: we will call this segment "S3". Thus, the entire interrupt recognition/response/return sequence might be represented as follows:

(We will use arbitrary addresses to simplify our discussion.)

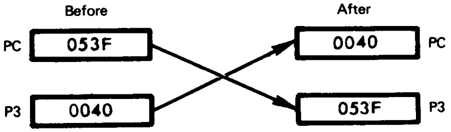


MAIN PROGRAM
003E
003F
0040
Interrupt request recognized at this point. SC/MP performs an XPPC P3 operation.
PC 0040
P3 053F
After the SC/MP responds to the interrupt:
PC 0540
P3 0040
INTERRUPT SERVICE ROUTINE
053F
0540
S1 Entry point and save routine
054F
0550
S2 Main body of service routine
055F
0560
S3 Restore routine and return to main program
056F
Last instruction of your interrupt service routine is XPPC P3.
After this instruction
Control is returned to Main Program resuming at point of interruption.
PC 0040
P3 056F

This sequence causes a proper return to the interrupted program but, as we have discussed, does not leave us with our desired pointer value (053F in this example) in P3. The solution requires us to rearrange the segments of our interrupt service routine as follows:



Now, our entry point for the interrupt service routine is still 0540, so we load P3 with a pointer of 053F as before. However, by rearranging the segments and adding a Jump instruction at the end of the second segment (S2), we can have the last instruction of our interrupt service routine located at 053F. When this instruction (XPPC P3) is executed the following operation occurs:



We have now returned control to the main program and we have also restored the contents of P3 to the required pointer value to allow servicing of subsequent interrupts.

One final point: the CPU's interrupt processing sequence resets the Interrupt Enable (IE) flag to zero. To allow subsequent interrupts to be serviced, your service routine must set the IE flag to "1". This would typically be the next to last instruction of your interrupt service routine. So the sequence of instructions would be:

```
        -
        -
        -
        IEN                  SET IE FLAG TO 1
        XPPC     P3          RETURN TO MAIN PROGRAM
                             FIRST INSTRUCTION OF SERVICE ROUTINE
```

SC/MP DMA AND MULTIPROCESSOR OPERATIONS

Because the SC/MP CPU is a low-cost, low-performance microprocessor, its designers anticipated that it would frequently be used in systems which include other devices of equal or greater intelligence and processing power. Accordingly, **logic is provided on the CPU which provides a simple yet effective method of operating in systems where the System Busses are shared.** The logic required to implement a shared-bus system is essentially the same regardless of whether the purpose is to allow another device (such as a high-speed peripheral) to perform a DMA operation or if it is required because there is more than one CPU operating in the system. There are a few rather subtle differences between the techniques used and we shall point these out as we proceed with our discussion.

SC/MP BUS-SHARING CONTROL SIGNALS

As we have already described, **three SC/MP signals are dedicated to bus-sharing activities; BREQ is an input/output signal which serves both as a bus-request and bus-busy signal, ENIN is effectively a bus-grant input signal, and ENOUT is an output signal that can be used to establish priorities in daisy chained configurations.** Let us begin by seeing how SC/MP might operate in a system which includes a DMA controller.

The DMA logic provided by the SC/MP CPU is nearly the inverse of that provided by other microcomputers in this book. Most CPUs assume that they always have control of the System Busses. If another system device requires access to the System Busses, it makes a request to a DMA controller which, in turn, inputs a signal to the CPU requesting that the CPU yield control of the busses. When the CPU has no need for the bus, it outputs an acknowledgement signal to the DMA controller which then sends a bus-grant signal to the requesting device. **The SC/MP CPU, however, competes for the System Busses just as any other system device:** it never assumes that it has control of the busses. Thus, there are really no special considerations that need be accounted for when designing DMA logic for systems that include the SC/MP CPU. The DMA controller can treat the CPU as simply another device (no different from a peripheral device, albeit the CPU might be assigned to a higher priority) that requires access to the System Busses. Therefore, a typical DMA application would only require the use of the SC/MP BREQ and ENIN signals as shown in Figure 3-10.

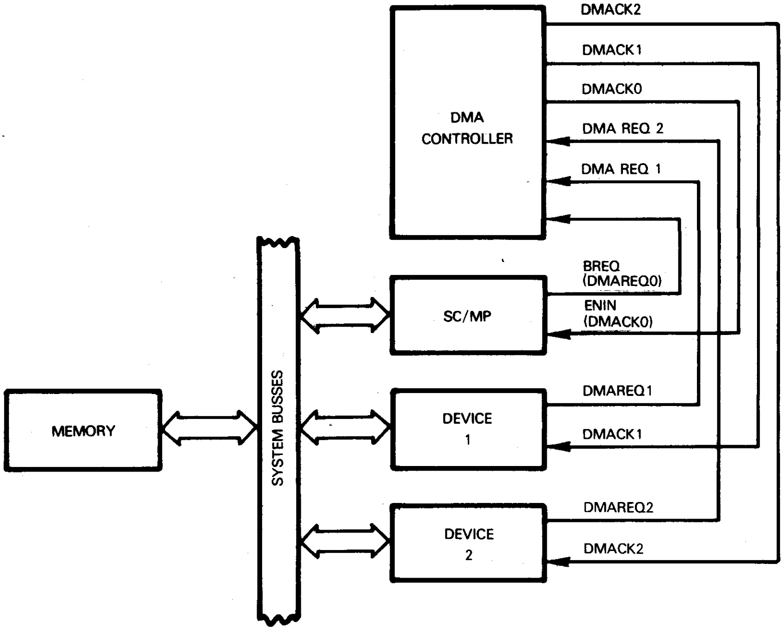


Figure 3-10. Using SC/MP In A System With Direct Memory Access

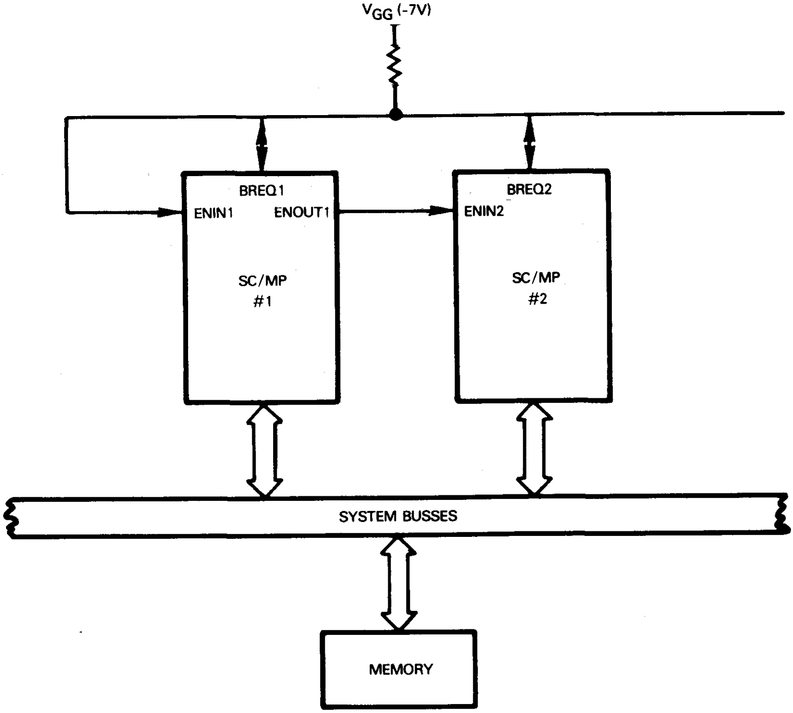
SC/MP IN MULTIPROCESSOR SYSTEMS

Now let us look at how the SC/MP bus-sharing logic might be used in a multiprocessor system. It is in such a system that the CPU's bus-sharing logic can be most appreciated. First, let us restate the rules which govern the conditions of the SC/MP ENOUT output signal.

1) ENOUT is always low while SC/MP is actually using the System Busses; that is, while the ENIN input and BREQ output are both high.

2) When SC/MP is not using the System Busses (either BREQ output or ENIN input low), ENOUT is held in the same state as the ENIN input.

The effect of these rules may not be immediately obvious. To see how they function to simplify bus-sharing, let us construct a simple multiprocessor system consisting of two SC/MP CPUs and some memory.

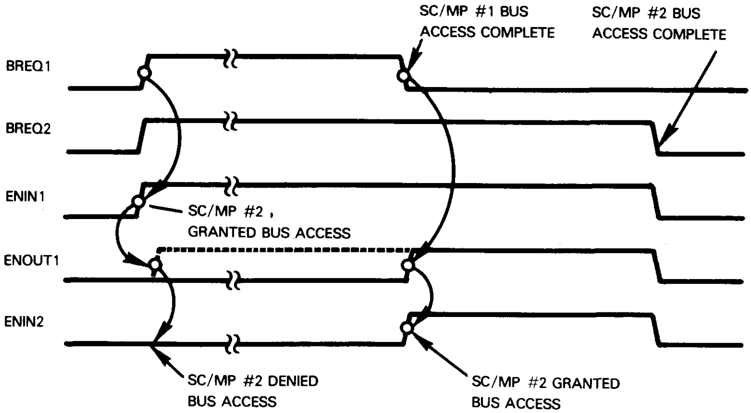


There are three possible situations that can exist with this configuration.

1) If one of the CPUs is currently using the bus, it is outputting a high on the BREQ line. This automatically prevents the other CPU from vying for the bus until the BREQ line goes low upon completion of the bus access by the first CPU.
2) If neither CPU is currently using the bus, the BREQ line is low. If one of the CPUs requires bus access, it can now output a high on the BREQ line. Once again, this will prevent the other CPU from subsequently vying for the bus.

Thus far there would seem to be no need for any control signals except the bidirectional BREQ line. However, it is when the third possible situation is encountered that the ENIN and ENOUT signals are needed.

3) If both CPUs require bus access at the same time, each will test the BREQ line and, finding it low, will output a high on BREQ. This simultaneous occurrence of requests for bus access is resolved by using the ENIN and ENOUT signals. The operation of these bus access signals to resolve this situation can be illustrated as follows:


SC/MP #1 BUS ACCESS COMPLETE
SC/MP #2 BUS ACCESS COMPLETE
BREQ1
BREQ2
ENIN1
SC/MP #2 , GRANTED BUS ACCESS
ENOUT1
ENIN2
SC/MP #2 DENIED BUS ACCESS
SC/MP #2 GRANTED BUS ACCESS

When the BREQ line goes high it applies a high input to the ENIN1 input of SC/MP #1. Since BREQ1 is also high at this time, SC/MP #1 now has access to the bus and it outputs a low on ENOUT1. This is applied to the ENIN2 input to SC/MP #2 and thus denies bus access by SC/MP #2. Notice that SC/MP #2 holds its BREQ2 output signal high even though its request has not yet been granted. When SC/MP #1 has finished its bus access, the BREQ1 output returns low. However, since the BREQ2 output is still high, ENIN1 remains high. This condition of BREQ1 low and ENIN1 high causes the ENOUT1 signal to go high thus enabling SC/MP #2.

This arrangement allows the first CPU in a daisy-chain string to have the highest priority for bus access and also automatically allows any other CPU to gain immediate access to the busses whenever they become available.

SC/MP CONTROL TECHNIQUES IN MULTIPROCESSOR APPLICATIONS

Now that we have described the way in which the bus-sharing logic of the SC/MP CPU can be used in a multiprocessor system, let us continue just a bit further and describe a few more common considerations that you must deal with if you are designing a multiprocessor system. We will limit this discussion primarily to hardware and control considerations since programming in a multiprocessor system can become quite complex and is beyond the scope of this book. However, the techniques we will describe here are the first step towards simplifying the programming for such a system.

The first operation that you must deal with in any microcomputer system is initialization of the system. This operation requires some additional thought when designing a multiprocessor system. Typically, one CPU will be the primary or controlling CPU: how do you ensure that this CPU has control of the system when power is first applied?

Figure 3-11 illustrates an easy method of establishing system control upon initialization. The system reset signal (NRST), which is generated at power-up, is applied to SC/MP #1. The Flag1 output from SC/MP #1 is then applied to the NRST input of SC/MP #2. Since the Flag 1 line is connected to a bit in the CPU's Status register which is set to zero on power-up, SC/MP #2 will be held in a reset condition until SC/MP #1 executes an instruction which sets that bit (and thus, the Flag 1 output line) high.

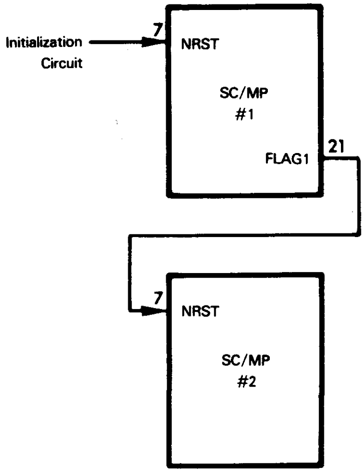


Figure 3-11. One Method Of Initializing A SC/MP Multiprocessor System

Of course, this method requires the FLAG 1 output from SC/MP #1 to be dedicated to this initialization operation. If this is a problem, you could use two separate initialization circuits with, for example, the RC time constant for the SC/MP #2 circuitry being greater than that of the circuitry for SC/MP #1. This approach, however, does not provide the positive control of the first method we described.

Once the multiprocessor system has been initialized and is running, the bus-sharing logic that we've already described will resolve contentions between the CPUs as far as access to System Busses is concerned. However, **there might be situations where we want to assure that one of the CPUs will be guaranteed immediate and extended access to the System Busses. This can also be accomplished quite easily with SC/MP as illustrated in Figure 3-12.**

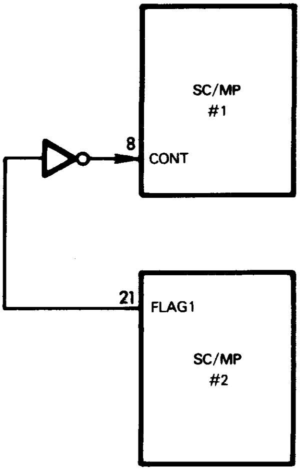


Figure 3-12. Forcing The Halt State In A SC/MP Multiprocessor System

In this illustration the FLAG 1 output of SC/MP #2 is inverted and applied to the CONT input of SC/MP #1. Now, if the F1 bit in the Status register of SC/MP #2 is set to "1", SC/MP #1 will be forced into the Halt state and is effectively removed from the system until the F1 bit is reset under program control.

THE SC/MP RESET OPERATION

A NRST low signal input to the SC/MP CPU initializes the microprocessor. While NRST is low, any in-process operations are automatically aborted and the CPU's strobes and address and data lines are floated. NRST must be held low for a minimum of two microcycles. After NRST goes high again, this is what happens:

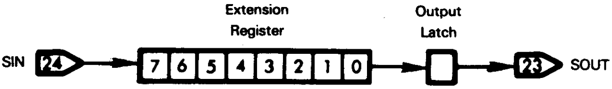
1) All of the programmable registers are cleared.
2) The first instruction is fetched from memory location 0001_{16}.
3) The Bus Request (BREQ) for this first input/output operation occurs within 6-1/2 microcycles after NRST goes high.

The NRST signal can be used at any time to reset the CPU, and must be used following power-up since SC/MP may power up in a random condition. After power has first been applied to the CPU, you should allow approximately 100 milliseconds for the oscillator and internal clocks to stabilize before applying the NRST signal.

SC/MP SERIAL INPUT/OUTPUT OPERATIONS

The SC/MP CPU not only has two of its 40 pins designated primarily for serial input/output operations, it also dedicates one instruction from its rather limited instruction set solely to serial I/O. Allocation of this amount of a CPU's resources for this purpose would seem unwarranted with most microprocessors: however, keep in mind that SC/MP is a very low cost device and intended primarily for use in slow-speed applications. It is quite likely that SC/MP will frequently be used to transfer data serially, so it is therefore not only reasonable but advantageous to provide straightforward methods of performing these operations. Let us look now at how this is done with SC/MP.

In our description of SC/MP's programmable registers, we described the Extension (E) register as an 8-bit register. **When the E register is used for serial I/O, it is actually a 9-bit register with connections to two of the device pins as shown in the figure below.**



When the SC/MP SIO (Serial Input/Output) instruction is executed, the contents of the Extension register are shifted right one bit position: the previous contents of bit 0 are loaded into the output latch and output on the SOUT pin, and the level (1 or 0) present at the SIN pin is loaded into bit 7 of the Extension register. The Extension register can be loaded from, and its contents can be transferred to the Accumulator. A typical serial output operation would thus consist of:

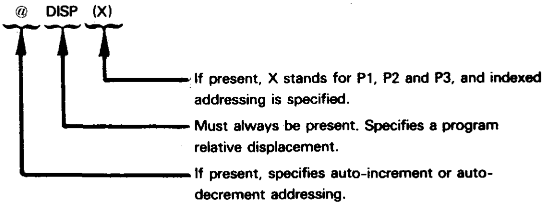
1) Loading the Accumulator with the data byte that is to be transmitted.
2) Transferring the contents of the Accumulator into the Extension register.
3) Performing eight SIO instructions to shift the contents of the Extension register into the output latch and out onto the SOUT pin.

Of course, this sequence does not cover all the programming requirements for serial data transfers. For example, your program must provide some method of timing the bit transmission. This is easily accomplished with SC/MP by using the Delay (DLY) instruction which can generate variable time delays ranging from 13 to 131,593 microcycles. For asynchronous operations, one of the SC/MP Flags which are connected to device pins can be pulsed each time a new bit is shifted out (or in) and one of the sense conditions inputs (SENSEA or SENSEB) can be tested to detect bit received/ready.

THE SC/MP INSTRUCTION SET

Table 3-4 lists the SC/MP instruction set.

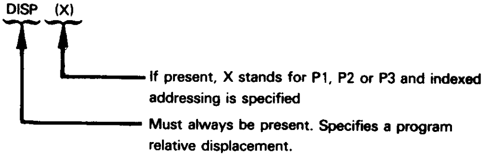
Memory reference instructions are shown as having either full or limited addressing capability. Full addressing capability is identified in the operand as follows:


@ DISP (X)
If present, X stands for P1, P2 and P3, and indexed addressing is specified.
Must always be present. Specifies a program relative displacement.
If present, specifies auto-increment or auto-decrement addressing.

Thus, the real options associated with full addressing capability are:

DISP	Direct, program relative addressing
DISP(X)	Direct, indexed addressing
@DISP(X)	Auto-increment or auto-decrement addressing

Limited addressing capabilities do not include the auto-increment and auto-decrement feature. The operand field for instructions with limited addressing capability is shown as follows:


DISP (X)
If present, X stands for P1, P2 or P3 and indexed addressing is specified
Must always be present. Specifies a program relative displacement.

The serial I/O instruction inputs serial data via the high order bit of the Extension register, and/or outputs serial data via the low order bit of the Extension register.

The serial I/O instruction works as a one-bit right shift of the Extension register contents, with bit 0 being shifted to the SOUT pin and the SIN pin being shifted into bit 7. This has been illustrated along with the logic description.

It is worth noting that SC/MP has no Jump-to-Subroutine instruction; rather the XPPC instruction is used to exchange the contents of the Program Counter with the contents of a Pointer register. In very simple applications (and those are the applications for which SC/MP is intended) this is a very effective scheme. Providing subroutines are not nested, a subroutine's beginning address may be stored in a Pointer register, then execution of XPPC moves the subroutine's starting address to the Program Counter, thereby executing the subroutine — but at the same time, the Program Counter contents are stored in the Pointer register, thus preserving the return address. At the conclusion of the subroutine, execution of another XPPC instruction is all that is needed to return from the subroutine. The only penalty paid is that one Pointer register is out of service while the subroutine is being executed. If all Pointer registers are needed by the subroutine, or if subroutines are nested, then the return address which is stored in the Pointer register must be saved in memory. In these more complicated applications, one of the Pointer registers will probably be used as a Stack Pointer, and addresses will be saved on the Stack.

This type of subroutine access, while it may appear primitive to a minicomputer programmer, is very effective in simple microcomputer applications.

The following symbols are used in Table 3-4.

AC	Accumulator
C	Carry status
DATA	An 8-bit binary data unit
DISP	An 8-bit signed binary displacement
E	The Extension register
EA	Effective address, determined by the instruction. Options are: DISP EA is [PC] + DISP DISP(X) EA is [X] + DISP @DISP(X) EA is [X] if DISP ≥ 0, EA is [X] + DISP if DISP < 0; in both cases [X]←[X] + DISP after EA is calculated.
E<i>	The ith bit of the Extension register
IE	Interrupt Enable
O	Overflow status
PC	Program Counter
X	One of the three Pointer registers
SIN	Serial Input pin
SOUT	Serial Output pin
SR	Status register
Z	Zero status
@	Auto-increment flag
X<y,z>	Bits y through z of a Pointer register. For example, X<7,0> represents the low order byte of one of the Pointer registers.
@DISP(X)	This designates the available addressing modes for the SC/MP, as described above. In all three of the addressing modes, if -128 is specified for DISP, the contents of the Extension register are used instead of DISP.
[ ]	Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.
[[ ]]	Implied memory addressing; the contents of the memory location designated by the contents of a register.
Λ	Logical AND
V	Logical OR
∀	Logical Exclusive-OR
←	Data is transferred in the direction of the arrow.
←→	Data is exchanged between the two locations designated on either side of the arrow.

Under the heading of STATUSES in Table 3-4, an X indicates statuses which are modified in the course of the instructions execution. If there is no X, it means that the status maintains the value it had before the instruction was executed.

Table 3-4. SC/MP Instruction Set Summary

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	O	
I/O	SIO		1			[E<I-1>]←[E<I>] SOUT ← [E0] [E7] ←SIN Shift the Extension register right one bit. Shift bit 0 of the Extension register to the output pin SOUT. Shift the data at input pin SIN into bit 7 of the Extension register.
PRIMARY MEMORY REF AND I/O	LD	@ DISP(X)	2			[AC]←[EA] Load Accumulator from addressed memory location.
	ST	@ DISP(X)	2			[EA]←[AC] Store Accumulator contents in addressed memory location.
SECONDARY MEMORY REFERENCE AND MEMORY OPERATE	ADD	@ DISP(X)	2	X	X	[AC]←[AC] + [EA] + [C] Add binary to Accumulator the addressed memory location's contents with Carry.
	DAD	@ DISP(X)	2	X		[AC]←[AC] + [EA] + [C] Add decimal to Accumulator the addressed memory location's contents with Carry.
	CAD	@ DISP(X)	2	X	X	[AC]←[AC] + [EA] + [C] Add complement of addressed memory location's contents with Carry to Accumulator.
	AND	@ DISP(X)	2			[AC]←[AC] ∧ [EA] AND Accumulator with addressed memory location's contents.
	OR	@ DISP(X)	2			[AC]←[AC] ∨ [EA] OR Accumulator with addressed memory location's contents.
	XOR	@ DISP(X)	2			[AC]←[AC] ⊻ [EA] Exclusive-OR Accumulator with addressed memory location's contents.
	ILD	@ DISP(X)	2			[EA]←[EA] + 1; [AC]←[EA] Increment addressed memory location's contents then load into Accumulator.
	DLD	@ DISP(X)	2			[EA]←[EA] - 1; [AC]←[EA] Decrement addressed memory location's contents, then load into Accumulator.

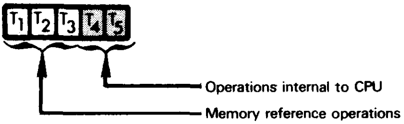
Table 3-4. SC/MP Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	O	
IMMEDIATE	LDI	DATA	2			[AC] ← DATA Load immediate into Accumulator.
IMMEDIATE OPERATE	ADI	DATA	2	X	X	[AC] ← [AC] + DATA + [C] Add binary immediate. Add Carry to result.
	DAI	DATA	2	X		[AC] ← [AC] + DATA + [C] Decimal add immediate. Add Carry to result.
	CAI	DATA	2	X	X	[AC] ← [AC] + DATA + [C]' Add the contents of the Accumulator to the complement of the immediate data value. Add Carry to result.
	ANI	DATA	2			[AC] ← [AC] ∧ DATA AND immediate.
	ORI	DATA	2			[AC] ← [AC] ∨ DATA OR immediate.
	XRI	DATA	2			[AC] ← [AC] ⊻ DATA Exclusive-OR immediate.
JUMP	JMP	DISP(X)	2			[PC] ← EA Unconditional jump to effective address.
JUMP ON CONDITION	JP	DISP(X)	2			If [AC] ≥ 0; [PC] ← EA If the Accumulator contents are greater than 0, jump to effective address.
	JZ	DISP(X)	2			If [AC] = 0; [PC] ← EA If the Accumulator contents equal 0, jump to effective address.
	JNZ	DISP(X)	2			If [AC] = 0; [PC] ← EA If the Accumulator contents are not 0, jump to effective address.

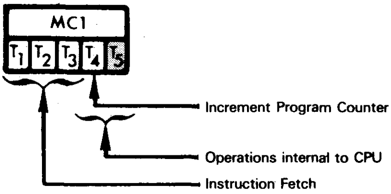
n summary, therefore, a clock period will normally have 9 segments, but may have 4, 5, , 7 or 8 segments.

Jote that **the only time you ever need to know about clock segmentation is when 'ou are creating your own clock signals.** If you use the 8224 Clock Signal Generator described later in this chapter) you can ignore clock signal segmentation.

:lock periods T_1, T_2 and T_3 of each machine cycle are used (with one exception) or memory reference operations. During periods T_4 and T_5 functions internal to he CPU are executed. These two clock periods can be used by external logic for a imited number of approved operations that do not involve the CPU:


T_1 T_2 T_3 T_4 T_5
Operations internal to CPU
Memory reference operations

'he first three clock periods of the first machine cycle are always used to fetch an nstruction from memory, and load it into the Instruction register. The first machine ycle always has at least four clock periods, with the Program Counter being increnented during T_4:


MC1
T_1 T_2 T_3 T_4 T_5
Increment Program Counter
Operations internal to CPU
Instruction Fetch

8080A INSTRUCTION STATUS

'he CPU identifies the operations that will occur during every nachine cycle by outputting status information on the Data 3us during clock period T_2. External logic uses SYNC and the $\Phi 1$ ulse at the start of T_2 to read status off the Data Bus. Timing is ilustrated in Figure 4-4.

f you are using an 8228 System Controller, it will decode status output on the)ata Bus during T_2. By combining this status information with the three control sigals: $\overline{\text{WR}}$, DBIN and HLDA, the 8228 System Controller is able to generate a set of bus ontrol signals which will interface industry standard memory devices and external ɔgic.

f you are not using an 8228 System Controller, then you must provide external ɔgic that decodes the Data Bus during $\Phi 1$ of T_2. Your external logic must generte control signals which will be active during subsequent clock periods, at which ime the Data Bus no longer holds status information.

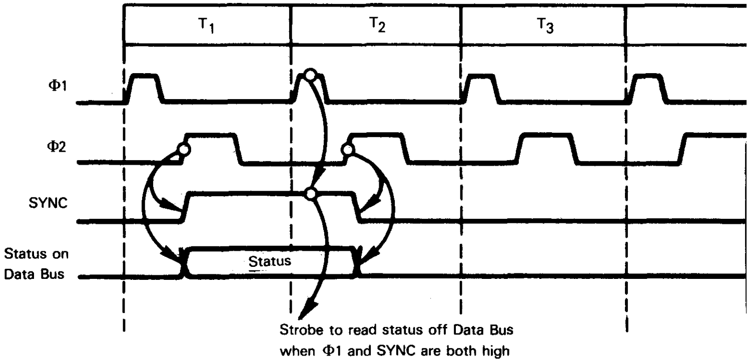

T1
T2
T3
Φ1
Φ2
SYNC
Status on Data Bus
Status
Strobe to read status off Data Bus when Φ1 and SYNC are both high

Figure 4-4. Status Output During T_2 Of Every Machine Cycle

Table 4-2 defines the statuses which may be output during clock period T_2. Table 4- defines the way in which statuses should be interpreted to identify the various possibl types of machine cycles.

Table 4-2. Statuses Output Via The Data Lines During The Second Clock Cycle Of An 8080A Machine Cycle

SYMBOLS	DATA BUS BIT	DEFINITION
HLTA	D3	Acknowledge signal for Halt instruction
INTA*	D0	Acknowledge signal for INTERRUPT request. Signal should be used to gate a Restart instruction onto the Data Bus when DBIN is active.
INP*	D6	Indicates that the Address Bus contains the address of an input device and the input device should be placed on the Data Bus when DBIN is active.
OUT	D4	Indicates that the Address Bus contains the address of an output device and the Data Bus will contain the output data when $\overline{WR}$ is active.
MEMR*	D7	Designates that the Data Bus will be used for memory read data.
M1	D5	Provides a signal to indicate that the CPU is in the fetch cycle for the first byte of an instruction.
STACK	D2	Indicates that the Address Bus holds the pushdown stack address from the Stack Pointer.
$\overline{WO}$	D1	Indicates that the operation in the current machine cycle will be a WRITE memory or OUTPUT function ($\overline{WO}$ = 0). Otherwise a READ memory, INPUT operation, or interrupt or Halt acknowledge will be executed.

*These three status bits can be used to control the flow of data onto the 8080A Data Bus.

Table 4-3. Statuses Output On The Data Bus For Various Types Of Machine Cycle

DATA BUS BIT	STATUS INFORMATION	TYPE OF MACHINE CYCLE									
		INSTRUCTION FETCH	MEMORY READ	MEMORY WRITE	STACK READ	STACK WRITE	INPUT READ	OUTPUT WRITE	INTERRUPT ACKNOWLEDGE	HALT ACKNOWLEDGE	INTERRUPT ACKNOWLEDGE WHILE HALT
D0	INTA	0	0	0	0	0	0	0	1*	0	1
D1	$\overline{\text{WO}}$	1	1	0	1	0	1	0	1	1	1
D2	STACK	0	0	0	1	1	0	0	0	0	0
D3	HLTA	0	0	0	0	0	0	0	0	1	1 (0)
D4	OUT	0	0	0	0	0	0	1	0	0	0
D5	M1	1	0	0	0	0	0	0	1	0	1
D6	INP	0	0	0	0	0	1	0	0	0	0
D7	MEMR	1	1	0	1	0	0	0	0	1 (0)	0

(0) Identifies status outputs of the NEC 8080A which differ from those of the Intel 8080A.

* This status is output as 0 by the NEC 8080A during a Call instruction being executed within the interrupt acknowledge process.

NSTRUCTION FETCH SEQUENCE

nstruction fetch timing is illustrated in Figure 4-5; events occur as follows:

eriod T_1 The leading edge of Φ2 triggers the SYNC high pulse, identifying period T_1.

WAIT is low, since the CPU is not in the Wait state.

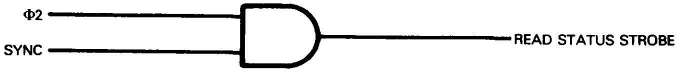
$\overline{\text{WR}}$ remains high since this is an instruction fetch cycle; data is not being written to memory.

The leading edge of Φ2 is used to set selected Data Bus lines high, providing external logic with status information as follows:

RI/$\overline{\text{WO}}$ (D1) The CPU is expecting data input.
M1 (D5) This is an instruction fetch period.
MEMR (D7) Data input is expected from memory.

The leading edge of Φ2 is used to set the required memory address on the address lines A0 to A15.

eriod T_2 External logic uses the Φ1 pulse of time period T_2 to read status off the Data Bus. The read status strobe may be created as follows:



Remember, if you are using an 8228 System Controller, it reads and decodes status for you.

Immediately after status has been output on the Data Bus, the Data Bus i free to receive the instruction object code. The address for the instructio object code will be on the Address Bus; this address appears on the Ad dress Bus during T_1, beginning with the rising edge of Φ2. The fact tha status has been output and the Data Bus is free to receive the instructio object code is indicated by DBIN being pulsed high. The DBIN high puls begins with the rising edge of Φ2 in T_2 and lasts exactly one cloc period.

Period T_3 While DBIN is high, external logic must place the addressed instructio code on the Data Bus. The CPU will store this data in the Instructio register —whence the Control Unit interprets it as an instruction code.

The Data Bus is floated at Φ2 during T_3. This means that the Data Bu has been disconnected from the CPU and can be used in any way by logi external to the CPU.

Period T_4 The Address Bus is floated at Φ2 during T_4.

The 8080A uses 1, 2 and 3 byte instructions. Each byte of a multibyte instructio requires its own instruction fetch. Exact timing for multibyte instructions is give later in this chapter, after the 8080A instruction set has been described.

A MEMORY READ OR WRITE OPERATION

So far as external logic is concerned, there is no difference between "read from memo ry" timing and instruction fetch timing — except that the M1 status (D5 on the Dat Bus) is high during an instruction fetch only. **Figure 4-5** therefore **applies to a memor read operation also.**

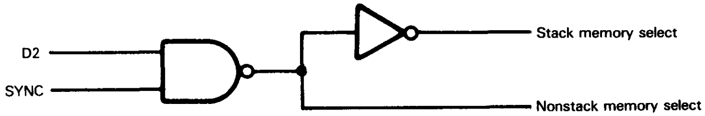
Since a memory read operation is executed during time periods T_1, T_2 and T_3 of machine cycle, the presence of a memory read operation in an instruction's execu tion sequence will add one machine cycle to instruction execution time.

Figure 4-6 shows timing and signal sequences for a memory write operation. Th signal sequences are identical to the instruction fetch sequence with the excep tion that DBIN remains low during T_2 and T_3, and different status signals are out put on the Data Bus during T_1.

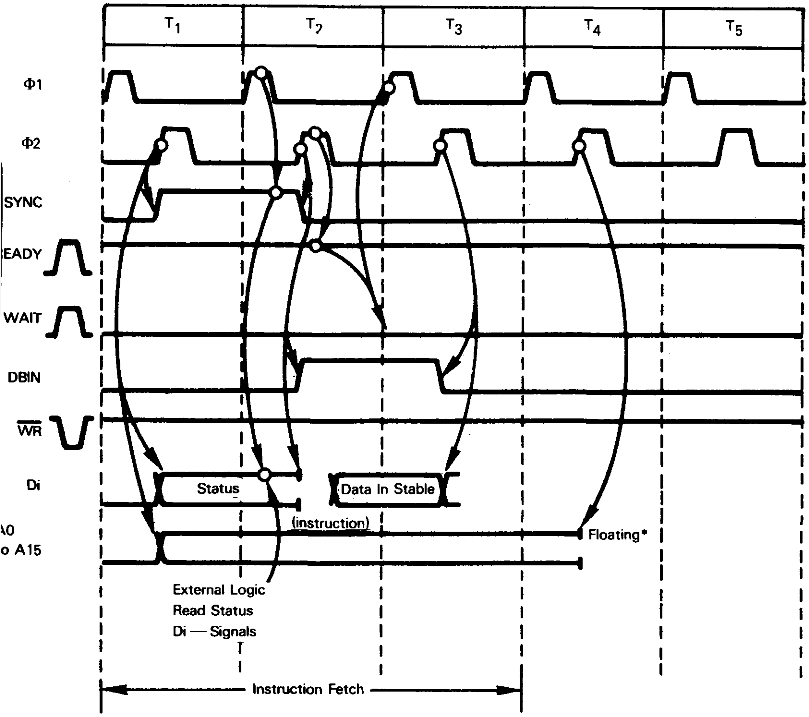
SEPARATE STACK MEMORY MODULES

One 8080A CPU can access two memory modules with overlapping memory ad dresses: a stack memory module and a nonstack memory module. Overlappin memory addresses can be used by the two memory modules, since Stack status (D high at Φ1 in T_2) can be used to select the stack memory, while lack of Stack status (D low at Φ1 in T_2) can be used to select nonstack memory. External logic must decod the address as referencing stack or nonstack memory.

Note that the 8228 System Controller does not generate a STACK control signa Nevertheless, if you wish, you may implement separate stack and nonstack memory with overlapping addresses; this requires your own status decode logic to isolate th Stack status. Such logic is quite simple, and may be illustrated as follows:



The only disadvantage associated with having a separate stack memory is that nor stack instructions cannot reference the stack memory.



'The NEC 8080A maintains the address on the Address Bus during T_4 and T_5.

Figure 4-5. 8080A Instruction Fetch Sequence

'HE WAIT STATE

\ Wait state may occur between clock periods T_2 and T_3. The Vait state frees external logic or memory from having to operte at CPU speed. Wait state timing is illustrated in Figure 4-7 nd Figure 4-8.

8080A SLOW MEMORIES

READY is low during Φ2 of T_2, the 8080A CPU will enter the Wait state following T_2. he Wait state consists of any number of clock periods during which the CPU performs o operations and maintains the levels of all output signals. The Wait state ends when EADY is input high. The CPU samples READY during every Φ2 pulse within the Wait tate; the Wait state will therefore end with the Φ1 pulse which follows a Φ2 pulse durıg which READY is sensed high.

lemory interface logic in any 8080A microcomputer system must be designed to nticipate that every memory access either will, or will not require a Wait state.

memory is as fast as the 8080A CPU, then READY will normally be held high, in antiipation of no Wait state. In Figures 4-7 and 4-8 a broken line is used to represent this READY normally high" case. Memory interface logic will pull READY low in order to insert one or more Wait machine cycles only in special circumstances; memory interface ɔgic has until Φ2 of T_2 to pull READY low.

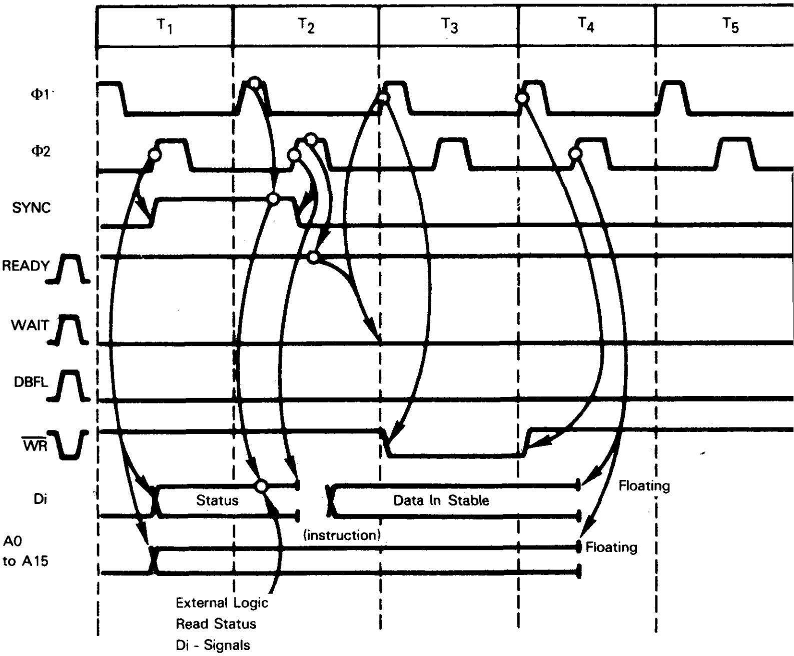


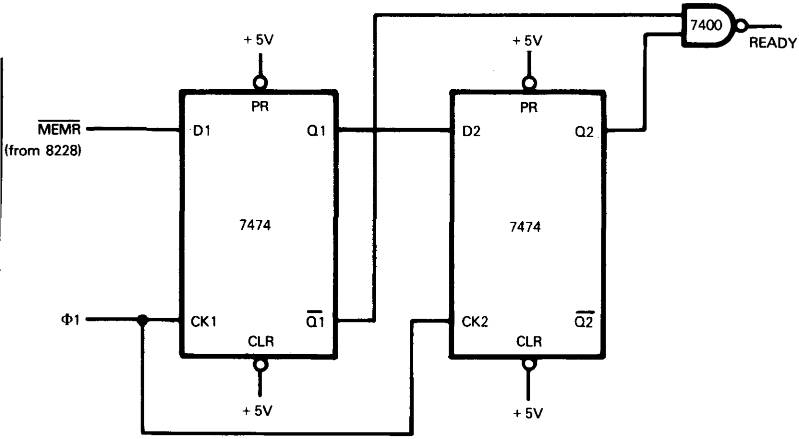
Figure 4-6. 8080A Memory Write Timing

If memory is slower than the 8080A CPU, then READY will normally be held low in an cipation of one or more Wait machine cycles occurring between T_2 and T_3. In tl special circumstance where no Wait state is needed, memory interface logic has un Φ2 of T_2 to set READY high.

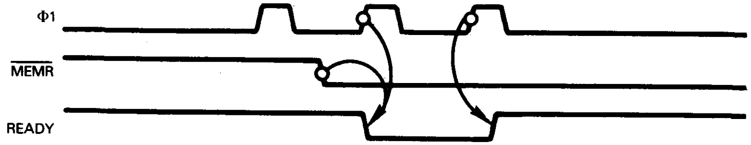
Note that $\overline{WR}$, if active, will be held low for the entire duration of a Wait state. This because if $\overline{WR}$ is to be set low, the transition occurs at Φ1 of T_2 and lasts until Φ1 T_4 — a period which completely encompasses the Wait state.

8080A WAIT STATE REQUEST LOGIC

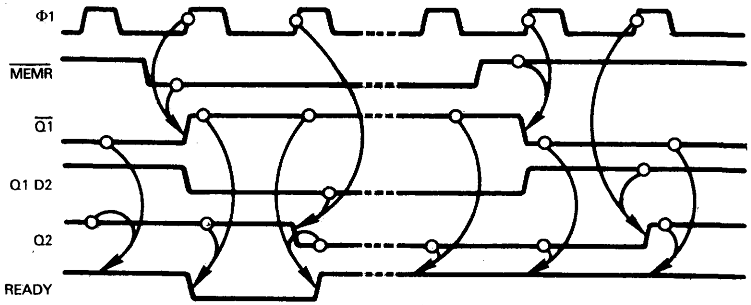
lelatively simple logic can be used to add a Wait state to a nachine cycle. Consider the following scheme:



ur goal, using the logic above, is to create a low READY pulse, which is one clock eriod wide, whenever $\overline{MEMR}$ makes a high-to-low transition.

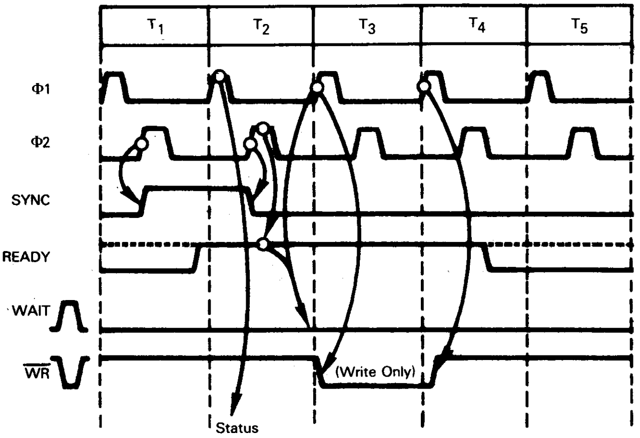


onsider the sequence of signal transitions in the logic we have illustrated above. At ach Φ1 clock pulse, transitions will occur as follows:



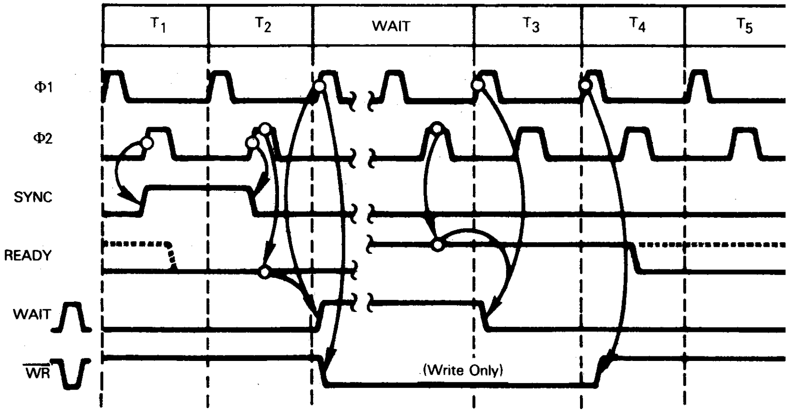
requires $\overline{Q1}$ and Q2 to be high simultaneously for READY to be low; and that condi-on exists for a single clock pulse.

bserve that you can use READY to trigger a one-shot in order to create a low EADY input of any duration.



▪▪▪▪▪▪▪▪ Represents alternate signal form for READY as described in text accompanying this figure.

Figure 4-7. The 8080A CPU Operating With Fast Memory And No Wait State



▪▪▪▪▪▪▪▪ Represents alternate signal form for READY as described in text accompanying this figure.

READY is false at Φ2 in T_2, so next Φ1 pulse initiates a Wait state, with WAIT set high by the leading edge of the Φ1 pulse. When READY is high at a Φ2 pulse, clock period T_3 will be initiated by the next Φ1 pulse and WAIT will be reset low.

Figure 4-8. The 8080A CPU Operating With Slow Memory And A Normal Wait State

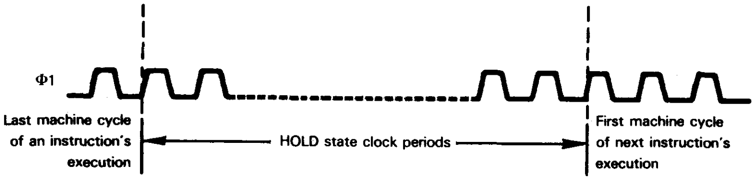
THE WAIT, HOLD AND HALT STATES

We have discussed the Wait state within an 8080A microcomputer system, no we have to look at two further states during which instructions are not e ecuted: the Hold and the Halt states.

The fact that there are three states within which instructions are not being executed frequently a source of confusion to 8080A users. Let us, therefore, clearly identify tl differences between these three states before continuing a discussion of the Hold and Halt states.

As we have already seen, the Wait state consists of one or more clock periods which are inserted within a machine cycle, giving external logic time to respond to a memory access. Thus, the Wait state consists of an indeterminate number of clock periods which occur within a machine cycle and extend the duration of that machine cycle.

The purpose of the Hold state is to float the System busses so that external logic can perform direct memory access operations. Conceptually, therefore, a Hold condition consists of any number of clock periods, occurring in between two machine cycles which define the termination of one instruction's execution and the initiation of the next instruction's execution:


Φ1
Last machine cycle of an instruction's execution
HOLD state clock periods
First machine cycle of next instruction's execution

The Hold state may be looked upon as a period of time during which the CPU goes into a state of suspended animation.

The Halt state results from the execution of a Halt instruction. The System Bus is not floated during a Halt state. During the Halt state, the CPU simply marks time. The purpose of a Halt state is to define those time intervals when there is nothing for the CPU to do; now when the CPU has nothing to do, it is only logical to assume that the CPU cannot know how long it will be before it has something useful to do. Typically a Halt condition will end when some external logic demands the service of the CPU. One method that external logic uses to demand CPU service is the interrupt request. The 8080A therefore requires an interrupt request to terminate the Halt state.

Let us now look at the Hold and Halt states in more detail.

THE HOLD STATE

The Hold state allows external logic to stop the CPU.

The Hold state is similar to the Wait state. During both states, signals output by the CPU are held constant; but the Data and Address Busses are floated in the Hold state only, not in the Wait state.

The Hold and Wait states are also initiated in different ways and they serve different functions.

The Wait state is initiated if external operations will not be completed during T_3. The purpose of the Wait state is to allow the CPU to operate with slow memories or external logic, therefore a Wait always occurs between clock periods T_2 and T_3.

A Hold state is initiated by the hold request input signal HOLD. The CPU acknowledges the onset of the Hold state by outputting HLDA high. If a HOLD is requested during a read or input operation ($RI/\overline{WO}$ (D1) high in T_2), then HLDA is set high by the leading edge of Φ1 in T_3. If a HOLD is requested during a write or output operation, then HLDA is set high by the leading edge of Φ1 in the cycle following T_3.

Note that even though HOLD is acknowledged and the Hold state is initiated in T_3 during a read memory or input data machine cycle, logic must still hold data steady on the Data Bus until the leading edge of Φ2 in T_3. This is because operations internal to the CPU will be executed normally during a HOLD. Operations internal to the CPU will only cease if the Hold state lasts for more cycles than would normally be present before the onset of the next T_1 cycle.

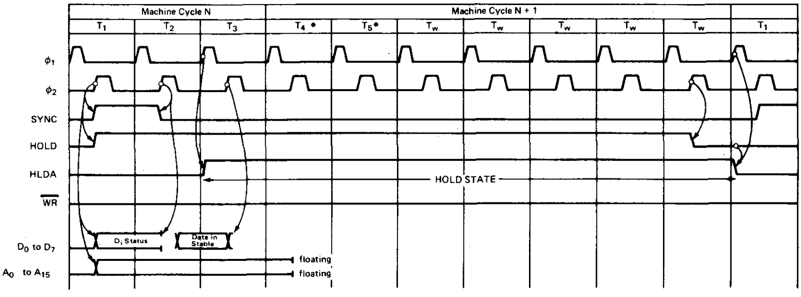
HOLD low will cause the end of the Hold state. HOLD low must coincide with the leading edge of Φ1 or Φ2, and will terminate the Hold state at the Φ1 pulse of the next machine cycle's T_1 clock period. The 8080A CPU will signal the end of the Hold state with HLDA false.

During the Hold state, the Data Bus and the Address Bus are floated. Floating begins at Φ2 in T_3 for a read operation and at Φ2 in the clock period following T_3 otherwise.

Figures 4-9 and 4-10 illustrate some variations on the Hold state.

The NEC 8080A and the Intel 8080A differ when a Hold is requested during a DAD instruction's execution. The NEC 8080A initiates the Hold as though a read operation was occurring, while the Intel 8080A initiates the Hold operation as though a write operation was occurring.

NEC 8080A HOLD DIFFERENCES



*optional, depending on instruction being executed

Figure 4-9A. Floating Of Data And Address Busses At Φ2 In T_3, For READ Operation Being Completed Prior To Onset Of Hold State

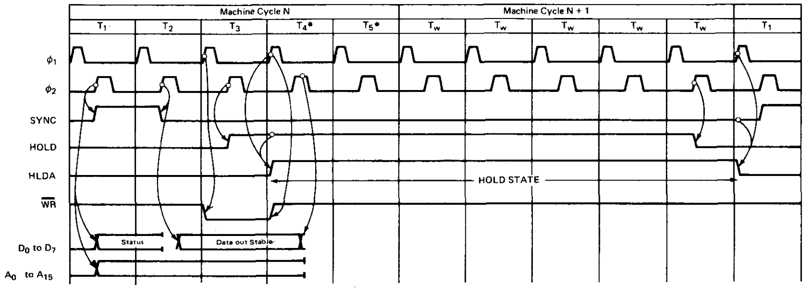


Figure 4-9B. Floating Of Data And Address Busses At Φ2 In T_4, For A WRITE, Or Any Non-READ Operation ($RI/\overline{WO}$=False)

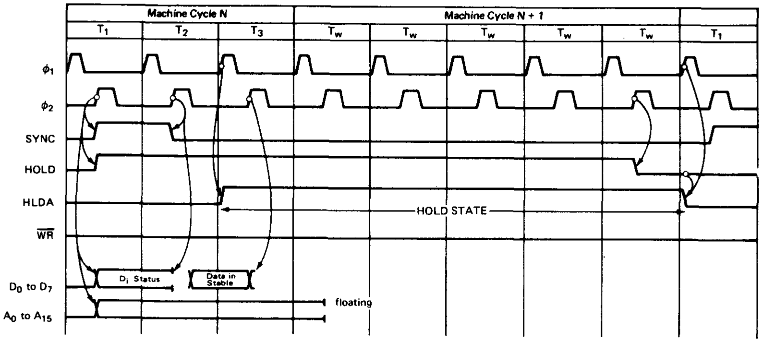


Figure 4-10A. Floating Of Data and Address Busses For READ Operation In A Three Clock Period Machine Cycle

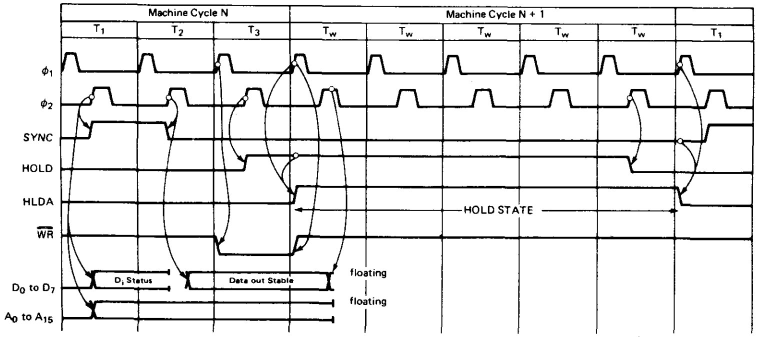


Figure 4-10B. Floating Of Data And Address Busses At Φ2 in T_1, For WRITE Or Any Non-READ Operation Being Completed Prior To Onset Of Hold State

THE HALT STATE AND INSTRUCTION

The Halt state is similar to the Wait state, except that it is initiated by a Halt instruction.

The Halt state is not initiated by READY low, although READY low is a necessary requirement for the onset of the Halt state. This means that READY high cannot be used to terminate a Halt state. Instead, **an interrupt request (INT high) must be used to terminate the Halt state.**

Note that if interrupts have been inhibited, the interrupt request (INT high) will never be acknowledged, and the only way to get out of a Halt state is to power down, then power up the CPU.

An anomaly of the Halt state is that the Data and Address Busses may be floated by entering the Hold state after entering the Halt state; that is, you can move into, and out of the Hold state while in the Halt state.

If the Hold state is entered after the Halt state, then the Hold state must be exited by setting HOLD low before exiting the Halt state.

During a HALT, a hold request signaled by HOLD will not be acknowledged if an interrupt has been requested (INT high) but not acknowledged (INTE high); i.e., the CPU will not enter the Hold state in the time between an interrupt being requested and acknowledged. Once the interrupt has been acknowledged (INTE low), the CPU may enter the Hold state.

Figure 4-30 illustrates signal sequences and timing for the Halt instruction (and state).

THE RESET OPERATION

A RESET high signal input to the 8080A CPU will clear the Program Counter and disable interrupts.

To properly perform the reset operation, RESET should be held high for at least three clock periods. During these three clock periods, reset operations are executed in the following sequence:

1) The Program Counter is cleared.
2) All interrupt requests are disabled.
3) Internal interrupt acknowledge logic (associated with signal INTE) is cleared.
4) Internal hold acknowledge logic (associated with signal HLDA) is cleared.

For as long as RESET is high, all 8080A CPU operations will be suspended.

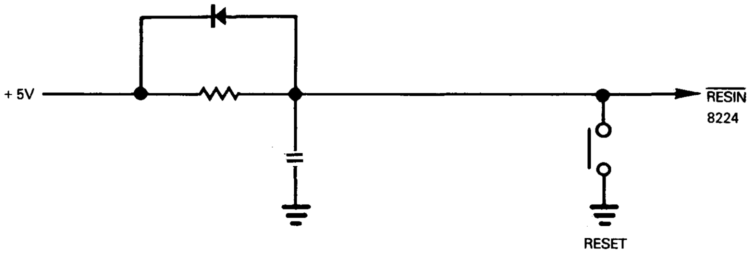
When RESET is reset low, instruction execution will resume with a T_1 clock period at the next Φ1 pulse. Since the Program Counter contains 0000, the first instruction executed following RESET will be the instruction stored in memory location 0000_{16}.

Interrupts remain disabled when program execution resumes.

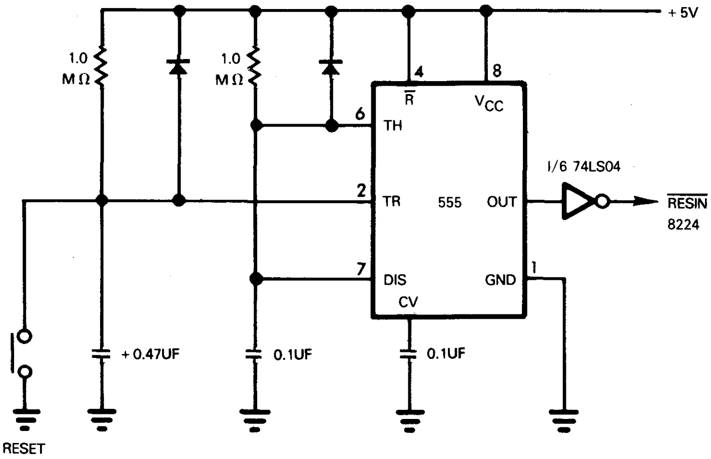
When you power up any 8080A system you must simultaneously reset it. Powering up does not reset or change anything within the 8080A. If you power up without resetting, then registers, including the Program Counter, will contain undefined data; thus program execution will immediately and erroneously begin at some random location of memory.

Here are two possible reset on power up logic implementations:

First a simple logic sequence:



Next a more complex, and more reliable one:



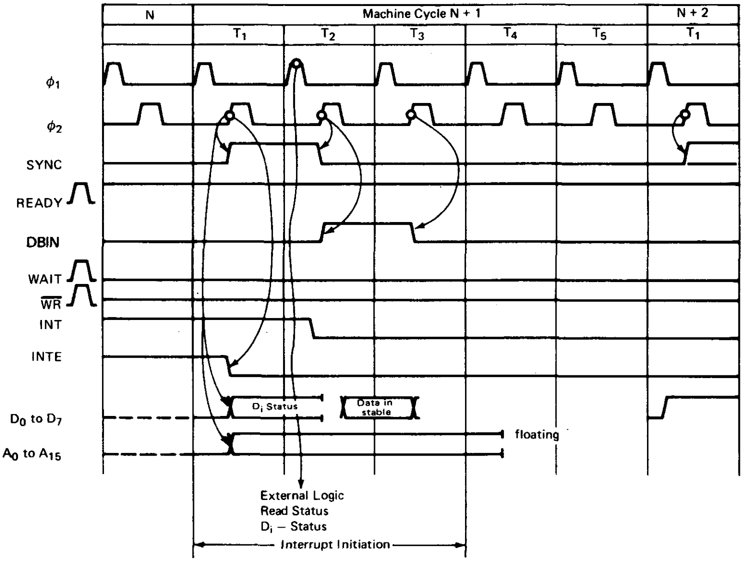


Figure 4-11. Interrupt Initiation Sequence

EXTERNAL INTERRUPTS

External logic may request an interrupt at any time by setting the INT input high. An interrupt request will only be acknowledged if interrupts have been enabled. Normally the EI (Enable Interrupts) and DI (Disable Interrupts) instructions are executed to enable and disable interrupts; however, interrupts are automatically disabled by the CPU during the RESET condition, and following an interrupt acknowledge.

The 8080A CPU outputs INTE high when interrupts have been enabled, and low when interrupts are disabled. If interrupts are enabled, then the 8080A CPU will acknowledge an interrupt request during the next T_1 clock period, on the rising edge of Φ2. At this time INTE is set low to reflect the fact that an interrupt acknowledge automatically disables interrupts. **Timing is illustrated in Figure 4-11.**

The 8080A CPU informs external logic that an interrupt has been acknowledged by outputting this status on the Data Bus:

D0 - INTA
D1 - RI/WO
D5 - M1

INTA is the principal interrupt acknowledge status; it is converted into a separate interrupt acknowledge control signal by the 8228 System Controller.

Once an interrupt has been acknowledged, the 8080A CPU enters an instruction fetch sequence — but with two differences:

1) Program Counter increment logic is suppressed.
2) Different statuses are output on the Data Bus during T_2. The statuses output on the Data Bus during various machine cycles are summarized in Table 4-3.

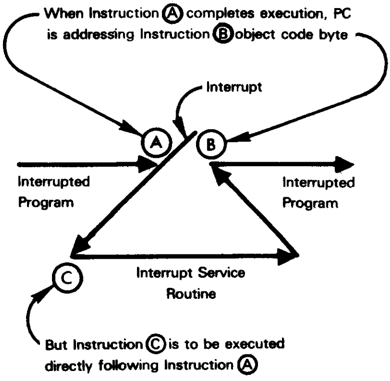
The different statuses output during T_2 of a normal, or a post-interrupt acknowledge instruction fetch are very important.

During a normal instruction fetch sequence, MEMR is output true on D7.

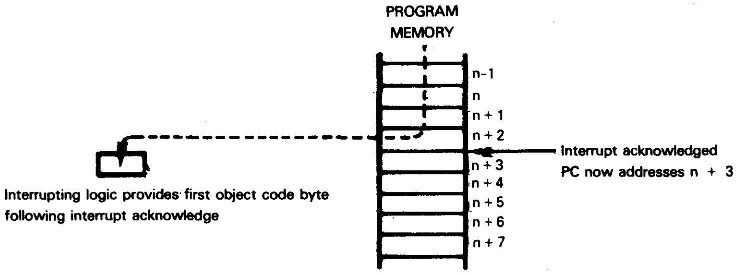
During the instruction fetch sequence which follows an interrupt acknowledge, MEMR is not output true on D7, but INTA is output true on D0.

Thus, external logic can differentiate between a normal instruction fetch and the instruction fetch sequence which follows an interrupt acknowledge.

It is very important that external logic be able to differentiate between a normal instruction fetch and an interrupt acknowledge instruction fetch. When the interrupt is acknowledged, the Program Counter is addressing an instruction which will not get executed until the interrupt service routine has completed execution:


When Instruction Ⓐ completes execution, PC is addressing Instruction Ⓑ object code byte
Interrupt
Ⓐ
Ⓑ
Interrupted Program
Interrupted Program
Ⓒ
Interrupt Service Routine
But Instruction Ⓒ is to be executed directly following Instruction Ⓐ

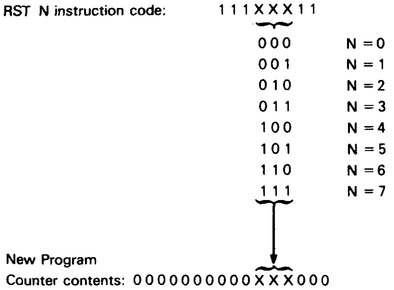
Therefore the first instruction executed following the interrupt acknowledge must save the Program Counter contents. The last instruction executed within the interrupt service routine restores the Program Counter contents.During the instruction fetch which follows an interrupt acknowledge, the Program Counter increment logic is suppressed, because the 8080A CPU expects the object code for the first interrupt service routine instruction to be supplied by the interrupting device instead of memory:



The object code provided by external logic during the instruction fetch which follows the interrupt acknowledge must be the object code for an instruction which will save the Program Counter contents for subsequent retrieval. There is only one instruction which will do this and that is a subroutine CALL instruction. Recall from Volume I that the subroutine CALL instruction will save the current Program Counter contents on the Stack, then will load a new starting address into the Program Counter. Thus, a subroutine CALL instruction satisfies the logical requirements for interrupt service routine initiation.

The normal way of terminating a subroutine is via a Return instruction. This instruction loads the Program Counter from the top of the Stack. The Return instruction will, therefore, satisfy the logical requirements for interrupt service routine termination.

There are two types of 8080A subroutine CALL instruction: the RESTART (RST) and the CALL. **The RST instruction is a one-byte subroutine CALL with the following object code:**

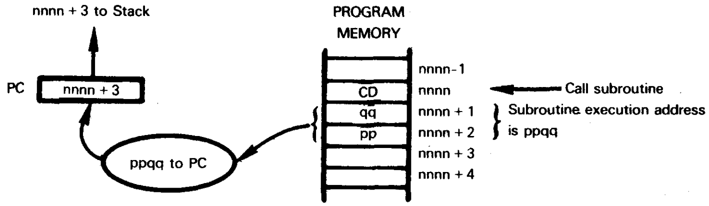


Therefore RST N instructions are equivalent to subroutine CALL instructions, with program execution branching as follows:

Subroutine

RST 0 branch to 0000_{16}
RST 1 branch to 0008_{16}
RST 2 branch to 0010_{16}
RST 3 branch to 0018_{16}
RST 4 branch to 0020_{16}
RST 5 branch to 0028_{16}
RST 6 branch to 0030_{16}
RST 7 branch to 0038_{16}

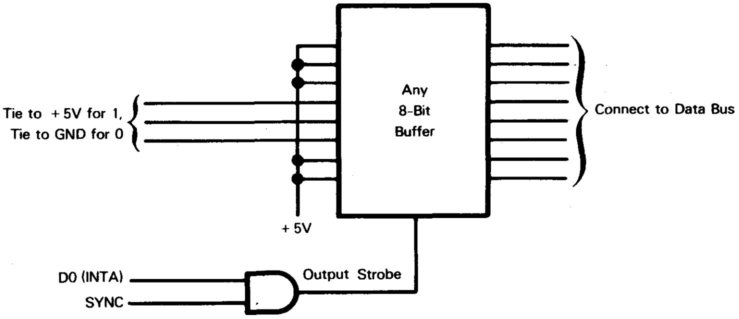
The CALL instruction is a typical three-byte, direct memory addressing subroutine call:



The address of the instruction following the subroutine call (nnnn+3) is saved on the Stack, to be retrieved subsequently by a Return instruction. The second and third CALL instruction object code bytes provide the address of the subroutine's first instruction; this address (ppqq) therefore is loaded into the Program Counter.

What is not clearly understood by many 8080A users is that external logic can respond to an interrupt acknowledge by inserting either an RST or a subroutine CALL instruction.

Responding to an interrupt acknowledge by inserting an RST instruction is very straightforward. The INTA status output during T_2 can be used to select external logic as the source of an object code, while the lack of an MEMR status can be used to suppress the normal instruction fetch which would occur from program memory. Thus, a simple 8-bit I/O buffer will generate a Restart instruction as follows:



With a little more effort, external logic can be designed to provide a subroutine CALL instruction's object code following the interrupt acknowledge. Providing the INTA status is used to suppress normal program memory accesses for the next three machine cycles, logic associated with the external interrupt request can supply the three consecutive object code bytes of a normal subroutine CALL instruction.

In a configuration that includes an 8228 System Controller, if the first object code byte received following INTA output is a CALL (CD_{16}), then the 8228 System Controller outputs two more INTA statuses for the next two machine cycles. Now external logic can use INTA as a signal which disables normal memory accesses, selecting external logic instead. For more details, see the 8228 System Controller description given later in this chapter.

If your configuration does not include an 8228 System Controller, then external logic must be quite complex if it responds to an interrupt acknowledge with a CALL instruction. These are the operations external logic must perform:

1) In response to INTA true, suppress normal memory references and transmit the code CD_{16} to the CPU. This code must be transmitted at the proper time, as an instruction code on the Data Bus.
2) Suppress normal memory accesses for the next two clock periods. Remember, there is no INTA true for these two periods.
3) During the next two clock periods, transmit the low order half, then the high order half of the interrupt service routine starting address. These two address bytes must be provided out of external logic, and their timing on the Data Bus must conform exactly to the second and third bytes of a CALL instruction.

If your configuration includes an 8259 Priority Interrupt Control Unit, then this device takes care of all logic associated with responding to an interrupt acknowledge with a CALL; the 8259 is described later in this chapter.

NEC 8080A INTERRUPT DIFFERENCES

The NEC 8080A does not handle the INTA signal in the same way as the Intel 8080A. In response to a Call instruction executed during an interrupt acknowledge, the NEC 8080A outputs INTA true for three machine cycles; in an Intel 8080A system an 8228 System Controller must be present for this to occur. The NEC 8080A D0 status output also differs at this time; see Table 4-3 for details.

The NEC 8080A responds to Restart instructions following an interrupt acknowledge in the same way as the Intel 8080A.

EXTERNAL INTERRUPTS DURING THE HALT STATE

NEC 8080A INTERRUPT DIFFERENCES

With all 8080A devices except the NEC 8080A, interrupt acknowledge logic during a Halt state is as illustrated in Figure 4-11. For the NEC 8080A, however, the interrupt acknowledge sequence differs slightly during the Halt state only. INTE is reset low by the NEC 8080A on the rising edge of $\Phi 2$ in clock period T_2; this is one clock period later than illustrated in Figure 4-11. Note that this difference in NEC 8080A response applies only to the interrupt acknowledge process occurring within a Halt state.

WAIT AND HOLD CONDITIONS FOLLOWING AN INTERRUPT

An interrupt cannot be acknowledged during a WAIT or HOLD condition, However, either of these conditions may occur following the interrupt acknowledge. For example, if there is insufficient time between $\Phi 1$ in T_2 and $\Phi 2$ in T_2 for external logic to fetch the required RST or CALL instruction, more time may be acquired by using the READY signal to generate a Wait state, as with any instruction's execution.

THE 8080A INSTRUCTION SET

Table 4-4 summarizes the 8080A instruction set; there is a significant departure in instruction set philosophy from the hypothetical microcomputer described in Volume I.

The 8080A is most efficiently programmed by making extensive use of the Stack and of subroutines. By providing a variety of Jump-to-Subroutine on Condition, and Return-from-Subroutine on Condition instructions, the 8080A allows the execution of subroutines to become an integral part of programmed logic sequences.

Observe that the 8080A has a number of 16-bit instructions; that is, instructions that operate on the 16-bit contents of the BC, DE or HL registers. These include 16-bit increment and decrement, 16-bit add, and 16-bit data moves.

The 16-bit instruction XTHL is particularly useful, since by allowing the top two Stack bytes to be exchanged with the HL registers, an easy method is provided for switching addresses.

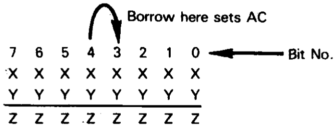
The DAA instruction modifies the A register contents to generate a binary coded decimal equivalent of the original binary value. If carries out of bit 3 or bit 7 result, these are reported in the Auxiliary Carry and Carry statuses, respectively. See Volume I for a discussion of the decimal adjust operation.

NEC 8080A INSTRUCTION SET DIFFERENCES

There are a few differences between NEC 8080A and Intel 8080A instruction execution.

For binary subtraction and BCD arithmetic the NEC 8080A performs operations in what is theoretically the "correct" fashion — which differs from the actual implementation of the Intel 8080A. Specifically, the NEC 8080A has a Subtract status (SUB) which is set after any addition is performed. Only the NEC 8080A has a Subtract status.

The NEC 8080A correctly sets and resets the Auxiliary Carry status (AC) during subtract operations, identifying any borrow by the low order digit as follows:



```
  7 6 5 4 3 2 1 0  ← Bit No.
  X X X X X X X X
- Y Y Y Y Y Y Y Y
= Z Z Z Z Z Z Z Z
```

X, Y and Z represent any binary digits.

Decimal subtraction for the Intel 8080A and NEC 8080A may be illustrated as follows, assuming the contents of Register C are to be subtracted from the contents of Register B:

```
INTEL 8080A          NEC 8080A
MVI   A,99H          MOV   A,B
SUB   C              SUB   C
ADD   B              DAA
DAA
```

In the instruction sequence illustrated above for the Intel 8080A, you cannot use the Subtract instruction directly since it works for binary arithmetic only. You must create the nine's complement of the subtrahend by subtracting it from 99. Then you add the minuend to the nine's complement of the subtrahend. Finally you decimal adjust the result.

In the case of the NEC 8080A you may use the Subtract instruction for either binary or BCD data.

For a complete discussion of decimal subtraction using the Intel 8080A, see "8080 PROGRAMMING FOR LOGIC DESIGN", Chapter 7.

The Carry and Auxiliary Carry statuses are also treated differently by the NEC and Intel 8080A. When Boolean instructions are executed by the Intel 8080A, the Carry status (C) is always reset; the Auxiliary Carry status (AC) is sometimes reset. The NEC 8080A leaves the Carry and Auxiliary Carry statuses alone when executing Boolean instructions.

When the AMD 9080A executes Boolean instructions it always clears both the Carry and Auxiliary Carry statuses.

THE BENCHMARK PROGRAM

Our benchmark program is coded for the 8080A as follows:

```
        LHLD    TABLE    ;LOAD ADDRESS OF FIRST FREE TABLE BYTE IN HL
        LXI     D,IOBUF  ;LOAD STARTING ADDRESS OF IOBUF IN DE
        LDA     IOCNT    ;LOAD I/O BUFFER LENGTH
        MOV     B,A      ;SAVE IN B
LOOP    LDAX    D        ;LOAD NEXT I/O BYTE
        INX     D        ;INCREMENT BUFFER ADDRESS
        MOV     M,A      ;STORE IN TABLE
        INX     H        ;INCREMENT TABLE ADDRESS
        DCR     B        ;DECREMENT BYTE COUNT
        JNZ     LOOP     ;RETURN FOR MORE BYTES
        SHLD    TABLE    ;AT END, RESTORE ADDRESS OF FIRST FREE TABLE
                         BYTE
```

The 8080A makes very few assumptions regarding the benchmark program.

The address of the first free byte in the data table is assumed to be stored in the first two bytes of the data table — addressed by the label TABLE. The immediate addressing instruction LHLD loads the contents of the first two bytes of the data table into the H and L registers. At the end of the program, the incremented table address is restored with the direct addressing instruction SHLD.

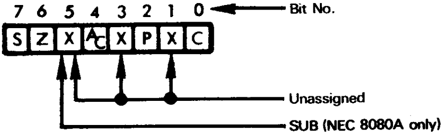
Since the I/O buffer starting address does not change, an Immediate instruction is used to load this address into the DE registers.

Since the number of occupied bytes in the I/O buffer may change, a direct addressing instruction, LDA, is used to load this buffer length into the Accumulator. It is then moved to the B register, since the Accumulator is used to transfer data within the program loop.

The 8080A program makes no assumptions regarding the location of either the I/O buffer, or the data table, but it does assume that the table is not more than 256 bytes long.

These are the abbreviations used in Table 4-4:

A	The Accumulator	
B	The B register	These are sometimes treated as a register pair
C	The C register	
D	The D register	These are sometimes treated as a register pair
E	The E register	
H	The H register	This register pair provides the implied memory address
L	The L register	
C	Carry status. In Table 4-4 C refers to Carry status, not to the C register.	
A_C	Auxiliary Carry status	
Z	Zero status	
S	Sign status	
P	Parity status	
SUB	Subtract status (present in the NEC 8080A only)	
I	The Instruction register	
I2	Second object code byte	
I3	Third object code byte	
PC	The Program Counter	
SP	The Stack Pointer	
PSW	The Program Status Word, which has bits assigned to status flags as follows:	



DATA	8-bit immediate data
DATA16	16-bit immediate data
DEV	An I/O device
REG	Register A, B, C, D, E, H or L
s	Source register
d	Destination register
M	Memory, address implied by HL
LABEL	A 16-bit address, specifying an instruction label
RP	A register pair: B for BC, D for DE, H for HL, SP for Stack Pointer
PORT	An I/O port, identified by a number between 0 and FF_{16}
ADDR	A 16-bit address, specifying a data memory byte
[ ]	Contents of location identified within brackets
[[ ]]	Memory byte addressed by location identified within brackets
$\overline{[\ ]}$	Complement of the contents of
←	Move data in direction of arrow
←→	Exchange contents of locations on either side of arrow
+	Add
—	Subtract

Λ	AND
V	OR
∀	XOR

The letter C is used to identify Carry status. Although C also identifies one of the 8080A registers, registers are always referenced generically in Table 4-4.

8080A CARRY STATUS NOMENCLATURE

Table 4-4. A Summary Of 8080A/9080A Microcomputer Instruction Set

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
I/O	IN	DEV	2							[A]←[DEV] Input to A from device DEV (DEV = 0 to 255)
	OUT	DEV	2							[DEV]←[A] Output from A to device DEV (DEV = 0 to 255)
PRIMARY MEMORY REFERENCE	LDAX	RP	1							[A]←[[RP]] Load A using address implied by BC (RP = B) or DE (RP = D)
	STAX	RP	1							[[RP]]←[A] Store A using implied addressing as for LDAX
	MOV	REG,M	1							[REG]←[[H,L]] Load any register using address implied by HL
	MOV	M,REG	1							[[H,L]]←[REG] Store any register using address implied by HL
	LDA	ADDR	3							[A]←[ADDR], i.e., [A]←[[I3, I2]] Load A, use direct addressing
	STA	ADDR	3							[ADDR]←[A], i.e., [[I3, I2]]←[A] Store A, use direct addressing
	LHLD	ADDR	3							[L]←[ADDR], [H]←[ADDR + 1], i.e., [L]←[[I3, I2]], [H]←[[I3, I2] + 1] Load H and L registers, use direct addressing
	SHLD	ADDR	3							[ADDR]←[L], [ADDR + 1]←[H] i.e., [[I3, I2]]←[L], [[I3, I2] + 1]←[H] Store H and L registers, use direct addressing

Table 4-4. A Summary Of 8080A/9080A Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	ADD	M	1	X	X	X	X	X	0	[A]←[A] + [[H,L]] Add to A
	ADC	M	1	X	X	X	X	X	0	[A]←[A] + [[H,L]] + [C] Add with Carry to A
	SUB	M	1	X	X	X	X	X	1	[A]←[A] - [[H,L]] Subtract from A
	SBB	M	1	X	X	X	X	X	1	[A]←[A] - [[H,L]] - [C] Subtract from A with borrow
	ANA	M	1	0**	X†**	X	X	X		[A]←[A] Λ [H,L]] AND with A
	XRA	M	1	0**	X†**	X	X	X		[A]←[A] ⊻ [[H,L]] Exclusive-OR with A
	ORA	M	1	0**	X†**	X	X	X		[A]←[A] V [[H,L]] OR with A
	CMP	M	1	X	X	X	X	X	1	Compare with A
	INR	M	1		X**	X	X	X	0	[[H,L]]←[[H,L]] + 1 Increment memory
	DCR	M	1		X**	X	X	X	1	[[H,L]]←[[H,L]]-1 Decrement memory
IMMEDIATE	LXI	RP,DATA16	3							[RP]←DATA16 Load 16-bit immediate data into BC (RP = B), DE (RP = D), HL (RP = H) or SP (RP = SP)
	MVI	M,DATA	2							[[H,L]]←DATA Load 8-bit immediate data into memory location with address implied by HL
	MVI	REG,DATA	2							[REG]←DATA Load 8-bit immediate data into any register

Table 4-4. A Summary Of 8080A/9080A Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
JUMP	JMP	ADDR	3							[PC]← ADDR Jump to instruction with label ADDR
	PCHL		1							[PC]←[H,L] Jump to instruction at location implied by HL
SUBROUTINE CALL AND RETURN (IMMEDIATE AND STACK)	CALL	ADDR	3							[[SP]]←[PC], [PC]← ADDR, [SP]←[SP]-2 Jump to subroutine starting at ADDR
	CC	ADDR	3							[[SP]]←[PC], [PC]← ADDR, [SP]←[SP]-2 Jump to subroutine if C = 1
	CNC	ADDR	3							[[SP]]←[PC], [PC]← ADDR, [SP]←[SP]-2 Jump to subroutine if C = 0
	CZ	ADDR	3							[[SP]]←[PC], [PC]← ADDR, [SP]←[SP]-2 Jump to subroutine if Z = 1
	CNZ	ADDR	3							[[SP]]←[PC], [PC]← ADDR, [SP]←[SP]-2 Jump to subroutine if Z = 0
	CP	ADDR	3							[[SP]]←[PC], [PC]← ADDR, [SP]←[SP]-2 Jump to subroutine if S = 0
	CM	ADDR	3							[[SP]]←[PC], [PC]← ADDR, [SP]←[SP]-2 Jump to subroutine if S = 1
	CPE	ADDR	3							[[SP]]←[PC], [PC]← ADDR, [SP]←[SP]-2 Jump to subroutine if even parity
	CPO	ADDR	3							[[SP]]←[PC], [PC]← ADDR, [SP]←[SP]-2 Jump to subroutine if odd parity
	RET		1							[PC]←[[SP]],[SP]←[SP]+2 Return from subroutine

Table 4-4. A Summary Of 8080A/9080A Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
SUBROUTINE CALL AND RETURN (IMMEDIATE AND STACK) (CONTINUED)	RC		1							[PC]←[[SP]], [SP]←[SP] + 2 Return from subroutine if C = 1
	RNC		1							[PC]←[[SP]], [SP]←[SP] + 2 Return from subroutine if C = 0
	RZ		1							[PC]←[[SP]], [SP]←[SP] + 2 Return from subroutine if Z = 1
	RNZ		1							[PC]←[[SP]], [SP]←[SP] + 2 Return from subroutine if Z = 0
	RM		1							[PC]←[[SP]], [SP]←[SP] + 2 Return from subroutine if S = 1
	RP		1							[PC]←[[SP]], [SP]←[SP] + 2 Return from subroutine if S = 0
	RPE		1							[PC]←[[SP]], [SP]←[SP] + 2 Return from subroutine if even parity
	RPO		1							[PC]←[[SP]], [SP]←[SP] + 2 Return from subroutine if odd parity
IMMEDIATE OPERATE	ADI	DATA	2	X	X	X	X	X	0	[A]←[A] + DATA Add immediate to A
	ACI	DATA	2	X	X	X	X	X	0	[A]←[A] + DATA + [C] Add with carry immediate to A
	SUI	DATA	2	X	X	X	X	X	1	[A]←[A]-DATA Subtract immediate from A
	SBI	DATA	2	X	X	X	X	X	1	[A]←[A]-DATA-[C] Subtract immediate with borrow from A
	ANI	DATA	2	0**	X†	X	X	X		[A]←[A] ∧ DATA AND immediate with A

Table 4-4. A Summary Of 8080A/9080A Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
IMMEDIATE OPERATE (CONTINUED)	XRI	DATA	2	0**	0**	X	X	X		[A]←[A]⊻DATA Exclusive-OR immediate with A
	ORI	DATA	2	0**	0**	X	X	X		[A]←[A] ∨ DATA OR immediate with A
	CPI	DATA	2	X	X	X	X	X		Compare immediate with A
JUMP ON CONDITION	JC	ADDR	3							[PC]← ADDR Jump if C = 1
	JNC	ADDR	3							[PC]← ADDR Jump if C = 0
	JZ	ADDR	3							[PC]← ADDR Jump if Z = 1
	JNZ	ADDR	3							[PC]← ADDR Jump if Z = 0
	JP	ADDR	3							[PC]← ADDR Jump if S = 0
	JM	ADDR	3							[PC]← ADDR Jump if S = 1
	JPE	ADDR	3							[PC]← ADDR Jump on even parity
	JPO	ADDR	3							[PC]← ADDR Jump on odd parity

Table 4-4. A Summary Of 8080A/9080A Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
REG-REG MOVE	MOV	ds	1							[REG]←[REG] Move any register (s) to any register (d)
	XCHG		1							[D]←→[H], [E]←→[L] Exchange DE with HL
	SPHL		1							[SP]←[HL] Transfer HL to SP
REGISTER-REGISTER OPERATE	ADD	REG	1	X	X	X	X	X	0	[A]←[A] + [REG] Add any register to A
	ADC	REG	1	X	X	X	X	X	0	[A]←[A] + [REG] + [C] Add with Carry any register to A
	SUB	REG	1	X	X	X	X	X	1	[A]←[A] - [REG] Subtract any register from A
	SBB	REG	1	X	X	X	X	X	1	[A]←[A] - [REG] - [C] Subtract any register with borrow from A
	ANA	REG	1	0**	X†	X	X	X		[A]←[A] Λ [REG] AND any register with A
	XRA	REG	1	0**	X†	X	X	X		[A]←[A] ⊻ [REG] Exclusive-OR any register with A
	ORA	REG	1	0**	X†	X	X	X		[A]←[A] V [REG] OR any register with A
	CMP	REG	1	X	X	X	X	X	1	Compare any register with A
	DAD	RP	1	X					0	[H,L]←[H,L] + [RP] Add to HL
REGISTER OPERATE	INR	REG	1		X**	X	X	X	0	[REG]←[REG] + 1 Increment any register
	DCR	REG	1		X**	X	X	X	1	[REG]←[REG] - 1 Decrement any register
	CMA		1							[A]←[$\overline{A}$] Complement A

Table 4-4. A Summary Of 8080A/9080A Microcomputer Instruction Set (Continued)

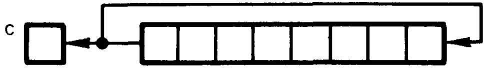
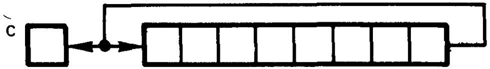
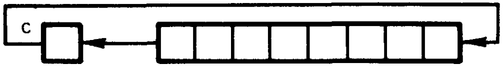
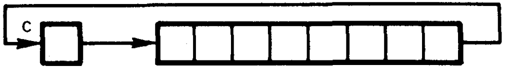
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
REGISTER OPERATE (CONTINUED)	DAA		1	X	X**	X	X	X		Decimal adjust A
	RLC		1	X						 Rotate A left with branch carry
	RRC		1	X						 Rotate A right with branch carry
	RAL		1	X						 Rotate A left with carry
	RAR		1	X						 Rotate A right with carry
	INX	RP	1							[RP]←[RP] + 1 Increment RP. RP = BC, DE, HL or SP
	DCX	RP	1							[RP]←[RP] - 1 Decrement RP
STACK	PUSH	RP	1							[[SP]]←[RP], [SP]←[SP] - 2 Push RP contents onto stack RP = BC, DE, HL or PSW
	POP	RP	1							[RP]←[[SP]], [SP]←[SP] + 2 Pop stack into RP RP = BC, DE, HL or PSW
	XTHL		1							[H,L]←→[[SP]] Exchange HL with top of stack

Table 4-4. A Summary Of 8080A/9080A Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
INTERRUPT	EI		1							Enable interrupts
	DI		1							Disable interrupts
	RST	N	1							Restart at addresses 8•N. N = 0 through 7.
STATUS	STC		1	1						[C]←1 Set Carry
	CMC		1	X						$[C] \leftarrow \overline{[C]}$ Complement Carry
	NOP		1							No operation
	HLT		1							Halt

Statuses: C = Carry
A_C = Carry out of bit 3
Z = Zero
S = Sign
P = Parity
X = Status set or reset
0 = Status reset
1 = Status Set
Blank = Status unchanged

* SUB status is present in NEC 8080A only

** NEC 8080A does not modify these status flags

† The AMD 9080A always resets A_C to 0 for all Boolean instructions

Table 4-5. A Summary Of Instruction Object Codes And Execution Cycles

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS	FIGURE
ACI	DATA	CE YY	2	7	4-15
ADC	REG	10001XXX	1	4	4-12
ADC	M	8E	1	7	4-15
ADD	REG	10000XXX	1	4	4-12
ADD	M	86	1	7	4-15
ADI	DATA	C6 YY	2	7	4-15
ANA	REG	10100XXX	1	4	4-12
ANA	M	A6	1	7	4-15
ANI	DATA	E6 YY	2	7	4-15
CALL	LABEL	CD ppqq	3	17	4-26
CC	LABEL	DC ppqq	3	11/17	4-26
CM	LABEL	FC ppqq	3	11/17	4-26
CMA		2F	1	4	4-12
CMC		3F	1	4	4-12
CMP	REG	10111XXX	1	4	4-12
CMP	M	BE	1	7	4-15
CNC	LABEL	D4 ppqq	3	11/17	4-26
CNZ	LABEL	C4 ppqq	3	11/17	4-26
CP	LABEL	F4 ppqq	3	11/17	4-26
CPE	LABEL	EC ppqq	3	11/17	4-26
CPI	DATA	FE YY	2	7	4-15
CPO	LABEL	E4 ppqq	3	11/17	4-26
CZ	LABEL	CC ppqq	3	11/17	4-26
DAA		27	1	4	4-12
DAD	RP	00XX1001	1	10(11)*	4-20
DCR	REG	00XXX101	1	5	4-13
DCR	M	35	1	10	4-14
DCX	RP	00XX1011	1	5	4-13
DI		F3	1	4	4-12
EI		FB	1	4	4-12
HLT		76	1	7	4-30
IN	PORT	DB YY	2	10	4-28
INR	REG	00XXX100	1	5	4-13
INR	M	34	1	10	4-14
INX	RP	00XX0011	1	5	4-13
JC	LABEL	DA ppqq	3	10	4-22
JM	LABEL	FA ppqq	3	10	4-22
JMP	LABEL	C3 ppqq	3	10	4-22
JNC	LABEL	D2 ppqq	3	10	4-22
JNZ	LABEL	C2 ppqq	3	10	4-22
JP	LABEL	F2 ppqq	3	10	4-22
JPE	LABEL	EA ppqq	3	10	4-22
JPO	LABEL	E2 ppqq	3	10	4-22
JZ	LABEL	CA ppqq	3	10	4-22
LDA	ADDR	3A ppqq	3	13	4-24
LDAX	RP	000X1010	1	7	4-15
LHLD	ADDR	2A ppqq	3	16	4-17

Table 4-5. A Summary Of Instruction Object Codes And Execution Cycles (Continued)

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS	FIGURE
LXI	RP,DATA16	00XX0001 YYYY	3	10	4-22
MOV	REG,REG	01dddsss	1	5(4)*	4-13
MOV	M,REG	01110sss	1	7	4-16
MOV	REG,M	01ddd110	1	7	4-15
MVI	REG,DATA	00ddd110 YY	2	7	4-15
MVI	M,DATA	36 YY	2	10	4-14
NOP		00	1	4	4-12
ORA	REG	10110XXX	1	4	4-12
ORA	M	B6	1	7	4-15
ORI	DATA	F6 YY	2	7	4-15
OUT	PORT	D3 YY	2	10	4-29
PCHL		E9	1	5	4-13
POP	RP	11XX0001	1	10	4-19
PUSH	RP	11XX0101	1	11	4-18
RAL		17	1	4	4-12
RAR		1F	1	4	4-12
RC		D8	1	5/11	4-27
RET		C9	1	10(11)*	4-19
RLC		07	1	4	4-12
RM		F8	1	5/11	4-27
RNC		D0	1	5/11	4-27
RNZ		C0	1	5/11	4-27
RP		F0	1	5/11	4-27
RPE		E8	1	5/11	4-27
RPO		E0	1	5/11	4-27
RRC		0F	1	4	4-12
RST	N	11XXX111	1	11	4-18
RZ		C8	1	5/11	4-27
SBB	REG	10011XXX	1	4	4-12
SBB	M	9E	1	7	4-15
SBI	DATA	DE YY	2	7	4-15
SHLD	ADDR	22 ppqq	3	16	4-25
SPHL		F9	1	5(4)*	4-13
STA	ADDR	32 ppqq	3	13	4-23
STAX	RP	000X0010	1	7	4-16
STC		37	1	4	4-12
SUB	REG	10010XXX	1	4	4-12
SUB	M	96	1	7	4-15
SUI	DATA	D6 YY	2	7	4-15
XCHG		EB		4	4-12
XRA	REG	10101XXX	1	4	4-12
XRA	M	AE	1	7	4-15
XRI	DATA	EE YY	2	7	4-15
XTHL		E3	1	18(17)*	4-21

ppqq	represents four hexadecimal digit memory address
YY	represents two hexadecimal data digits
YYYY	represents four hexadecimal data digits
X	represents an optional binary digit
ddd	represents optional binary digits identifying a destination register
sss	represents optional binary digits identifying a source register

* The NEC 8080A has five instructions with unique execution times, defined above by (N)* where N is the number of NEC 8080A instruction cycles.

INSTRUCTION EXECUTION TIMES AND CODES

Table 4-5 lists instructions in alphabetic order, showing object codes and execution times, expressed as machine cycles.

Where two instruction cycles are shown, the first is for "condition not met" whereas the second is for "condition met".

Detailed timing for instructions is provided by Figures 4-12 through 4-30. Table 4-5 identifies the timing diagram that applies to each instruction.

Instruction object codes are represented as two hexadecimal digits for instructions without variations.

Instruction object codes are represented as eight binary digits for instructions with variations; the binary digit representation of variations is then identifiable.

The second and third bytes of multibyte instruction codes are not shown. These bytes contain either data or addresses and their contents are self-evident.

The NEC 8080A has four instructions with execution times that differ from the Intel 8080A. These four instructions are the Register Move (MOV), the Return (RET), the 16-bit Add (DAD), and the Exchange instructions XTHL and SPHL.

NEC 8080A INSTRUCTION EXECUTION TIME DIFFERENCES

SUPPORT DEVICES THAT MAY BE USED WITH THE 8080A

Of the microprocessors described in this book none have a wider variety of support devices than the 8080A. These support devices are described in the rest of Chapter 4. Most of the devices described were originally developed by Intel, although a few were not. Note that the 8224 Clock Generator and the 8228 System Controller devices are used so routinely with the 8080A that they frequently are looked upon as a three-chip CPU. An exception is the TMS5501 made by Texas Instruments; it cannot be used with an 8228 System Controller.

In addition to the support devices described in this chapter, **the MILE bidirectional 8-bit port may be used with an 8080A.** The MILE is manufactured by National Semiconductor for use with the PACE and SC/MP microprocessors; **it is described in Chapter 14.**

A number of general purpose support devices are described in Chapter 21. These are support devices that may be used with any microprocessor and are specific to none.

One generalization that can be made regarding 8080A support devices is that the 8080A is so well endowed with support logic that it will rarely make much sense to use another microprocessor's support part in preference.

It is very difficult to use 6800 support devices with the 8080A because 6800 support devices require a synchronizing strobe signal which is difficult to generate within an 8080A system.

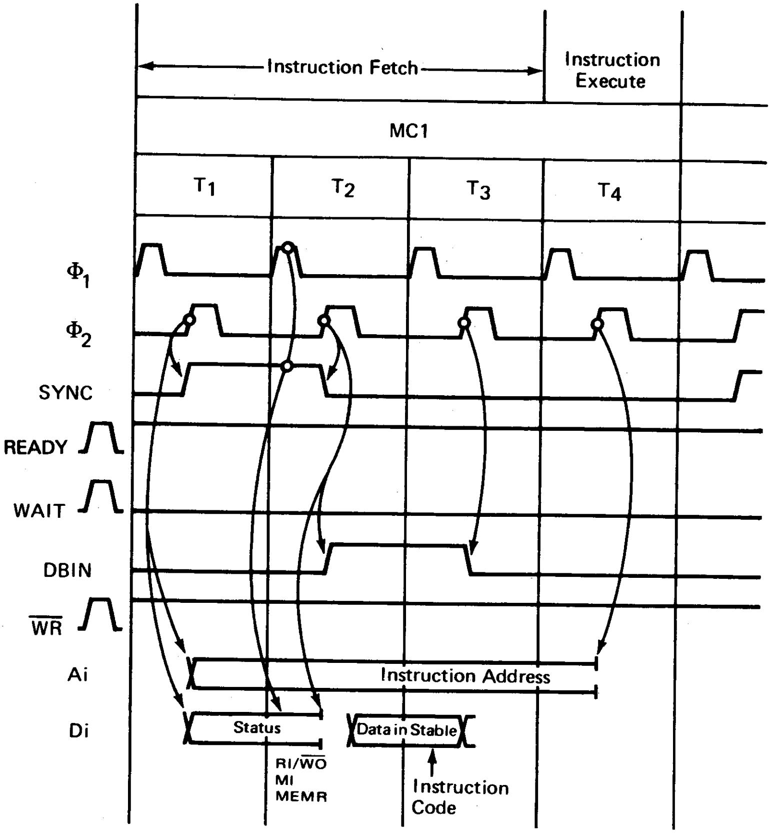


Figure 4-12. Signal Sequences And Timing For Instructions:
STC, CMC, CMA, NOP, RLC, RRC, RAL, RAR, XCHG, EI,
DI, DAA, ADD R, ADC R, SUB R, SBB R, ANA R, XRA R, ORA R, CMP R

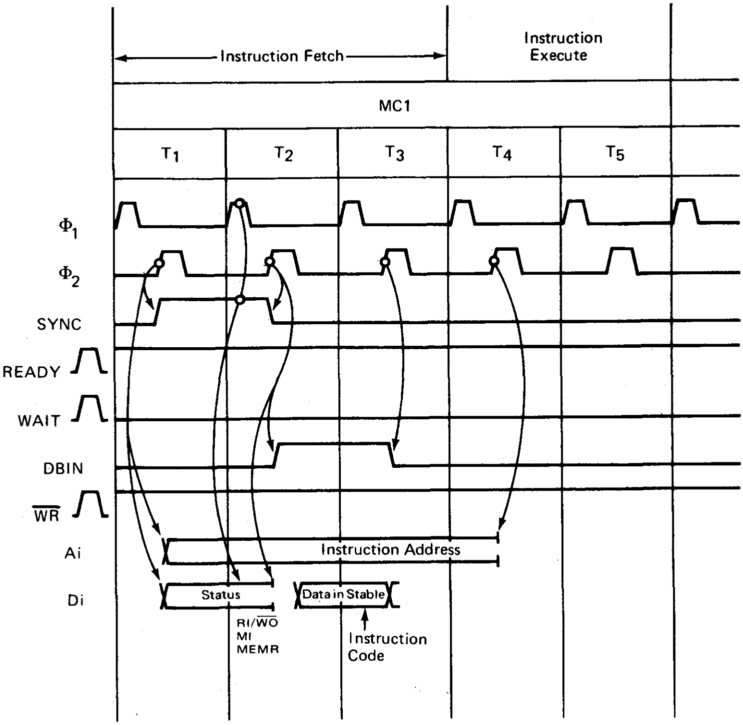


Figure 4-13. Signal Sequences And Timing For Instructions: INR, DCR, MOV REG REG, SPHL, PCHL, DCX, INX

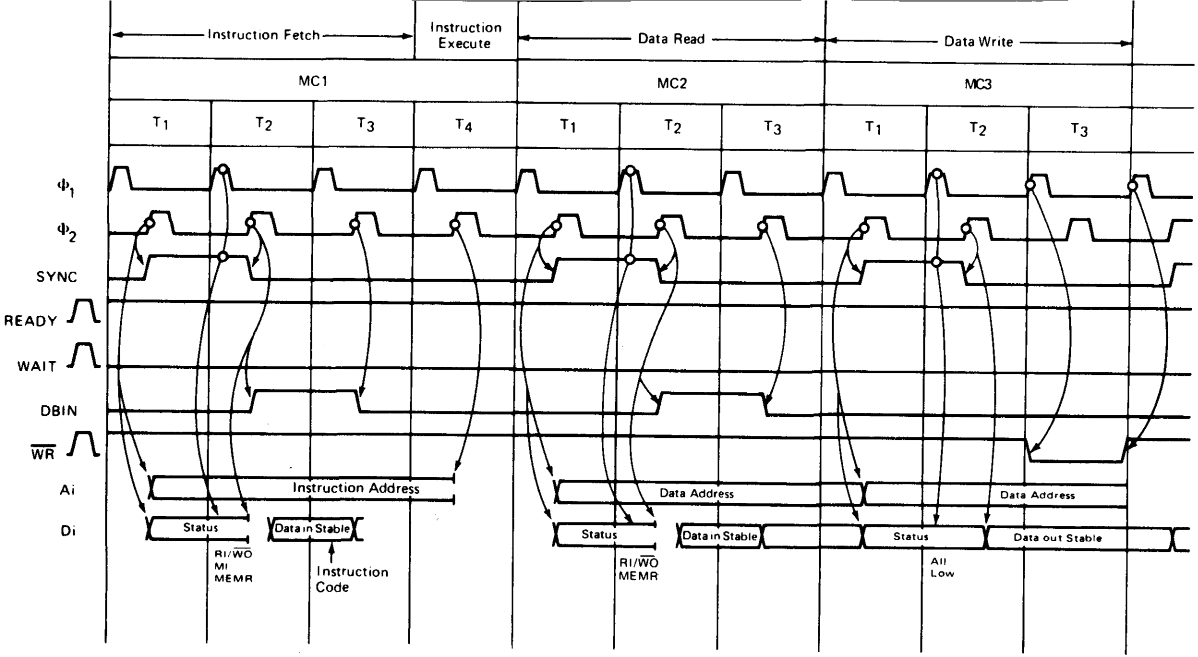


Figure 4-14. Signal Sequences And Timing For Instructions: DCR, INR, MVI M

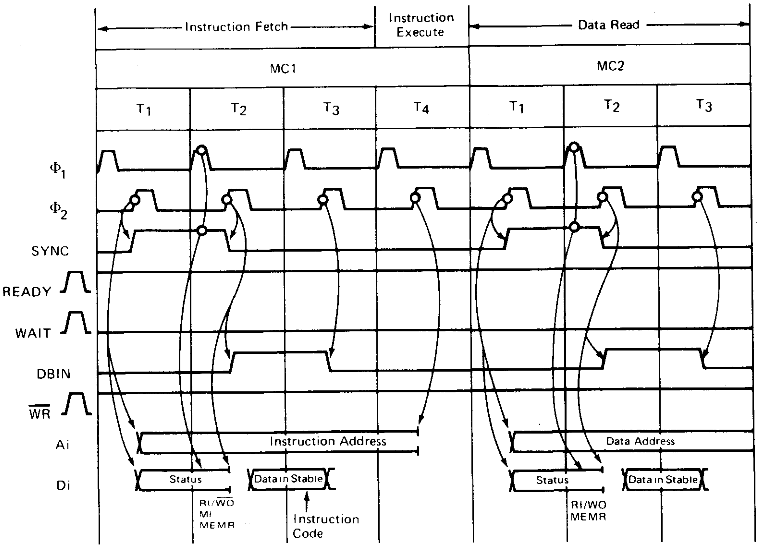


Figure 4-15. Signal Sequences And Timing For Instructions: LDAX, MOV REG M, ADI, ACI, SUI, SBI, ANI, XRI, ORI, CPI, MVI R, ADD M, ADC M, SUB M, SBB M, ANA M, XRA M, ORA M, CMP M

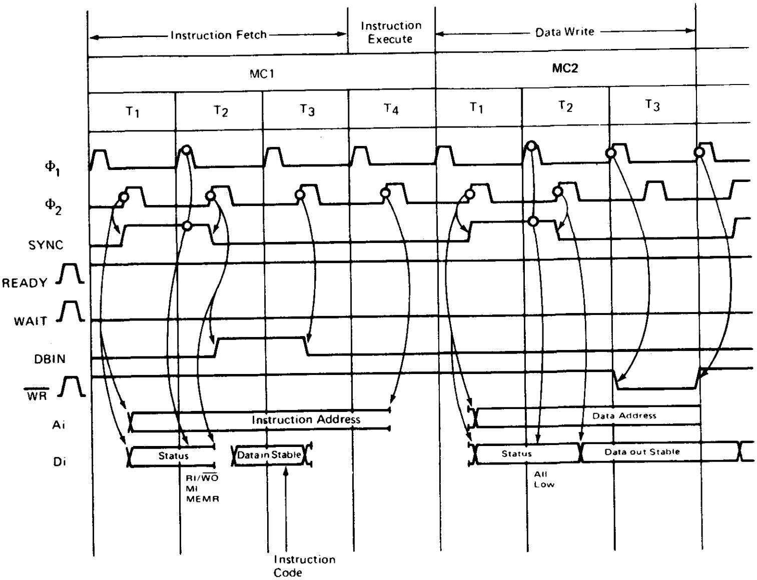


Figure 4-16. Signal Sequences And Timing For Instructions: STAX, MOV M REG

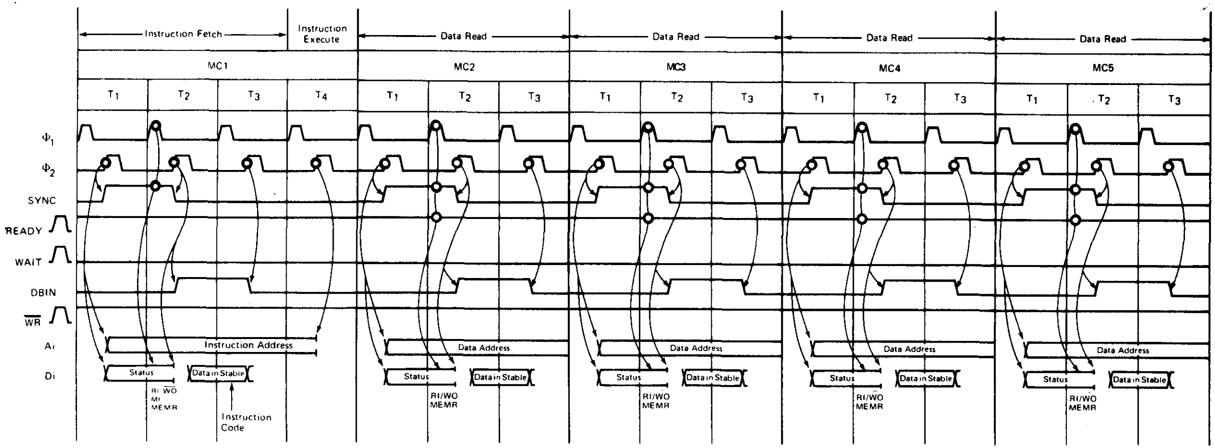


Figure 4-17. Signal Sequences And Timing For Instructions: LHLD

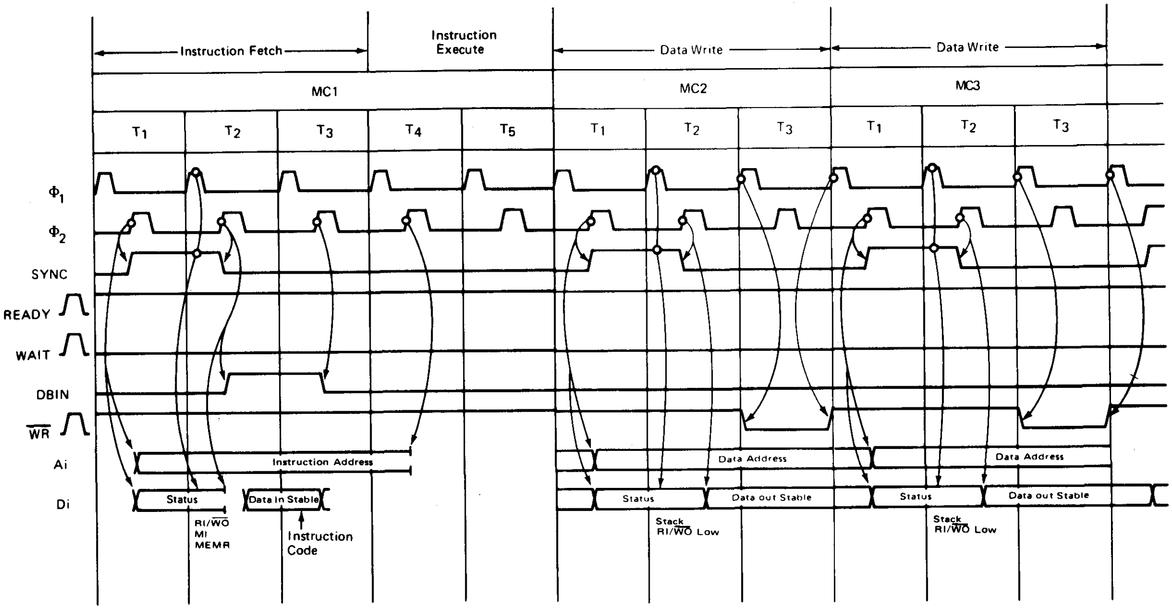


Figure 4-18. Signal Sequences And Timing For Instructions: PUSH, RST

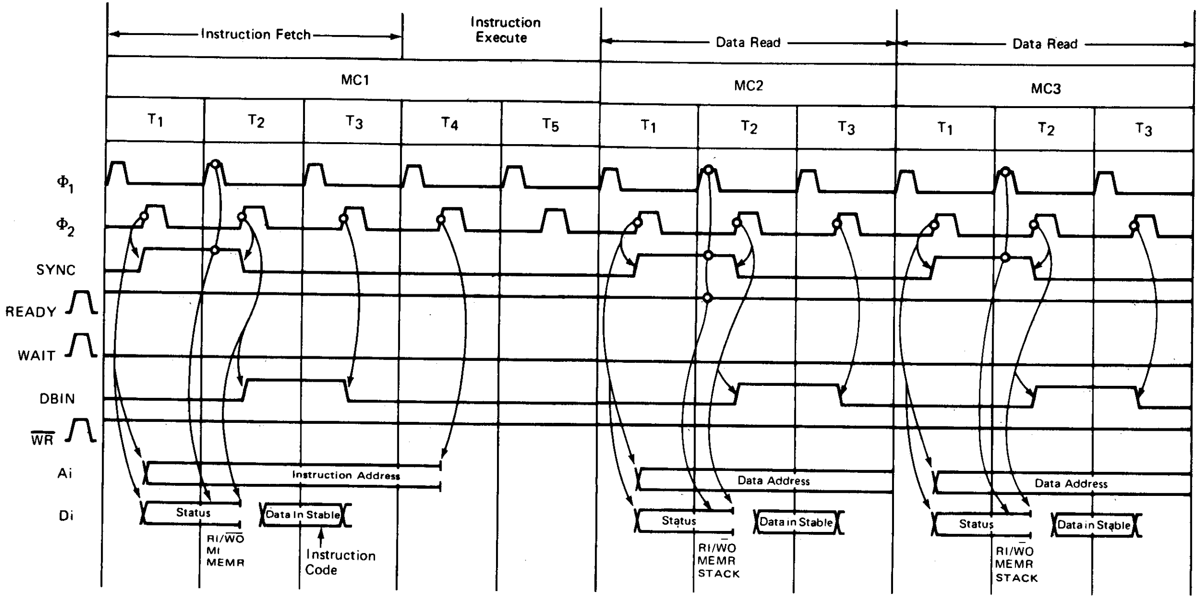


Figure 4-19. Signal Sequences And Timing For Instructions:
POP, RET

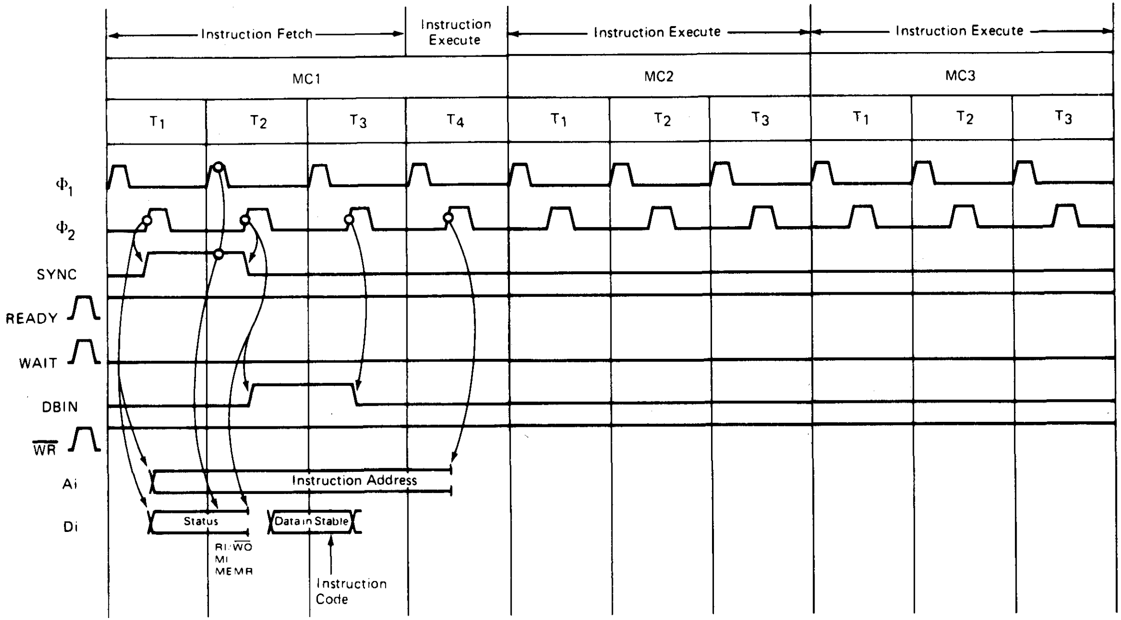


Figure 4-20. Signal Sequences And Timing For Instructions: DAD

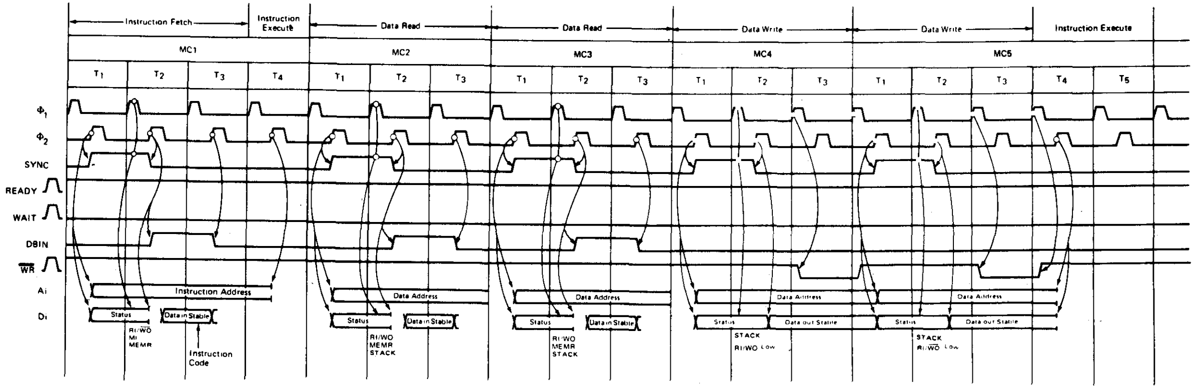


Figure 4-21. Signal Sequences And Timing For Instructions: XTHL

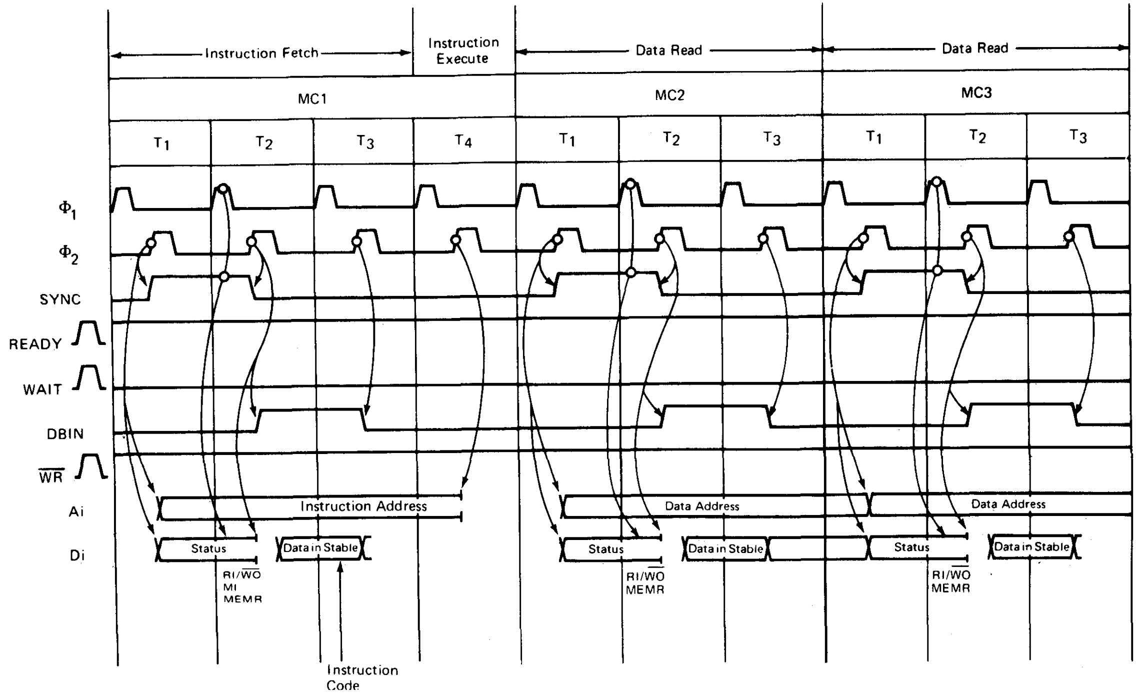


Figure 4-22. Signal Sequences And Timing For Instructions: LXI, JMP, JNZ, JZ, JNC, JC, JPO, JPE, JP, JM

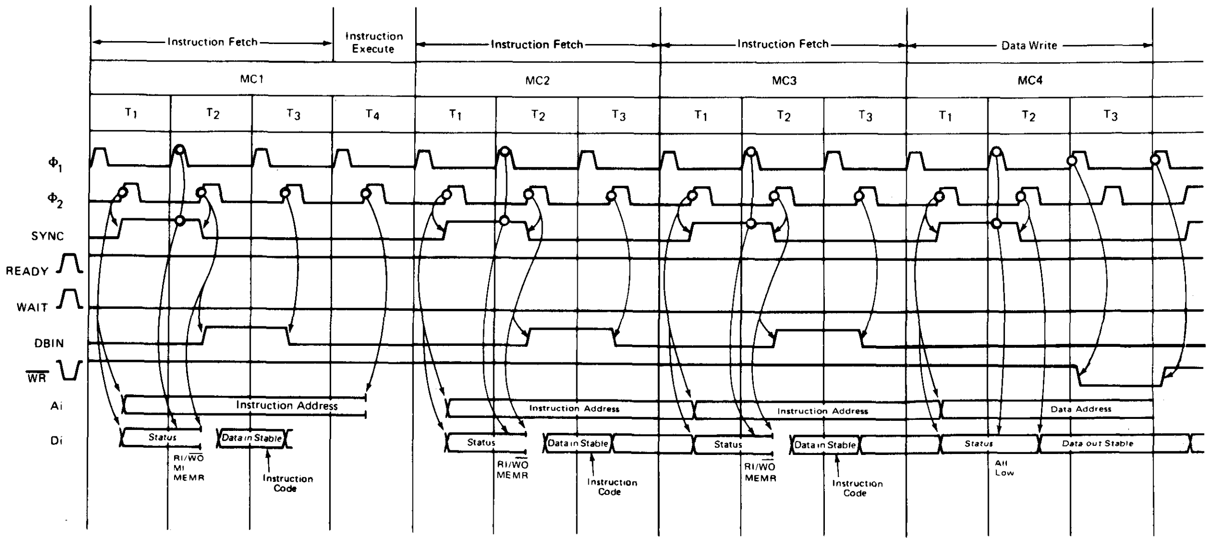


Figure 4-23. Signal Sequences And Timing For Instructions: STA

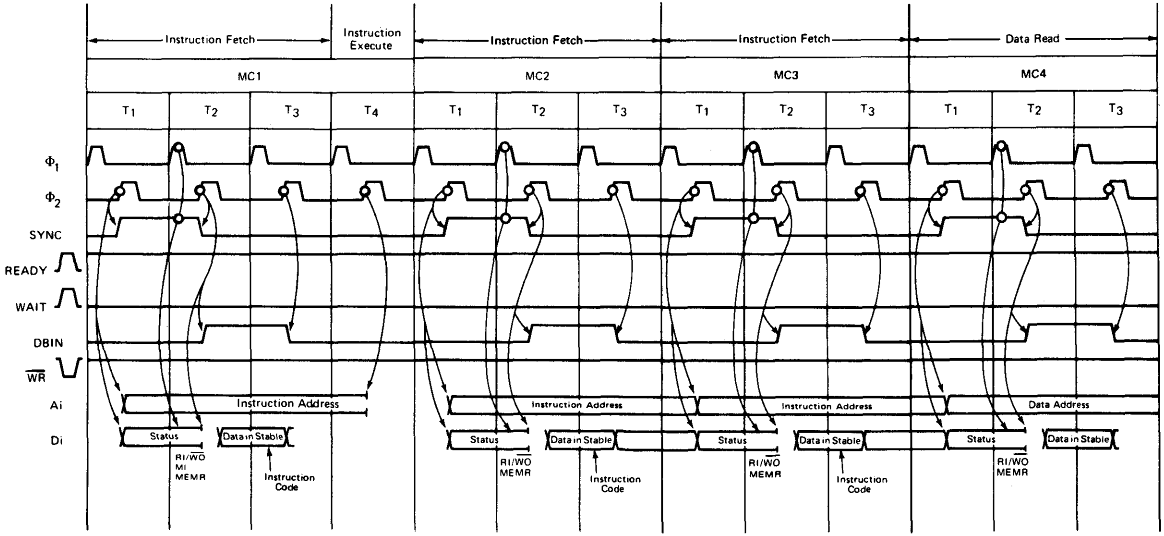


Figure 4-24. Signal Sequences And Timing For Instructions:
LDA

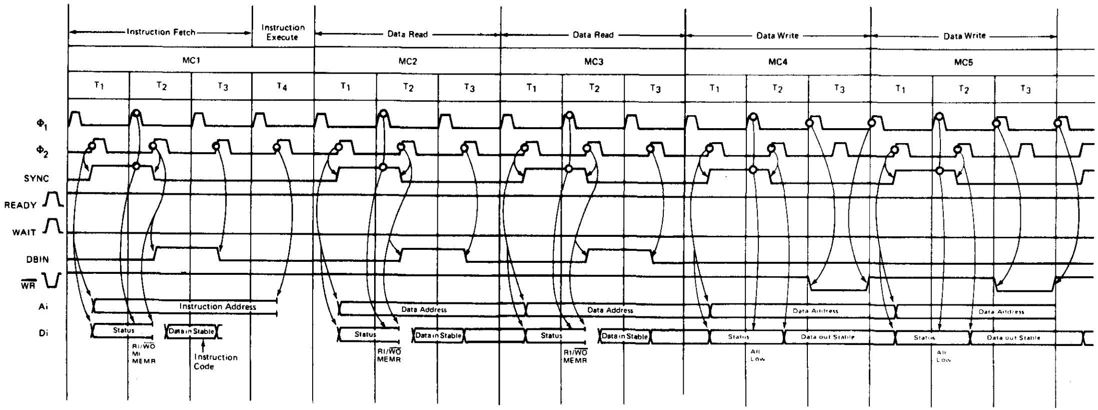


Figure 4-25. Signal Sequences And Timing For Instructions: SHLD

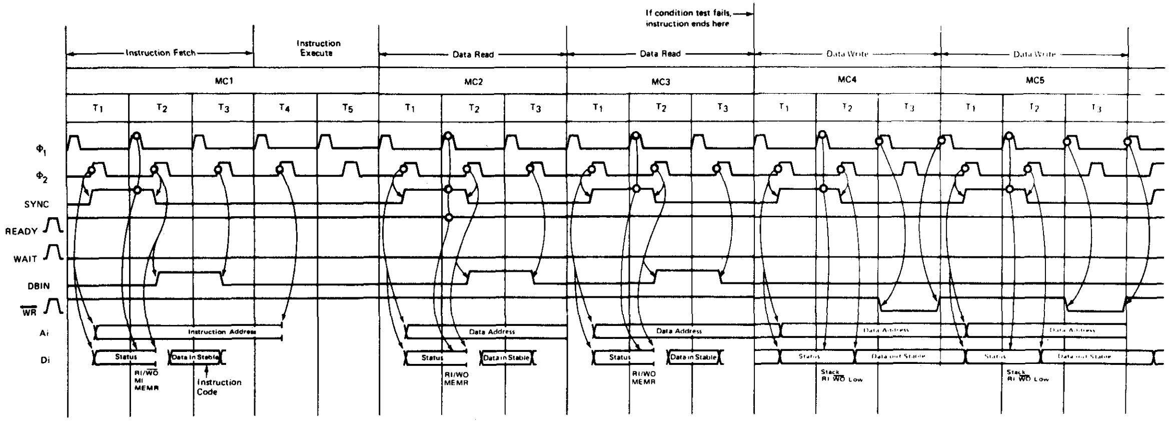


Figure 4-26. Signal Sequences And Timing For Instructions:
CALL, CNZ, CZ, CNC, CC, CPO, CPE, CP, CM

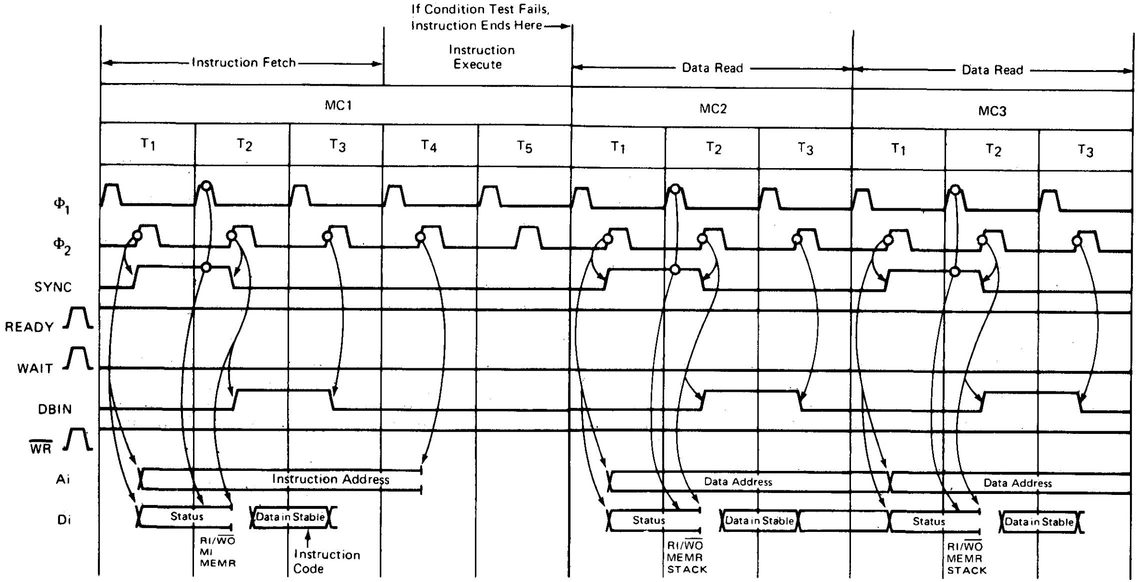


Figure 4-27. Signal Sequences And Timing For Instructions: RNZ, RZ, RNC, RC, RPO, RPE, RP, RM

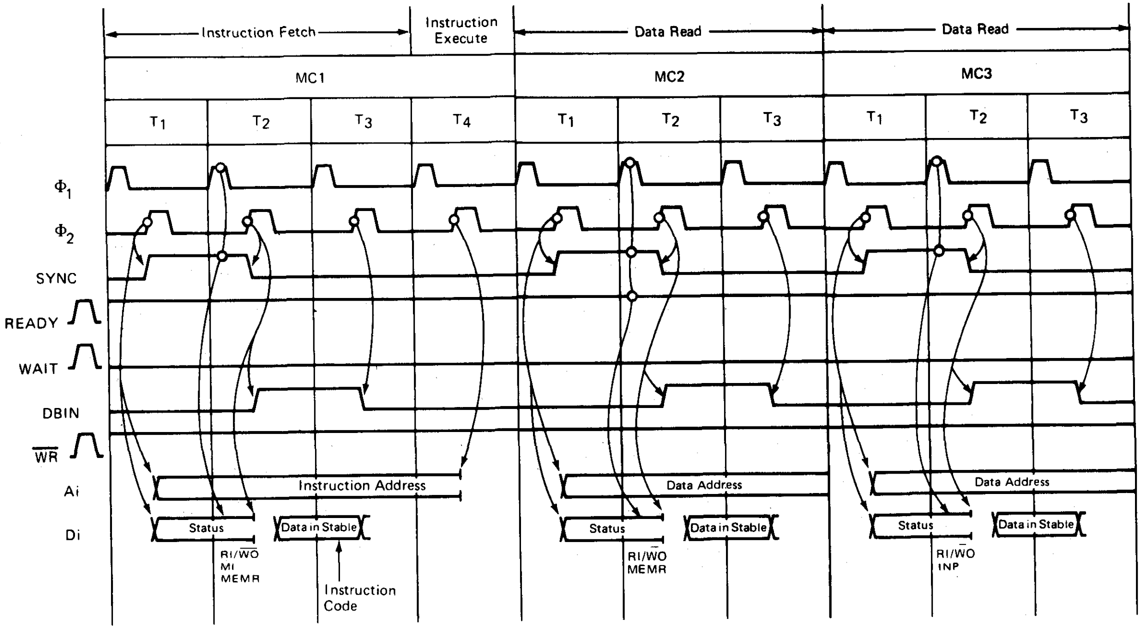


Figure 4-28. Signal Sequences And Timing For Instructions:
IN

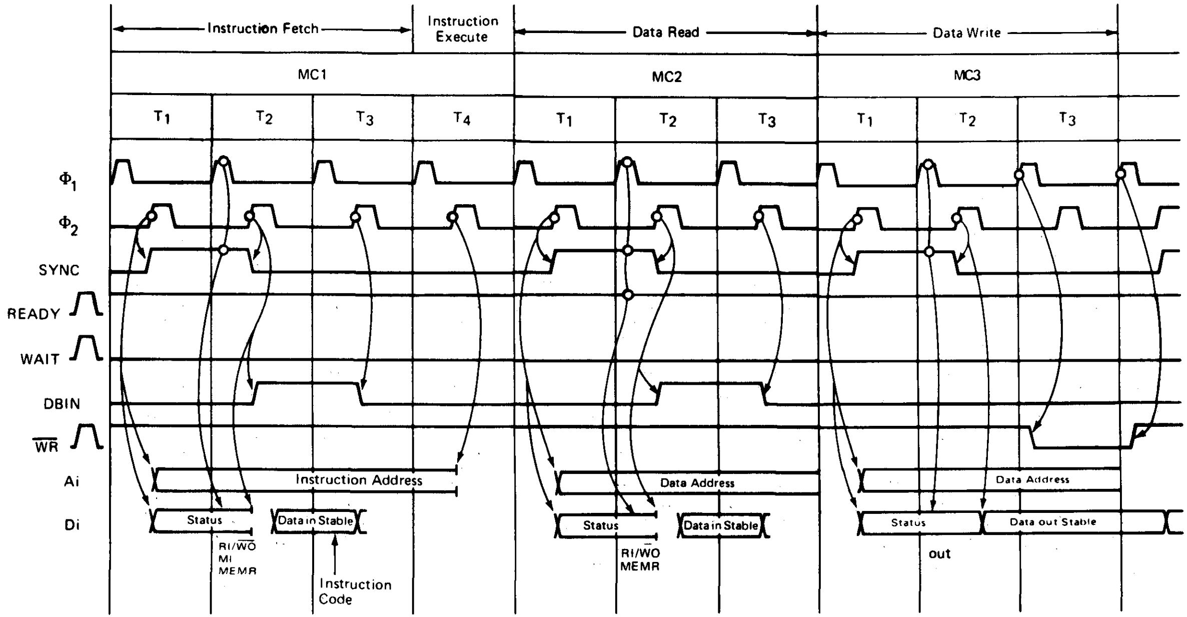


Figure 4-29. Signal Sequences And Timing For Instructions: OUT

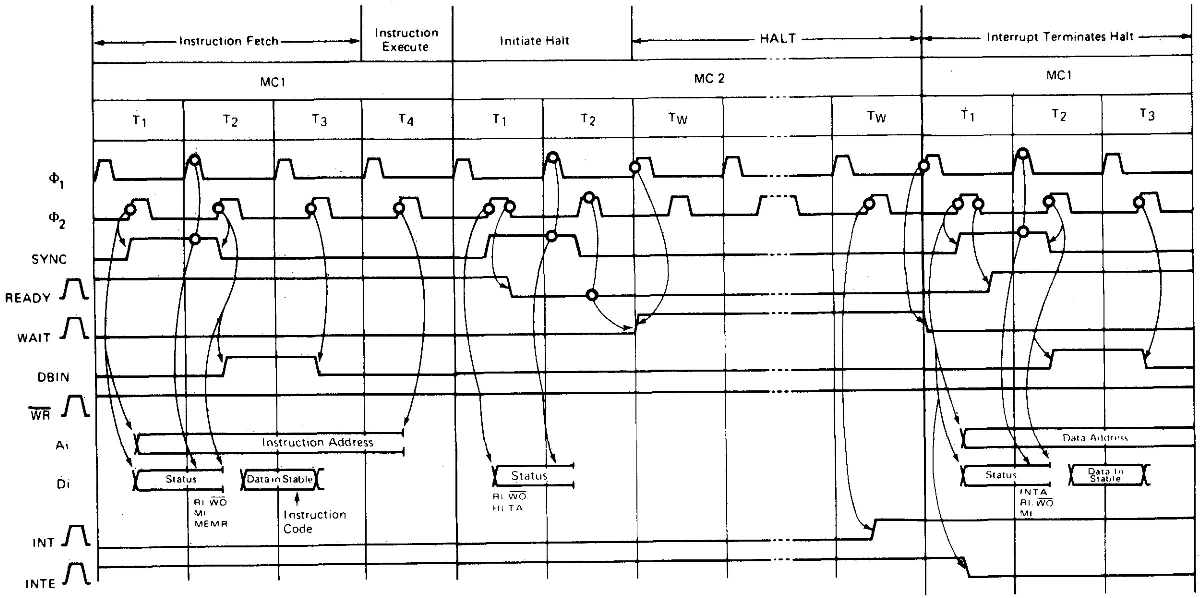


Figure 4-30. Signal Sequences And Timing For Instructions: HLT

THE 8224 CLOCK GENERATOR AND DRIVER

The primary purpose of this device is to provide the 8080A CPU with its required Φ1 and Φ2 clock signals. Coincidentally, the 8080A READY and RESET inputs are created, with correct synchronization. Recall that these two signals must be synchronized with Φ2.

Logic implemented on the 8224 Clock Generator corresponds generally to the block labeled "Clock Logic" in Figure 4-1. To be completely accurate, however, a small portion of the Bus Interface Logic should also be illustrated as provided by the 8224 device.

8224 CLOCK GENERATOR PINS AND SIGNALS

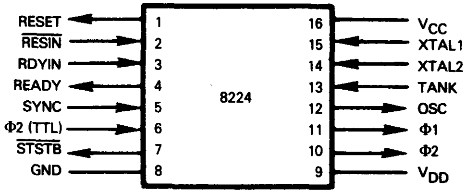
8224 pins and signals are illustrated in Figure 4-31. Figure 4-33 illustrates the 8224 connected to an 8080A CPU and an 8228 System Controller.

Signals may be divided between timing logic and control logic.

8224 CLOCK SIGNALS

Clock frequency is controlled by a crystal connected to the XTAL1 and XTAL2 pins. Crystal frequency must be exactly nine times the required clock period. The fastest clock period supported today is 250 nanoseconds, provided by the AMD 9080A. 500 nanosecond clock periods are standard. Since crystal frequency has to be nine times the clock period, the usual 500 nanosecond clock will require an 18 MHz frequency crystal.

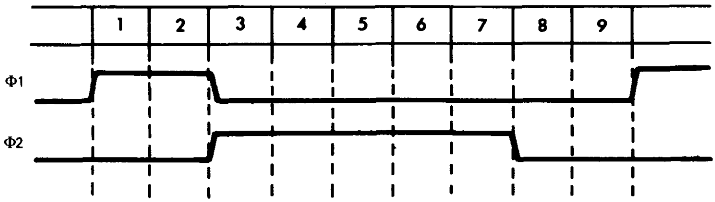
If an overtone mode crystal is employed, then it must be supported by an external LC network, connected to the TANK input. This is standard clock logic practice; microprocessor clock logic represents no special case, therefore overtone mode crystals are not discussed further.



PIN NAME	DESCRIPTION	TYPE
RESET	Control signal output to 8080A	Output
$\overline{\text{RESIN}}$	Reset logic input	Input
RDYIN	Ready logic input	Input
READY	Control signal output to 8080A	Output
SYNC	Control signal input from 8080A	Input
Φ2 (TTL)	TTL level duplicate of Φ2	Output
$\overline{\text{STSTB}}$	Sync signal output to 8228	Output
XTAL1,XTAL2	External crystal connections	Input
TANK	Overtone crystal extra input	Input
OSC	Crystal oscillator waveform	Output
Φ1, Φ2	Clock signals to 8080A	Output
V_{CC},V_{DD},GND	Power and Ground	

Figure 4-31. 8224 Clock Generator Signals And Pin Assignments

The principal clock signals output are Φ1 and Φ2, as required by the 8080A CPU. These two clock signals are derived from a divide-by-nine counter that defines Φ1 and Φ2 as follows:

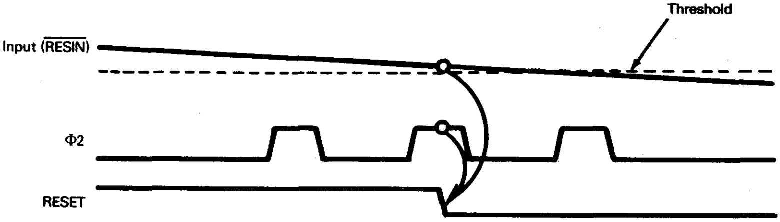


Two additional timing signals are output:

The crystal oscillator frequency is output as OSC.

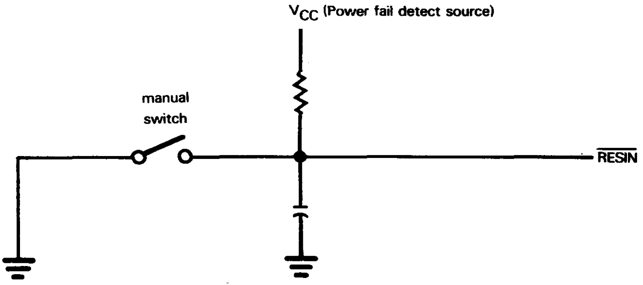
A TTL level duplicate of Φ2 is also output for general use within the microcomputer system.

The RESET input signal required by the 8080A CPU is usually generated by special external logic to provide sharp signal edges and synchronization with the Φ2 clock pulse. Consider one common use of RESET — to detect power failure. A vague input may have to be converted into a crisp RESET as follows:

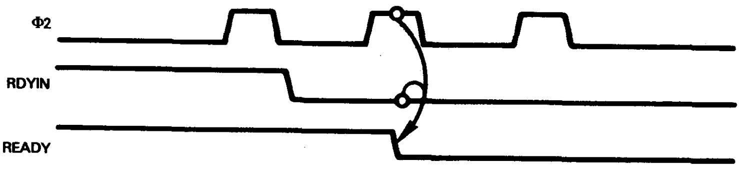


The 8224 Clock Generator will accept a sloppy input, as illustrated above by $\overline{\text{RESIN}}$, and in response will create a sharp RESET output that conforms to the requirements of the 8080A CPU. A Schmitt trigger within the logic of the 8224 clock chip creates the appropriate reset logic level change when $\overline{\text{RESIN}}$ falls below a threshold level.

RESET is also frequently connected to manually operated switches; this allows the microcomputer system to be reset by human intervention. The following simple circuit creates the appropriate $\overline{\text{RESIN}}$ input to the 8224 Clock Generator so that either power failure or an external switch may reset the CPU:


V_{CC} (Power fail detect source)
manual switch
$\overline{\text{RESIN}}$

READY logic accepts an asynchronous RDYIN signal and creates a synchronous READY input to the 8080A CPU:



One further signal created by the 8224 Clock Generator is the status strobe signal $\overline{\text{STSTB}}$, which is required by the 8228 System Controller. This signal is of very little interest to a user since it simply accepts an 8080A SYNC output and converts it into the required 8228 $\overline{\text{STSTB}}$ input.

When comparing the 8080A microcomputer system with other devices, it would be inaccurate to dismiss the 8224 Clock Generator simply as an additional device — which must be added to an 8080A system, supplying logic which is commonly found on competing CPU chips. Do not forget the reset logic capability provided by the 8224 Clock Generator.

It can be argued that the 8080A CPU creates an artificial restriction — that RESET and READY inputs must be synchronized with Φ2; therefore the fact that the 8224 does this for you, simply eliminates a self imposed problem that should never have been there in the first place. This reasoning has merit, but the ability of the 8224 to receive a ragged $\overline{\text{RESIN}}$ input is a valuable feature that should not be overlooked.

THE 8228 AND 8238 SYSTEM CONTROLLER AND BUS DRIVER

The 8228 System Controller consists of a bidirectional bus driver, plus control signal generation logic. The 8238 System Controller advances $\overline{\text{I/OW}}$ and $\overline{\text{MEMW}}$ to give large memories more time to respond to a memory write.

BUS DRIVER LOGIC

A large number of memory and I/O devices may be connected directly to the 8228 bidirectional Data Bus; such connections to the 8080A Data Bus would not be feasible. Remember, memory devices leak current even when they are not selected; therefore, even the passive load of unselected memory devices connected directly to an 8080A CPU will leak more current than is available.

When comparing the 8080A microcomputer system with an alternate microcomputer system, you should look carefully at the fan out provided by the alternate CPU.

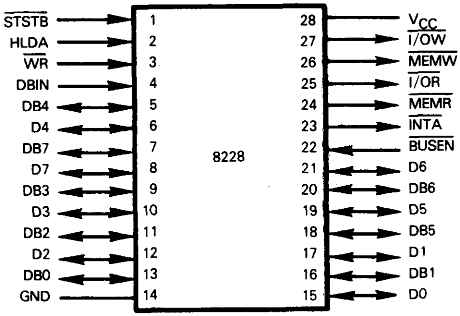
If the alternate CPU busses need to be buffered, then the 8228 System Controller becomes the equivalent 8080A system device; as such it does not represent an economic liability.

If the alternate CPU busses do not need to be buffered, then the 8228 System Controller represents an additional device, peculiar to the 8080A system.

CONTROL SIGNAL LOGIC

The 8228 combines the three 8080A control signals: $\overline{\text{WR}}$, DBIN and HLDA, with the statuses output on the Data Bus during T_2 in order to generate bus control signals as follows:

MEMR status on D7 true, with DBIN true generates $\overline{\text{MEMR}}$ true
OUT status on D4 false, with WR true generates $\overline{\text{MEMW}}$ true
INP status on D6 true, with DBIN true generates $\overline{\text{I/OR}}$ true
OUT status on D4 true, with $\overline{\text{WR}}$ true generates $\overline{\text{I/OW}}$ true
INTA status on D0 true generates $\overline{\text{INTA}}$ true



8228
$\overline{\text{STSTB}}$ 1
HLDA 2
$\overline{\text{WR}}$ 3
DBIN 4
DB4 5
D4 6
DB7 7
D7 8
DB3 9
D3 10
DB2 11
D2 12
DB0 13
GND 14
28 V_{CC}
27 $\overline{\text{I/OW}}$
26 $\overline{\text{MEMW}}$
25 $\overline{\text{I/OR}}$
24 $\overline{\text{MEMR}}$
23 $\overline{\text{INTA}}$
22 $\overline{\text{BUSEN}}$
21 D6
20 DB6
19 D5
18 DB5
17 D1
16 DB1
15 D0

PIN NAME	DESCRIPTION	TYPE
D0 - D7	Data Bus connection to CPU	Bidirectional
DB0 - DB7	Data Bus to external logic	Bidirectional
$\overline{\text{STSTB}}$	Status strobe input from 8224	Input
HLDA	Hold acknowledge input from CPU	Input
$\overline{\text{WR}}$	Data output strobe, input from CPU	Input
DBIN	Data input strobe, input from CPU	Input
I/OW	I/O write control output	Output
MEMW	Memory write control output	Output
$\overline{\text{I/OR}}$	I/O read control output	Output
$\overline{\text{MEMR}}$	Memory read control output	Output
$\overline{\text{INTA}}$	Interrupt acknowledge control	Output
$\overline{\text{BUSEN}}$	DB Bus float/enable control input	Input
V_{CC}, GND	Power and Ground	

Figure 4-32. 8228 System Controller Signals And Pin Assignments

8228 SYSTEM CONTROLLER PINS AND SIGNALS

8228 pins and signals are illustrated in Figure 4-32.

D0 through D7 represent the bidirectional Data Bus connection between the 8228 System Controller and the 8080A CPU; it is referred to as the "Processor Data Bus".

DB0 through DB7 represent the high fan out, bidirectional Data Bus accessed by external logic; it is referred to as the "System Data Bus".

$\overline{\text{WR}}$, DBIN and HLDA represent the control signals of the same name that are output by the 8080A CPU

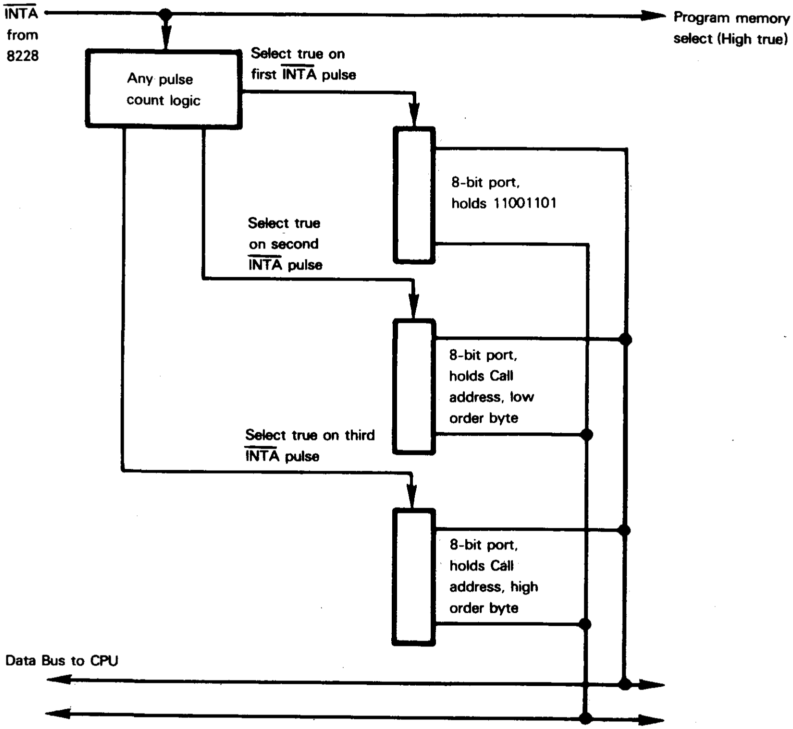
All control bus signals use active low logic and may be defined as follows:

$\overline{\text{MEMR}}$ — a read from memory strobe
$\overline{\text{MEMW}}$ — a write to memory strobe
$\overline{\text{I/OR}}$ — a read from external I/O strobe
$\overline{\text{I/OW}}$ — a write to external I/O strobe
$\overline{\text{INTA}}$ — interrupt acknowledge

Control signal timing is given in Figure 4-34.

The interrupt acknowledge signal $\overline{\text{INTA}}$ has two special features which need to be explained. This signal may be tied to a +12 Volt power supply through a 1K Ohm resistor, in which case 8228 logic assumes that there is only one possible interrupting source within the microcomputer system. **Now the 8228 will automatically insert the object code for an RST 7 instruction in response to the interrupt acknowledge.** This means that external logic does not need to supply the first post-interrupt instruction's object code. Of course, this means that all interrupt service routines effectively begin with the execution of an RST 7 instruction.

If external logic responds to the $\overline{\text{INTA}}$ low pulse by supplying the first byte of a CALL instruction's object code (11001101), then the 8228 System Controller will automatically generate two more $\overline{\text{INTA}}$ low pulses for the next two machine cycles. See Figure 4-34 for $\overline{\text{INTA}}$ pulse timing within the machine cycle. Now external logic can use the $\overline{\text{INTA}}$ pulse as a memory deselect and an interrupt acknowledge logic select. Here is a very general illustration of external logic that responds to an interrupt acknowledge by supplying the CPU with a three-byte CALL instruction's object code:



$\overline{\text{INTA}}$ from 8228
Program memory select (High true)
Any pulse count logic
Select true on first $\overline{\text{INTA}}$ pulse
8-bit port, holds 11001101
Select true on second $\overline{\text{INTA}}$ pulse
8-bit port, holds Call address, low order byte
Select true on third $\overline{\text{INTA}}$ pulse
8-bit port, holds Call address, high order byte
Data Bus to CPU

Recall that the NEC 8080A generates three $\overline{\text{INTA}}$ low output pulses in response to a Call instruction object code being returned during the interrupt acknowledge process. But the NEC 8228 System Controller does not assume that these three low $\overline{\text{INTA}}$ pulses will occur. Thus **the NEC 8228 System Controller may be used with an NEC 8080A or any other 8080A.** In every case the NEC 8228 will generate three low $\overline{\text{INTA}}$ output pulses when external logic responds to an interrupt acknowledge by providing a Call instruction object code.

The status strobe $\overline{\text{STSTB}}$ which is output by the 8224 Clock Generator is a variation of the SYNC output from the 8080A CPU. $\overline{\text{STSTB}}$ synchronizes the 8228 System Controller and is of no other concern to an 8080A user.

$\overline{\text{BUSEN}}$ is an external input to the 8228 System Controller. This is a very useful signal because it allows external logic to float the Data Bus. **When this signal is input low, the bidirectional bus driver logic of the 8228 System Controller presents a high impedance to the external Data Bus, thus allowing external logic to gain access to this bus.** We will have more to say about $\overline{\text{BUSEN}}$ later in this chapter when describing the 8257 Direct Memory Access control device.

Figure 4-33 illustrates the way in which the 8080A CPU normally combines with the 8224 Clock Generator and the 8228 System Controller. These three devices are frequently looked upon as a single entity.

THE 8251 PROGRAMMABLE COMMUNICATION INTERFACE

8251 USART

This is a typical serial I/O device, referred to generically as a Universal Synchronous/Asynchronous Receiver/Transmitter (USART). The 8251 is very similar to the serial I/O device which was described conceptually in Volume I, Chapter 5. The description that follows therefore assumes you have read Volume I, or otherwise understand serial I/O device concepts.

Figure 4-35 illustrates that part of our general microcomputer system logic which has been implemented on the 8251 USART.

Table 4-6 summarizes the performance options available to you when using the 8251 USART in a microcomputer system. The 8251 USART is packaged as a 28-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL compatible.

Of the 8080A support devices described in this chapter the 8251 is the most difficult to use. Unless you carefully understand logic associated with every signal and function — and write appropriate 8251 control programs — the device will malfunction. Some problems associated with using the 8251 USART have been removed by AMD in their 9551 upward replacement device. If you are designing new applications, we recommend you use the 9551 USART in preference to the 8251 USART.

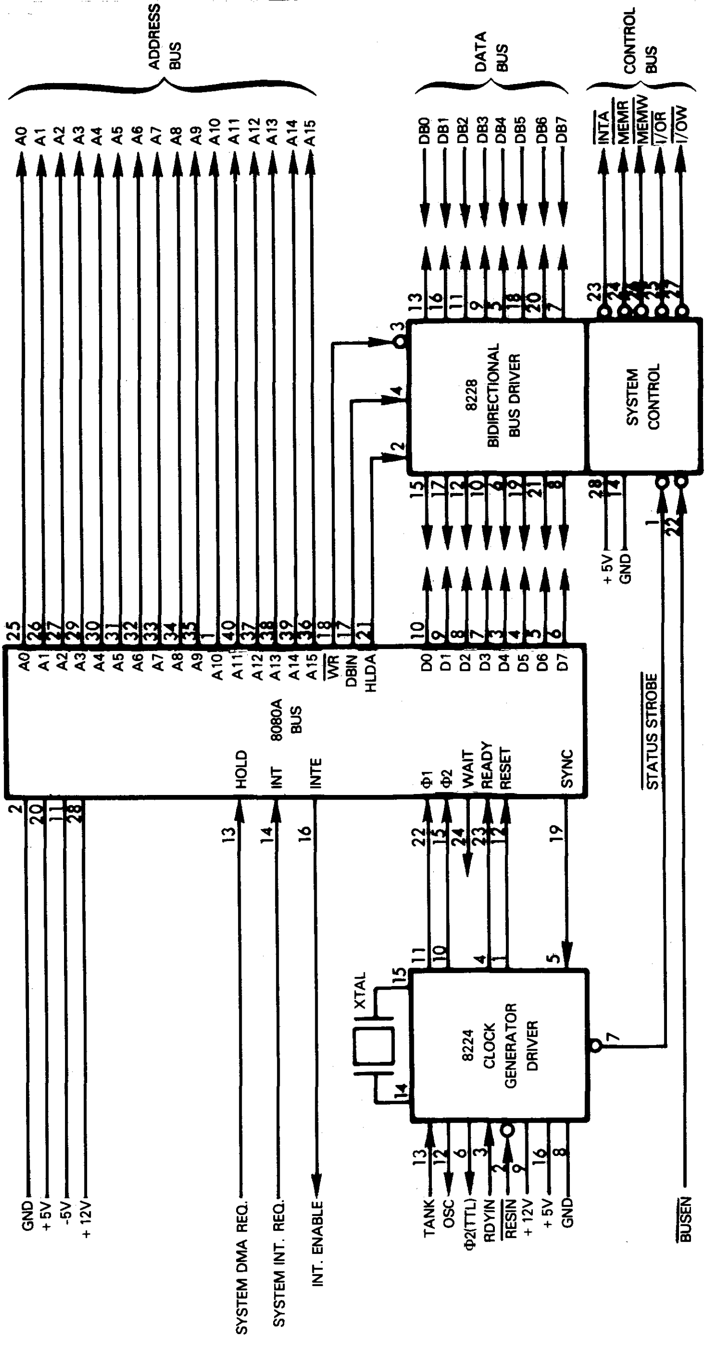


Figure 4-33. A Standard, Three Device 8080A Microcomputer System

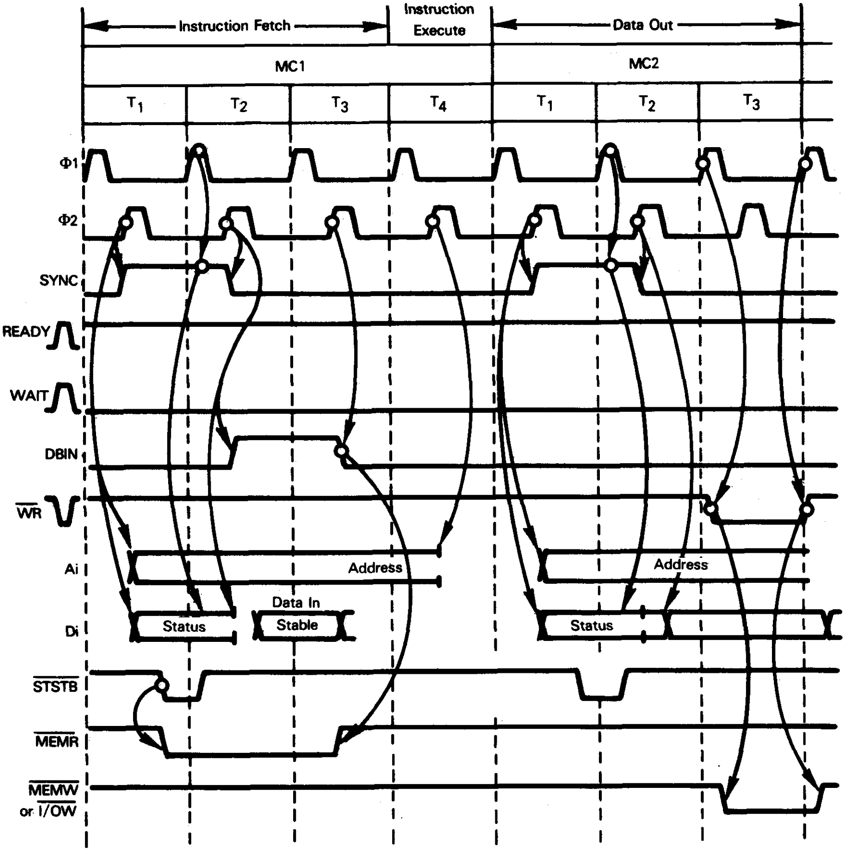


Figure 4-34. Timing For Control Signals Output By The 8228 System Controller

Table 4-6. A Summary Of 8251 USART Options

FEATURE	SYNCHRONOUS OPTIONS	ASYNCHRONOUS OPTIONS
Bits per character	5, 6, 7 or 8	5, 6, 7 or 8
Baud Rate	DC to 56K Baud	DC to 615/16K Baud
Errors detected	Parity, Overrun	Parity, Overrun, Framing
Synchronization	Internal or External	Does not apply
Clock Rate**	Baud Rate	1*, 16 or 64 times Baud Rate
Stop Bits	Does not apply	1, 1½ or 2

*x1 Asynchronous is a special case sometimes referred to as "isosynchronous".

**The clock rate cannot exceed the microcomputer system clock frequency.

8251 USART PINS AND SIGNALS

8251 USART pins and signals are illustrated in Figure 4-36. Signals may be divided into the following four categories:

1) **CPU Interface and Control**
2) **Serial Input**
3) **Serial Output**
4) **Modem Control**

We will first consider CPU Interface and Control signals.

8251 USART CPU INTERFACE

D0 - D7 constitutes an 8-bit, bidirectional Data Bus. When data is output to the 8251 USART from the CPU, either a byte of parallel data, or a Control Code may be transmitted. The 8251 USART will convert data bytes into a serial data stream and output the serial data stream to external logic. Control codes define the serial data protocol that will be used by the 8251 USART.

Either data or status may be input from the 8251 USART to the CPU via the Data Bus. Data consists of an 8-bit, parallel data unit extracted from the serial input data stream. Status consists of the contents of a USART Status register. A program executed by the CPU can examine individual bits from the Status register and conditionally execute different logic sequences depending on statuses indicated.

8251 USART ADDRESSING

The 8251 USART can be accessed by the CPU either as two I/O ports, or as two memory locations. 8251 select logic consists of a master chip select input $\overline{\text{CS}}$, and a Control/Data select C/$\overline{\text{D}}$. So long as $\overline{\text{CS}}$ is low, the USART will be connected to the Data Bus. $\overline{\text{CS}}$ must therefore be derived from Address Bus lines so that two I/O port addresses or two memory addresses cause $\overline{\text{CS}}$ to be generated low; one of the two select addresses must set C/$\overline{\text{D}}$ low while the other select address sets C/$\overline{\text{D}}$ high.

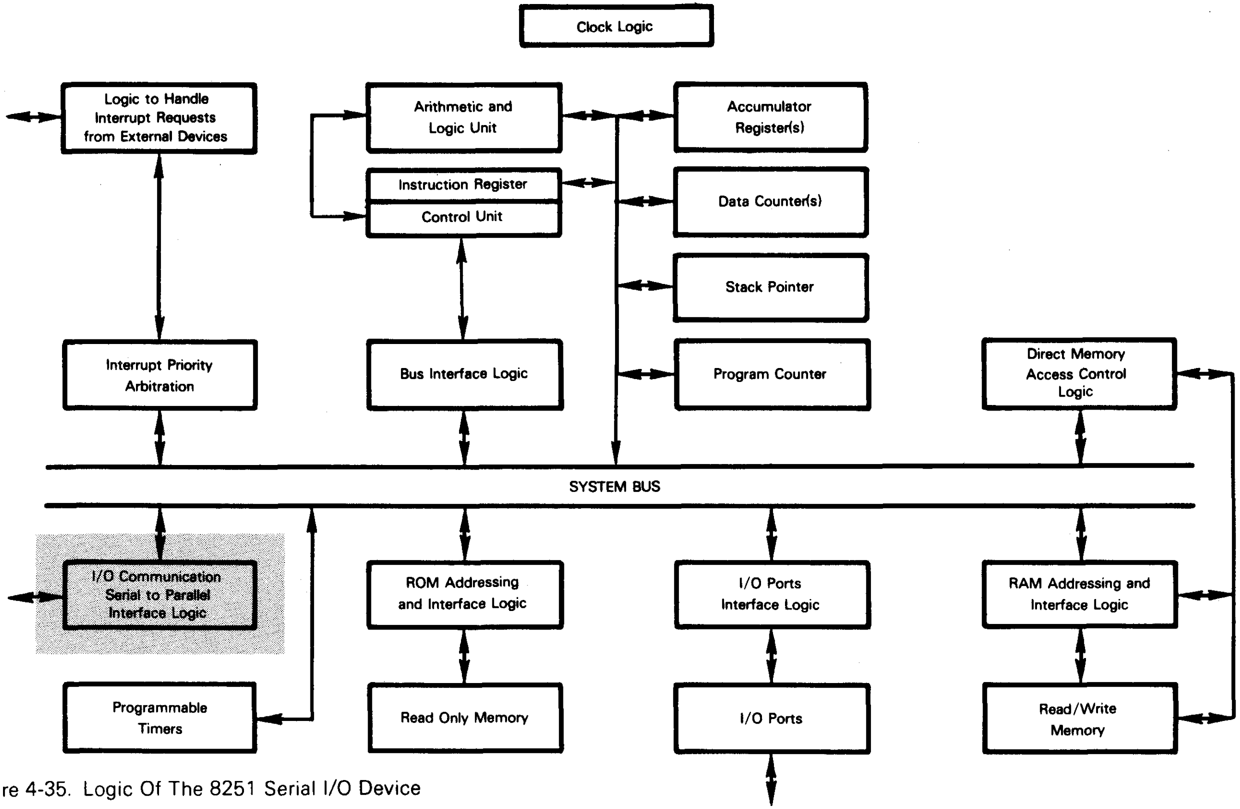
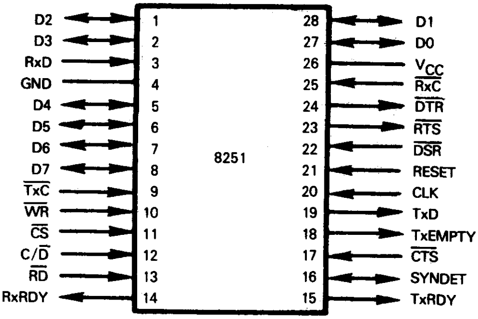

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Instruction Register
Control Unit
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

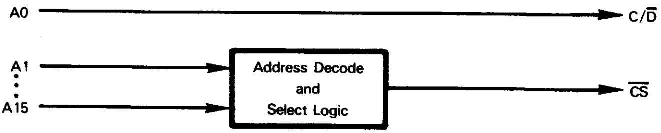
Figure 4-35. Logic Of The 8251 Serial I/O Device


D2 1
D3 2
RxD 3
GND 4
D4 5
D5 6
D6 7
D7 8
$\overline{\text{TxC}}$ 9
$\overline{\text{WR}}$ 10
$\overline{\text{CS}}$ 11
C/$\overline{\text{D}}$ 12
$\overline{\text{RD}}$ 13
RxRDY 14
8251
28 D1
27 D0
26 V_{CC}
25 $\overline{\text{RxC}}$
24 $\overline{\text{DTR}}$
23 $\overline{\text{RTS}}$
22 $\overline{\text{DSR}}$
21 RESET
20 CLK
19 TxD
18 TxEMPTY
17 $\overline{\text{CTS}}$
16 SYNDET
15 TxRDY

PIN NAME	DESCRIPTION	TYPE
D0 - D7	Data Bus	Bidirectional
RESET	System reset	Input
CLK	Device Clock	Input
C/$\overline{\text{D}}$	Control or Data select	Input
$\overline{\text{RD}}$	Read data or status on Data Bus	Input
$\overline{\text{WR}}$	Write Data or Control on Data Bus	Input
$\overline{\text{CS}}$	Chip Select	Input
$\overline{\text{DSR}}$	Data set ready indicator	Input
$\overline{\text{DTR}}$	Data terminal ready indicator	Output
$\overline{\text{CTS}}$	Clear to send data indicator	Input
$\overline{\text{RTS}}$	Request to send data indicator	Output
TxD	Serial data output	Output
TxRDY	Transmitter ready to receive data on Data Bus	Output
TxEMPTY	Transmitter empty. Has no data to transmit	Output
$\overline{\text{TxC}}$	Serial transmit data clock	Input
RxD	Serial data input	Input
RxRDY	A character is ready to transmit on Data Bus	Output
$\overline{\text{RxC}}$	Serial data input clock	Input
SYNDET	Synchronous data detect or force	Bidirectional
V_{CC}, GND	Power and Ground	

Figure 4-36. 8251 Programmable Serial Communications Interface Signals And Pin Assignments

If the 8251 is being selected via two memory addresses, select logic may be illustrated in this elementary form:



Address decode and select logic illustrated above generates a single low $\overline{\text{CS}}$ signal by decoding the 15 address lines A1 - A15. There is, of course, no reason why 15 address lines must be decoded; and in many small systems, it may be possible to generate $\overline{\text{CS}}$ out of a single address line — or perhaps two address lines. The one thing that you must insure is that the address decode logic which selects $\overline{\text{CS}}$ low does not also select any memory location. So long as the 8251 address space does not overlap memory address spaces, no problems will arise.

Even the selection of A0 as a single address line connected to C/$\overline{D}$ is arbitrary. What it says is that two sequential memory addresses will be set aside for the 8251:

XXXXXXXXXXXXXXX0 — **Data select**
XXXXXXXXXXXXXXX1 — **Control/Status select**

The direct generation of $\overline{CS}$ and C/$\overline{D}$ from the Address Bus, as illustrated above, **will work providing you are using an 8228 System Controller to generate all read and write control strobes.** But the 8080A CPU outputs undefined information on the Address Bus during clock periods when the Address Bus is not being floated, and is not being used for a memory read or write operation. This being the case, select logic based entirely on decoding the Address Bus can spuriously cause devices to consider themselves selected, even though no read or write operation is in progress. But so long as no read or write strobe occurs during such spurious selections, no harm is done; and providing you use an 8228 System Controller in your 8080A configuration, you can be sure that read and write strobes will only occur at the proper time. If you are generating your own read and write control logic, then you must remember that any select logic based on direct Address Bus decoding can cause the 8251 USART to become selected randomly — at times when no USART access operation is supposed to be in progress.

If you use I/O instructions to access an 8251 USART, then address lines A0 - A7 are the active address carriers. Select logic that generates $\overline{CS}$ and C/$\overline{D}$ can now ignore A8 - A15.

A programmer will access the 8251 USART as two I/O ports or two memory addresses — one of which inputs or outputs data while the other inputs status or outputs controls.

Every input or output operation will be identified by the execution of either a memory reference instruction or an I/O instruction. In response to instruction execution, the 8228 System Controller will generate $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$ or $\overline{I/OW}$. These control signals are connected to $\overline{WR}$ and $\overline{RD}$ as follows:

If the 8251 USART is accessed as two memory locations:

$\overline{MEMR}$ ———————————— $\overline{RD}$
$\overline{MEMW}$ ———————————— $\overline{WR}$

If the 8251 USART is accessed as two I/O ports:

$\overline{I/OR}$ ———————————— $\overline{RD}$
$\overline{I/OW}$ ———————————— $\overline{WR}$

8251 BUS CONTROL SIGNALS

When $\overline{WR}$ is low, the CPU is outputting data or a control word to the USART

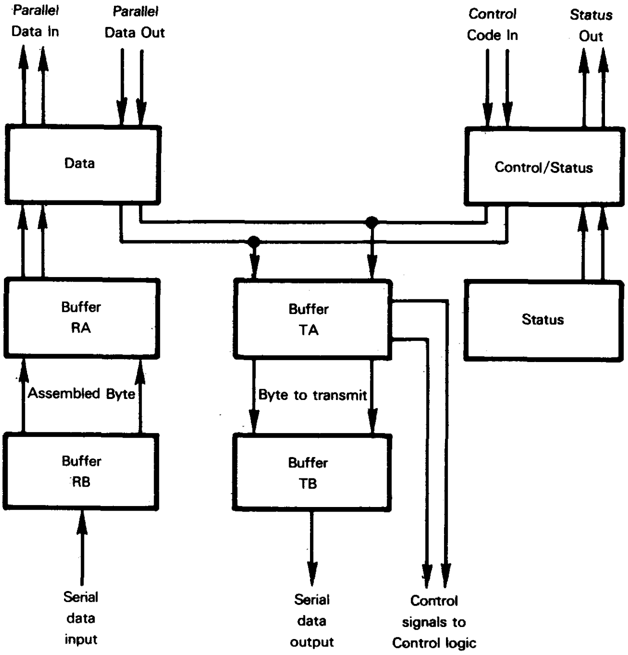
When $\overline{RD}$ is low, the CPU is inputting data or status from the USART.

There are two additional CPU interface and control signals: RESET and CLK. RESET is a typical system reset signal which, when input high, forces the 8251 USART into an idle mode. In this mode, all controls are reset — which means that subsequent operation must begin with instructions executed by the CPU to define the type of operation which is to follow; this is done using Control Codes which we will define shortly.

CLK is a clock signal input, usually connected to the Φ2 TTL level output of the 8224 Clock Generator. Note carefully that this clock input does not control either the serial transmit rate or the serial receive rate. For the 8251, this clock is used for internal timing within the USART. However, it does have to be at least 30 times the receive or transmit rate in synchronous mode and 13 times the receive or transmit rate in asynchronous mode. Also, because of device electrical specifications, CLK must be more than 0.74 MHz and less than 2.38 MHz. For the 9551 in asynchronous mode CLK must be greater than 4.5 times the frequency of either transmit or receive clocks.

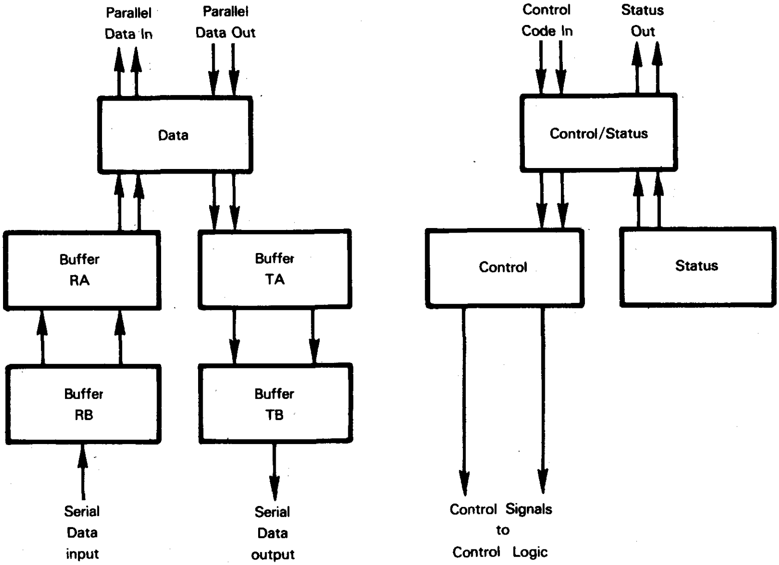
8251 USART DATA TRANSFER AND CONTROL

There are a number of buffers via which data flows into and out of the 8251 USART. These data flows may be illustrated as follows:



The AMD 9551 has a separate Control register for Control codes. Thus the AMD 9551 data flows may be illustrated as follows:

9551 CONTROL COMMAND BUFFER



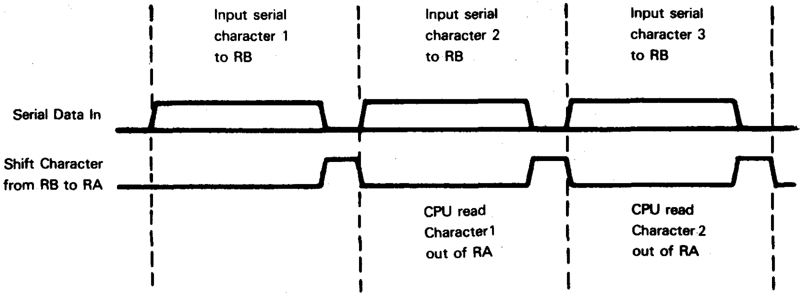
Before discussing individual signals which control data flows within the 8251 USART, consider the implications of the illustration above.

When serial data is input, 8-bit parallel data units are assembled in buffer RB.

As soon as serial data output logic has exhausted the contents of Buffer TB, it seeks the next character to be output in TA; TA contents are shifted into buffer TB.

Any memory read or input instruction which accesses the Data Address of the 8251 USART will access the contents of buffer RA. Therefore buffer RA contents must be read while the next sequential data byte is being assembled in buffer RB.

Timing for serial data input may be illustrated conceptually as follows:



Input serial character 1 to RB
Input serial character 2 to RB
Input serial character 3 to RB
Serial Data In
Shift Character from RB to RA
CPU read Character1 out of RA
CPU read Character 2 out of RA

When the next sequential byte has been assembled in buffer RB, it will be shifted to buffer RA, erasing prior buffer RA contents. If these prior buffer RA contents have not been read, an overrun error will be reported in the Status register.

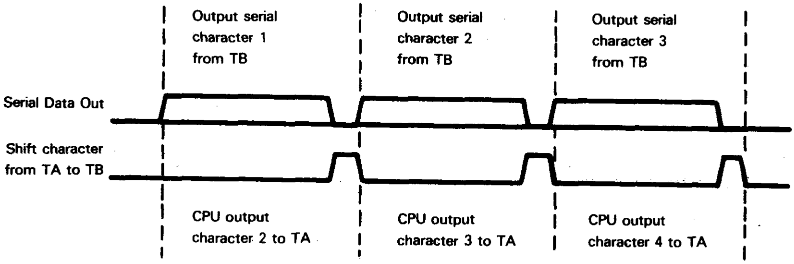
If for the moment we ignore Control Commands, then serial data output is essentially the reverse of the serial data input.

8251 USART logic converts the parallel contents of buffer TB into a serial data stream according to the protocol selected.

As soon as serial data output logic has exhausted the contents of buffer TB, it seeks the next character to be output in TA; TA contents are shifted into buffer TB.

If buffer TA is empty, then in synchronous mode, 8251 USART logic inserts a SYNC character into buffer TB. In asynchronous mode, 8251 USART logic transmits a break or mark character, depending on the selected option.

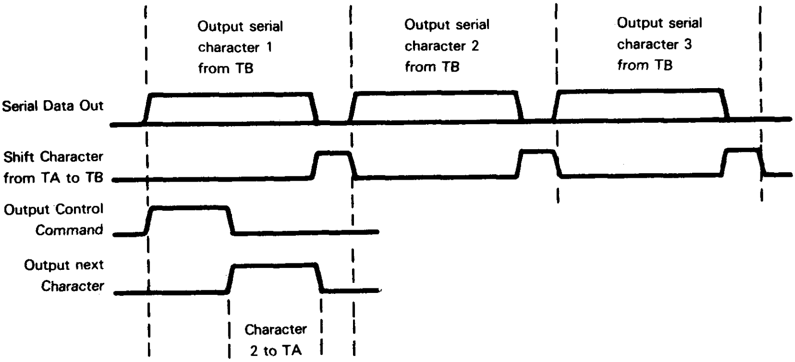
A program executed by the 8080A must transmit the next data byte to TA while the previous data byte is being output serially from TB. Conceptually, timing may be illustrated as follows:


Output serial character 1 from TB
Output serial character 2 from TB
Output serial character 3 from TB
Serial Data Out
Shift character from TA to TB
CPU output character 2 to TA
CPU output character 3 to TA
CPU output character 4 to TA

A Control Command is written out by accessing the 8251 USART Control/Status address; the Control Command byte is loaded into buffer TA. This is the same buffer which holds any data waiting to be transferred to buffer TB. The transient contents of buffer TA, following arrival of a Control Command, modifies logic throughout the 8251 USART to reflect the requirements of the Control Command just received. **There is no permanent Control Command buffer in the 8251, but remember, the 9551 does have a separate Control Command buffer.**

The one point of considerable confusion which arises when using an 8251 USART is the fact that Control Commands and data waiting to be output share buffer TA. If a Control Command is output while buffer TA holds a valid data byte, then the valid data byte will be lost — and the Control Command will be substituted as the next byte of data to be serialized and output; **however, defensive programming will keep you out of trouble. We are soon going to describe control signals that accompany transmitted serial data; you can use these control signals to keep out of trouble — and we will explain exactly how.**

If you are issuing Control Commands to an 8251 USART while outputting serial data, be sure to output the next data byte as soon as you have written out the Control Command. Furthermore, make sure that both output operations occur within the time taken to serialize and output the data byte in TB:



Output serial character 1 from TB
Output serial character 2 from TB
Output serial character 3 from TB
Serial Data Out
Shift Character from TA to TB
Output Control Command
Output next Character
Character 2 to TA

9551 USART DIFFERENCES

Since the 9551 USART has a separate Control Command buffer, none of the preceding comments regarding defensive programming apply to the 9551. When using the 9551 you can output Control Commands at any time without fear of affecting the serial data output.

Although the 9551 has a separate Control Command buffer, **you can replace an 8251 with a 9551 without having to change any 8251 programs.** The defensive programming techniques which are necessary when using an 8251 are harmless when using a 9551. But the reverse is not true. It is possible to write programs which output Control Commands to the 9551 at times which would be invalid when using an 8251. **If you have a 9551 in your system, then do not switch to an 8251** since there is a high probability that the switch from a 9551 to an 8251 will require many programming changes.

The Status register contents may be read by a memory read or input instruction that accesses the Control/Status address of the 8251 USART Reading status has no impact whatsoever on reading or writing data, or writing Control Commands.

Table 4-7 summarizes the data flows which occur in response to the control and select signals $C/\overline{D}$, $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$.

We are now going to look at serial transmit and receive logic, along with associated control signals.

Table 4-7. 8251 Data Flow Paths

$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	OPERATION
0	0	1	0	Parallel data in from RA
0	1	0	0	Parallel data out to TA
1	0	1	0	Status Out
1	1	0	0	Control code to TA
X	X	X	1	8251 not selected
X	1	1	X	Illegal state

TRANSMITTING SERIAL DATA — SERIAL TRANSMIT CONTROL SIGNALS

8251 SERIAL TRANSMIT LOGIC

Consider serial transmit logic. There are four signals associated with this logic.

8251 ISOSYNCHRONOUS SERIAL I/O

The actual serial data output occurs via TxD. The serial data output rate is controlled by the transmit clock signal which must be input on $\overline{\text{TxC}}$. $\overline{\text{TxC}}$ may or may not be derived from the microcomputer system clock. Remember, the actual rate at which asynchronous serial data is output may be 1/16 the $\overline{\text{TxC}}$ clock rate, or 1/64 of this rate. Asynchronous, serial data output occurring at exactly the clock rate is referred to as isosynchronous serial I/O.

The high-to-low transition of TxC clocks data transfers.

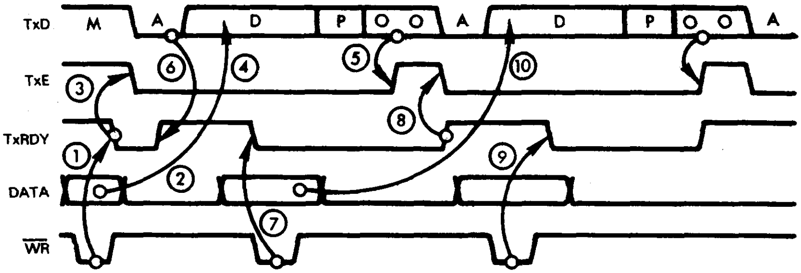
Two control signals are generated by transmit logic: TxRDY and TxE.

TxRDY is output high as soon as buffer register TA contents have been shifted into TB — and TA can be loaded with the next data byte to be output. TxRDY is reset low when the next data byte is output to TA. **But beware: the TxRDY status will be available on the TxRDY pin only when the 8251 USART is enabled to transmit — that is, when $\overline{\text{CTS}}$ is low and TxE is high. However, the TxRDY bit in the Status buffer, which we will discuss later, is always set when the buffer register TA is empty, whether or not the 8251 USART has been enabled.**

TxE is output high as soon as data in TA has been serialized and output. TxE remains high until a valid data byte is shifted from TB into TA.

8251 ASYNCHRONOUS TRANSMIT

Asynchronous serial output timing may be illustrated as follows:



M	=	Marking
A	=	Start bit
D	=	Data bits
P	=	Parity bit
O	=	Stop bits

Let us consider the event sequence illustrated by the timing diagram above. Initially a mark is being output via TxD.

Data ① arrives and is strobed into TA by $\overline{\text{WR}}$ ②

Since TxE is high, TB is empty; so TA contents are immediately transferred to TB ③ TxE is reset low.

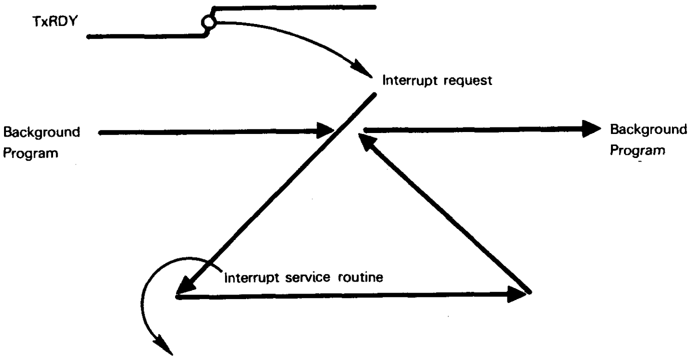
Data ① is now serialized and output ④ from TB.

As soon as data output ④ is complete, as identified by the output of stop bits, TB is empty, so TxE is set high ⑤

However, as soon as TA contents were transferred to TB ③ , TA became empty, so TxRDY was set high again ⑥ . This high was used to trigger transfer of the next data byte to TA ⑦ . Therefore when TxE is set high ⑤ , data is waiting in TA. This data is immediately transferred to TB, resetting TxE ⑧ , and again setting TxRDY high ⑨ .

The next data byte ⑦ is now serialized and output ⑩ from TB.

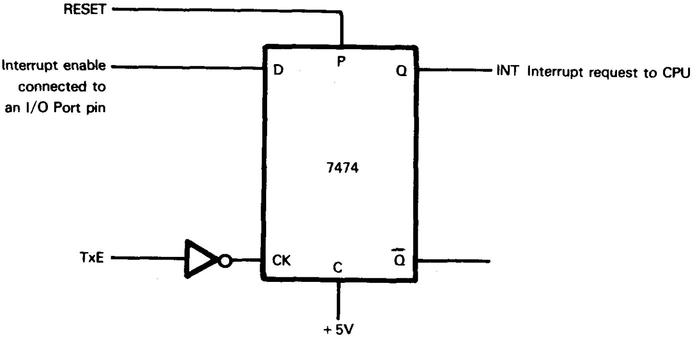
You should use the TxRDY signal or status to time transmission of data bytes to an 8251 USART. The TxRDY signal frequently is used to generate an interrupt request; the resulting interrupt service routine transmits the next data byte to the USART. This may be illustrated as follows:



```
PUSH    PSW      SAVE A AND PSW ON THE STACK
PUSH    H        SAVE H AND L ON THE STACK
LHLD    SOUT     LOAD CURRENT SERIAL DATA OUTPUT BUFFER ADDRESS
MOV     A,M      LOAD NEXT BYTE FOR SERIAL OUTPUT
OUT     PORTN    OUTPUT TO 8251
POP     H        RETURN FROM INTERRUPT SERVICE ROUTINE
POP     PSW
RET
```

When data output speed is not critical you can poll the Status register of the 8251 USART in order to determine when TxRDY is low — and another data byte may be output to the 8251 USART

By using TxE and TxRDY together you can determine when Control codes may be output to an 8251 USART. The high-to-low transition of TxE marks the point at which data has been transferred from TA to TB — and TA is empty. Thus you may use TxE to correctly time Control Command outputs. Consider the following flip-flop used to create a Control Command interrupt request:


RESET
Interrupt enable connected to an I/O Port pin
D
P
Q
INT Interrupt request to CPU
7474
TxE
CK
C
Q̄
+5V

The trailing edge of TxE is used to clock the flip-flop which requires a low D input for a valid interrupt request to be output. This low D input must be provided as an appropriate output from an I/O port. When you are ready to output a Control Command, the following instruction sequence will ensure that the Control Command is output correctly:

```
        MVI     A,ENABLE    ;LOAD DATA WITH PIN SETTING FOR D LOW
        OUT     PORTN       ;D IS CONNECTED TO A PIN OF PORT N
LOOP    NOP                 ;WAIT FOR INTERRUPT
        JMP     LOOP
MAIN PROGRAM CONTINUES HERE
        -
        -
        -
CONTROL COMMAND OUTPUT PROGRAM OCCURS ANYWHERE IN YOUR PROGRAM
AS A SERVICE ROUTINE
        -
        -
        -
```

8251 SYNCHRONOUS TRANSMIT

Synchronous serial transmit timing is essentially the same as the asynchronous timing we have just described. The only differences pertain to protocol — synchronous characters are not framed by start and stop bits; instead a stream of synchronous characters is preceded by SYNC characters. Also SYNC characters are inserted into the data stream whenever a valid character is not ready to be transmitted.

But beware of two 8251 synchronous, serial transmit peculiarities:

1) SYNC characters are loaded into TB via TA. If you output a command while a SYNC character is being loaded into TB, you will finish up with a hybrid character that is neither SYNC nor Command Code.

2) At least one valid data character must be transmitted before a SYNC character can be generated.

RECEIVING SERIAL DATA

8251 SERIAL RECEIVE LOGIC

Now consider receive logic signals.

Serial data is input via the RxD pin.

Serial data is strobed in by the $\overline{\text{RxC}}$ input clock signal which, like the $\overline{\text{TxC}}$ clock signal, is usually not derived from the microcomputer system clock. The low-to-high transition of $\overline{\text{RxC}}$ clocks data transfers.

Receive logic uses two control signals, RxRDY and SYNDET

Whenever an 8-bit data unit has been extracted from the serial input data stream, **RxRDY goes high to inform the CPU that the data has been shifted into buffer RA and may be read.** If the CPU does not read the contents of RA before the next byte of data is assembled in RB, then an overrun error will occur. The missed data byte will be lost, but an overrun error will be reported in the Status register.

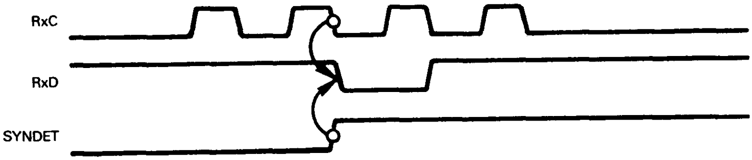
8251 HUNT MODE IN SERIAL I/O

Consider synchronous serial data input.

When the 8251 USART is receiving synchronous serial data, it initially seeks one or more SYNC characters at the beginning of the data stream. **In order to detect SYNC characters, the 8251 USART must be in Hunt mode.** You place the 8251 USART in Hunt mode by outputting an appropriate control character to the 8251 USART. In the Hunt mode, the eight bits of RB are compared with the SYNC character each time RB is filled. When a match results, the USART leaves Hunt mode and frames incoming bits as 8-bit units. When two SYNC characters are used, the USART leaves Hunt mode only if two sequential characters loaded into RB match two SYNC characters.

The SYNDET bidirectional signal operates exactly as described in Volume I, Chapter 5. SYNDET is used in synchronous mode only. In synchronous mode, SYNDET is output high after one or two SYNC characters have been detected ahead of a data stream. **Since SYNC characters are transmitted whenever there are gaps in the data stream, they may also be received at any time — not just during the beginning of transmission. The 8251 USART does not remove SYNC characters from the data stream; it treats them as data bytes. However, SYNDET is output high to identify SYNC characters so that they may be discarded. SYNDET will be reset by a Status Read.**

SYNDET is input true by external logic when external synchronization has been selected:

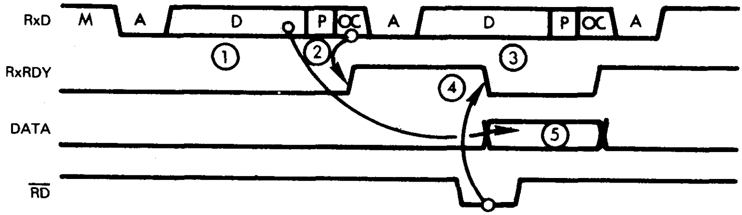


Thus by using SYNDET as an input, external logic can identify the first bit of the first serial data stream byte.

Now consider asynchronous serial data input.

Asynchronous receive timing may be summarized as follows:

8251 SERIAL ASYNCHRONOUS RECEIVE



A = Start bit
D = Data bits
P = Parity bit
O = Stop bit

Data ① is being assembled in RB. As soon as the data byte has been assembled, it is shifted to RA and RxRDY is set true ②

While the next data byte is being assembled in RB ③ , the CPU must execute instructions which read the contents of RA. As soon as RA has been read, RxRDY is eturned false ④ and data ① appears on the Data Bus ⑤

8251 SERIAL SYNCHRONOUS RECEIVE

8251 MODEM CONTROL LOGIC

Synchronous serial receive timing is essentially the same as asynchronous receive timing, as was the case with ransmit logic. Once again, the only differences pertain to protocol.

Modem control signals are absolutely standard.

The 8251 indicates it is ready using $\overline{\text{DTR}}$. The modem eplies when it is ready using $\overline{\text{DSR}}$.

Once the USART and the modem are both ready, individual data transmittal sequences are initiated by the USART outputting a request to send on RTS which must be acknowledged by the data set via CTS.

But there are some anomalies associated with 8251 Modem control logic.

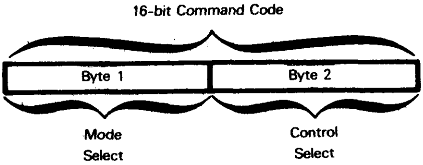
You output $\overline{\text{DTR}}$ using an appropriate Control code, and you receive back $\overline{\text{DSR}}$, which is eported as a status; we will describe Control codes and statuses shortly. It is up to your program logic to make sure that nothing inappropriate happens until $\overline{\text{DSR}}$ is input low; 8251 USART logic sets a Status register bit to reflect $\overline{\text{DSR}}$ — and that is all.

$\overline{\text{CTS}}$, on the other hand, influences 8251 logic. It requires a low $\overline{\text{CTS}}$ input for 8251 ransmit logic to be enabled.

8251 USART CONTROL CODES AND STATUS

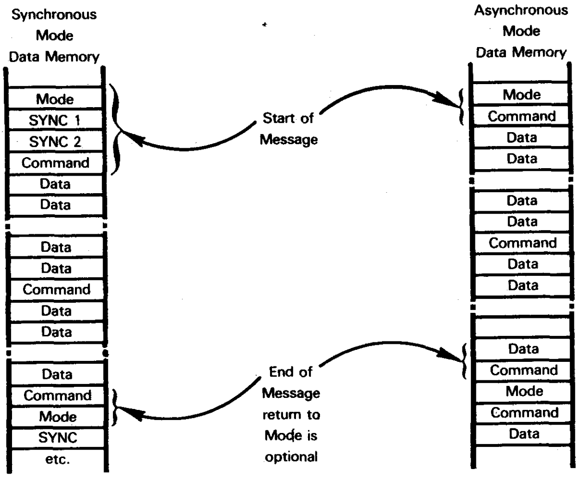
n order to select from among the numerous options provided by the 8251 USART, here is the equivalent of a 16-bit Command which must be appropriately filled with two Command Codes. There is also an 8-bit status, maintained in a read only buffer. As we illustrated earlier, a single control/status address accesses status when ou read and controls when you write.

Since two Command Codes are written to one address, the two codes are distinguished by referring to one as a "Mode Select", and the other as a "Control Select":



When the 8251 USART is first powered up, or after it is reset, USART logic assumes a Mode Select Command Code. If the Mode Select Command specifies synchronous mode, the 8251 logic expects the subsequent command input to be the SYNC character — or, in systems with two SYNC characters, that the next two bytes will be SYNC1 and SYNC2. As soon as a Mode Select Command Code — and SYNC characters, if needed — has been received, 8251 USART logic switches to expecting Control Select Codes; this persists until the 8251 is reset, or a special Control Select is output forcing the choice back to Mode Select.

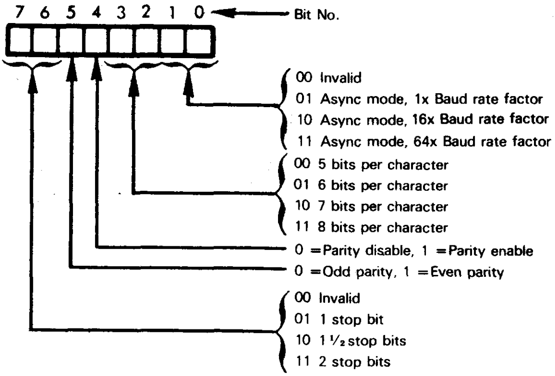
In terms of data transfer operations, therefore, the CPU will transmit Mode, Command and Data bytes as follows:



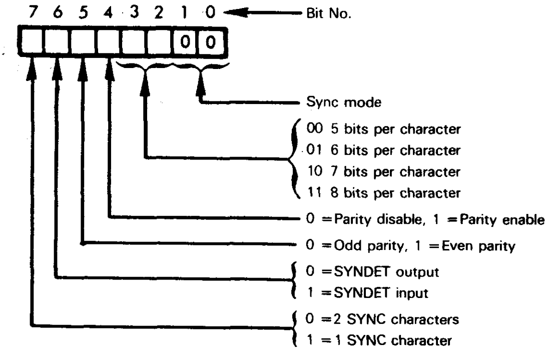
If two independent programs control a single USART, the situation may arise in which an internal reset command arrives when the USART expects a SYNC character. The command would be accepted as a SYNC character, and no reset would occur. This error can be avoided by preceding the internal reset command word with three all-zero command inputs to the USART.

8251 MODE CONTROL BYTE FORMAT

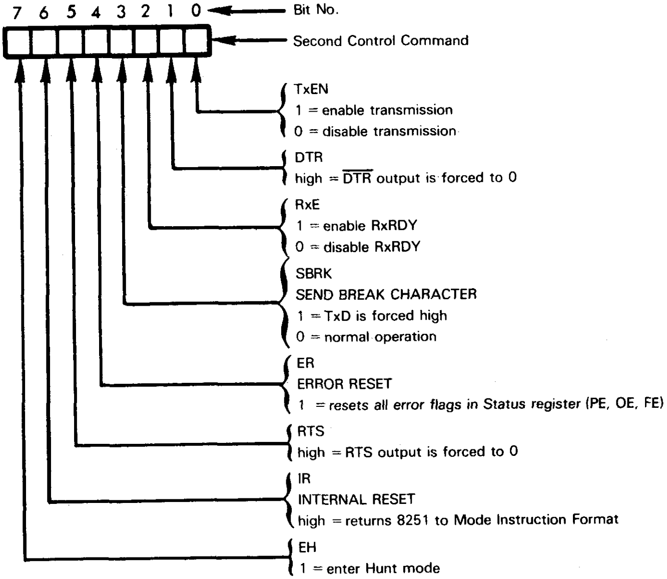
These are the ways in which a data byte, output as a mode (Control byte 1) will be interpreted by 8251 logic. First there is asynchronous mode control:



Synchronous mode control is defined thus:



The second Control Command data byte will be interpreted in a very different way. Whereas the Mode Command defines the operating environment, **the Control Command causes instant action, as follows:**



Let us look at the various Control Command bits. See Volume I, Chapter 5 for a discussion of what individual signals do in serial I/O systems.

Bit 0 is the transmit enable control TxEN. Data can be transmitted only while TxEN is high. Table 4-8 defines the operations which can occur depending on the condition of bit TxEN and signals TxE and TxRDY

Control Command Bit 1 directly sets the level of the $\overline{\text{DTR}}$ signal output. $\overline{\text{DTR}}$ is used by data communications systems that are operated automatically or under program control. $\overline{\text{DTR}}$ output low indicates to logic beyond the microcomputer system that the 8251 is ready to communicate. Notice that Bit 1 of the Control Command is the complement of the $\overline{\text{DTR}}$ control output level.

Bit 2 of the Control Command enables and disables the receive control signal RxRDY. Bit 2 does not enable or disable receive logic. Providing receive logic is otherwise enabled, the 8251 will receive data whether or not RxRDY has been enabled. If RxRDY has been enabled by the Control Command, then it will indicate the condition of data being received — as we have already described.

Control Command Bit 3, if set to 1, will immediately interrupt serial data output and force TxE high. This is a break signal.

Control Command Bit 4 is the error reset bit. When you read the Status register, if you find any of the Status register error bits are true, then you must clear the error status bits by outputting a Control Command with 1 in Bit 4.

Control Command Bit 5 is the complement of the $\overline{\text{RTS}}$ signal output. $\overline{\text{RTS}}$ is used by modem handshaking control logic; it has no effect whatsoever on the serial receive logic of the 8251 USART.

Control Command Bit 6, when high, causes the next command output to be interpreted as a Mode Select and not a Control Command. Returning from a sequence of Control Commands to a Mode Select constitutes a Reset operation within the 8251 USART.

Control Command Bit 7 applies to synchronous operation only. When high, this bit causes the 8251 USART to enter the Hunt mode — at which time receive logic will look for SYNC characters, either one or two, as specified by the Mode Select.

Frequently, Control Command Bit 2 (RxE) is used as a receive enable for asynchronous operations while Control Command Bit 7 (EH) is used as a synchronous receive enable. In either case, remember that prior data in buffers RA and RB are not cleared. **You should therefore discard the first two bytes read after enabling receive logic via Control Command bits 2 or 7.**

When a Control Command is received by the 8251, signals are set or reset immediately as specified by appropriate Control Command bits, and flip-flops are set for events which cannot occur immediately. TxEN, $\overline{\text{DTR}}$, RxE, SBRK and $\overline{\text{RTS}}$ cause signals to be set or reset immediately. ER, EH and IR cause flags to be set.

The ER flag remains set until error conditions have been cleared as identified by the Status register; ER is then automatically reset. You do not have to output another Control Command in order to automatically reset ER.

The IR flag, when set, causes the next command output to be interpreted as a mode control. The IR flag remains set until the next mode control is output at which time the IR flag is reset.

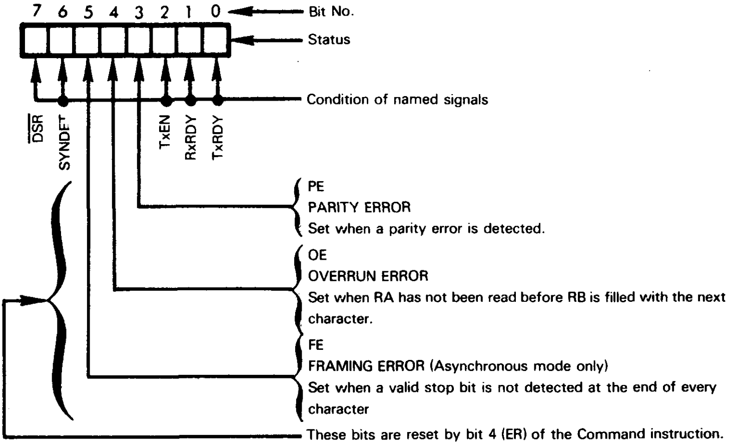
The EH flag, when set, remains set until one or two SYNC characters have been detected. SYNC is then reset.

Table 4-8. Transmit Operations As A Function Of TxE, TxRDY, And TxEN

TxEN	TxE	TxRDY	OPERATION
0	X	X	Transmitter is disabled.
1	0	0	TB is currently transmitting and an additional byte is in TA, ready for transmission.
1	0	1	Byte is shifting from TB to TxD. TA is available to receive a new byte from the CPU.
1	1	0	TB byte has been transmitted. A new byte is waiting for transmission. This is a transient condition.
1	1	1	TA and TB are empty. TxD continues to mark if 8251 is in the asynchronous mode. TxD will send Sync pattern if 8251 is in the synchronous mode.

When status is read from the 8251, the status byte is interpreted as follows:

8251 STATUS FORMAT


7 6 5 4 3 2 1 0
Bit No.
Status
Condition of named signals
DSR
SYNDET
TxEN
RxRDY
TxRDY
PE
PARITY ERROR
Set when a parity error is detected.
OE
OVERRUN ERROR
Set when RA has not been read before RB is filled with the next character.
FE
FRAMING ERROR (Asynchronous mode only)
Set when a valid stop bit is not detected at the end of every character
These bits are reset by bit 4 (ER) of the Command instruction.

The parity, overrun and framing errors simply report when these errors occur. 8251 logic makes no attempt to correct these errors in any way. Corrections, if any, are the responsibility of the program monitoring 8251 execution.

Similarly, the six signal condition bits merely report associated signal levels: 1 for high, 0 for low.

THE NEC μPD379 SYNCHRONOUS RECEIVER/TRANSMITTER

It might be argued that many of the problems found on the 8251 USART result from an attempt to combine synchronous and asynchronous serial I/O logic on a single chip. NEC takes a different approach, providing separate devices for synchronous and asynchronous serial I/O. The μPD379 is a synchronous serial I/O device. Although the μPD379 has been designed to work with 8080A microprocessors, in fact, it can be used with almost any microprocessor described in this book.

The μPD379 provides standard synchronous I/O protocol and Synchronous Data Link Control (SDLC) protocol.

Table 4-9 summarizes operating characteristics of the μPD379 as compared to the 8251 devices.

The μPD379 is packaged as a 42-pin DIP; it uses the three standard 8080A power supplies: V_{DD} = +12V, V_{CC} = +5V and V_{BB} = -5V.

Figure 4-35 illustrates that part of general microcomputer system logic which has been implemented on the μPD379.

Table 4-9. A Comparison Of 8251/9551 And μPD379 Synchronous Serial Data Transfer Capabilities

FEATURE	8251/9551	μPD379
Synchronous	Yes	Yes
Asynchronous	Yes	No
Bits per character	5, 6, 7 and 8	8 only
Errors detected	Parity, overrun	Overrun, underrun
Synchronization	Internal or External	Internal only
SYNC characters	1 or 2	1 only
Baud rate	DC to 56K Baud	DC to 800K Baud
Clock rate	x1, x16, x64	x1
Protocols	Standard only	Standard or SDLC
Duplex	Half duplex only	Half or full duplex
Modem Controls	$\overline{\text{DSR}}$, $\overline{\text{DTR}}$, $\overline{\text{CTS}}$, $\overline{\text{RTS}}$	None
Buffering	Double	Double

A μPD379 DEVICE OVERVIEW

μPD379 design philosophy differs markedly from that of the 8251. The μPD379 has separate 8-bit parallel input and output data pins whereas the 8251 has a single 8-bit bidirectional parallel data port. The μPD379 has no Control and Status registers; rather, it relies on input and output signals to achieve the same result.

The μPD379 has both advantages and disadvantages.

Advantages are that full duplex operation is possible — that is to say, transmit and receive logic can operate simultaneously. Also, control inputs may be created continuously and status may be sampled continuously by logic lying between the CPU and the μPD379 device; external logic can cope with status and controls much faster than programming steps could accomplish the same task.

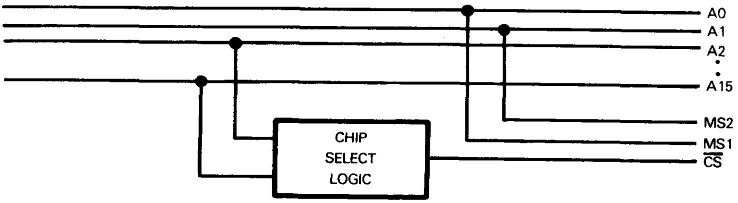
But in order to avail yourself of μPD379 high speed advantages you must have additional logic which will add to overall system cost. Not only must you have two sets of connections from I/O ports to the system Data Bus, you may also need logic to create appropriate control signals and hold status outputs.

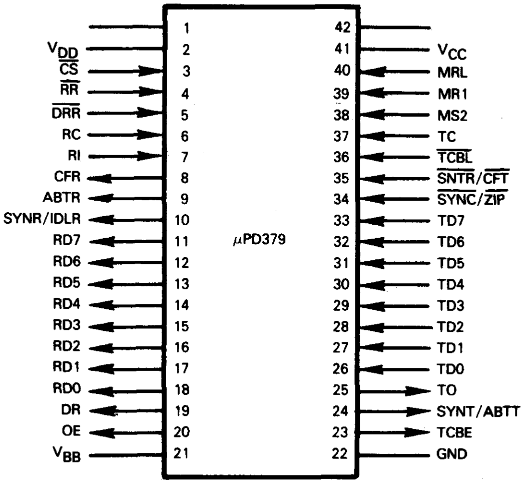
μPD379 PINS AND SIGNALS

μPD379 pins and signals are illustrated in Figure 4-37. We will examine signals while discussing operating modes of the μPD379.

Table 4-10 summarizes signal conditions when the μPD379 device is not selected ($\overline{\text{CS}}$ high) and following receive logic reset ($\overline{\text{RR}}$ low). Table 4-10 also summarizes set and reset conditions for signals, where relevant.

To the programmer the μPD379 device will appear to be four separately addressable I/O ports or memory locations. These locations are accessed via the chip select ($\overline{\text{CS}}$) and mode select signals (MS1, MS2) as follows:





PIN NAME		DESCRIPTION	TYPE
RD0 - RD7		Parallel Data Out	Output
TD0 - TD7		Parallel Data In	Input
$\overline{RR}$		Receive Logic Reset	Input
$\overline{CS}$		Chip Select	Input
MS1, MS2		Mode Select	Input
MRL		Mode Select input strobe	Input
TO		Serial data output	Output
TC		Transmitter clock	Input
TCBE		Transmitter character buffer empty	Output
SYNT	(S)	A SYNC character is being output	Output
ABTT	(D)	An abort pattern is being output	
$\overline{SNTR}$	(S)	Reset SYNT	Input
$\overline{CFT}$	(D)	Reset TCBE and transmit closing flag	
$\overline{TCBL}$		Strobe for all data input via TD0 - TD7	Input
RI		Serial data input	Input
RC		Receiver clock	Input
DR		Receiver buffer full	Output
$\overline{DRR}$		Reset DR	Input
CFR	(D)	Closing flag received	Output
ABTR	(D)	Abort pattern received	Output
SYNR	(S)	SYNC character received	Output
IDLR	(D)	Idle pattern received	
OE		Overrun error	Output
$\overline{SYNC}$	(S)	SYNC character input via TD0 - TD7	Input
$\overline{ZIP}$	(D)	Prohibit zero insertion pattern	
V_{CC}, V_{BB}, V_{DD}, GND		Power and Ground	

(S) Simple synchronous mode interpretation

(D) SDLC mode interpretation

Figure 4-37. μPD379 Synchronous Serial Communications Interface Signals And Pin Assignments

The mode select inputs MS1 and MS2 are latched on the rising edge of signal MRL; **MRL** therefore **acts as an address input strobe.** You may connect MRL to the SYNC timing signal output by the 8080A or the $\overline{\text{STSTB}}$ status strobe signal output by the 8224 Clock Generator Driver device. Since SYNC is positive true, it must be inverted in order to create MRL

Alternatively, **you can connect MS1 and MS2 to Data Bus lines; MRL is then generated from the write strobe signal $\overline{\textbf{WR}}$.** $\overline{\text{CS}}$ must still be decoded from the Address Bus.

μPD379 MODES

In standard synchronous mode the μPD379 operates in a manner akin to the 8251.

In SDLC mode the μPD379 conforms to SDLC protocol.

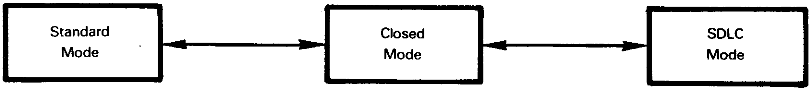
In Closed mode no serial input or output is taking place.

μPD379 MODE SPECIFICATION

MS1 and MS2 define modes as follows:

MODE	MS1	MS2
Closed	0	0 or 1
Standard	1	0
SDLC	1	1

You must move between Standard and SDLC via the Closed mode. Never attempt to switch directly between Standard and SDLC modes. This may be illustrated as follows:



You cannot be operating simultaneously in Standard and SDLC modes. For example, the μPD379 device will operate in full duplex mode; that is to say, you can simultaneously transmit and receive serial data. But the data being transmitted and received must both be subject to Standard synchronous protocol, or to SDLC protocol. One cannot be subject to Standard protocol while the other is subject to SDLC protocol.

CLOSED MODE

Closed mode and mode select logic are illustrated in Figure 4-38.

Before beginning operation in either Standard or SDLC mode you must enter Closed mode.

μPD379 SYNC CHARACTER

Prior to operating in Standard synchronous mode you must specify the SYNC character which will be used for all synchronization operations. You use the SYNC/$\overline{\text{ZIP}}$ signal in order to specify the SYNC character. If this signal is high when you select Closed mode then the Japanese industrial standard 16_{16} (ASCII SYN code) is assumed to be the SYNC character. If this signal is low when you select the Closed mode, then the SYNC character is strobed in by TCBL on the parallel data input pins TD0 - TD7. These two cases may be illustrated as follows; first specification of 16_{16} as the SYNC character:

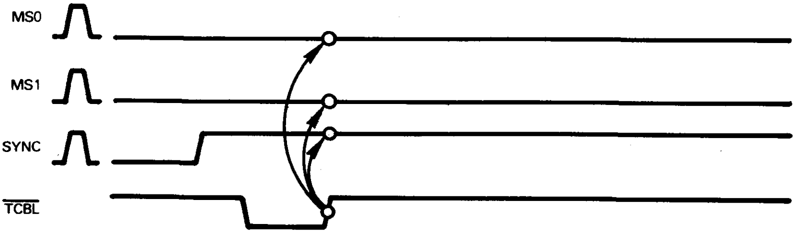
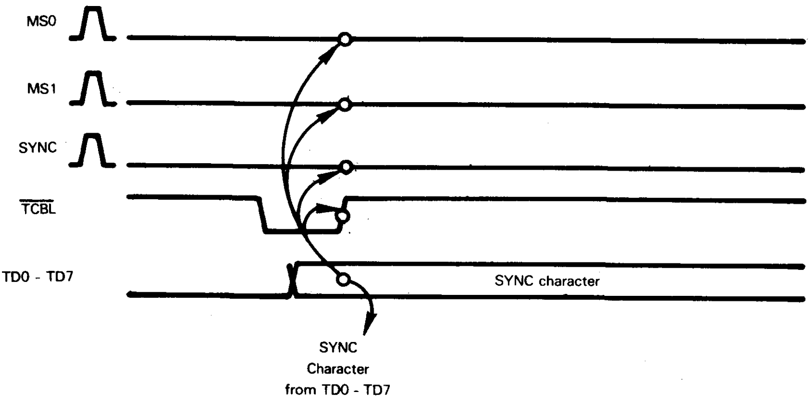


Table 4-10. Signal Setting, Resetting And Disable Summary

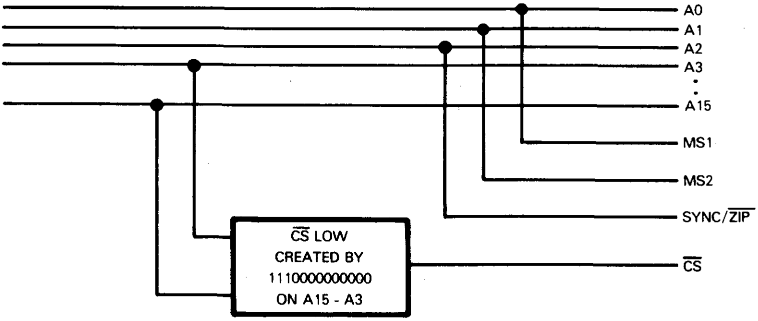
Signal	Condition when $\overline{CS}$ is high	Condition following $\overline{RR}$	Setting Condition	Resetting Condition
RD0 - RD7	High impedance	1	*	*
TD0 - TD7	Unaffected	Unaffected	*	*
$\overline{RR}$	Disabled	*	*	*
$\overline{CS}$	*	Unaffected	*	*
MS1, MS2	Unaffected	Unaffected	*	*
MRL	Disabled	Unaffected	*	*
TO	Unaffected	Unaffected	*	*
TC	Unaffected	Unaffected	*	*
TCBE	High impedance	Unaffected	Transmitter character buffer empty	$\overline{TCBL}$ low $\overline{CFT}$ low (SDLC only) ½ bit prior to ABTT going high (SDLC only)
SYNT	High impedance	Unaffected	SYNC character transmitted	$\overline{SNTR}$ low or start of data character transmit
ABTT	High impedance	Unaffected	Abort pattern transmitted	Flag pattern transmitted
$\overline{SNTR}$	Disabled	Unaffected	*	*
$\overline{CFT}$	Disabled	Unaffected	*	*
$\overline{TCBL}$	Disabled	Unaffected	*	*
RI	Unaffected	Unaffected	*	*
RC	Unaffected	Unaffected	*	*
DR	High impedance	0	Received data loaded into Receiver Buffer Register	$\overline{DRR}$ low. Rising edge of OE (SDLC only) Abort flag (SDLC only)
$\overline{DRR}$	Disabled	Unaffected	*	*
CFR	High impedance	0	$7E_{16}$ flag received (SDLC only)	Rising edge of DR or OE (SDLC only)
ABTR	High impedance	0	Abort flag received (SDLC only)	Rising edge of DR or OE (SDLC only)
SYNR	High impedance	0	SYNC character detected	DR high on a non-SYNC character
IDLR	High impedance	0	Idle pattern detected	Rising edge of DR or OE
OE	High impedance	0	Overrun error	DR low on data transfer to Receiver Buffer Register (Standard) Rising edge of DR (SDLC)
SYNC	Disabled	Unaffected	*	*
$\overline{ZIP}$	Disabled	Unaffected	*	*
Signal	Condition when CS is high	Condition following RR	Setting Condition	Resetting Condition

* Signal transition is a function of CPU or external operation only.

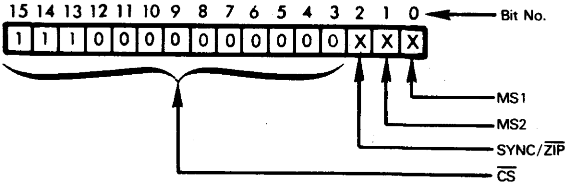
Here is timing to specify any other SYNC character:



If you create MS0 and MS1 from Address Bus lines, consider creating the SYNC input from another Address Bus line. By way of illustration here is one possibility:



The illustration above arbitrarily assumes that A15 - A3 creates CS low for the signal pattern 1110000000000. For this arbitrary case the μPD379 will be accessed by these addresses:



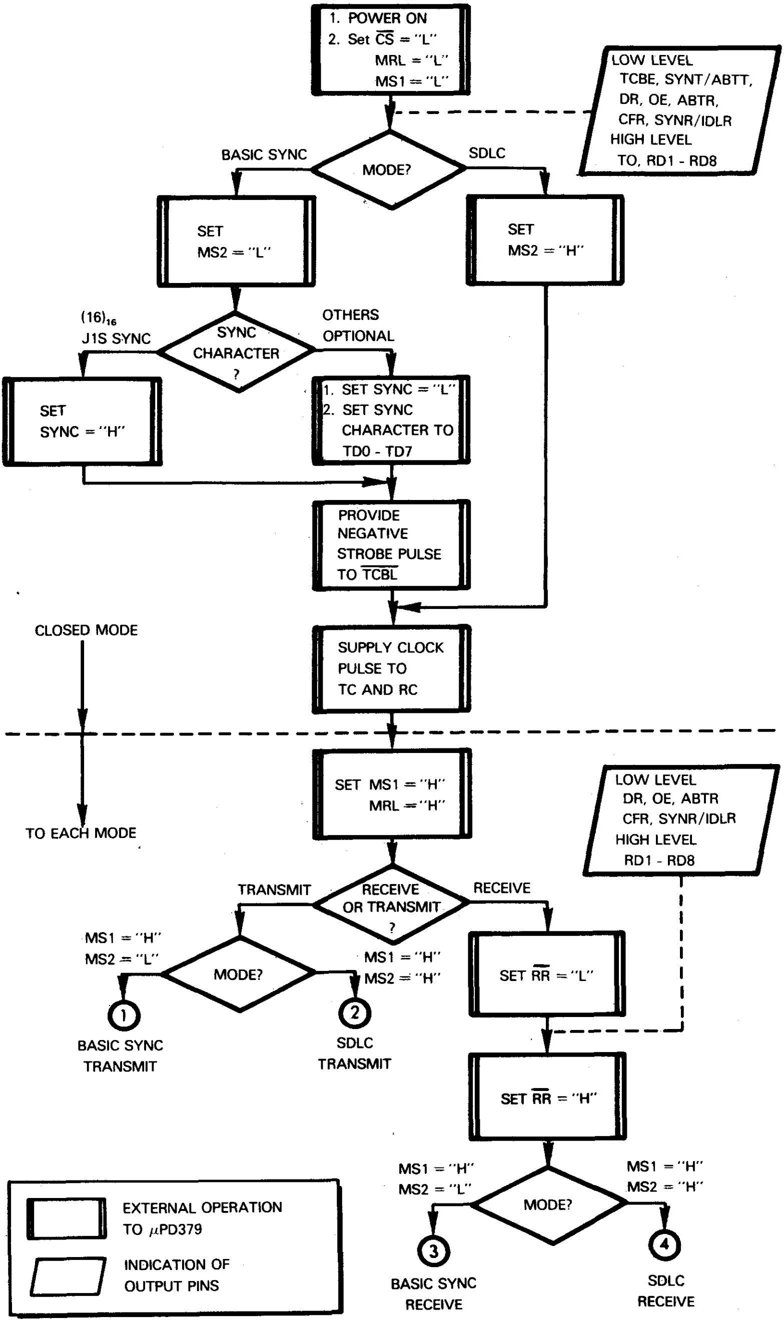


Figure 4-38. Initialization And Mode Select Operation Flow Chart

Memory addresses will be interpreted as follows:

Memory Address	Interpretation
E000	Closed mode, SYNC character being output as data on the Data Bus.
E001	Standard mode, parallel data may be written to, or read from this address.
E002	Same as E000
E003	SDLC mode with Zero insert prohibited; parallel data may be written to, or read from this address.
E004	Closed mode, 16_{16} specified as SYNC character. Ignore the Data Bus.
E005	Same as E001
E006	Same as E004
E007	SDLC mode with Zero insert enabled; parallel data may be written to, or read from this address.

These interpretations will become effective when mode is strobed in by MRL high and the SYNC character is strobed in by $\overline{\text{TCBL}}$.

The SYNC character is used by standard synchronous serial I/O protocol as described in Volume I, Chapter 5; that is to say the SYNC character is used to synchronize the start of a new serial data input stream; it is also used as a filler character when no valid character is available to transmit.

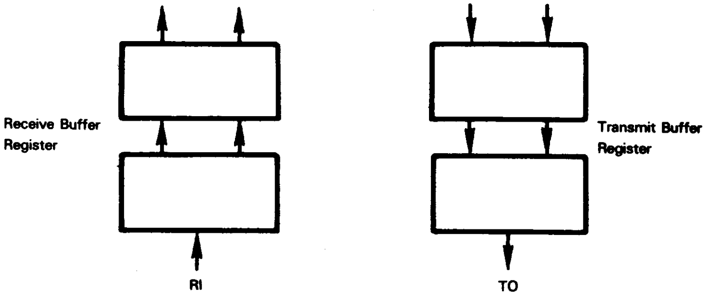
The SYNC character has no meaning, and is not used in SDLC protocol.

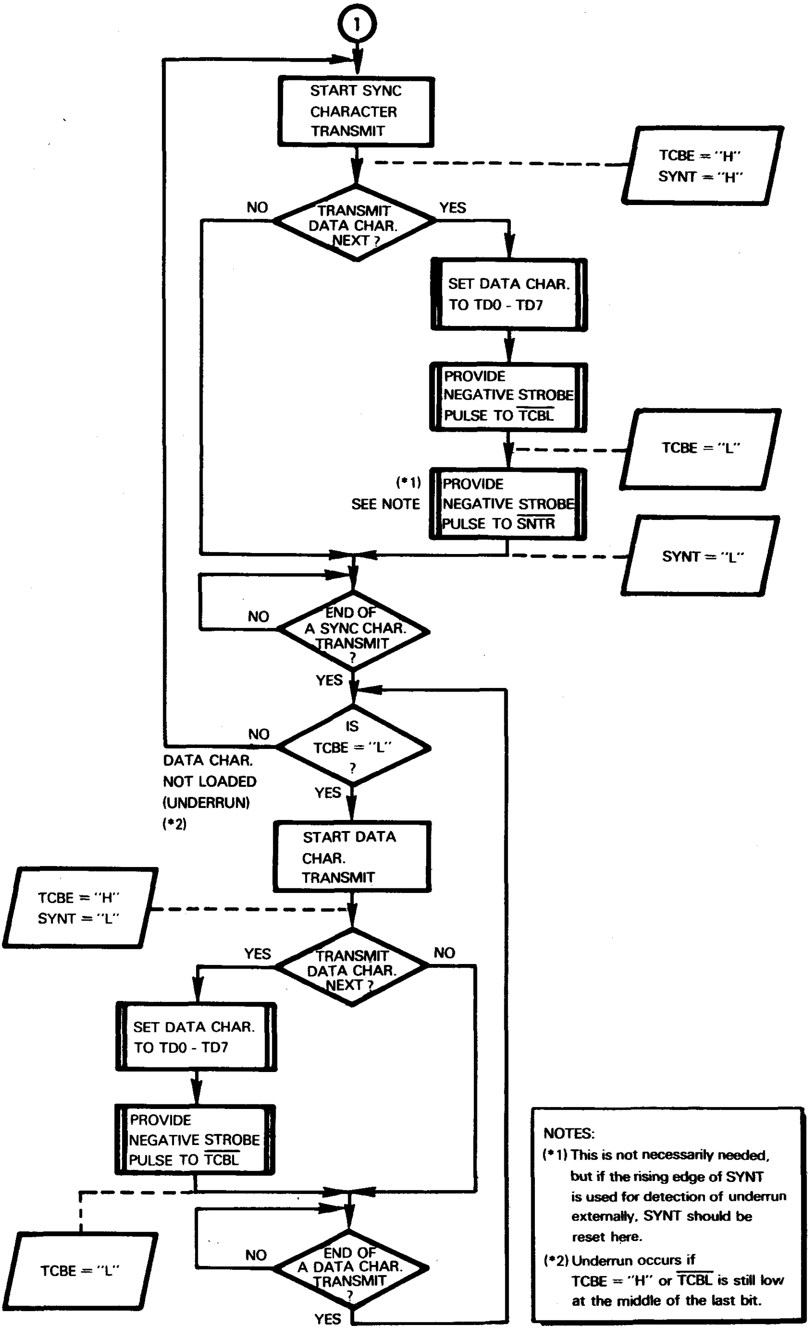
There are some important points you should note regarding the way in which the μPD379 handles the SYNC characters when operating under standard synchronous protocol. From the discussion in Chapter 5 of Volume I, recall that **either one or two SYNC characters may precede a new serial data stream.** This is an option which you can select under program control when using the 8251 USART.

When using the μPD379, the number of SYNC characters preceding a new serial data stream is not a programmable option; rather, separate control signals are used to identify a SYNC character being transmitted (SYNT) and a SYNC character having been received (SYNR). It is up to the CPU, or logic external to the μPD379, to force the transmission of two SYNC characters, and to require the receipt of two SYNC characters if that is the option you must have.

DATA BUFFERING

The μPD379 like the 8251 has a single buffer register for transmit and receive logic. This may be illustrated as follows:




1
START SYNC CHARACTER TRANSMIT
TCBE = "H"
SYNT = "H"
NO
TRANSMIT DATA CHAR. NEXT ?
YES
SET DATA CHAR. TO TD0 - TD7
PROVIDE NEGATIVE STROBE PULSE TO $\overline{\text{TCBL}}$
TCBE = "L"
(*1) SEE NOTE
PROVIDE NEGATIVE STROBE PULSE TO $\overline{\text{SNTR}}$
SYNT = "L"
NO
END OF A SYNC CHAR. TRANSMIT ?
YES
NO
IS TCBE = "L" ?
DATA CHAR. NOT LOADED (UNDERRUN) (*2)
YES
START DATA CHAR. TRANSMIT
TCBE = "H"
SYNT = "L"
YES
TRANSMIT DATA CHAR. NEXT ?
NO
SET DATA CHAR. TO TD0 - TD7
PROVIDE NEGATIVE STROBE PULSE TO $\overline{\text{TCBL}}$
TCBE = "L"
NO
END OF A DATA CHAR. TRANSMIT ?
YES

NOTES:

(*1) This is not necessarily needed, but if the rising edge of SYNT is used for detection of underrun externally, SYNT should be reset here.

(*2) Underrun occurs if TCBE = "H" or $\overline{\text{TCBL}}$ is still low at the middle of the last bit.

Figure 4-39. Standard Synchronous Transmission Operations Flow Chart

Thus as described for the 8251 USART, you have the time it takes to transmit or receive a single character during which you must write or read the next character.

TRANSMITTING AND RECEIVING SERIAL DATA UNDER STANDARD SYNCHRONOUS PROTOCOL

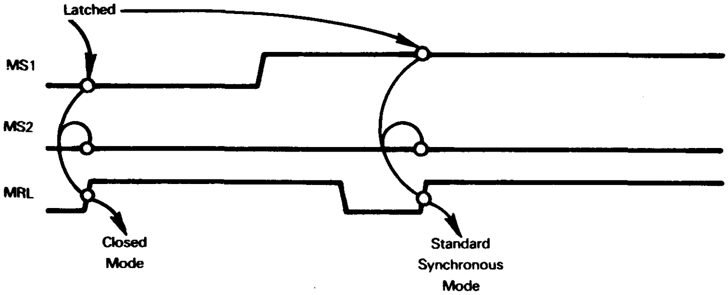
Let us now look at the signals used by the μPD379 when transmitting and receiving data using standard synchronous protocol.

First consider data transmission. Operations are illustrated in Figure 4-39.

μPD379 STANDARD TRANSMIT

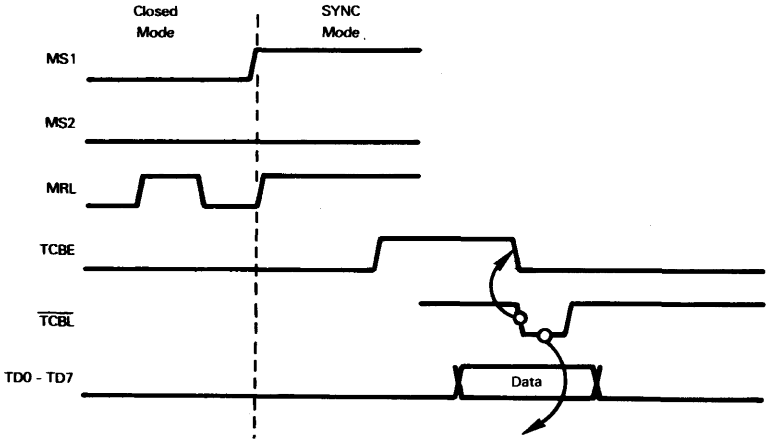
Initially you will identify either **the** standard Japanese SYNC character (16_{16}) or you will provide some other **SYNC character;** this specification is made in Closed mode, as already described.

Next you select standard synchronous protocol via the MS1 and MS2 signals. These signals must have valid levels while MRL makes a low-to-high transition. The sequence of going from Closed mode to Standard synchronous mode may be illustrated as follows:



Since serial and parallel data are transmitted and received via separate and distinct pins, **you do not have to specify whether a transmit or receive operation is going to occur;** in fact, both can occur simultaneously.

Once standard synchronous protocol has been selected, subsequent operations are quite elementary. First, **you must output parallel data.** You do this by writing to the memory location or I/O port address which correspnds to the μPD379 device operating in Standard synchronous mode. Timing may be illustrated as follows:

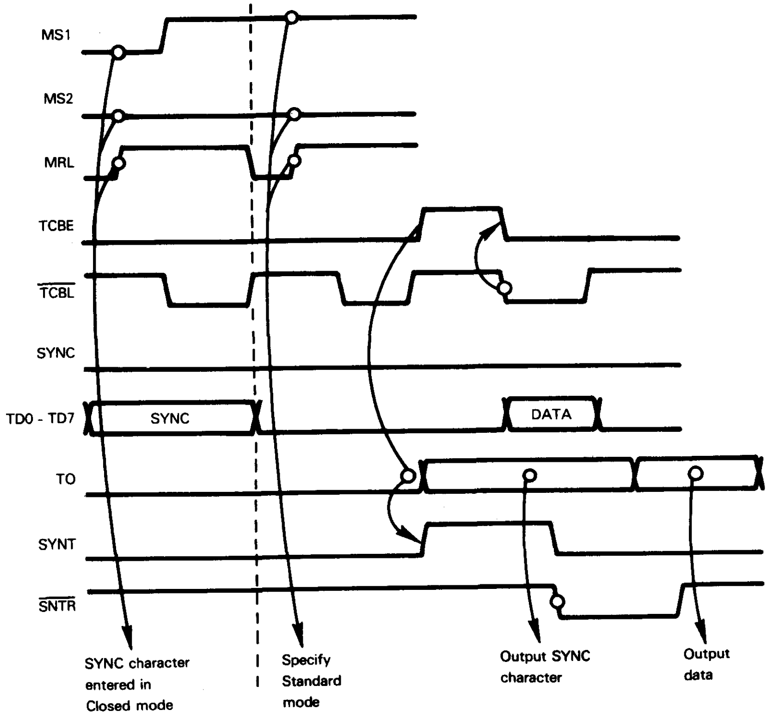


The data input pins TD0 - TD7 will be connected directly to the Data Bus.

TCBE is a control signal outpt by the μPD379 device to indicate that the transmit buffer is empty — therefore the CPU should write parallel data for transmittal. How you use the TCBE output is up to you, but observe that it will go high as soon as Standard synchronous mode has been established. TCBE high could be used to create an interrupt request, or it could be trapped in a buffer, whence the CPU could read it as status. TCBE will subsequently be reset by a low $\overline{\text{TCBL}}$ pulse. $\overline{\text{TCBL}}$ may be connected directly to the memory or I/O write control signal ($\overline{\text{MEMW}}$ or $\overline{\text{I/OW}}$) output by the 8228 System Controller.

As soon as Standard synchronous mode has been selected, **the μPD379 device will output a single synchronous character,** identified by SYNT high. In order to reset SYNT you must pulse $\overline{\text{SNTR}}$ low. If you wish to output two SYNC characters, then you may use the SYNT high pulse as a signal to output another SYNC character, as though it were data, before resetting SYNT with a low $\overline{\text{SNTR}}$ pulse.

If you are outputting two SYNC characters at the head of a serial data stream, then you can write the second SYNC character as parallel data while the first SYNC character is being output. If you are using a single SYNC character at the head of your serial data stream, then, while this SYNC character is being output, **you must write out the first data character to the μPD379 device.** If you are using two SYNC characters, then you must write the first data character while the second SYNC character is being output. Once the parallel data character has been written to the μPD379 device it will be output serially, least significant bits first. TCBE identifies those time intervals during which the Transmit Character Buffer is empty and may be written into. TCBE is therefore equivalent to the TxRDY signal of the 8251 USART. Timing may be illustrated as follows:



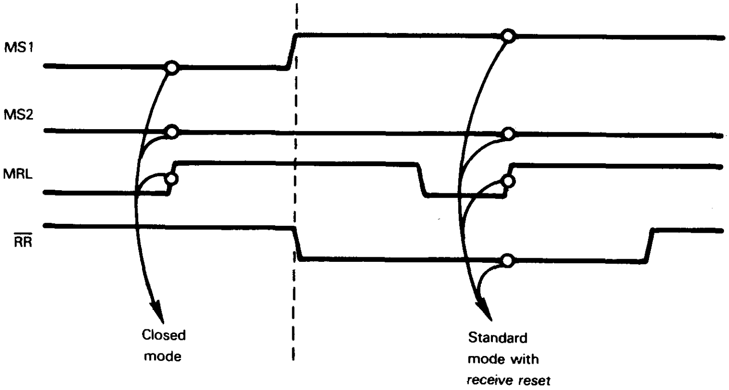
So long as there is a valid data character waiting to be transmitted, serial data transmission will proceed as illustrated above. **If at any time a valid character is not ready to transmit, then the μPD379 device will insert a SYNC character.** When a SYNC character is inserted in the middle of a serial data stream, then the SYNT signal is pulsed high; $\overline{\text{SNTR}}$ does not have to be pulsed low in order to reset SYNT in the middle of a transmit stream. SYNT remains high only for the duration of the SYNC character being output in the middle of serial data transmission.

μPD379 STANDARD RECEIVE

Let us now consider signals which accompany serial data being received using standard synchronous protocol. Figure 4-40 illustrates operations.

Initially, you must specify the nature of the SYNC character. This is done in Closed mode, as we have already described.

Once the SYNC character has been specified, you select standard synchronous protocol using the MS1 and MS2 inputs in exactly the same way as described for standard synchronous transmit logic. When beginning a receive operation, however, you must also reset the receive logic by inputting $\overline{\text{RR}}$ low to the μPD379 device. Timing may be illustrated as follows:



You are now ready to start receiving serial data. First, receive logic must detect one or two SYNC characters. As soon as the first SYNC character has been detected SYNR will go high.

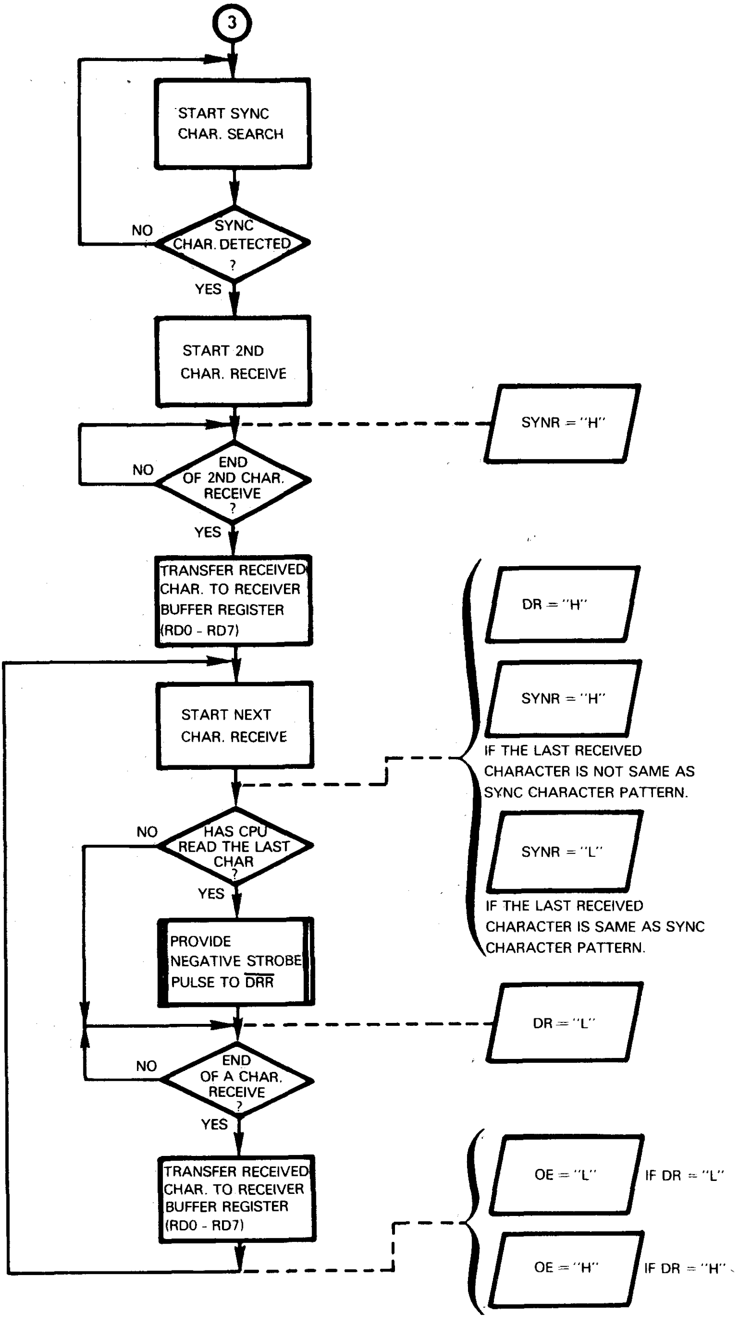
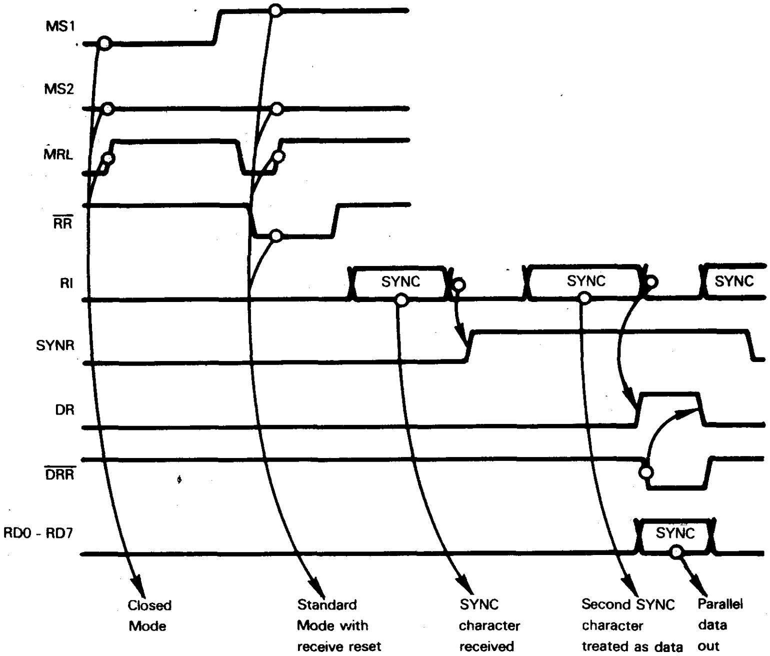

3
START SYNC CHAR. SEARCH
SYNC CHAR. DETECTED ?
NO
YES
START 2ND CHAR. RECEIVE
SYNR = "H"
END OF 2ND CHAR. RECEIVE ?
NO
YES
TRANSFER RECEIVED CHAR. TO RECEIVER BUFFER REGISTER (RD0 - RD7)
DR = "H"
SYNR = "H"
IF THE LAST RECEIVED CHARACTER IS NOT SAME AS SYNC CHARACTER PATTERN.
START NEXT CHAR. RECEIVE
HAS CPU READ THE LAST CHAR ?
NO
YES
SYNR = "L"
IF THE LAST RECEIVED CHARACTER IS SAME AS SYNC CHARACTER PATTERN.
PROVIDE NEGATIVE STROBE PULSE TO $\overline{DRR}$
DR = "L"
END OF A CHAR. RECEIVE ?
NO
YES
TRANSFER RECEIVED CHAR. TO RECEIVER BUFFER REGISTER (RD0 - RD7)
OE = "L"
IF DR = "L"
OE = "H"
IF DR = "H"

Figure 4-40. Standard Synchronous Receiving Operations Flowchart

SYNR will remain high until the first non-SYNC character has been detected. This may be illustrated as follows:



As soon as a single SYNC character has been accepted, the DR and $\overline{DRR}$ signals become active. At the conclusion of each character assembled, whether it be a SYNC character or not, DR goes high. DR will remain high until reset by a low $\overline{DRR}$ pulse. You will normally use DR as a signal to the CPU indicating that another byte of data must be read from the μPD379 device. Thus, DR is equivalent to the RxRDY signal of the 8251 USART. $\overline{DRR}$ will typically be connected to the memory read control signal ($\overline{MEMR}$) output by the 8228 System Controller. Thus, the process of reading parallel data from the μPD379 device will automatically reset DR.

An overrun error will occur if the CPU has not read parallel data from the Receive Data Buffer before the next parallel data byte has been assembled and is transmitted to the Receive Data Buffer. Within the μPD379 device this condition is identified by DR still being high when the next byte of data is transmitted to the Receive Data Buffer. If this situation occurs, then the overflow error signal OE will go high and data will be lost.

OE remains high until the next byte of data is transmitted to the Receive Data Buffer while DR is low.

Whenever a SYNC character is received, either at the head of the serial data stream or in the middle of a serial data stream, SYNR remains high from the conclusion of the SYNC character to the conclusion of the next non-SYNC character. No signal is required to reset SYNR.

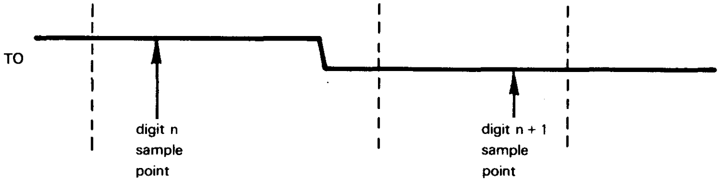
Table 4-10 summarizes the conditions which set and reset control signals associated with standard synchronous protocol.

SERIAL DATA

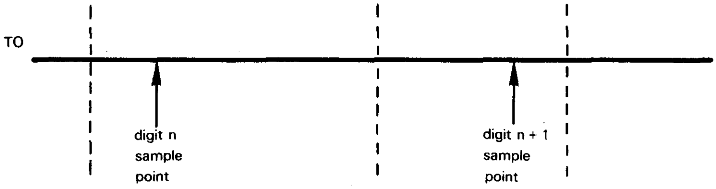
NRZI SERIAL DATA

SDLC protocol demands that all serial data is transmitted and received using "Non-Return to Zero Inverted" (NRZI) format. The μPD379 does not use NRZI serial data.

NRZI specifies that serial data signal "change of state" or "same state", occurring across single bit time intervals, define 0 bits and 1 bits, respectively. A 0 bit may be illustrated as follows:



A 1 bit may be illustrated as follows:

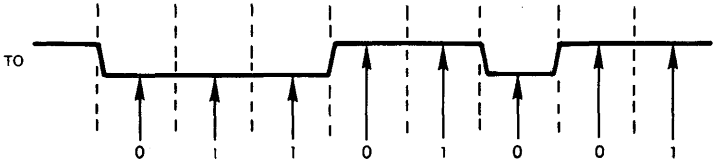


TRANSMITTING AND RECEIVING DATA UNDER SDLC PROTOCOL

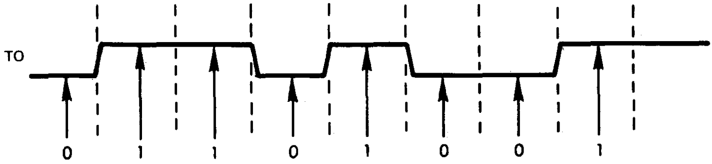
SDLC PROTOCOL

Since SDLC protocol has not been described in Volume I, "Basic Concepts", we will begin our discussion of μPD379 SDLC operation with some basic information on SDLC protocol itself. This discussion of SDLC is by no means a complete definition of SDLC protocol; however it does provide sufficient background to understand the operation of the μPD379 device.

The serial bit stream 01101001 would become the following serial signal:



The μPD379 outputs a high signal for a 1 bit and a low signal for a 0 bit. Thus 01101001 would become the following serial signal, as output by the μPD379:

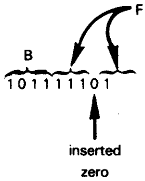


SDLC FLAG

In between transmission of valid data, but while the data link is not idle, a flag pattern ($7E_{16}$) is transmitted repeatedly.

SDLC ZERO INSERTION

Whenever five consecutive 1 bits appear in a single data character, a 0 bit is inserted following the fifth 1 bit. Thus, BF_{16} would become the following serial bit stream:



This is referred to as Zero insertion; it may be suppressed under program control. If Zero insertion is provided, then transmitting logic will create a dummy 0 bit whenever five consecutive 1 bits appear in a data stream, within a single 8-bit character. The receiving device must strip this synchronization pattern from data being received.

If an underflow occurs, that is, a valid data character is not ready to be transmitted when transmit logic requires one, then an abort pattern (FF_{16}) is output. The abort is followed by an output of flag patterns ($7E_{16}$) which terminate when the next valid character is ready to be transmitted.

When no data is being transmitted, an idle pattern is output. The idle pattern consists of 15 successive 1 bits. This may be likened to Marking in standard synchronous I/O protocol.

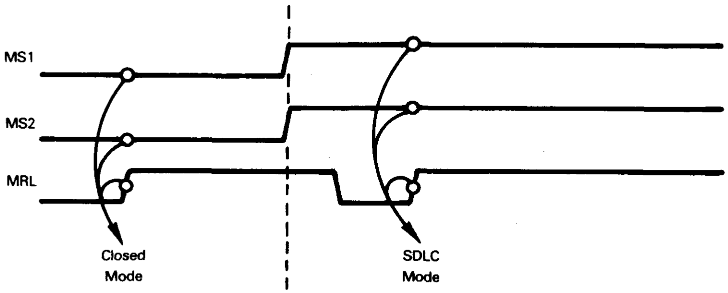
In order to avoid an abort, you may output a flag pattern ($7E_{16}$) when no valid data is available for transmission. The flag pattern will also be transmitted at the end of a valid data stream. The term Closing Flag is used to describe the use of the flag pattern as a filler or terminator in a serial data stream.

Serial data bits are transmitted and clocked under SDLC protocol exactly as described in Volume I, Chapter 5 for standard synchronous protocol.

μPD379 SDLC TRANSMIT

Let us now consider serial data transmission under SDLC protocol. Figure 4-41 illustrates operations involved.

You must enter SDLC mode from Closed mode either to transmit or receive serial data. The mode select sequence may be illustrated for entry into SDLC mode:



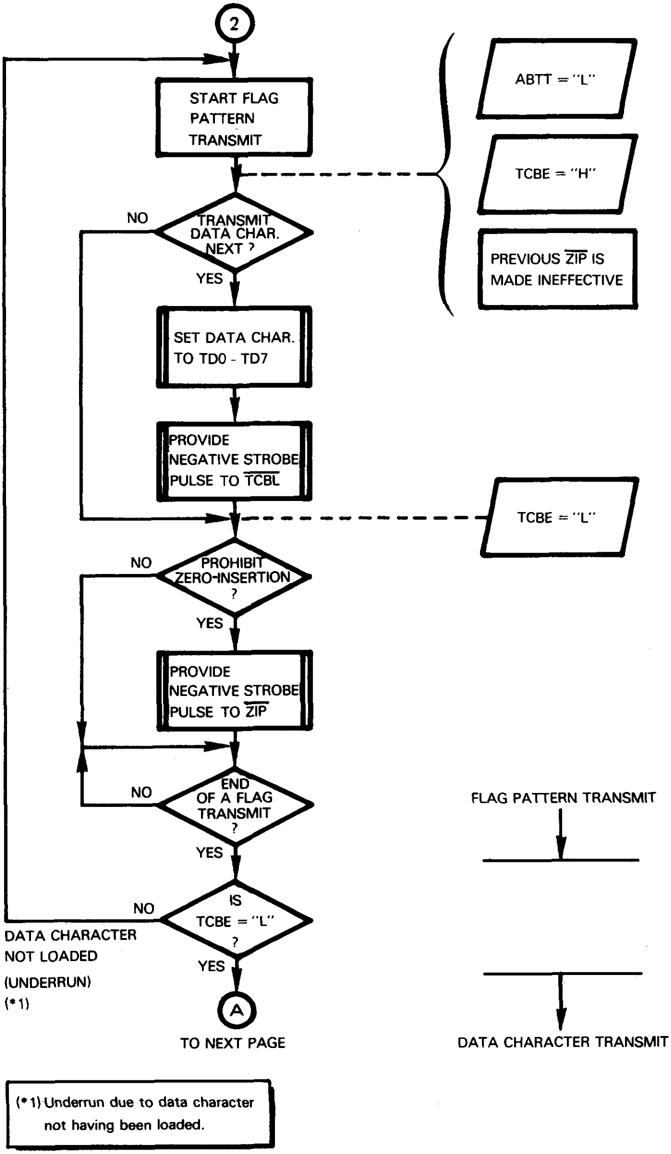

2
START FLAG PATTERN TRANSMIT
ABTT = "L"
TCBE = "H"
PREVIOUS $\overline{ZIP}$ IS MADE INEFFECTIVE
TRANSMIT DATA CHAR. NEXT ?
NO
YES
SET DATA CHAR. TO TD0 - TD7
PROVIDE NEGATIVE STROBE PULSE TO $\overline{TCBL}$
TCBE = "L"
PROHIBIT ZERO-INSERTION ?
NO
YES
PROVIDE NEGATIVE STROBE PULSE TO $\overline{ZIP}$
END OF A FLAG TRANSMIT ?
NO
YES
FLAG PATTERN TRANSMIT
IS TCBE = "L" ?
NO
YES
DATA CHARACTER NOT LOADED (UNDERRUN) (*1)
A
TO NEXT PAGE
DATA CHARACTER TRANSMIT
(*1) Underrun due to data character not having been loaded.

Figure 4-41. SDLC Transmit Operations Flow Chart

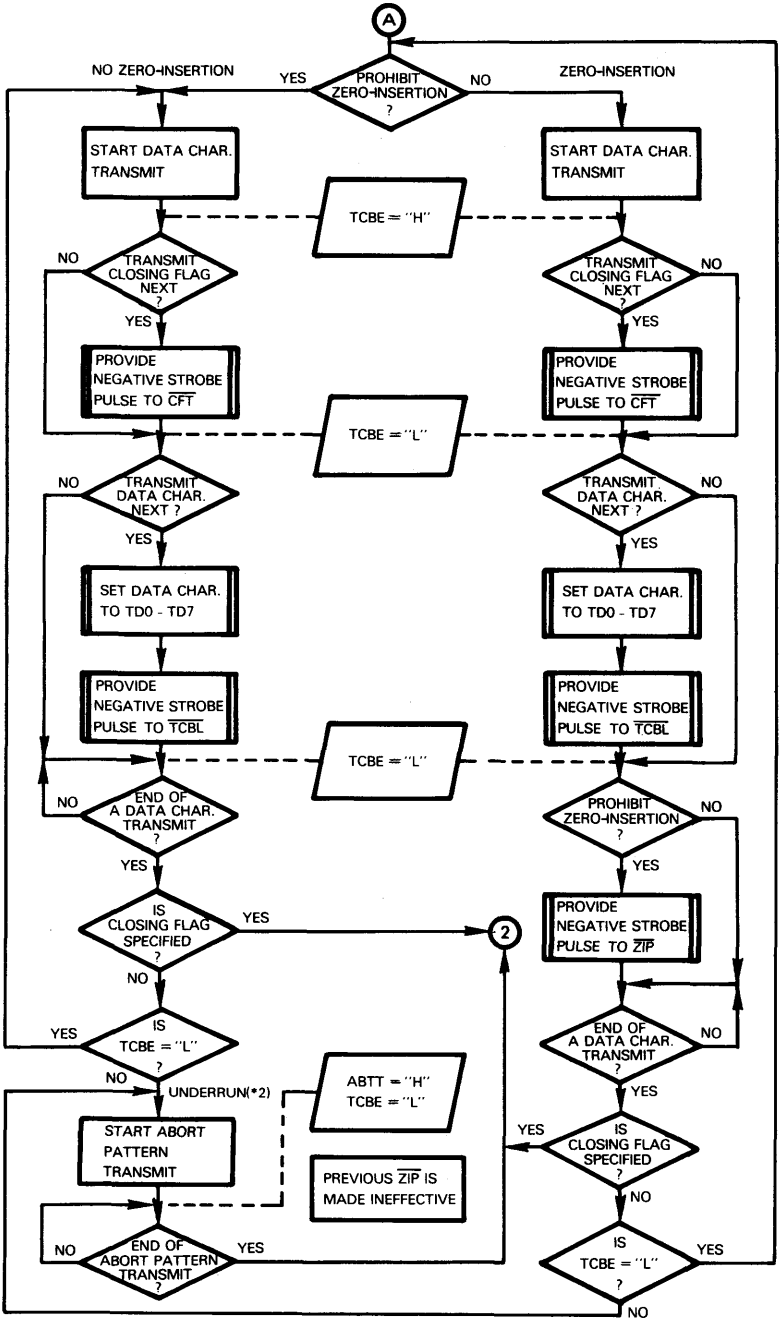
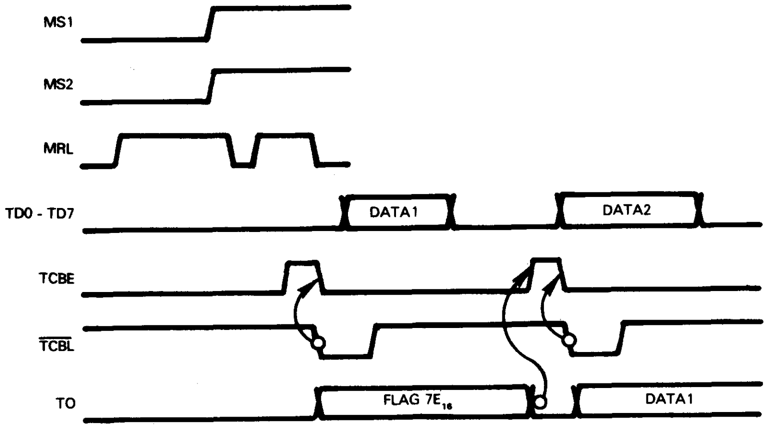

A
NO ZERO-INSERTION
YES
PROHIBIT ZERO-INSERTION ?
NO
ZERO-INSERTION
START DATA CHAR. TRANSMIT
START DATA CHAR. TRANSMIT
TCBE = "H"
NO
TRANSMIT CLOSING FLAG NEXT ?
YES
TRANSMIT CLOSING FLAG NEXT ?
NO
YES
PROVIDE NEGATIVE STROBE PULSE TO $\overline{CFT}$
PROVIDE NEGATIVE STROBE PULSE TO $\overline{CFT}$
TCBE = "L"
NO
TRANSMIT DATA CHAR. NEXT ?
YES
TRANSMIT DATA CHAR. NEXT ?
NO
YES
SET DATA CHAR. TO TD0 - TD7
SET DATA CHAR. TO TD0 - TD7
PROVIDE NEGATIVE STROBE PULSE TO $\overline{TCBL}$
PROVIDE NEGATIVE STROBE PULSE TO $\overline{TCBL}$
TCBE = "L"
NO
END OF A DATA CHAR. TRANSMIT ?
YES
PROHIBIT ZERO-INSERTION ?
NO
YES
PROVIDE NEGATIVE STROBE PULSE TO $\overline{ZIP}$
IS CLOSING FLAG SPECIFIED ?
YES
2
NO
END OF A DATA CHAR. TRANSMIT ?
NO
YES
YES
IS TCBE = "L" ?
NO
UNDERRUN(*2)
ABTT = "H"
TCBE = "L"
START ABORT PATTERN TRANSMIT
YES
IS CLOSING FLAG SPECIFIED ?
NO
PREVIOUS $\overline{ZIP}$ IS MADE INEFFECTIVE
NO
END OF ABORT PATTERN TRANSMIT ?
YES
IS TCBE = "L" ?
YES
NO

Figure 4-41. SDLC Transmit Operations Flow Chart (Continued)

As soon as SDLC mode has been entered, the μPD379 device will start transmitting flag patterns. TCBE will go high since the Transmit Character Buffer is empty. You output parallel data by writing to the μPD379 device as described for standard synchronous protocol. **Parallel data is output while a flag pattern is being transmitted** as follows:



TCBE is an output signal which you may use in any way to trigger transmittal of parallel data to the μPD379 device. TCBE high may generate an interrupt request, or TCBE may terminate at some external buffer whence it is read as status by the CPU.

$\overline{\text{TCBL}}$ will frequently be tied to the write control strobe $\overline{\text{MEMW}}$ or $\overline{\text{I/OW}}$.

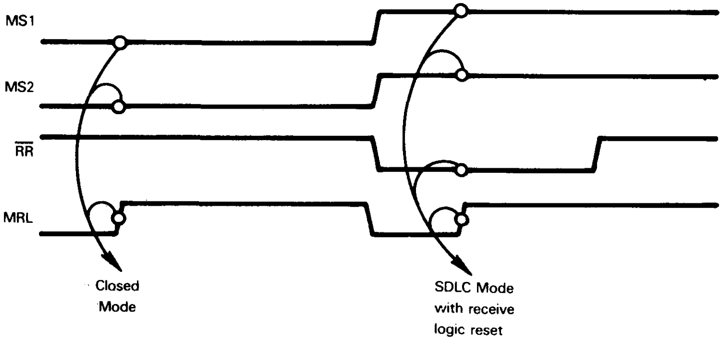
If Zero insertion has been prohibited, then the 6-bit synchronizing pattern will not be transmitted. **Zero insertion is prohibited by a low input via the $\overline{\text{ZIP}}$ signal.** Zero insertion will be prohibited beginning with the character following that within which the $\overline{\text{ZIP}}$ low pulse occurs. Zero insertions will be suppressed until the next Closing Flag or abort pattern is transmitted.

If a valid data byte will not be ready for transmission when needed by transmit logic, $\overline{\text{CFT}}$ must be input low to the μPD379 device in advance of the absent parallel data. You must also input a low $\overline{\text{CFT}}$ signal while transmitting the last data character of any valid data stream. The $\overline{\text{CFT}}$ low pulse causes flag patterns to be output upon termination of the currently transmitted character.

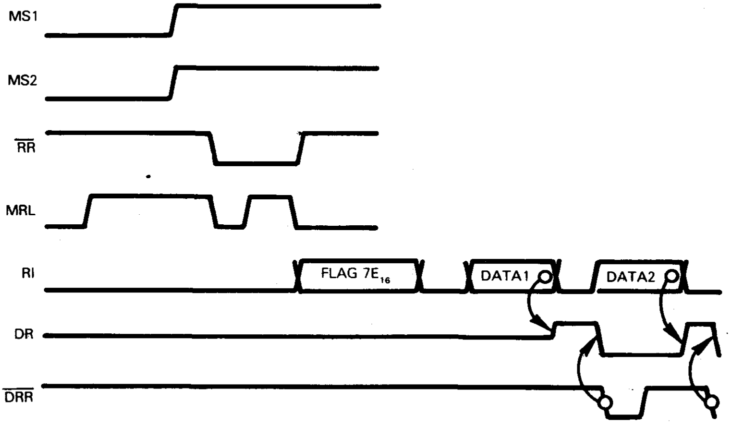
If $\overline{\text{CFT}}$ has not been input low to the μPD379 device and parallel data is not ready for transmittal, then TCBE will be high at the conclusion of the current data character's transmission. At this time **an Abort pattern will be output via TO together with a high signal output via the ABTT line.** The Abort pattern will be followed by a sequence of flag patterns.

μPD379
SDLC
RECEIVE

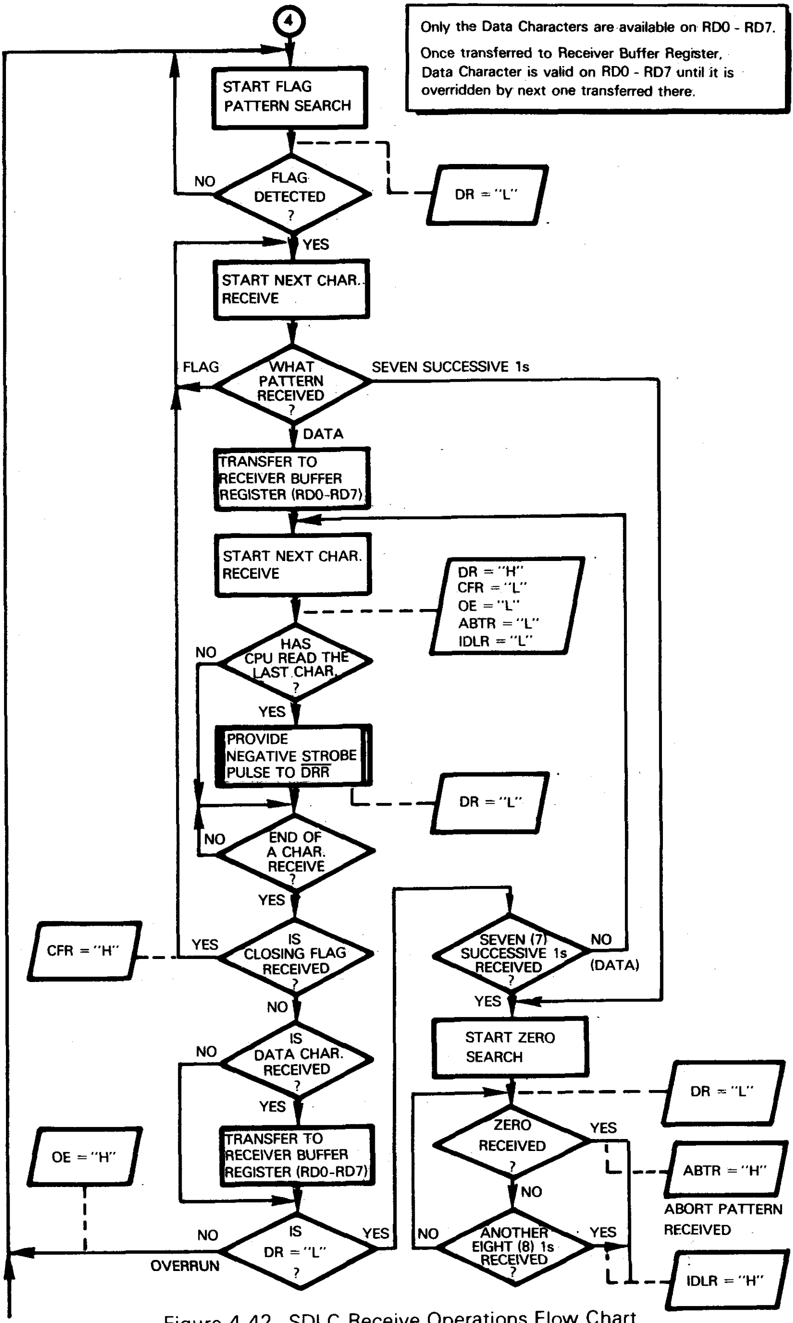
Now let us examine serial data being received under SDLC protocol. Figure 4-42 illustrates operations performed. Initialization of receive logic closely follows standard synchronous protocol which we have already described. **You must enter Closed mode and then select SDLC; and using $\overline{\text{RR}}$ you must reset receive logic.** Timing may be illustrated as follows:



Receive logic will now start searching the input stream for a flag pattern ($7E_{16}$). The flag pattern will not have the 6-bit Zero insertion synchronization code associated with it. This is how the μPD379 knows it is a flag. **As soon as a flag pattern has been detected, receive logic will start to assemble data bytes, stripping out the Zero insertion synchronization pattern, if present. As soon as a data character has been assembled and is in the Receive Data Buffer, DR goes high. DR remains high until reset by $\overline{\text{DRR}}$.** As described for standard synchronous receive logic, you will normally tie the read control strobe $\overline{\text{MEMR}}$ or $\overline{\text{I/OR}}$ to $\overline{\text{DRR}}$ in order to reset DR. Timing may be illustrated as follows:



If data in the Receive Data Buffer is not read before the next parallel byte is loaded into the Receive Data Buffer, then an overflow error occurs. The overflow error is identified by the OE signal being output high. The OE signal remains high until the Receive Data Buffer contents are read; the falling edge of $\overline{\text{DRR}}$ resets OE as well as DR.



4
Only the Data Characters are available on RD0 - RD7.
Once transferred to Receiver Buffer Register, Data Character is valid on RD0 - RD7 until it is overridden by next one transferred there.
START FLAG PATTERN SEARCH
FLAG DETECTED ?
NO
YES
DR = "L"
START NEXT CHAR. RECEIVE
WHAT PATTERN RECEIVED ?
FLAG
SEVEN SUCCESSIVE 1s
DATA
TRANSFER TO RECEIVER BUFFER REGISTER (RD0-RD7)
START NEXT CHAR. RECEIVE
DR = "H"
CFR = "L"
OE = "L"
ABTR = "L"
IDLR = "L"
HAS CPU READ THE LAST CHAR. ?
NO
YES
PROVIDE NEGATIVE STROBE PULSE TO DRR
DR = "L"
END OF A CHAR. RECEIVE ?
NO
YES
CFR = "H"
IS CLOSING FLAG RECEIVED ?
YES
NO
SEVEN (7) SUCCESSIVE 1s RECEIVED ?
NO
(DATA)
YES
START ZERO SEARCH
IS DATA CHAR. RECEIVED ?
NO
YES
DR = "L"
ZERO RECEIVED ?
YES
NO
ABTR = "H"
ABORT PATTERN RECEIVED
OE = "H"
TRANSFER TO RECEIVER BUFFER REGISTER (RD0-RD7)
IS DR = "L" ?
NO
YES
OVERRUN
ANOTHER EIGHT (8) 1s RECEIVED ?
NO
YES
IDLR = "H"

Figure 4-42. SDLC Receive Operations Flow Chart

The μPD379 device uses control outputs to identify special characters and conditions detected in the serial input stream. **An idle condition, identified by 15 consecutive 1 bits, causes IDLR to go high.** IDLR is reset on the rising edge of DR when data is subsequently read or on the rising edge of OE if an overrun error occurs.

If an Abort pattern (FF_{16}) is received then ABTR is output high. ABTR is reset on the rising edge of DR or OE.

When a valid closing flag ($7E_{16}$) is received CFR goes high. CFR is reset on the rising edge of DR or OE.

THE μPD369 UNIVERSAL ASYNCHRONOUS RECEIVER/TRANSMITTER

The μPD369 Universal Asynchronous Receiver/Transmitter is the second of two serial I/O devices manufactured by NEC in support of the 8080A (or any other microprocessor). It is a functional equivalent of the industry standard 1602 UART from Western Digital. The only difference is the 42-pin package which allows the standard 8080A power supplies to be used.

UART
1602
INDUSTRY
STANDARD

Capabilities of the μPD369 are very similar to asynchronous operation of the 8251; therefore the discussion of the μPD369 device which follows concentrates on differences between the 8251 and the μPD369. For a discussion of asynchronous serial I/O, see Volume I, Chapter 5. For a discussion of asynchronous serial I/O operations within an 8080 microcomputer system, see the preceding description of the 8251 USART.

There are three major differences between operating capabilities of the μPD369 and the 8251:

1) The 8251 allows transmit and receive clocks to operate at the baud rate, at 16x the baud rate or 64x the baud rate; the μPD369 clocks operate at 16x the baud rate only.
2) The 8251 has a primitive set of modem control signals — $\overline{\text{DSR}}$, $\overline{\text{DTR}}$, $\overline{\text{CTS}}$, $\overline{\text{RTS}}$. The μPD369 has no modem control signals.
3) The μPD369 can operate in full duplex or half duplex mode; having separate 8-bit parallel data input and output ports. It is quite simple to operate transmit and receive logic of the μPD369 simultaneously. The 8251 can be used in half duplex mode only.

Table 4-11 compares the operating characteristics of the μPD369 and the 8251 USART.

The μPD369, like the μPD379, uses input and output signals for Control and Status information. By way of contrast, the 8251 has Control and Status registers which must be written into and read as though they were data locations.

Figure 4-35 illustrates that part of our general microcomputer system logic which has been implemented on the μPD369 device.

The μPD369 device is packaged as a 42-pin DIP. It uses the three standard 8080A power supplies: +5V (V_{CC}), -5V (V_{BB}) and +12V (V_{DD}).

μPD369 DEVICE PINS AND SIGNALS

μPD369 device pins and signals are illustrated in Figure 4-43. Signals will be described with reference to the 8251 USART, the equivalent device. For a discussion of the manner in which various signals are used, refer to the 8251 USART device description.

As compared to the 8251, the μPD369 has a very primitive CPU interface. There is no chip select signal, nor are there individually addressable buffers within the device.

The parallel data output lines (RR0 - RR7) and the parallel data input lines (TR0 - TR7) will both be connected to the System Data Bus.

There are four signals which essentially act as read/write strobes: RRD, $\overline{\text{THRL}}$, CRL and SFD.

Table 4-11.A Comparison Of 8251/9551 And μPD369 Asynchronous Serial Data Transfer Capabilities

FEATURE	8251/9551	μPD369
Synchronous	Yes	No
Asynchronous	Yes	Yes
Bits Per Character	5, 6, 7 or 8	5, 6, 7 or 8
Error Detected	Parity, Overrun, Framing	Parity, Overrun, Framing
Stop Bits	1, 1-1/2 or 2	1 or 2, 1-1/2 with 5 data bits only
Baud Rate	DC to 9.6K baud	DC to 50K baud
Clock Frequency	x1, x16 or x64	x16
Duplex	Half duplex only	Half or full duplex
Modem Controls	$\overline{\text{DSR}}$, $\overline{\text{DTR}}$, $\overline{\text{CTS}}$, $\overline{\text{RTS}}$	None
Buffering	Double	Double

When RRD is low the output pins RR0 - RR7 are connected to the Receive register; RRD acts as a data read strobe.

A low pulse occurring at $\overline{\text{THRL}}$ loads data via TR0 - TR7 to a Transmit Holding buffer. $\overline{\text{THRL}}$ therefore acts as a data write strobe.

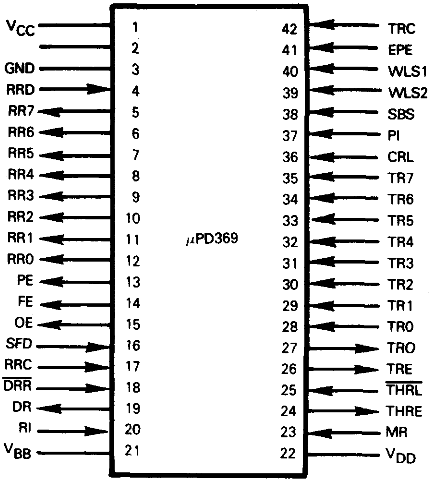
CRL acts as a control input strobe. A high level on CRL causes the various μPD369 control signal inputs to be sampled.

SFD acts as a status read strobe. A low level on SFD connects the status signals to appropriate logic within the μPD369, allowing status to be sampled.

RRD, $\overline{\text{THRL}}$, CRL and SFD must act as select signals as well as enable strobes, they must be generated as the AND of appropriate control signals plus memory or I/O port addresses.

MR is a master reset input. When this signal is pulsed high for at least 500 nanoseconds the μPD369 device is reset. Transmit and receive buffers are cleared; signals FE OE, PE and DRR are reset low while TRO, THRE, and TRE are set high.

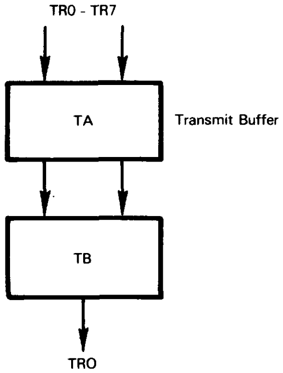
Let us consider in detail signals associated with transmit logic. The actual seria data output occurs via TRO. Serial data output is clocked by a clock signal inpu via TRC. TRC clock frequency must be 16x the baud rate for serial data being outpu via TRO.



PIN NAME	DESCRIPTION	TYPE
RR0 - RR7	Parallel received data output	Output
TR0 - TR7	Parallel transmitted data input	Input
MR	*Master reset*	Input
TRO	Serial data output	Output
TRC	Transmit clock	Input
PI	Parity inhibit	Input
SBS	Stop bits select	Input
WLS1,WLS2	Word length select	Input
EPE	Odd/even parity select	Input
CRL	Control load	Input
TRE	Transmit register empty	Output
THRE	Transmit holding register empty	Output
$\overline{\text{THRL}}$	Transmit holding register load	Input
RI	*Serial data input*	Input
RRC	Receiver clock	Input
PE	Parity error	Output
FE	Framing error	Output
OE	Overrun error	Output
SFD	Status flag disconnect	Input
DR	Data received	Output
$\overline{\text{DRR}}$	Data received reset	Input
RRD	Parallel data read strobe	Input
V_{BB},V_{CC},V_{DD},GND	Power and Ground	

Figure 4-43. μPD369 Universal Asynchronous Receiver/Transmitter Interface Signals And Pin Assignments

Transmitted data is double buffered. This may be illustrated as follows:



TRE and THRE are control signals associated with transmit logic. TRE goes high as soon as Register TB is empty. TRE remains high until data has been transmitted from TA to TB and serial data transmission resumes.

THRE goes high as soon as TA is empty. THRE remains high until fresh data has been written into Register TA; THRL is the control signal which causes this write to occur.

TRE and THRE are equivalent to the TxE and TxRDY signals, respectively, of the 8251 USART.

Serial data transmit operations are controlled by signals PI, EPE, SBS, WLS1 and WLS2. These signals cause options to be selected as follows:

PI: 1 = No parity bit. 0 = Add parity bit.
EPE: When PI is 0, EPE = 1 selects even parity. EPE = 0 selects odd parity.

WLS1, WLS2 select data bit lengths as follows:

WLS2	WLS1	
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

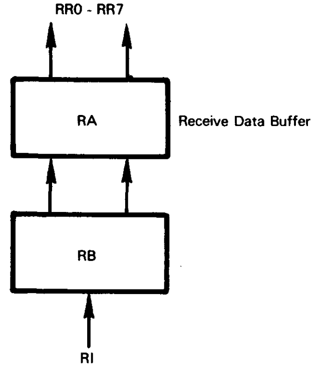
SBS: 1 = 2 stop bits. 0 = 1 stop bit.
With 5 data bits, SBS = 1 forces 1-1/2 stop bits — as required by Baudot code.

The above transmit option signals are sampled while CRL is high. CRL may be used as a control output strobe, or CRL may be held high — in which case the output option signals are constantly sampled. If you are using the μPD369 device with one fixed set of transmit options, then write CRL to +5V with PI, EPE, SBS, WLS1 and WLS2 tied appropriately to power and ground. If you are modifying transmit options under program control, then the option signals will be tied to appropriate lines of the System Data Bus; CRL must be created as the AND of the write enable control signal plus an appropriate address decoded off the Address Bus. This method of creating CRL has already been described.

Let us now look at receive logic.

Serial data is received via RI. The receive baud rate is controlled by the receiver clock signal RRC which must be 16x the baud rate.

Receive logic is double buffered. This may be illustrated as follows:

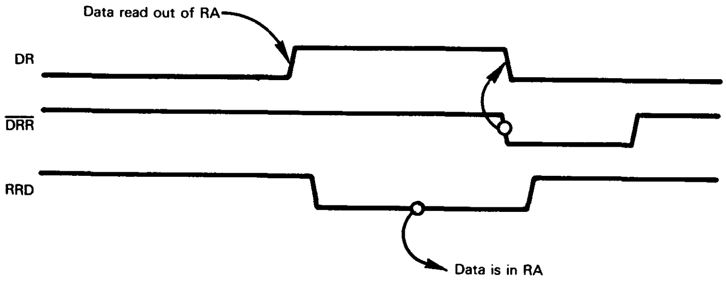


As soon as a byte of data has been assembled in RB and is loaded into RA, signal DR goes high. **Signal DR is** thus **equivalent to the RxRDY control signal of the 8251 USART.**

Receive control signals of the μPD369 are the same as the industry standard UARTs, based on the Western Digital 1602.

RRD, when high, disconnects the parallel data output pins RR0 - RR7 from the Receive Data Buffer RA. Conversely, RRD low connects the parallel data output pins to RA. RRD may therefore be used as a read control and device select strobe; this use of RRD has already been described.

But $\overline{\text{RRD}}$ does not reset to DR; $\overline{\text{DRR}}$ must be pulsed low before DR will be reset. This may be illustrated as follows:



$\overline{\text{DRR}}$ thus acts as a separate acknowledge signal. $\overline{\text{DRR}}$ and RRD may be derived from the same source.

Parity errors, framing errors and overrun errors in the received data are detected. These errors are identified by the PE, FE and OE status output signals. So long as SFD is high, the three output status signals are disconnected from status logic within the μPD369 device. SFD must be pulsed low in order to sample these status lines. SFD may therefore be derived as the AND of a read control strobe plus an appropriate memory address decode. This generation of SFD has been described. If SFD is derived in this fashion, then PE, FE and OE must be connected to lines of the System Data Bus; status will be read out of the μPD369 device by reading data out of the memory location or I/O port associated with SFD creation.

Figure 4-44 flowcharts a valid event sequence for using the μPD369.

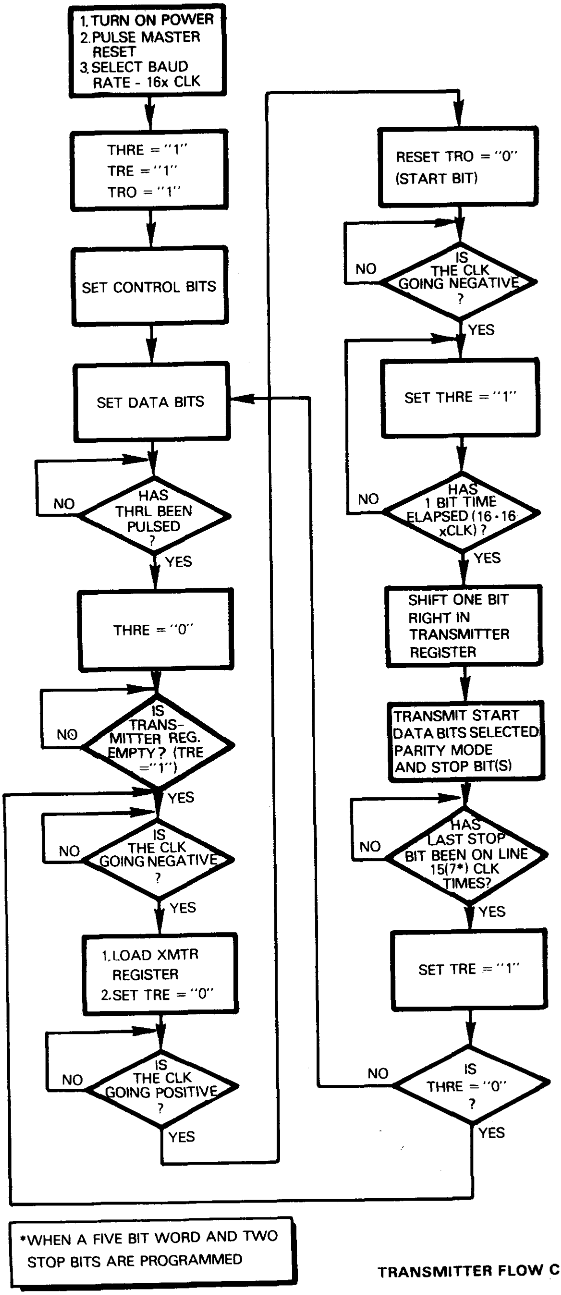

1. TURN ON POWER
2. PULSE MASTER RESET
3. SELECT BAUD RATE - 16x CLK
THRE = "1"
TRE = "1"
TRO = "1"
SET CONTROL BITS
SET DATA BITS
HAS THRL BEEN PULSED ?
NO
YES
THRE = "0"
IS TRANS-MITTER REG. EMPTY ? (TRE ="1")
NO
YES
IS THE CLK GOING NEGATIVE ?
NO
YES
1. LOAD XMTR REGISTER
2. SET TRE = "0"
IS THE CLK GOING POSITIVE ?
NO
YES
RESET TRO = "0" (START BIT)
IS THE CLK GOING NEGATIVE ?
NO
YES
SET THRE = "1"
HAS 1 BIT TIME ELAPSED (16 • 16 xCLK) ?
NO
YES
SHIFT ONE BIT RIGHT IN TRANSMITTER REGISTER
TRANSMIT START DATA BITS SELECTED PARITY MODE AND STOP BIT(S)
HAS LAST STOP BIT BEEN ON LINE 15(7*) CLK TIMES?
NO
YES
SET TRE = "1"
IS THRE = "0" ?
NO
YES
*WHEN A FIVE BIT WORD AND TWO STOP BITS ARE PROGRAMMED
TRANSMITTER FLOW CHART

Figure 4-44. Operations Flow Chart For μPD369 Asynchronous Serial I/O Operations

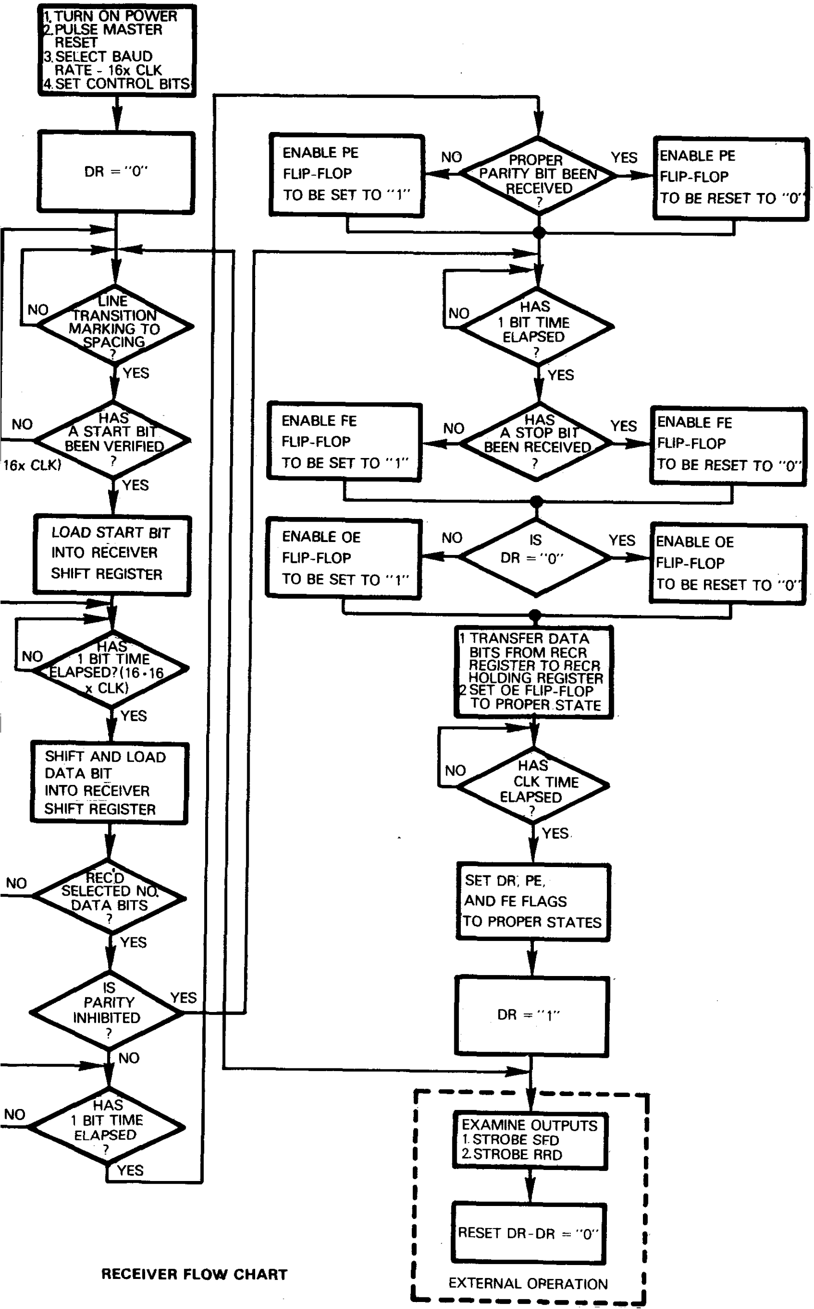

1. TURN ON POWER
2. PULSE MASTER RESET
3. SELECT BAUD RATE - 16x CLK
4. SET CONTROL BITS
DR = "0"
LINE TRANSITION MARKING TO SPACING ?
NO
YES
HAS A START BIT BEEN VERIFIED ?
NO
16x CLK)
YES
LOAD START BIT INTO RECEIVER SHIFT REGISTER
HAS 1 BIT TIME ELAPSED? (16-16 x CLK)
NO
YES
SHIFT AND LOAD DATA BIT INTO RECEIVER SHIFT REGISTER
REC'D SELECTED NO. DATA BITS ?
NO
YES
IS PARITY INHIBITED ?
YES
NO
HAS 1 BIT TIME ELAPSED ?
NO
YES
ENABLE PE FLIP-FLOP TO BE SET TO "1"
NO
PROPER PARITY BIT BEEN RECEIVED ?
YES
ENABLE PE FLIP-FLOP TO BE RESET TO "0"
HAS 1 BIT TIME ELAPSED ?
NO
YES
ENABLE FE FLIP-FLOP TO BE SET TO "1"
NO
HAS A STOP BIT BEEN RECEIVED ?
YES
ENABLE FE FLIP-FLOP TO BE RESET TO "0"
ENABLE OE FLIP-FLOP TO BE SET TO "1"
NO
IS DR = "0"
YES
ENABLE OE FLIP-FLOP TO BE RESET TO "0"
1 TRANSFER DATA BITS FROM RECR REGISTER TO RECR HOLDING REGISTER
2 SET OE FLIP-FLOP TO PROPER STATE
HAS CLK TIME ELAPSED ?
NO
YES
SET DR, PE, AND FE FLAGS TO PROPER STATES
DR = "1"
EXAMINE OUTPUTS
1. STROBE SFD
2. STROBE RRD
RESET DR-DR = "0"
EXTERNAL OPERATION
RECEIVER FLOW CHART

Figure 4-44. Operations Flow Chart For μPD369 Asynchronous Serial I/O Operations (Continued)

THE 8255 AND 8255A PROGRAMMABLE PERIPHERAL INTERFACE (PPI)

The 8255 PPI is a general purpose I/O device. Even though it has been designed for use within an 8080A microcomputer system, in fact, it can be used with almost any microprocessor.

The 8255 PPI provides 24 I/O pins, which may be configured as one, two or three I/O ports. 16 of the I/O pins are latched; eight of the I/O pins are simply buffered.

Figure 4-45 illustrates that part of our general microcomputer system logic which has been implemented on the 8255 PPI.

The 8255 PPI is packaged as a 40-pin DIP. It uses a single +5V power suppiy. All inputs and outputs are TTL compatible. The device is implemented using n-Channel MOS technology.

There are two differences between the 8255A and the 8255.

8255 AND 8255A DIFFERENCES

The 8255 is limited to working with 8080A devices having clock signals with 500 nanosecond cycle times or longer. The 8255A will work with clock signals as short as 250 nanoseconds — that is, with the fastest 8080A CPUs on the market today.

The 8255, when reset, outputs the $\overline{\text{OBF}}$ control signal low in Mode 1; external logic thus has no initial data request control signal. The 8255A, when reset, outputs $\overline{\text{OBF}}$ high in Mode 1; thus external logic receives an initial request to transmit data to the 8255A.

Electrical characteristics of the 8255 and the 8255A differ significantly — as defined in the data sheets at the end of this chapter.

8255 PPI PINS AND SIGNALS

The 8255 pins and signals are illustrated in Figure 4-46. Pins and signals are deceptively straightforward; the power of this device lies in its internal architecture and operating features.

Consider first the various Data Busses.

D0 - D7 represent the bidirectional Data Bus, via which all communications between the CPU and the 8255 occur.

PA0 - PA7, PB0 - PB7 and PC0 - PC7 represent Data Busses connected to the three 8 bit I/O ports A, B and C. All parallel I/O communications with external logic occur over one of these three I/O port busses.

There are three device select pins: $\overline{\text{CS}}$, A0 and A1.

8255 PPI SELECT LOGIC

$\overline{\text{CS}}$ is the master chip select. When a low signal is input at this pin, the 8255 is selected. A0 and A1 allow one of four registers within the 8255 to be addressed. $\overline{\text{CS}}$, A0 and A1 combine to address individual registers within the 8255 as follows:

$\overline{\text{CS}}$	A0	A1	Selected
0	0	0	I/O Port A
0	1	0	I/O Port B
0	0	1	I/O Port C
0	1	1	A Control, write only buffer
1	X	X	8255 not selected

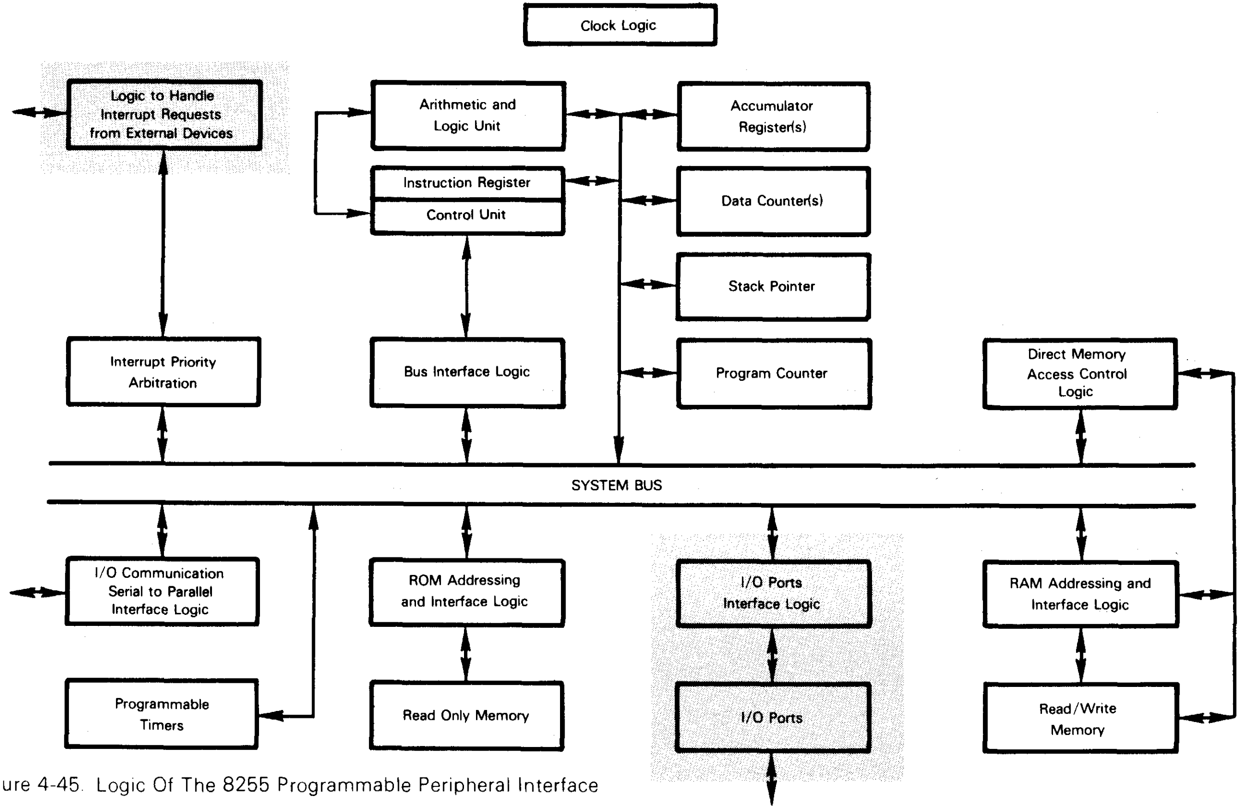
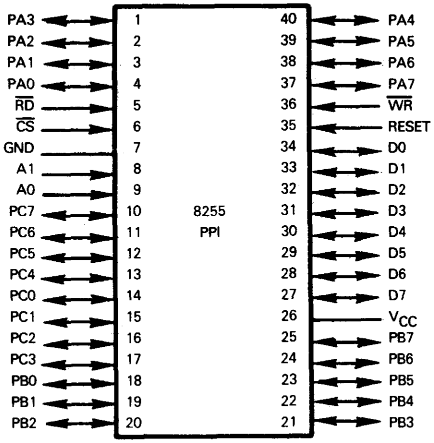

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Instruction Register
Control Unit
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

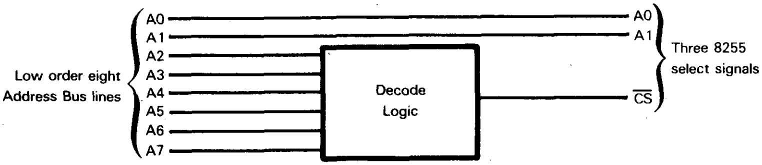
Figure 4-45. Logic Of The 8255 Programmable Peripheral Interface



PIN NAME	DESCRIPTION	TYPE
D0 - D7	Bidirectional Data Bus	Bidirectional
PA0 - PA7	Eight I/O pins, designated as Port A	Bidirectional
PB0 - PB7	Eight I/O pins, designated as Port B	Bidirectional
PC0 - PC7	Eight I/O pins, designated as Port C upper and Port C lower	Bidirectional
$\overline{RD}$	Read from device control	Input
$\overline{WR}$	Write to device control	Input
RESET	System reset	Input
$\overline{CS}$	Device select	Input
A0,A1	I/O port select	Input
V_{CC},GND	Power and Ground	

Figure 4-46. 8255 Programmable Peripheral Interface Device Signals And Pin Assignments

Thus, an 8255 PPI appears to the CPU either as four I/O ports, or as four memory locations. This is how select logic might access an 8255 PPI as four I/O ports:



Suppose $\overline{CS}$ is output low when decode logic receives 010011 as input, and $\overline{CS}$ is output high otherwise. The 8255 PPI will respond to I/O port addresses as follows:

A7	A6	A5	A4	A3	A2	A1	A0	
0	1	0	0	1	1	0	0	$4C_{16}$ selects I/O Port A
0	1	0	0	1	1	0	1	$4D_{16}$ selects I/O Port B
0	1	0	0	1	1	1	0	$4E_{16}$ selects I/O Port C
0	1	0	0	1	1	1	1	$4F_{16}$ selects Control buffer

If the 8255 PPI is accessed as four memory addresses, then $\overline{CS}$ would be decoded out of A2 through A15.

8255 PPI READ AND WRITE CONTROL

$\overline{\text{RD}}$ and $\overline{\text{WR}}$ are active low control signals which enable read and write operations within a selected 8255 PPI register. When $\overline{\text{RD}}$ is low, the addressed register contents are read to the CPU. When $\overline{\text{WR}}$ is low, the CPU will write into the addressed register.

When the 8255 PPI is selected as four I/O ports, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ will be connected to the $\overline{\text{I/OR}}$ and $\overline{\text{I/OW}}$ control signals, respectively, output by the 8228 System Controller.

If the 8255 PPI is accessed as four memory locations, $\overline{\text{RD}}$ must be connected to $\overline{\text{MEMR}}$ and $\overline{\text{WR}}$ must be connected to $\overline{\text{MEMW}}$.

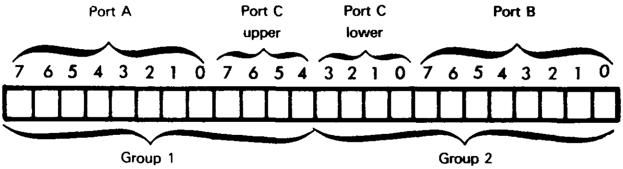
Of course, if there is no 8228 System Controller in the configuration, you will have to generate $\overline{\text{RD}}$ and $\overline{\text{WR}}$ by whatever means are available.

The only signals remaining to be explained are RESET, power and ground; and these are all self-evident. RESET is a typical system reset signal. When input high this clears all 8255 registers.

Power must be connected to a single +5V power supply.

8255 PPI OPERATING MODES

The four 8255 PPI addressable locations consist of three I/O ports and a Control buffer. I/O Port C may be divided into two 4-bit I/O ports, accessed as a single addressable unit. Ports may be illustrated as follows:



Ports are accessed as three independent, 8-bit parallel units (Ports A, B and C).

Any eight pins from I/O Ports B and C may be used to source 1 milliamp of current at 1.5 volts. This allows Darlington transistors to be driven directly. Darlington transistors are used by devices such as printers and high voltage displays.

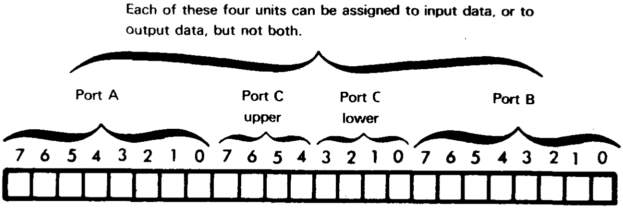
Some 8255 options allow ports to be divided into two groups, illustrated as Group 1 and Group 2 above. In a group, Port A or Port B acts as a data transfer port, while selected bits of Port C function as control signals.

You identify the way in which ports will operate by loading an appropriate command into the Control buffer.

You have three options.

8255 PPI MODE 0

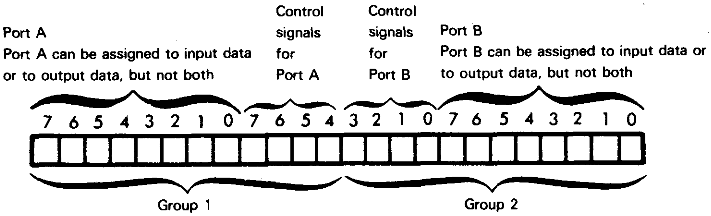
Mode 0 represents simple I/O. *In this mode each I/O port can be* defined as a simple input port, or a simple output port. This may be illustrated as follows:


Each of these four units can be assigned to input data, or to output data, but not both.
Port A
Port C upper
Port C lower
Port B
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0

Observe that Port C upper and lower halves may be defined separately as input and/or output ports.

8255 PPI MODE 1

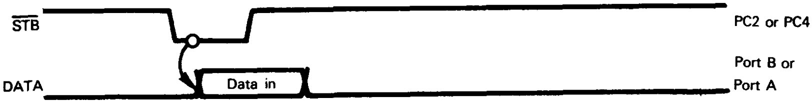
Mode 1 allows I/O Ports A and B to be defined as either input or output ports, with Port C providing handshaking control signals. Mode 1 may be illustrated as follows:


Port A
Port A can be assigned to input data or to output data, but not both
Control signals for Port A
Control signals for Port B
Port B
Port B can be assigned to input data or to output data, but not both
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
Group 1
Group 2

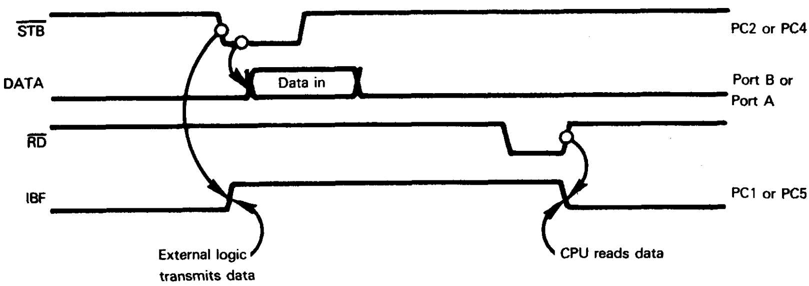
In Mode 1, you must define Port A, Port B, Port C upper and Port C lower as input or output ports; these are four independent data transfer direction definitions. In the case of Port C upper and Port C lower, some pins are set aside as control signals — as we are about to describe. Port C control signals ignore port input or output definitions; unassigned Port C pins operate as either input or output signal carriers, depending on the direction defined for the Port C upper and Port C lower.

We will now examine the simple handshaking provided by Mode 1.

If Port A or B has been assigned as an input port, then external logic must be able to indicate that new input data has been placed at the input port; an input strobe control signal ($\overline{\text{STB}}$) serves this purpose:

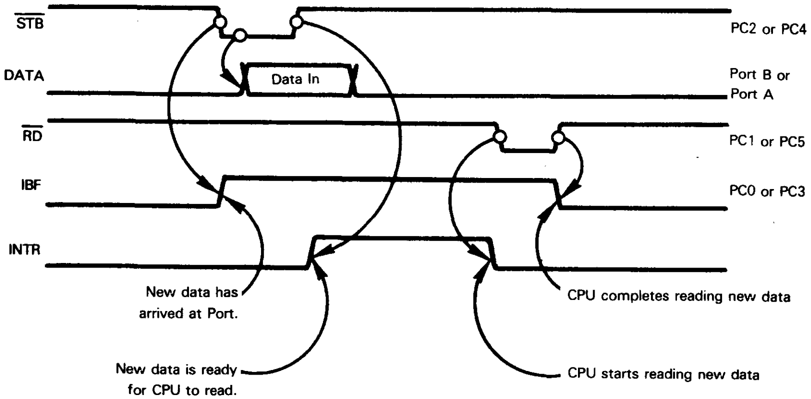


$\overline{\text{STB}}$ low causes new data to be strobed into the port's input buffer. A second control signal (IBF) is output high while the input buffer is full: that is, from the instant $\overline{\text{STB}}$ strobes data into the buffer, until the CPU executes an instruction which reads the buffer contents:



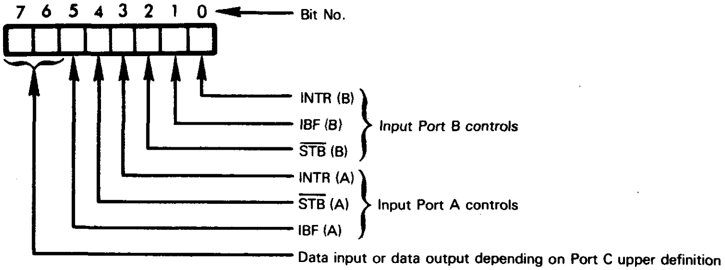
$\overline{\text{RD}}$ low identifies the execution of an instruction which reads the input port's buffer contents. Thus while IBF is high, there is unprocessed data in the input port's buffer. External logic must not input new data while IBF is high; if it does, prior unprocessed data will be destroyed.

How is the CPU to know when fresh data is in an I/O port's buffer and must be read? IBF will not do this job. IBF goes high when data starts entering the I/O port buffer — at which point external logic must be told not to try sending any more data. The CPU must not try to read an I/O port's buffer contents until data transfer into the port is complete. Each port is therefore provided with a control output which can serve to interrupt the CPU when data has been loaded into an I/O port's buffer:

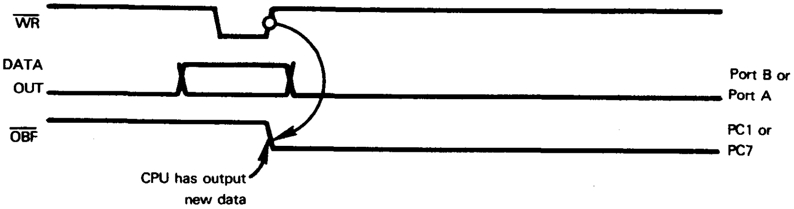

STB
DATA
RD
IBF
INTR
Data In
PC2 or PC4
Port B or Port A
PC1 or PC5
PC0 or PC3
New data has arrived at Port.
New data is ready for CPU to read.
CPU completes reading new data
CPU starts reading new data

I/O Port C pins 7 and 6 do not handle input control signals; they may be used in any way to handle bidirectional data.

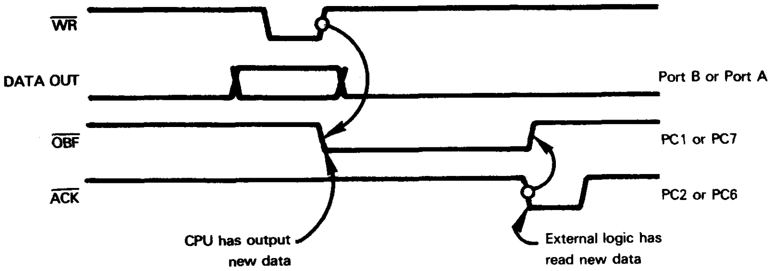
In summary, Port C provides Mode 1 input control signals as follows:



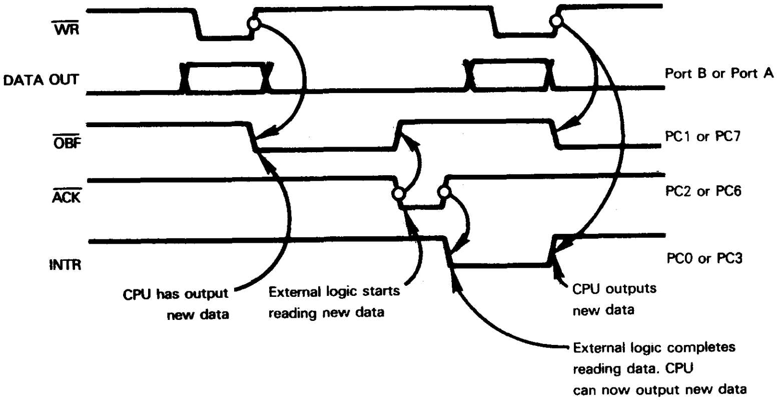
Now look at the control signals needed by a Mode 1 output port. As soon as the CPU writes data into an I/O port buffer, a control signal $\overline{OBF}$ is output low, telling external logic that new data is ready to be read:



External logic must acknowledge that it has read the new data by inputting $\overline{\text{ACK}}$ low:

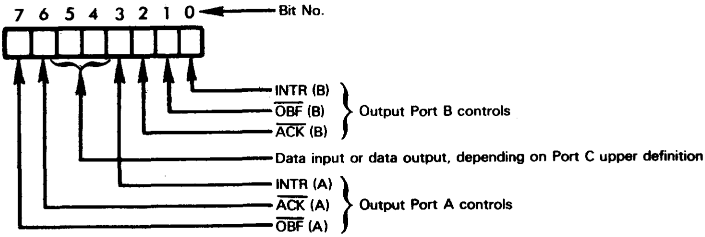


If the CPU again outputs new data while $\overline{\text{OBF}}$ is low, then it will destroy the prior output data — which external logic has not yet read. An interrupt control, similar to the input interrupt control, therefore tells the CPU when it is safe to output new data:



In output Mode 1, Port C pins 4 and 5 do not handle control signals; they may be used in any way to handle bidirectional data.

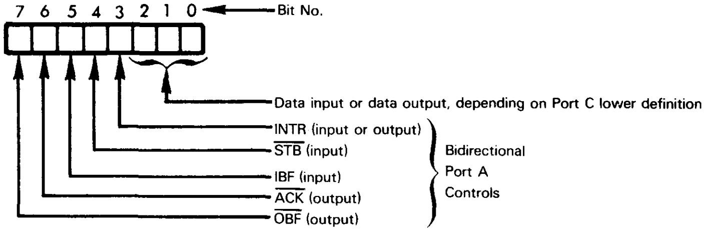
In summary, Port C provides Mode 1 output signals as follows:


7 6 5 4 3 2 1 0
Bit No.
INTR (B)
$\overline{\text{OBF}}$ (B)
$\overline{\text{ACK}}$ (B)
Output Port B controls
Data input or data output, depending on Port C upper definition
INTR (A)
$\overline{\text{ACK}}$ (A)
$\overline{\text{OBF}}$ (A)
Output Port A controls

When an 8255 PPI is reset, all I/O port bits — and associated control signals — are reset. This is very significant when operating the 8255 PPI in Mode 1 or 2 since **external logic will initially receive a low $\overline{\text{OBF}}$ signal. Thus, external logic will initially think that the 8255 output buffer is full,** and will not transmit data. You must read and discard the first data byte following a Reset before external logic will receive a high $\overline{\text{OBF}}$ signal, thus initiating normal handshaking.

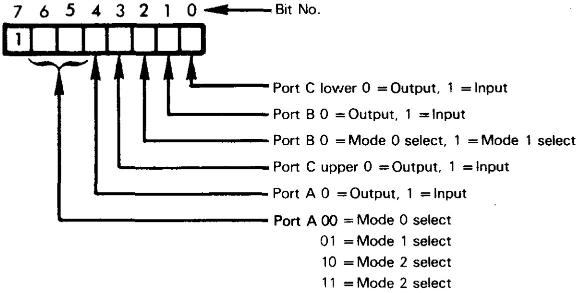
Following a Reset, the new 8255A initially outputs $\overline{\text{OBF}}$ high; data transfer under Mode 1 or Mode 2 will begin without program intervention.

Mode 2 allows I/O Port A to act as a bidirectional data port. Bidirectional handshaking control signals are provided by I/O Port C. Bidirectional I/O control signals are a simple combination of the Mode 1 input and output control signals, and may be illustrated as follows:

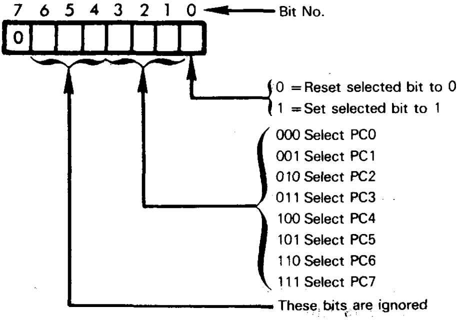

7 6 5 4 3 2 1 0
Bit No.
Data input or data output, depending on Port C lower definition
INTR (input or output)
STB (input)
IBF (input)
ACK (output)
OBF (output)
Bidirectional Port A Controls

Note carefully that with a single exception of Mode 2, no restrictions are placed on the mode combinations which can be selected within an 8255 PPI. Table 4-12 summarizes allowed mode combinations and identifies the resulting I/O port pin assignments.

You make I/O port assignments by writing a control word to the 8255 Control port address. This is how the Control Word is interpreted:


7 6 5 4 3 2 1 0
Bit No.
1
Port C lower 0 = Output, 1 = Input
Port B 0 = Output, 1 = Input
Port B 0 = Mode 0 select, 1 = Mode 1 select
Port C upper 0 = Output, 1 = Input
Port A 0 = Output, 1 = Input
Port A 00 = Mode 0 select
01 = Mode 1 select
10 = Mode 2 select
11 = Mode 2 select

At any time you can also write a Control Word to the 8255 Control port address in order to set or reset individual bits of Port C. If a Control Word is to modify I/O Port C bits, then it must have the following format:



The 8255 PPI has no Status register; instead you should read the contents of I/O Port C, and interpret bits as shown in Table 4-12.

Table 4-12. Allowed 8255 I/O Port Assignments And Pin Utilization

PORT A		PORT B		PORT C (Individual Bit Assignments)							
Mode	Assignment	Mode	Assignment	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
0	I or O	0	I or O	I or O	I or O	I or O	I or O	I or O	I or O	I or O	I or O
0	I or O	1	I	I or O	I or O	I or O	I or O	I or O	$\overline{\text{STB}}$ (B)	IBF (B)	INTR (B)
0	I or O	1	O	I or O	I or O	I or O	I or O	I or O	$\overline{\text{ACK}}$ (B)	$\overline{\text{OBF}}$ (B)	INTR (B)
1	I	0	I or O	I or O	I or O	IBF (A)	$\overline{\text{STB}}$ (A)	INTR (A)	I or O	I or O	I or O
1	O	0	I or O	$\overline{\text{OBF}}$ (A)	$\overline{\text{ACK}}$ (A)	I or O	I or O	INTR (A)	I or O	I or O	I or O
1	I	1	I	I or O	I or O	IBF (A)	$\overline{\text{STB}}$ (A)	INTR (A)	$\overline{\text{STB}}$ (B)	IBF (B)	INTR (B)
1	I	1	O	I or O	I or O	IBF (A)	$\overline{\text{STB}}$ (A)	INTR (A)	$\overline{\text{ACK}}$ (B)	$\overline{\text{OBF}}$ (B)	INTR (B)
1	O	1	I	$\overline{\text{OBF}}$ (A)	$\overline{\text{ACK}}$ (A)	I or O	I or O	INTR (A)	$\overline{\text{STB}}$ (B)	IBF (B)	INTR (B)
1	O	1	O	$\overline{\text{OBF}}$ (A)	$\overline{\text{ACK}}$ (A)	I or O	I or O	INTR (A)	$\overline{\text{ACK}}$ (B)	$\overline{\text{OBF}}$ (B)	INTR (B)
2	I/O	0	I	$\overline{\text{OBF}}$ (A)	$\overline{\text{ACK}}$ (A)	IBF (A)	$\overline{\text{STB}}$ (A)	INTR (A)	I or O	I or O	I or O
2	I/O	0	O	$\overline{\text{OBF}}$ (A)	$\overline{\text{ACK}}$ (A)	IBF (A)	$\overline{\text{STB}}$ (A)	INTR (A)	I or O	I or O	I or O
2	I/O	1	I	$\overline{\text{OBF}}$ (A)	$\overline{\text{ACK}}$ (A)	IBF (A)	$\overline{\text{STB}}$ (A)	INTR (A)	$\overline{\text{STB}}$ (B)	IBF (B)	INTR (B)
2	I/O	1	O	$\overline{\text{OBF}}$ (A)	$\overline{\text{ACK}}$ (A)	IBF (A)	$\overline{\text{STB}}$ (A)	INTR (A)	$\overline{\text{ACK}}$ (B)	$\overline{\text{OBF}}$ (B)	INTR (B)

I means input only.
O means output only.
I or O means input or output, but not both, depending on how port has been defined.
I/O means bidirectional.

8255 PPI USE PROBLEMS

There are some non-obvious features of the 8255 PPI which you should be aware of.

When I/O Port A or B is assigned as an output port, individual port pins may output high or low signals; this is accomplished by writing data to the I/O port before assigning it as an output port. When you assign either or both halves of I/O Port C as an output port, all signals will initially be low.

When operating in Mode 0, beware of assigning upper and lower halves of I/O Port C to input and output. Some users have reported that when you read from I/O Port C you may alter output signal levels; when you write to I/O Port C you may write into the input signal's buffer, modifying any bit settings created by input signals. This is a problem that does not always appear.

In Modes 1 and 2 you must use the Control code to write to I/O Port C, one bit at a time. You cannot write an 8-bit pattern to I/O Port C. Also, in Modes 1 and 2, all control signals must be initialized by writing to the appropriate bit of I/O Port C using the appropriate Control code.

8255 PPI interrupt logic must be enabled. In Modes 1 and 2 observe that interrupt requests may be output via INTRA and INTRB (pins 3 and 0 of I/O Port C, respectively). These interrupt requests will only occur if you first output a high level to these bits of I/O Port C using the appropriate Control codes.

THE 8212 8-BIT INPUT/OUTPUT PORT

This is a simple, nonprogrammable, yet versatile and economic I/O device, capable of providing total I/O interface logic in small microcomputer systems; in larger microcomputer systems consider the 8212 I/O port for specific local I/O needs.

This device has a data latch, buffer and interrupt logic. The data latch consists of eight D-type flip-flops whose Q outputs are connected to tristate, non-inverting output buffers. In order to make effective use of the 8212 I/O port, it is important to understand the logic of this device; therefore it is illustrated in Figure 4-47. We will refer to this figure for clarification when describing individual signals.

In order to understand the latching and buffering of the 8212 I/O port, **note that the eight flip-flops are sensitive to the level of the C input and not to signal transitions.** When the clock input C is high, the Q outputs will track the D inputs. When the clock inputs C are low, the Q outputs hold their prior levels and are disconnected from the D inputs.

Interrupt logic provided by the 8212 I/O port is functionally similar to the handshaking logic described for Modes 1 and 2 of 8255 PPI operation.

Figure 4-45, which defines 8255 PPI logic bounds, also illustrates that portion of our general microcomputer systems which can be implemented using an 8212 I/O port.

8212 I/O PORT PINS AND SIGNALS

8212 I/O port pins and signals are illustrated in Figure 4-48. We will describe these pins and signals with reference to Figure 4-47.

DI0 - DI7 are data input pins. They are connected to the eight data inputs of the D-type flip-flops within the 8212 I/O port. The Q outputs of these flip-flops are connected to tristate buffers, which in turn output data via pins DO0 - DO7.

Data flow is controlled by the mode select (MD), strobe (STB), and device select logic.

Before this device can be selected, a low input must occur at $\overline{\text{DS1}}$ and a high input must occur at DS2.

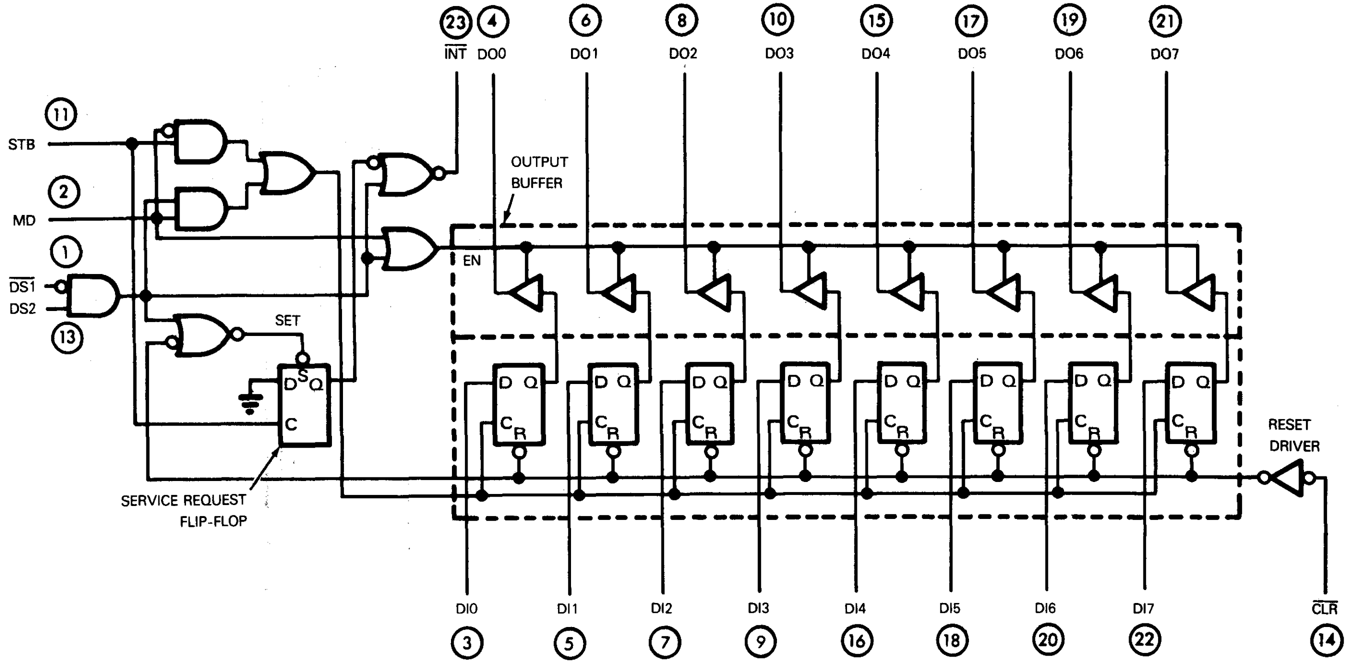
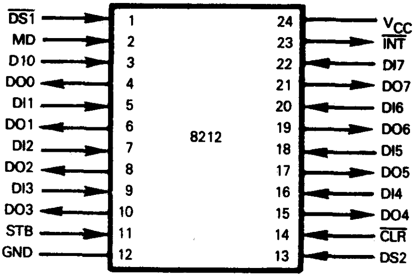


Figure 4-47. Internal Logic Of The 8212 I/O Port



PIN NAME	DESCRIPTION	TYPE
DI0 - DI7	Input Data Bus	Input
DO0 - DO7	Output Data Bus	Output
$\overline{DS1}$,DS2	Device select	Input
MD	Mode select	Input
STB	Data strobe	Input
$\overline{CLR}$	Device Clear	Input
$\overline{INT}$	Interrupt request	Output
V_{CC}	Power	
GND	Ground	

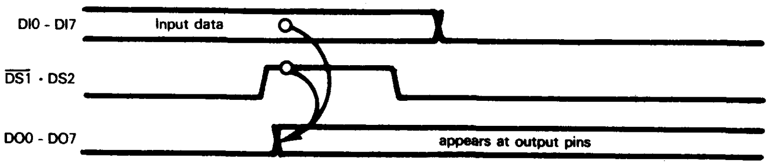
Figure 4-48. 8212 Input/Output Port Signals And Pin Assignments

The eight D-type flip-flops may be clocked either by device select logic, or by the strobe input (STB), depending on the condition of the mode input (MD).

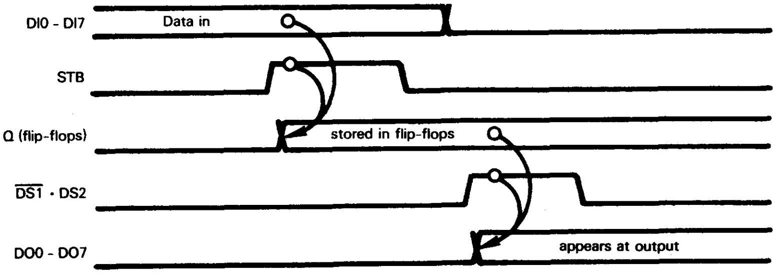
If MD is input high, then device select will clock the flip-flops, and the output buffers are permanently enabled by the high MD level; hence **this is defined as output mode.**

If the MD signal is input low, then it negates chip select logic as a contributor to the flip-flop clock; and it does not enable the output buffers. Now the flip-flop clock inputs must be derived by the strobe input STB, and device select enables the output buffers. With this configuration STB can strobe data into the flip-flop at any time, whether or not the device has been selected. Device select enables the output buffers and occurrence of data at pins DO0 - DO7; hence **MD low is referred to as an input mode.**

To summarize, when MD is high, device select logic strobes data at DI0 - DI7 into the flip-flops, and data flows through immediately to DO0 - DO7:

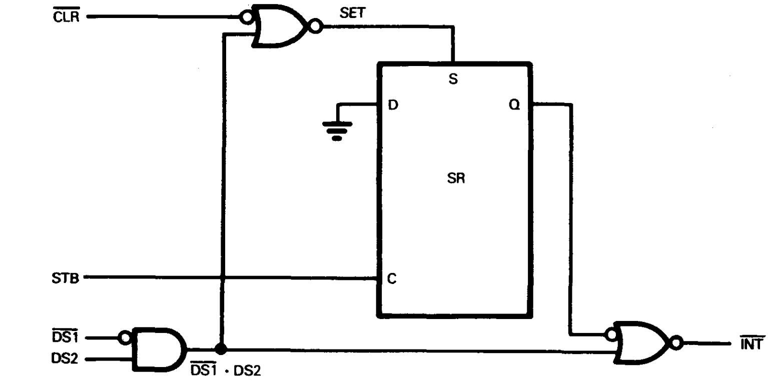


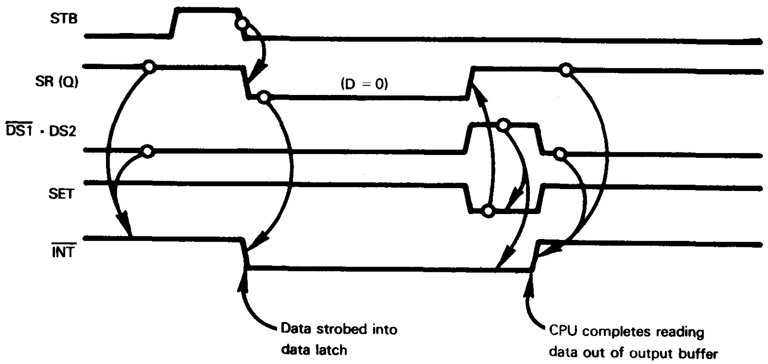
When MD is low, STB may at any time strobe data at DI0 - DI7 into the D-type flip-flops; the 8212 I/O port need not be selected for this to occur. Device select logic controls the instant at which new data is output via DO0 - DO7:



The D-type flip-flops may be reset at any time by setting the $\overline{\text{CLR}}$ input low.

The interrupt request signal $\overline{\text{INT}}$ functions much as it does in 8255 PPI Modes 1 and 2 operation. When the 8212 I/O port is used for data input, recall that STB high clocks data from DI0 - DI7 into the flip-flops. The $\overline{\text{CPU}}$ must be informed of this event. The high-to-low transition of STB clocks the service request flip-flop SR, causing SR (Q) to output low, since SR (D) is tied to ground. SR (Q) low is inverted at the $\overline{\text{INT}}$ NOR gate, therefore the NOR gate receives a high input and $\overline{\text{INT}}$ is output low. Thus **$\overline{\text{INT}}$ informs the CPU that data has been strobed into the data flip-flops.** Assuming that $\overline{\text{CLR}}$ is high, the device select logic outputs a second high to the NOR gate which drives the SR flip-flop SET input. SET is therefore input low, forcing SR (Q) high; SR (Q) high translates into a low input to the NOR gate generating $\overline{\text{INT}}$, which is forced high in consequence. Device select logic is output high when the contents of the 8212 I/O port is read, therefore **the process of reading port contents sets $\overline{INT}$ high again.** This may be illustrated as follows:





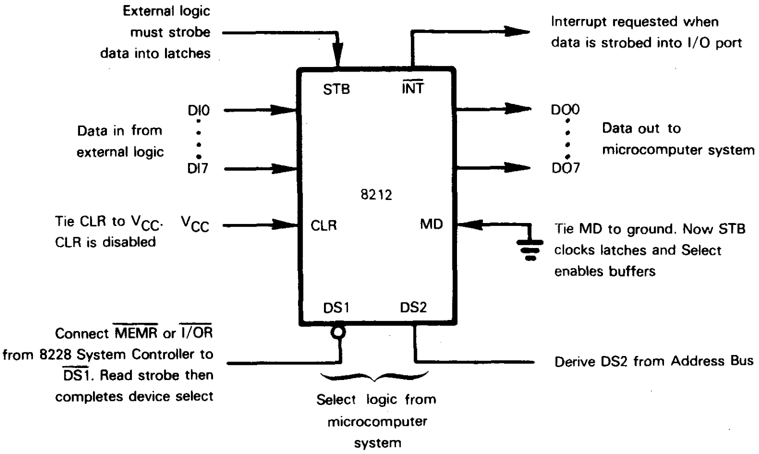
STB
SR (Q)
(D = 0)
$\overline{DS1} \cdot DS2$
SET
$\overline{INT}$
Data strobed into data latch
CPU completes reading data out of output buffer

8212 I/O PORT UTILIZATION OPTIONS

We will now look at some of the ways in which an 8212 I/O port can be used. Let us start by trying to emulate 8255 PPI modes — for a single 8255 I/O port.

Consider the 8212 I/O port being used for data input:

8212 I/O PORT USED FOR INPUT WITH HANDSHAKING


External logic must strobe data into latches

Interrupt requested when data is strobed into I/O port

Data in from external logic

Data out to microcomputer system

Tie CLR to V_{CC}. CLR is disabled

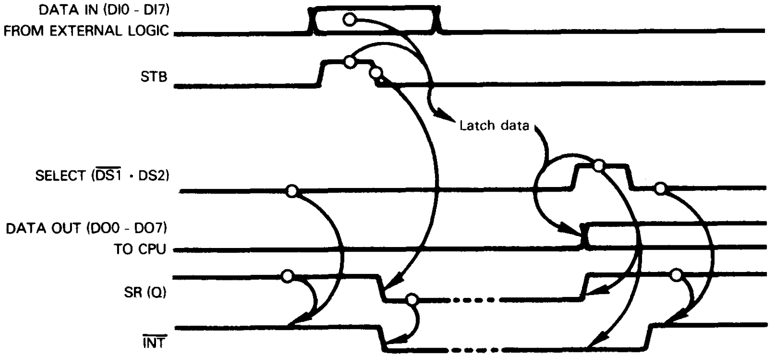
Tie MD to ground. Now STB clocks latches and Select enables buffers

Connect $\overline{\text{MEMR}}$ or $\overline{\text{I/OR}}$ from 8228 System Controller to $\overline{\text{DS1}}$. Read strobe then completes device select

Derive DS2 from Address Bus

Select logic from microcomputer system

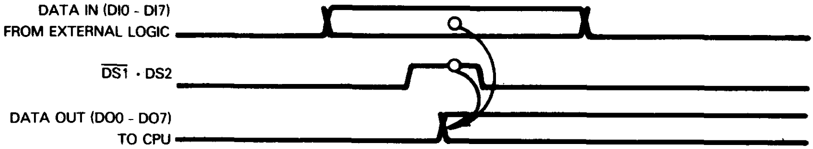
Input timing may be illustrated as follows:


DATA IN (DI0 - DI7) FROM EXTERNAL LOGIC
STB
Latch data
SELECT ($\overline{\text{DS1}}$ · DS2)
DATA OUT (DO0 - DO7) TO CPU
SR (Q)
$\overline{\text{INT}}$

As illustrated above, STB is a handshaking signal, equivalent to the 8255 $\overline{\text{STB}}$ control; $\overline{\text{INT}}$ can be used to signal the CPU that new data is in the latches and may be read. $\overline{\text{INT}}$ must serve the functions of 8255 IBF and INTR controls; **we have** thus **approximated 8255 Mode 1 input.**

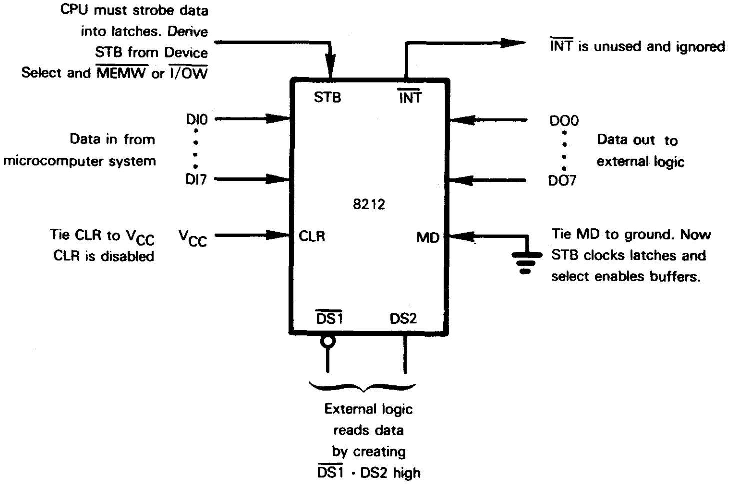
8212 I/O PORT USED FOR SIMPLE INPUT

We can simplify data input protocol by tying STB high (to V_{CC}). DATA IN passes through the latches continuously and SELECT enables DATA OUT. You can ignore $\overline{INT}$. **The 8212 is now operating as a simple, gated buffer,** where SELECT enables the output of whatever data occurs at DI0 - DI7; **this is equivalent to 8255 Mode 0 input.** Timing may be illustrated as follows:



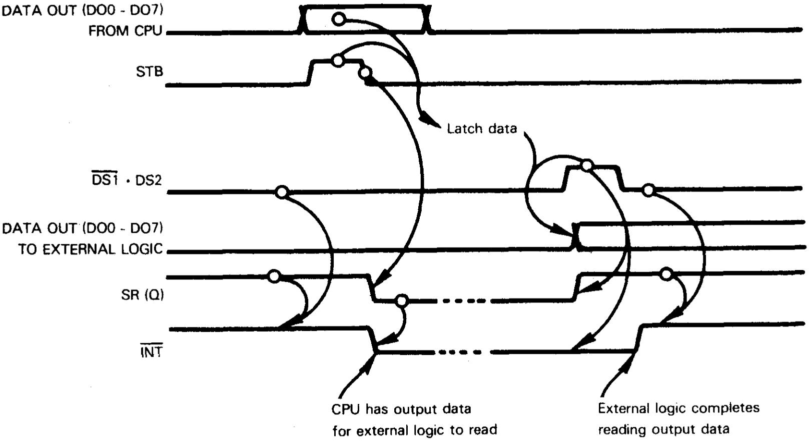
8212 I/O PORT USED FOR OUTPUT WITH HANDSHAKING

The 8212 I/O Port being used for output may be illustrated as follows:



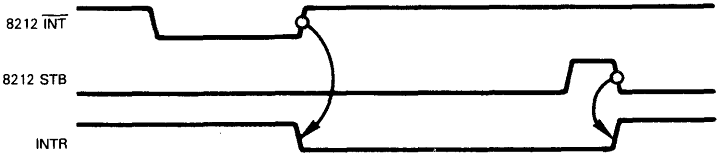
CPU must strobe data into latches. Derive STB from Device Select and $\overline{MEMW}$ or $\overline{I/OW}$
$\overline{INT}$ is unused and ignored
STB
$\overline{INT}$
DI0
DI7
Data in from microcomputer system
DO0
DO7
Data out to external logic
8212
Tie CLR to V_{CC} CLR is disabled
V_{CC}
CLR
MD
Tie MD to ground. Now STB clocks latches and select enables buffers.
$\overline{DS1}$
DS2
External logic reads data by creating $\overline{DS1}$ · DS2 high

This is a simple reversal of the input logic just described. The CPU creates STB and external logic creates DS1•DS2. DI0 - DI7 is connected to the microcomputer system Data Bus and DO0 - DO7 is connected to external logic. INT goes low when the CPU strobes data into the I/O port; INT therefore informs external logic that data is ready to be output. External logic resets INT upon reading the data. This timing may be illustrated as follows:


DATA OUT (DO0 - DO7) FROM CPU
STB
Latch data
$\overline{\text{DS1}}$ • DS2
DATA OUT (DO0 - DO7) TO EXTERNAL LOGIC
SR (Q)
$\overline{\text{INT}}$
CPU has output data for external logic to read
External logic completes reading output data

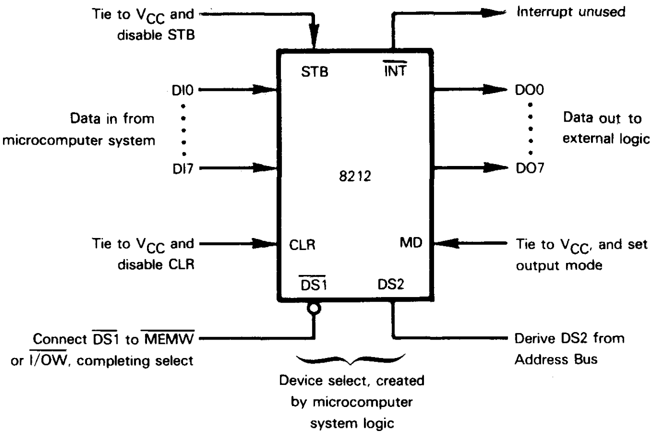
This use of the 8212 I/O port for data output is directly comparable to 8255 Mode 1 output.

8212 $\overline{\text{INT}}$ serves the same function as 8255 OBF. 8212 $\overline{\text{DS1}}$•DS2 is equivalent to 8255 ACK. The 8255 interrupt request INTR, however, must be derived from external logic as follows:



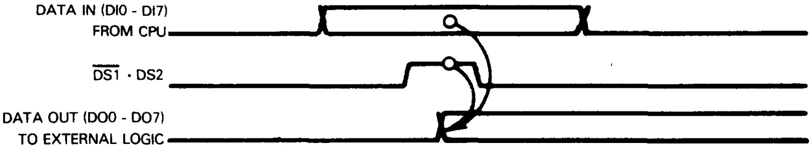
The 8212 I/O port can be configured for simple, Mode 0 output as follows:

8212 I/O PORT USED FOR SIMPLE OUTPUT



Tie to V_{CC} and disable STB
Interrupt unused
STB
$\overline{\text{INT}}$
DI0
DO0
Data in from microcomputer system
Data out to external logic
DI7
DO7
8212
Tie to V_{CC} and disable CLR
CLR
MD
Tie to V_{CC}, and set output mode
$\overline{\text{DS1}}$
DS2
Connect $\overline{\text{DS1}}$ to $\overline{\text{MEMW}}$ or $\overline{\text{I/OW}}$, completing select
Derive DS2 from Address Bus
Device select, created by microcomputer system logic

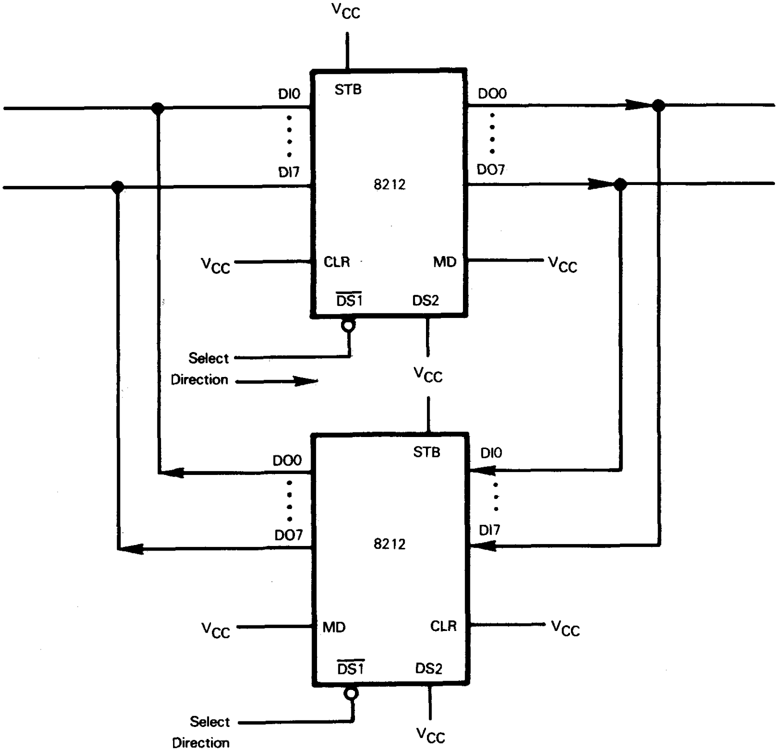
Simple output timing may be illustrated as follows:



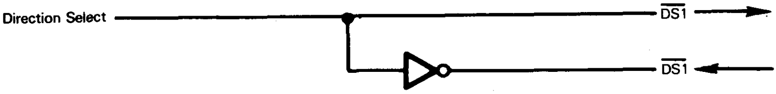
DATA IN (DI0 - DI7) FROM CPU
$\overline{\text{DS1}}$ · DS2
DATA OUT (DO0 - DO7) TO EXTERNAL LOGIC

8212 I/O PORTS AS BIDIRECTIONAL BUS DRIVERS

In many small microcomputer configurations two 8212 I/O ports may be wired with one for simple input, the other for simple output, then combined back-to-back to form a bidirectional bus driver:



Device select logic is the key to this use of the 8212 I/O ports. Tie DS2 of both devices to V_{CC}, then use $\overline{DS1}$ to select one device or the other:



Whatever signal is controlling the direction of data flow on the bus becomes the "Direction select". In its simple form it is connected to $\overline{DS1}$ of one 8212; inverted it is connected to $\overline{DS1}$ of the other 8212.

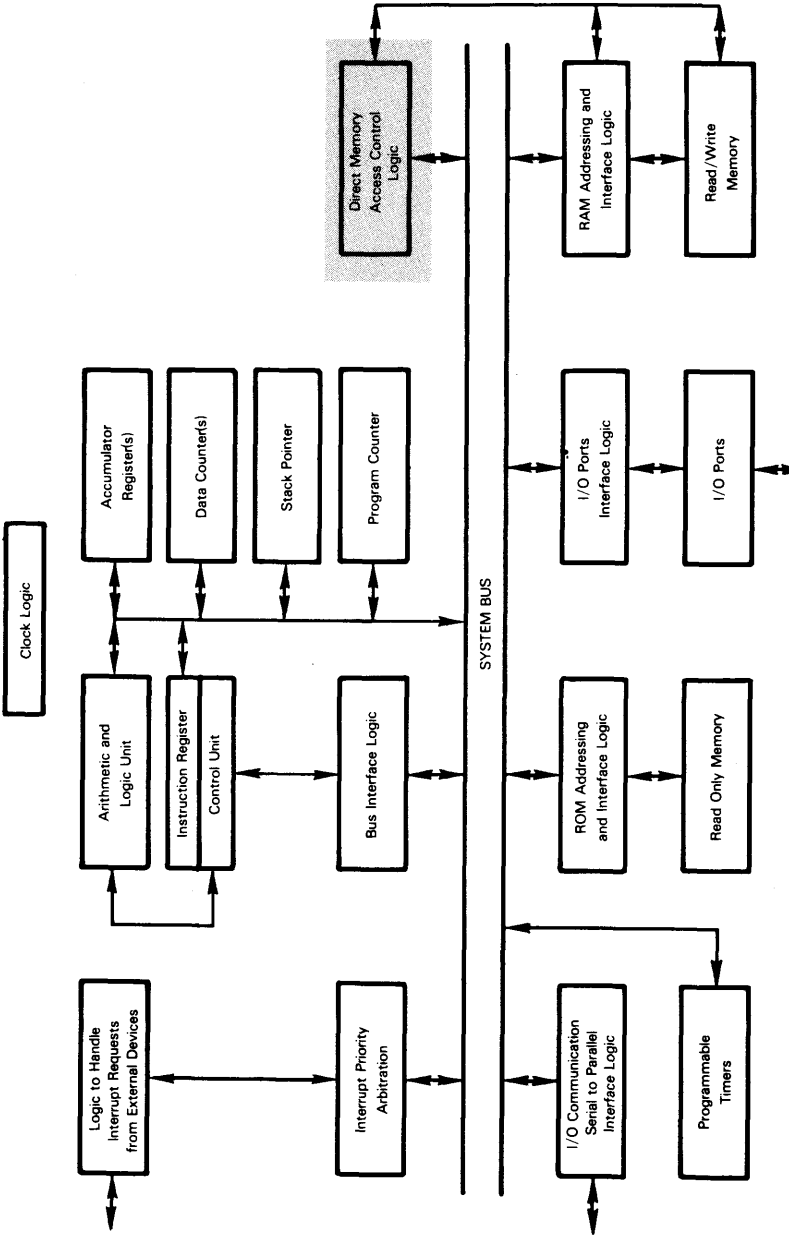

Clock Logic
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Direct Memory Access Control Logic
Arithmetic and Logic Unit
Instruction Register
Control Unit
Bus Interface Logic
Logic to Handle Interrupt Requests from External Devices
Interrupt Priority Arbitration
SYSTEM BUS
RAM Addressing and Interface Logic
Read/Write Memory
I/O Ports Interface Logic
I/O Ports
ROM Addressing and Interface Logic
Read Only Memory
I/O Communication Serial to Parallel Interface Logic
Programmable Timers

Figure 4-49. Logic Of The 8257 DMA Controller

THE 8257 DIRECT MEMORY ACCESS CONTROLLER

This device provides 8080A based microcomputer systems with logic to support four direct memory access channels.

From our discussion of the 8080A CPU, recall that this microprocessor supports direct memory access by providing a Hold state. During the Hold state, the CPU disconnects itself from system busses, allowing external logic to access memory by mimicking CPU signals on the Address, Data and Control Busses. Logic of the 8257 DMA Controller creates the necessary Hold state and performs the necessary CPU mimicking operations.

Because 8080A DMA logic is unique to this microprocessor, the 8257 DMA Controller is not general purpose; it is difficult to use with other microprocessors described in this book.

It is important to note that the 8257 DMA Controller requires an 8212 I/O port, or some equivalent device, to demultiplex Data Bus pin outputs.

Noteworthy features of the 8257 DMA Controller are:

1) The ability to assign permanent priorities to the four DMA channels, or to rotate priorities on a round robin basis.
2) By reducing the number of DMA channels to three, one DMA channel can be used for the recursive DMA transfer of fixed length or chained records.

Figure 4-49 illustrates that part of our general microcomputer system logic which has been implemented on the 8257 DMA Controller device.

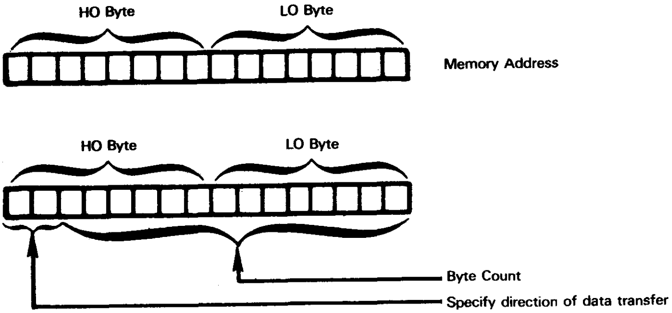
The 8257 DMA Controller chip is fabricated using N-channel depletion load MOS technology. It is packaged as a 40-pin ceramic or plastic DIP. All signals are TTL compatible.

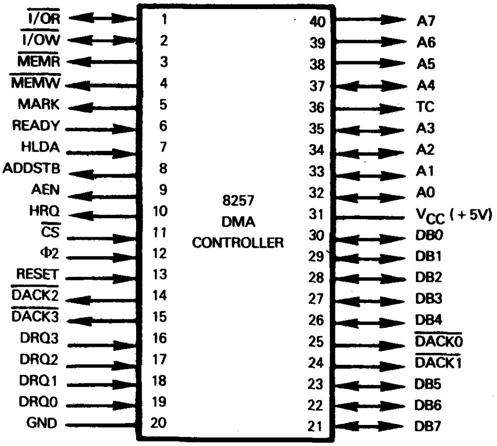
8257 DMA CONTROLLER PINS AND SIGNALS

Figure 4-50 summarizes 8257 DMA pins and signals. Many of these signals have 8080A counterparts, therefore we will describe them within the context of a general 8257 device discussion.

8257 DMA CONTROLLER REGISTERS

Logic associated with each DMA channel centers on two 16-bit registers. One register holds a memory address; the other contains a byte count, plus DMA direction control:





$\overline{\text{I/OR}}$ 1
$\overline{\text{I/OW}}$ 2
$\overline{\text{MEMR}}$ 3
$\overline{\text{MEMW}}$ 4
MARK 5
READY 6
HLDA 7
ADDSTB 8
AEN 9
HRQ 10
$\overline{\text{CS}}$ 11
Φ2 12
RESET 13
$\overline{\text{DACK2}}$ 14
$\overline{\text{DACK3}}$ 15
DRQ3 16
DRQ2 17
DRQ1 18
DRQ0 19
GND 20
8257
DMA
CONTROLLER
40 A7
39 A6
38 A5
37 A4
36 TC
35 A3
34 A2
33 A1
32 A0
31 V_{CC} (+5V)
30 DB0
29 DB1
28 DB2
27 DB3
26 DB4
25 $\overline{\text{DACK0}}$
24 $\overline{\text{DACK1}}$
23 DB5
22 DB6
21 DB7

PIN NAME	DESCRIPTION	TYPE
DB0 - DB7	Bidirectional Data Bus and high order Address byte	Tristate, Bidirectional
A0 - A4	Four low order address lines and register select lines	Tristate, Bidirectional
A5 - A7	Simple address lines	Output
$\overline{\text{I/OR}}$	Device read input and external logic read output	Tristate, Bidirectional
$\overline{\text{I/OW}}$	Device write input and external logic write output	Tristate, Bidirectional
$\overline{\text{MEMR}}$	Memory read control	Tristate, Output
$\overline{\text{MEMW}}$	Memory write control	Tristate, Output
MARK	128 (or 256) byte count indicator	Output
TC	Penultimate byte indicator	Output
READY	Memory ready or not ready	Input
HRQ	Hold request to CPU	Output
HLDA	Hold acknowledge from CPU	Input
ADDSTB	Address on Data Bus strobe	Output
AEN	DMA bus control enable and system controller disable	Output
$\overline{\text{CS}}$	Device select	Input
Φ2	Clock input (Φ2 TTL from 8224)	Input
RESET	System reset	Input
DRQ0 - DRQ3	DMA service request	Input
$\overline{\text{DACK0}}$ - $\overline{\text{DACK3}}$	DMA service acknowledge	Output
V_{CC}, GND	Power and Ground	

Figure 4-50. 8257 DMA Controller Signals And Pin Assignments

The transfer of any block of data via DMA begins with an initial memory address, byte count and data transfer direction specification being loaded into appropriate registers. As each byte of data is transferred, the Address register contents are incremented and the Byte Count register contents are decremented.

Each of the four DMA channels provided by a single 8257 device has its own pair of Address and Byte Count registers; that means there are eight such registers in total. Each register is addressed as a single I/O port or memory location. For this to be possible, 8257 logic requires each register to be accessed twice; on the first access, the low order byte of the selected register will be accessed, while on the second access

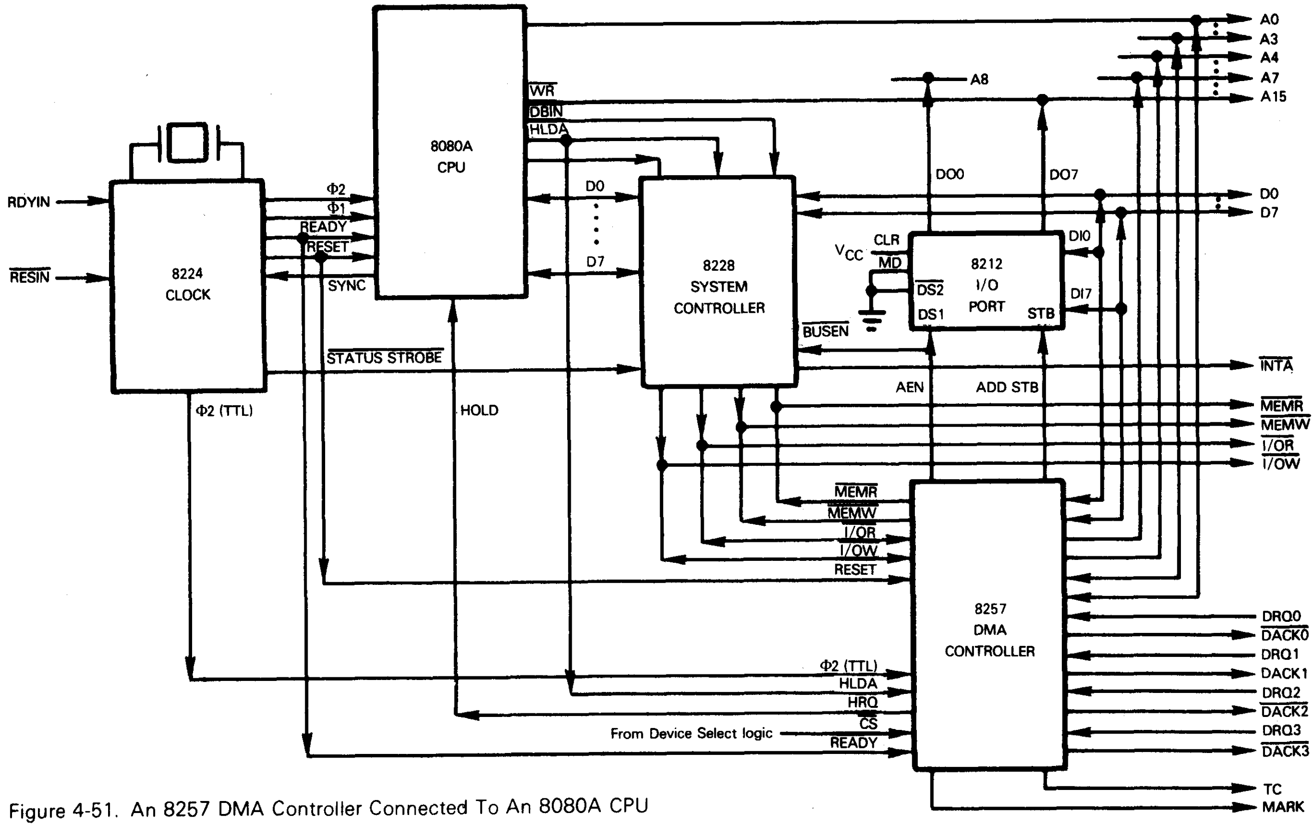
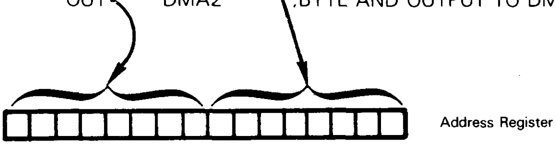


Figure 4-51. An 8257 DMA Controller Connected To An 8080A CPU

the high order byte of the selected register will be accessed. This may be illustrated as follows:

```
        MVI     A, DMALO   ;LOAD LOW ORDER DMA STARTING ADDRESS
        OUT     DMA2       ;BYTE AND OUTPUT TO DMA CHANNEL 2
        MVI     A, DMAHI   ;LOAD HIGH ORDER DMA STARTING ADDRESS
        OUT     DMA2       ;BYTE AND OUTPUT TO DMA CHANNEL 2
```



Observe that a single I/O port address, represented above by the symbol DMA2, accesses different halves of a 16-bit register, when executed sequentially.

As this access procedure would imply, it is mandatory that whenever you access one of the 8257 DMA Controller's 16-bit registers, you must execute access instructions in pairs. For example, were you to transfer a block of data that is less than 256 bytes in length, you would still have to write a 00 value into the selected DMA channel's Counter register high order byte. If you did not do so, then on the next access of the Counter register, you would automatically select the high order register byte — which may not be what you planned on.

The need to access DMA controller 16-bit registers via pairs of instructions has two non-obvious ramifications.

First, never let $\overline{CS}$ become true while $\overline{I/OR}$ or $\overline{I/OW}$ are low; the 8257 will be fooled into thinking you just accessed a Controller register, and that will mess up your execution of instruction pairs.

Second, disable interrupts while executing the pair of Controller register accesses. A chance interrupt splitting the two register access instructions could cause a lot of problems.

8257 DMA CONTROLLER ADDRESS BUS

8257 DMA CONTROLLER DATA BUS

The contents of the selected DMA Channel Address register are output as a memory address during each DMA access. Of course, during any single DMA access, only one DMA channel will be active — and its Address register contents will be placed on the Address Bus. **The low order eight bits of the address are output via address lines A0 - A7; the high order eight bits of the address are output on the data lines D0 - D7.** The data lines are bidirectional since they are also used to transfer data from the CPU into any of the 8257 registers or to read registers' contents back to the CPU. Thus **the data lines D0 - D7 must be demultiplexed; this is done typically using an 8212 I/O port, as illustrated in Figure 4-51.**

Recall that DMA control logic is never involved in the actual transfer of data under DMA control. Thus the data pins D0 - D7 of the 8257 device never transfer data to or from memory under DMA. If you are unclear on this point, refer again to the discussion on direct memory access given in "An Introduction To Microcomputers: Volume I".

The four low order address lines A0 - A3 are bidirectional. In order to keep down the number of pins required by the 8257 DIP, **these four low order address lines are used to select individual addressable locations within the 8257 device.** This is necessary because the 8257 has nine addressable registers; therefore the usual procedure of including register select with device select logic is not viable. Four register select pins would be needed, and that many are simply not available.

Table 14-13 summarizes the way in which 8257 addressable locations are identified by the low order four address lines.

Table 14-13. Addressing Registers Within The 8257 DMA Controller

SELECT CODE				
A3	A2	A1	A0	ADDRESSED 8257 LOCATION
0	0	0	0	DMA Channel 0 Address register*
0	0	0	1	DMA Channel 0 Byte Count register*
0	0	1	0	DMA Channel 1 Address register*
0	0	1	1	DMA Channel 1 Byte Count register*
0	1	0	0	DMA Channel 2 Address register*
0	1	0	1	DMA Channel 2 Byte Count register*
0	1	1	0	DMA Channel 3 Address register*
0	1	1	1	DMA Channel 3 Byte Count register*
1	0	0	0	Command register on output or write Status register on input or read

*These are 16-bit registers. On alternate accesses, the low order, then the high order byte is addressed. The low order byte is accessed first following a Reset or Power-up.

The chip select input $\overline{\text{CS}}$ must be low before any addressable location within the 8257 device can be accessed. Thus $\overline{\text{CS}}$ must be low for any of the select codes illustrated in Table 4-13 to apply. But $\overline{\text{CS}}$ must be deselected while any DMA operation is actually taking place; that is to say, the 8257 device is only selected while it is a slave device on the System Bus and is being accessed by the CPU under program control. Once the 8257 device is a master of the System Bus and is controlling the transfer of data, then it is no longer selectable. In fact, **logic within the 8257 device disables $\overline{\text{CS}}$ as an input while conducting an actual DMA data transfer.** This is necessary since the 8257 DMA device could otherwise inadvertently select itself.

8257 DMA CONTROLLER PRIORITY ARBITRATION

External logic requiring direct memory access makes its request via one of the four request lines DRQ0 - DRQ3. If more than two sources simultaneously request direct memory access, then **you can program the 8257 Controller to use one of two priority arbitration schemes:**

1) In the default case, priorities are permanently assigned as follows: DRQ0 then DRQ1, then DRQ2, then DRQ3.
2) An alternative priority arbitration scheme assigns priorities on a round robin basis; the channel which just had access immediately becomes the lowest priority channel. This may be illustrated as follows:

Channel Just Serviced:	Start	DRQ2	DRQ1	DRQ0
First Priority:	DRQ0	DRQ3	DRQ2	DRQ1
Second Priority:	DRQ1	DRQ0	DRQ3	DRQ2
Third Priority:	DRQ2	DRQ1	DRQ0	DRQ3
Fourth Priority:	DRQ3	DRQ2	DRQ1	DRQ0

The shaded requests are selected.

The four acknowledge signals $\overline{\text{DACK0}}$ - $\overline{\text{DACK3}}$ are used to tell external logic when a direct memory access request has been acknowledged and is being processed.

External logic will use the direct memory access request and acknowledge signals in one of two ways:

8257 DMA BYTE BY BYTE

8257 DMA BURST MODE

1) Under normal circumstances, external logic will raise its DRQ signal whenever it is ready for a direct memory access; it will lower the signal upon receiving a $\overline{\text{DACK}}$ response.

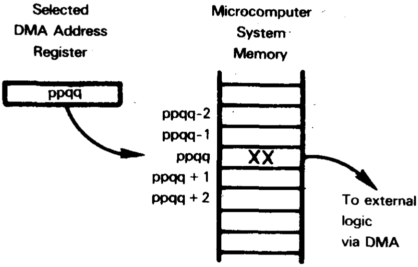
2) In burst mode, external logic will leave its direct memory access request DRQ true while an entire block of data is transferred via DMA. We will have more to say about this mode of operation later.

Now consider in more detail the events that constitute a direct memory access.

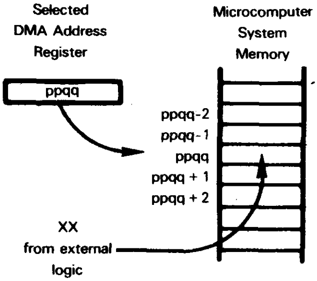
For a direct memory access sequence to begin, one or more DMA requests must be present on lines DRQ0 - DRQ3.

Three types of DMA access may occur:

1) During **a DMA read,** the contents of the memory location selected by the DMA Address register will be transmitted to external logic:



2) During **a DMA write,** external logic must provide a byte of data which will be written into the memory word addressed by the DMA Address register:



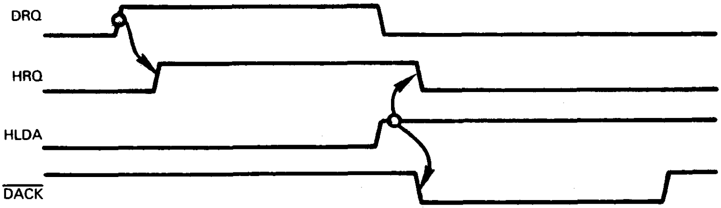
3) **A DMA verify** cycle is also provided. During a DMA verify cycle, the 8257 device takes control of the System Bus in the usual way; however, no data is actually transferred between memory and external logic. DMA verify cycles are typically used by external logic which transfers data in records — and must be given time to compute cyclical redundancy characters which are appended to the record. For example, the DMA verify cycle would be used with external logic interfacing magnetic storage devices such as cassettes and floppy disks.

The type of DMA access which is about to occur will have been defined previously under program control via the two high order Byte Count register bits for the selected DMA channel. As this would imply, it is **the program being executed by the 8080A CPU that determines the nature of the DMA access.**

There is no reason why external logic could not define the type of DMA access; however, it makes more sense for the CPU to do so since any program being executed by the CPU must know whether data is being transferred into or out of memory via DMA.

When the 8257 device receives a DMA request via a DRQ signal, it issues a Hold request to the CPU via HRQ. This Hold request is maintained until acknowledged by the CPU via HLDA. At this point the 8257 device has control of the System Bus, usually for a time of four clock periods — the time required for a single memory access to occur. However, the 8257 device may extend the Hold period for slow external devices by inserting Wait state clock periods, in the same manner as the CPU performs this operation.

Timing for the onset of a DMA transfer may be illustrated as follows:



Notice that selected external logic does not receive its acknowledge via $\overline{\text{DACK}}$ until the onset of the HOLD condition, after HLDA has been received true from the CPU.

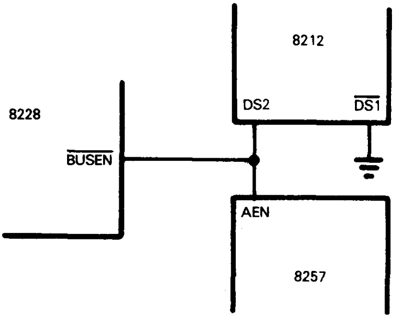
While the 8257 device has control of the System Bus, it mimics a CPU read or write operation. Notice that the 8257 has all of the necessary System Bus signals:

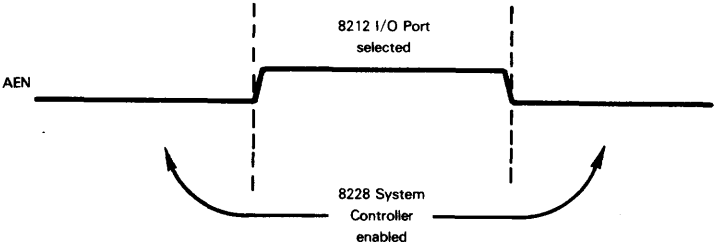
	Three-Device, 8080A Signals from Figure 4-33		8257 Signals from Figure 4-50	
8080A	A0 - A15	Address Bus	A0 - A7	8257
8228	DB0 - DB7	Data Bus	DB0 - DB7	8257
8228(25)	$\overline{\text{I/OR}}$	I/O Read strobe	$\overline{\text{I/OR}}$	8257(1)
8228(27)	$\overline{\text{I/OW}}$	I/O Write strobe	$\overline{\text{I/OW}}$	8257(2)
8228(24)	$\overline{\text{MEMR}}$	Memory read strobe	MEMR	8257(3)
8228(26)	$\overline{\text{MEMW}}$	Memory write strobe	$\overline{\text{MEMW}}$	8257(4)
8080A(23)	READY	Memory ready/not ready	READY	8257(6)
8080A(12)	RESET	System reset	RESET	8257(13)

The 8257 Controller has two special signals, needed by the 8080A-8257 interface; they are ADD STB and AEN.

AEN enables and disables the 8228 System Controller. When low, AEN provides the 8228 System Controller with the required low $\overline{\text{BUSEN}}$ input. When high, AEN provides the 8228 System Controller with a high $\overline{\text{BUSEN}}$ input, which causes the 8228 System Controller to disconnect itself. AEN is also connected to the 8212 DS2 select; the 8212

$\overline{\text{DS1}}$ select is grounded, therefore AEN high becomes the 8212 select. AEN logic may be illustrated as follows:



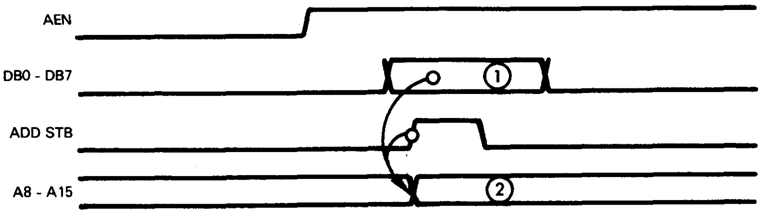


If I/O devices are accessed via I/O instructions in your microcomputer system, then you have another problem — which you must use AEN to solve. Remember, I/O device select logic will decode Address Bus lines A0 - A7 while $\overline{\text{I/OR}}$ or $\overline{\text{I/OW}}$ are true. Both $\overline{\text{I/OR}}$ and $\overline{\text{I/OW}}$ are used by the 8257 as control signals, and that can cause I/O devices to spuriously select themselves. To solve this problem, make $\overline{\text{AEN}}$ low a prerequisite for I/O device selection.

ADD STB is a strobe output which identifies the high order byte of an address being output on the 8257 Controller Data Bus. ADD STB must therefore be used by external logic to demultiplex the Data Bus outputs from the 8257 DMA Controller. Data Bus contents, while ADD STB is high, must be routed to the high order eight Address Bus lines.

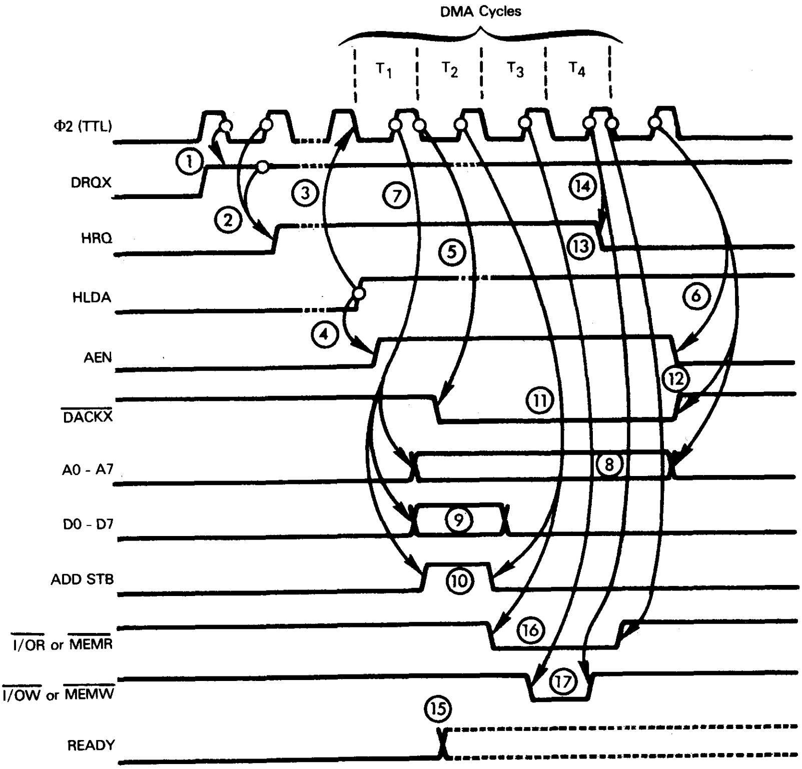
In summary, therefore, AEN is a master enable signal whereas ADD STB is an address strobe signal which occurs while AEN is high.

There are many ways in which external logic can use AEN and ADD STB; the use illustrated in Figure 4-51 is just one possibility, and shows the 8212 I/O port configured as an input port with handshaking; our previous discussion of the 8212 I/O port covers this port use in detail. As configured in Figure 4-51, the 8212 I/O port becomes selected whenever AEN is output high — at which time the 8228 System Controller is disconnected from the System Bus. The 8212 I/O port continuously receives as inputs whatever happens to be transferred into, or out of the 8257 DMA Controller via its data pins. However, only at the instant when ADD STB makes a low-to-high transition, will the data inputs be latched. At this instant the data inputs will be the high order byte of a memory address. As illustrated in Figure 4-51, the output buffer of the 8212 I/O port will be continuously enabled, since select logic true will span the entire period of the ADD STB pulse. For this reason, the moment a new high order address byte is strobed into the 8212 data latches, it will be output immediately — and will be maintained on the high order eight address lines until ADD STB is pulsed high again. Timing may be illustrated as follows:



The value output on DB0 - DB7 while ADD STB is high ① is maintained as the value output via A8 - A15 ② until another ADD STB pulse occurs.

We are now ready to describe timing for complete DMA access cycles. Timing may be illustrated as follows:



Let us look at the sequence of events illustrated by the timing diagram above.

The logic sequence which initiates a DMA access begins two clock periods in advance of the DMA access. That is to say, once the 8257 DMA Controller senses a high DRQ input, it will take at least two clock periods before the 8257 device gains control of the System Bus. DRQ is sampled on the trailing edge of the Φ2 clock, as illustrated by ① above. Upon sensing one or more DRQ lines high, 8257 logic requests a Hold condition by outputting HRQ high on the leading edge of the next Φ2 pulse, as illustrated by ② .

The CPU will respond to HRQ high by entering a Hold state. The Hold state is acknowledged by HLDA being output high. This will occur prior to the trailing edge of the Φ2 pulse, as illustrated by ③ above. This marks the beginning of the clock cycles during which the 8257 device takes control of the System Bus. While it is possible for HLDA to be output high on the Φ2 clock pulse that follows HRQ being output high, this may not be the case. In fact, the time delay between HRQ and HLDA determines the maximum rate at which data may be transferred under DMA control — a topic we will address shortly.

As soon as the 8257 device gains control of the System Bus, it outputs AEN high in order to disable the 8228 System Controller, and select the 8212 Data Bus demultiplexing I/O port. This is illustrated by ④ above. Note that it is AEN being output high which disables the 8228 System Controller and gives the 8257 device complete mastery of the System Bus.

Now that the 8257 device has control of the System Bus, it can start mimicking read or write signals. For either a DMA read or write, the required memory address must appear on the Address Bus. The low order byte of the memory address is output via the 8257 A0 - A7 pins, on the leading edge of the Φ2 clock during the DMA T_1 clock period. This is illustrated by ⑦ above. The low order address byte is output directly, from the Address register for the selected DMA channel, to the low order eight address lines of the System Address Bus. The address remains stable until the leading edge of the first Φ2 clock pulse cycle following T_4. This is illustrated by ⑥ above.

The high order byte of the memory address is output via the data pins D0 - D7 of the 8257 device. This is illustrated above by ⑨ . We have just discussed how ADD STB is pulsed high (⑩ above) in order to latch the Data Bus output into the 8212 I/O demultiplexing port. Though not shown in the timing diagram illustrated above, the ADD STB high pulse (⑩ above) will cause the contents of the Data Bus (⑨ above) to appear on the high order eight address lines of the 'System Address Bus. Thus all 16 address lines have been output.

If a DMA read operation is to occur, then the $\overline{MEMR}$ low pulse (⑯ above) strobes data from memory onto the System Data Bus. Data must be stable for the duration of this pulse. On the next Φ2 high-going pulse $\overline{I/OW}$ goes low (⑬ above) to indicate that data is stable and can be read by external logic.

During a DMA write operation, $\overline{I/OR}$ low (⑯ above) strobes data from external logic onto the Data Bus. Then a low level on $\overline{MEMW}$ (⑰ above) indicates that data is stable and can be written to memory.

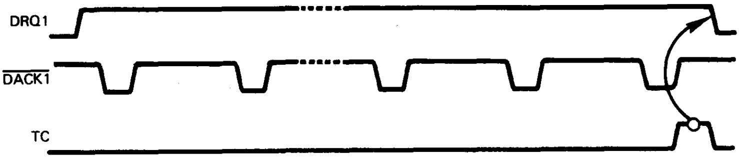
In the event that external logic cannot respond to a write within the four clock periods of a normal DMA cycle, then the READY signal must be input low in advance of the Φ2 pulse during T_2, as illustrated by ⑮ above.

The termination of the DMA operation is timed by the T_4 Φ2 pulse and the Φ2 pulse of the next clock period. Termination logic is illustrated by ⑥ , ⑩ and ⑫ . This termination logic is quite straightforward; it simply dovetails with the sequence in which events occur during the first clock period of any normal 8080A instruction execution machine cycle.

Note very carefully that the timing illustrated above represents the sequence of events surrounding data transfer between external logic and memory. When 8257 DMA device internal registers are accessed, either to read or write a byte of data, timing is exactly the same as it would be for any memory reference or I/O instruction's execution.

There are two signals output by the 8257 device which are very valuable when transferring blocks of data via DMA; these two signals are TC and MARK.

TC is a terminal count signal. This signal is output on the terminal data byte of a record — that is, when the count in the currently selected Count register is 0, indicating that the byte being transferred via this DMA channel will be the last for the current data block. **Since the last byte is byte 0, the byte count should be initiated with N-1, if N bytes are to be transferred.** External logic may use the TC signal in order to initiate "end of data transfer" operations. If, for example, blocks of data are being transferred under burst mode, then external logic will lower its DMA request following the next DMA acknowledge, in order to prepare for the next block of data transfer. This may be illustrated as follows for DMA Channel 1:



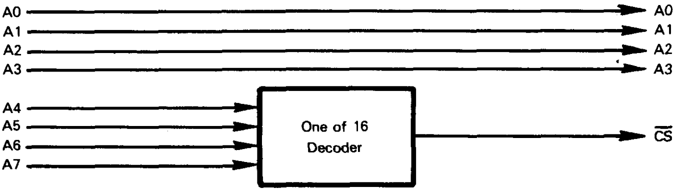
The MARK signal outputs true at 128-byte intervals.

The MARK signal is particularly useful when the 8257 device is being used to create interface logic for magnetic storage devices such as floppy disks or tape cassettes which store data in contiguous 128-byte or 256-byte records.

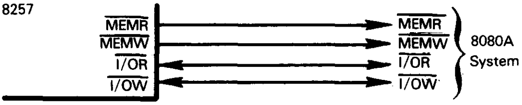
PROGRAMMING THE 8257 DMA CONTROLLER

To the programmer, the 8257 DMA Controller will appear to be either 16 I/O ports or 16 memory addresses. Assuming that address pins A0 - A7 of the 8257 device are connected to corresponding address lines of the Address Bus, then the I/O port or memory addresses will be contiguous.

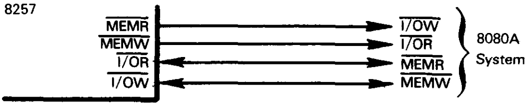
Address lines A0 - A3 are bidirectional so that they may be used additionally as register addressing inputs once a particular 8257 device has been selected via $\overline{\text{CS}}$. No rules peculiar to the 8257 device govern the way in which $\overline{\text{CS}}$ should be generated. Here is one simple possibility, where the 8257 device is being accessed as 16 I/O ports:



If the 8257 device is being accessed as 16 I/O ports then control lines should be connected as follows:



These are the only changes needed if you are going to access the 8257 device as 16 memory locations:



As we are going to describe shortly, there are additional changes in the program control word interpretation when the 8257 device is being accessed as memory locations.

You must load appropriate information into Address registers, Count registers and a programmed Command register, before you can initiate transfer under DMA control.

Loading the Address register is self-evident; using two contiguous memory write or I/O instructions, you must load the low and high order bytes of the starting memory address for the next DMA transfer.

Next you must load the byte count into the associated Count register. The two high order bits of the Count register are read and write control bits.

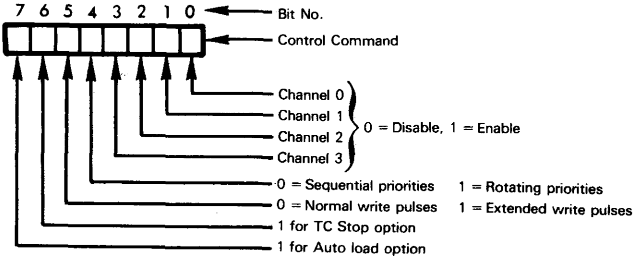
The contents of the two high order Count register bits will have different interpretations depending on whether the 8257 device is being accessed as 16 or I/O memory locations. The two cases are defined as follows:

15 14 ← Bit No.

15	14	I/O Ports	Memory Locations
0	0	Verify DMA Cycle	Verify DMA Cycle
0	1	Write DMA Cycle	Read DMA Cycle
1	0	Read DMA Cycle	Write DMA Cycle
1	1	Illegal	Illegal

8257 CONTROL COMMAND

Actual DMA transfers are enabled, along with programmable options, by outputting an appropriate Control Command to the Control register. This is how individual bits of the Control register will be interpreted:



Reset automatically sets all Control Command bits to 0.

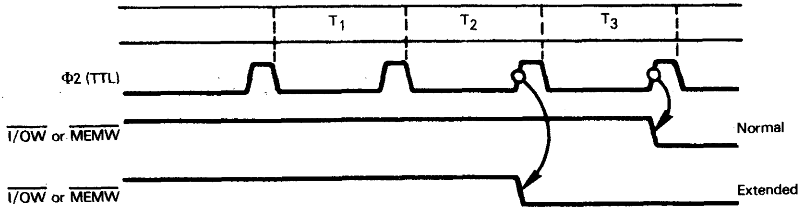
Always disable DMA channels when they are not in use. Never enable a DMA channel until all its registers have been completely programmed. Look upon a channel enable as the DMA initialization.

Let us look at the Control Command in more detail.

Bits 0 through 3 simply enable or disable the four DMA channels of the 8257 device.

Bit 4 allows you to select the rotating priority or permanent priority option when resolving external DMA request conflicts.

Extending the memory write or I/O pulse, via bit 5, is an intriguing feature of the 8257 device. What this programmable option does is allow the memory write or I/O write pulse to go true sooner than normal in a time period, giving external logic more time to identify the need for Wait states. This may be illustrated as follows:



T_1
T_2
T_3
Φ2 (TTL)
$\overline{I/OW}$ or $\overline{MEMW}$
Normal
$\overline{I/OW}$ or $\overline{MEMW}$
Extended

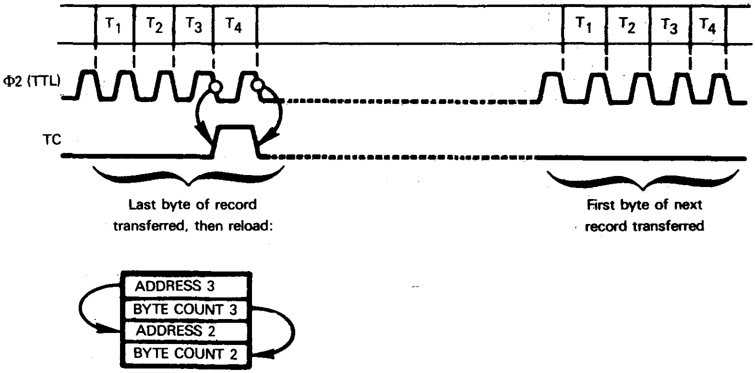
Notice that slow external logic will not need Wait states when reading from memory. So long as data has been read off the Data Bus, the data can be manipulated after the CPU has regained system control.

If Control Command bit 6 is set, then a high TC pulse will automatically disable the appropriate channel after its last byte is transferred. This ensures that DMA transfers will cease even if external logic fails to lower its DMA request. The channel can be re-enabled only when the 8080A CPU writes a Control Command which sets that channel's enable bit.

In auto load (Control Command bit 7 set) Channel 2 is not affected by TC Stop.

The auto load option allows a sequence of records to be transferred via DMA, without having to re-initialize the Address register and Byte Count register prior to each record's transfer.

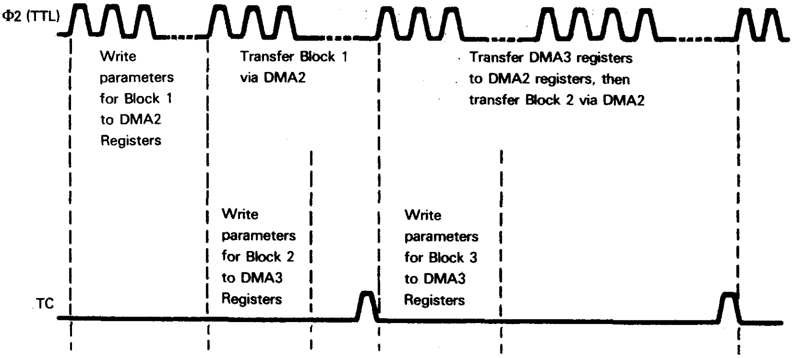
The auto load option only works with DMA Channel 2 and it requires DMA Channel 3 to be set aside as a Channel 2 buffer. While a data record is being transferred via DMA Channel 2, program logic must load appropriate address and byte count information into Channel 3 registers in order to identify the next record which will be transferred via DMA Channel 2. When the DMA Channel 2 Byte Count register counts out, Channel 3 registers' contents are automatically read, nondestructively, into Channel 2 registers; and the next data record is transferred. This may be illustrated as follows:



When Channel 2's Address and Byte Count registers are loaded, the same data will be loaded automatically in Channel 3's registers.

Now if parameters do not change with each re-execution of a DMA transfer via Channel 2, then you can simply leave the Channel 2 initial address and byte count values in Channel 3 registers. Suppose for example, that a 256-byte record must be continuously re-read out of a fixed memory buffer. This assumes that program logic is able to stay ahead of the DMA transfer continuously re-loading the data buffer.

But it is perfectly reasonable to transfer chained records under DMA control, whereby for each succeeding record transfer read via DMA, a different area of memory is accessed. Now while one record is being transferred under DMA control via DMA Channel 2, parameters for the next record may be loaded into DMA Channel 3 registers. This may be illustrated as follows:

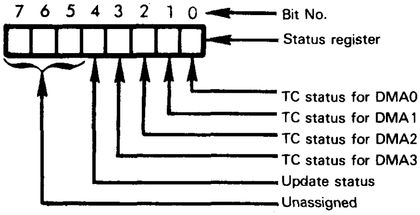


Φ2 (TTL)
Write parameters for Block 1 to DMA2 Registers
Transfer Block 1 via DMA2
Transfer DMA3 registers to DMA2 registers, then transfer Block 2 via DMA2
Write parameters for Block 2 to DMA3 Registers
Write parameters for Block 3 to DMA3 Registers
TC

It is only necessary to ensure that you do not attempt to write data into Channel 3 registers at the very instant that this data is being transferred to Channel 2 registers. In order to make sure that this does not happen, a special status (bit 4 of the Status register) goes true from the terminal count (TC high pulse), until the beginning of the next record's first DMA access.

8257 DMA CONTROLLER STATUS FLAGS

The Update status is just one of eight status flags that may be read at any time out of the 8257 device's Status register. As is standard in most 8080A-type peripheral devices, the address of the Status register is identical to the address of the Control Command register. The Status register is accessed when reading and the control accessed when writing. **Here is the way in which Status register bits have been assigned:**



The Update status flag we have just described.

The Terminal Count flags simply indicate the condition of the Terminal Count signal, as it would apply to individual DMA channels. Recall that there is only one terminal count signal output, and this signal must be shared by all four DMA channels. If more than one DMA channel is active, then the Terminal Count signal will become true during the penultimate DMA access for every DMA channel.

Programs should therefore use Terminal Count status bits in order to determine whether a record being transferred via any DMA channel has been completely transferred.

When you read the contents of the Status register, all bits except the update flag are automatically cleared. The update flag is not affected by a status read.

DMA TRANSFER RATES

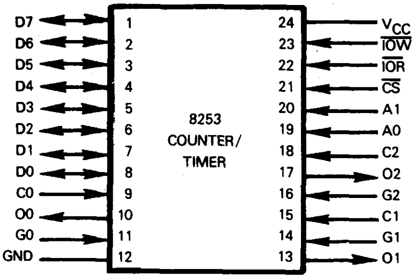
Recall from our discussion of the Hold state earlier in this chapter, that the CPU will only acknowledge a Hold request upon completing all memory accesses; this is illustrated in Figure 4-10. The CPU will therefore overlap the Hold state to some extent, while completing CPU only operations during machine cycles 4 and/or 5.

Note also that a Hold request will be acknowledged at the conclusion of a machine cycle, even if an instruction is in mid-execution. The SHLD instruction, for example, requires five machine cycles to execute, as illustrated in Figure 4-25. Nevertheless, were the 8257 device to request a Hold via HRQ during machine cycle 2, then the Hold would be acknowledged at the conclusion of MC2 (T_3) — that is right before the start of machine cycle 3. Under these circumstances, the complete DMA transfer would occur between machine cycles 2 and 3 of the SHLD instruction's execution.

THE 8253 PROGRAMMABLE COUNTER/TIMER

This is a programmable device which contains three sets of timing logic. Each set of timing logic can be programmed independently as an interval timer, a one-shot, a clock signal generator, or an event counter.

The 8253 Counter/Timer is a very significant device in any digital logic oriented microcomputer application. One of the more striking general weaknesses of microprocessors on the market today is their inflexibility when it comes to processing individual signals. The 8253 Counter/Timer goes a long way towards



PIN NAME	DESCRIPTION	TYPE
D0 - D7	Data Bus	Tristate, Bidirectional
C0	Timer 0 clock	Input
O0	Timer 0 output	Output
G0	Timer 0 gate	Input
C1	Timer 1 clock	Input
O1	Timer 1 output	Output
G1	Timer 1 gate	Input
C2	Timer 2 clock	Input
O2	Timer 2 output	Output
G2	Timer 2 gate	Input
A0, A1	Register select	Input
$\overline{\text{CS}}$	Chip select	Input
$\overline{\text{IOR}}$	Read control	Input
$\overline{\text{IOW}}$	Write control	Input
V_{CC}, GND	Power and Ground	

Figure 4-52. 8253 Counter/Timer Signals And Pin Assignments

lleviating this problem — either within an 8080A, or in any other microcomputer ystem — since the 8253 can be used with any microprocessor described in this ook.

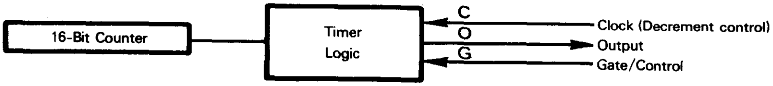
f yours is a digital logic application, we urge you to consider the possibilities of he 8253.

he 8253 is fabricated using N-channel, depletion load technology; it is packaged n a 24-pin plastic DIP. All pins are TTL compatible.

253 COUNTER/TIMER PINS AND SIGNALS

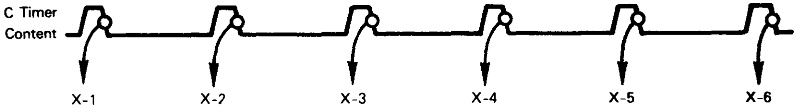
253 Counter/Timer pins and signals are illustrated in Figure 4-52. These pins and sigals are described in conjunction with a general discussion of 8253 organization logic nd capabilities.

ach of the three sets of timer logic consists of a 16-bit counter and three control ignals:



8253 COUNTER/ TIMER OPERATION

The counter will decrement by one on each falling edge of the clock signal C. Note that this signal may be synchronous or asynchronous. If synchronous, it is a traditional clock signal and the timer is being used to measure time intervals:



X represents any initial 16-bit Timer value.

If the clock signal is asynchronous, then the timer is being used as an event counter; it counts the number of times the clock signal input has made a high-to-low transition.

Clock frequencies may vary from 0 (DC) to 2 MHz.

The Gate input (G) is used variously to initiate or suspend timer operation.

Timer results are output via the output signal (O), which may be used as a one-shot pulse, interrupt request or simple control signal.

To the programmer, the 8253 Counter/Timer appears as four memory locations or I/O ports. These are selected as follows:

$\overline{CS}$	A0	A1	
0	0	0	Select Counter/Timer 0
0	0	1	Select Counter/Timer 1
0	1	0	Select Counter/Timer 2
0	1	1	Select Control register
1	X	X	Device not selected

The signals $\overline{RD}$ and $\overline{WR}$, are standard read and write control signals. If the 8253 Counter/Timer is being accessed as I/O ports, then $\overline{RD}$ and $\overline{RW}$ will be connected to $\overline{I/OR}$ and $\overline{I/OW}$, respectively; these are I/O read and write control signals output by the 8228 System Controller. If the 8253 Counter/Timer is being accessed as memory locations, then $\overline{RD}$ and $\overline{RW}$ will be connected to $\overline{MEMR}$ and $\overline{MEMW}$, respectively; these are the memory read and write control signals output by the 8228 System Controller.

A low $\overline{RD}$ pulse causes data from the selected 8253 location to be placed on the Data Bus.

A low $\overline{WR}$ pulse causes data on the Data Bus to be placed in the addressed 8253 location.

The Control register is a write-only location. If you attempt to read from this location, then no operation will occur and the Data Bus will be floated at the 8253 Counter/Timer interface.

The Data Bus is floated while the 8253 Counter/Timer is selected and neither $\overline{RD}$ or $\overline{WR}$ is pulsed low.

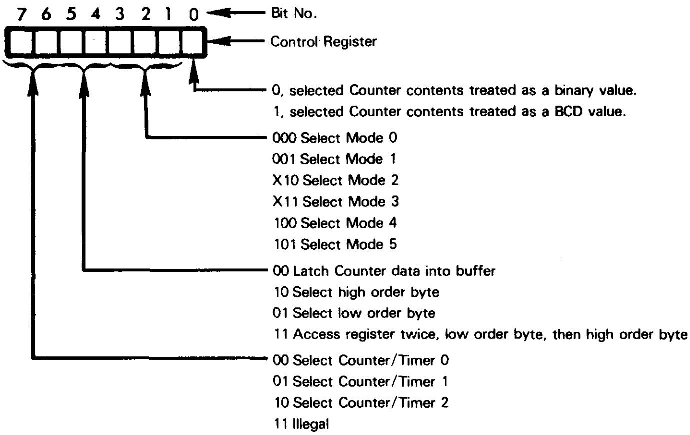
Notice that each 16-bit Counter/Timer register is accessed as a single addressable unit. When reading into or writing out of 16-bit registers, the prior Control code determines whether the high order byte or the low order byte will be accessed.

Data may be written into, or out of the 8253 Counter/Timer via the data pins D0 - D7. The Control register, however, is a write only location; its contents cannot be read.

8253 COUNTER/TIMER PROGRAMMABLE OPTIONS

We will begin our discussion of 8253 Counter/Timer options by describing the Control code which must be written into the Control register. Subsequently the various operating modes will be discussed, along with appropriate examples.

This is the general format for the control code:


7 6 5 4 3 2 1 0
Bit No.
Control Register
0, selected Counter contents treated as a binary value.
1, selected Counter contents treated as a BCD value.
000 Select Mode 0
001 Select Mode 1
X10 Select Mode 2
X11 Select Mode 3
100 Select Mode 4
101 Select Mode 5
00 Latch Counter data into buffer
10 Select high order byte
01 Select low order byte
11 Access register twice, low order byte, then high order byte
00 Select Counter/Timer 0
01 Select Counter/Timer 1
10 Select Counter/Timer 2
11 Illegal

X may be 0 or 1.

When a Counter/Timer is "selected" via Control code bits 7 and 6, the selection applies to the rest of the control code; in other words, you are specifying which Counter/Timer is having its options set.

It does not matter which Counter/Timer you select via a control code, the next 8253 access can still address any one of the three Counter/Timers — or the Control register. The byte of the Counter/Timer that will be selected, however, depends on the previously executed control code.

When the 8253 is reset, or in the absence of any Control codes output for a Counter/Timer, all 0 values are assumed for relevant Control code bits.

Bit 0 of the Control code allows you to select binary or decimal numeric logic for each Counter. If you select binary logic, then you may load a Counter with any value, ranging from 0000_{16} through $FFFF_{16}$. The Counter times out when it decrements to 0; therefore the largest value you can load is 0000_{16}. When decrementing with binary logic, this will cause a timeout after $65{,}536_{10}$ counts.

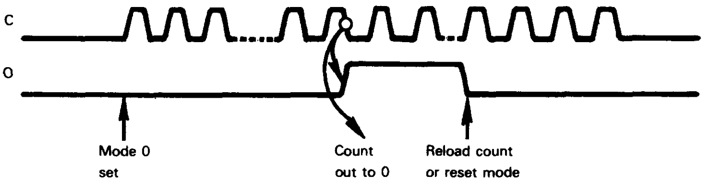
If you select decimal mode for any Counter/Timer, then you must load the Timer with some valid decimal number in the range 0000_{10} through 9999_{10}. The Counter/Timer will decrement only in the range of valid decimal numbers. Once again, however, loading 0000_{10} will provide the longest time interval, equivalent to 10,000 counts.

8253 COUNTER/ TIMER MODES

We will now examine the six modes in which the 8253 Counter/Timer may operate. But first a word of clarification. **The three Counter/Timers are completely separate and independent entities,** such that neither the mode nor the logic condition of one Timer has any impact whatsoever on another Timer. That is to say, the three Timers can be operated in any combination of modes and they can be started or stopped independently.

Let us now consider the six modes in detail.

Mode 0. This is referred to as "interrupt-on-terminal-count" and may be illustrated as follows:



When a Control code is output selecting Mode 0 for a Counter, that Counter's output O is set low. The Counter must be loaded with an appropriate count, before or after the Counter is selected with Mode 0. As soon as the Counter has been loaded, it will start the count down; on decrementing to 0, the output will go high and will remain high until the Counter is reloaded or reset.

Now the instant at which the Counter starts to decrement will depend on the byte access which was specified by bits 5 and 4 of the Control code. This may be illustrated as follows:

Control Bits

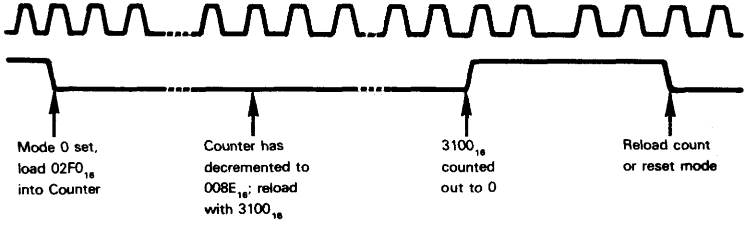
5	4	
1	0	Store data in high order Counter byte, clear low order Counter byte and start decrementing.
0	1	Store data in low order Counter byte, clear high order Counter byte and start decrementing.
1	1	Expect two Counter byte accesses. Store first byte in low order Counter eight bits, store second byte in high order Counter eight bits; after receiving second byte, start decrementing.

The following instruction sequence accesses Counter/Timer 1 in Mode 0, specifying binary counting and 128 counts before 0 will be output high:

```
MVI     A,50H    ;OUTPUT CONTROL CODE
OUT     7
MVI     A,80H    ;OUTPUT INITIAL COUNTER/TIMER VALUE
OUT     5
```

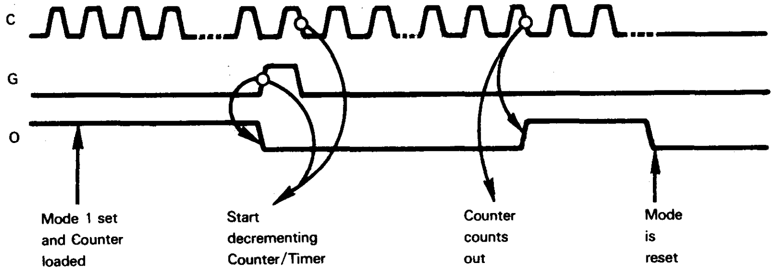
This instruction sequence assumes I/O addresses 4, 5, 6 and 7 select the 8253 device.

In Mode 0 you can read the Counter/Timer contents at any time — but this requires a special programming technique, since you are reading a value which may change while being read; this special technique is described after the six modes. **You can also write into the Counter/Timer at any time;** but if you do, the Counter/Timer will start decrementing again, beginning with the new value loaded into the Counter/Timer. This may be illustrated as follows:

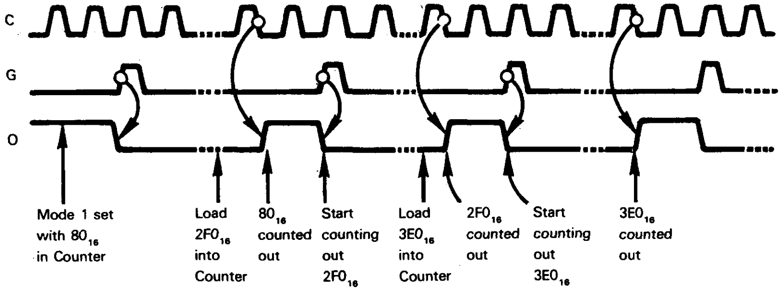


If you write into a Counter/Timer while the Counter/Timer is decrementing, then the decrementing process stops as soon as you write the first byte into the Counter/Timer. The decrementing process starts again as soon as you have written the second byte into the Counter/Timer. The Counter/Timer is inactive during the time interval between writing the first and second bytes.

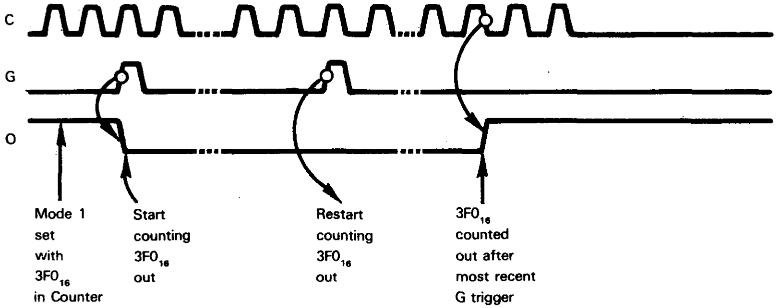
Mode 1. This is referred to as a programmable one-shot mode and may be illustrated as follows:


C
G
O
Mode 1 set and Counter loaded
Start decrementing Counter/Timer
Counter counts out
Mode is reset

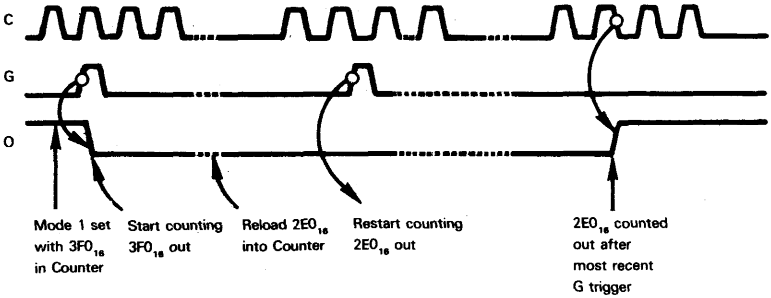
Mode 1 differs from Mode 0 in that the Counter does not start to decrement until triggered by input G. Since input G initiates the time interval, reloading the Counter while it is decrementing has no effect on the current time interval. This may be illustrated as follows:


C
G
O
Mode 1 set with 80_{16} in Counter
Load $2F0_{16}$ into Counter
80_{16} counted out
Start counting out $2F0_{16}$
Load $3E0_{16}$ into Counter
$2F0_{16}$ counted out
Start counting out $3E0_{16}$
$3E0_{16}$ counted out

The one-shot is retriggerable. Whenever a rising edge of G occurs, the time interval will be restarted, whether or not the Counter has counted out to 0. This may be illustrated as follows:


C
G
O
Mode 1 set with $3F0_{16}$ in Counter
Start counting $3F0_{16}$ out
Restart counting $3F0_{16}$ out
$3F0_{16}$ counted out after most recent G trigger

If the Counter was reloaded after G had triggered a one-shot, then on a retrigger, the new Counter value will be used. This may be illustrated as follows:



C
G
O
Mode 1 set with $3F0_{16}$ in Counter
Start counting $3F0_{16}$ out
Reload $2E0_{16}$ into Counter
Restart counting $2E0_{16}$ out
$2E0_{16}$ counted out after most recent G trigger

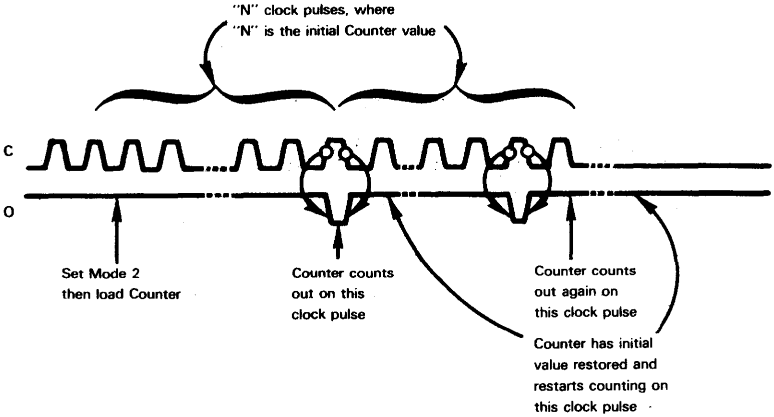
The following instruction sequence accesses Counter/Timer 0 logic in Mode 1, with BCD timing assumed, and 3000_{16} as the initial Timer constant:

```
        MVI     23H     ;OUTPUT CONTROL CODE
        OUT     7
        MVI     30H     ;OUTPUT 3000 DECIMAL TO
        OUT     4       ;TIMER 0
```

The above instruction sequence assumes that the 8253 device is selected by I/O port addresses 4, 5, 6 and 7.

The Control code output to I/O Port 7 specifies that a byte of data will be output only to the high order eight bits of the Timer. The low order eight bits will be set to 00. Thus the decimal value 3000 is created in the Timer by outputting 30 to the high order eight bits.

Mode 2. This is referred to as the rate generator mode and may be illustrated as follows:

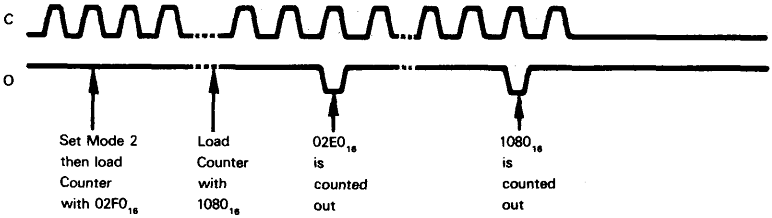


"N" clock pulses, where
"N" is the initial Counter value
C
O
Set Mode 2 then load Counter
Counter counts out on this clock pulse
Counter counts out again on this clock pulse
Counter has initial value restored and restarts counting on this clock pulse

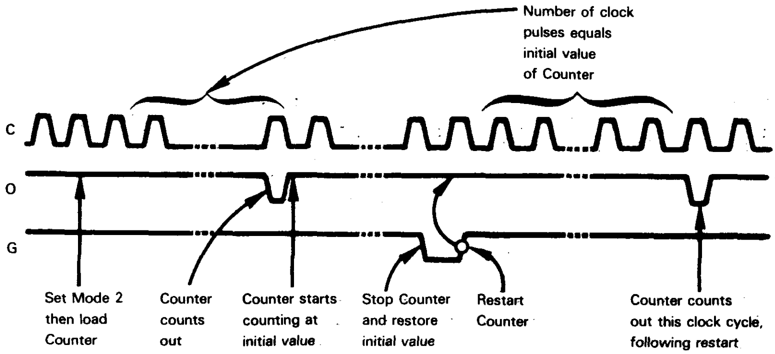
The Counter register does not have to be reloaded. Once set with an initial value, low O pulses will be output indefinitely, with the Counter register contents controlling the number of C pulses between each O pulse.

In Mode 2, the Counter identifies a time interval which will occur between low pulses of output O. The output O low pulse lasts 1 clock pulse.

If you reload the Counter, the current time interval is not affected, but the new Counter value will determine the duration of the next time interval. This may be illustrated as follows:


C
O
Set Mode 2 then load Counter with 02F0$_{16}$
Load Counter with 1080$_{16}$
02E0$_{16}$ is counted out
1080$_{16}$ is counted out

In Mode 2, input G operates as a reset signal. If input low, G immediately forces O high. When subsequently input high, G initiates a new time interval, thus input G may be used to synchronize a Mode 2 Counter with external logic. This may be illustrated as follows:


Number of clock pulses equals initial value of Counter
C
O
G
Set Mode 2 then load Counter
Counter counts out
Counter starts counting at initial value
Stop Counter and restore initial value
Restart Counter
Counter counts out this clock cycle, following restart

Notice that in Mode 2, the output O remains high in the time interval between a Control code selecting Mode 2 and an initial Counter value being loaded into the selected Counter. Thus, whereas input G can be used to synchronize a Counter in mid-execution, program control can be used to synchronize the Counter initially.

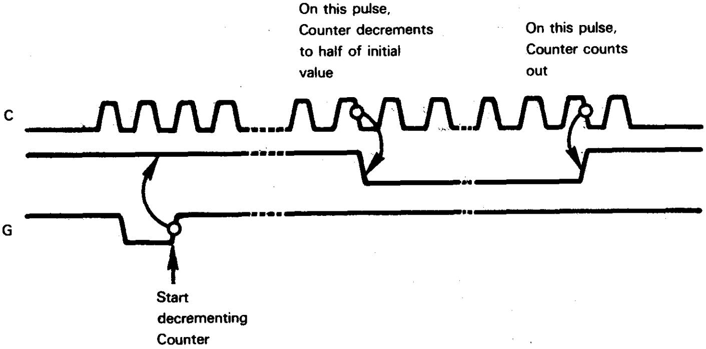
The following instruction sequence illustrates Timer 2 logic being accessed in Mode 2:

```
MVI     B4H     ;OUTPUT CONTROL CODE
OUT     7
MVI     F0H     ;OUTPUT 02F0 TO COUNTER/TIMER 2
OUT     6
MVI     02H
OUT     6
```

The instruction sequence illustrated above shows the two-byte value 02F0 being loaded into the Counter register associated with Timer 2 logic. This specification is made by setting bits 5 and 4 of the Control code to 11; it also means that two bytes of initial Counter value must be output in succession, with the low order byte being output first. The instruction sequence again assumes that the Counter/Timer will be accessed as I/O Ports 4, 5, 6 and 7. I/O Port 6 therefore corresponds to the Counter register for Timer 2 logic, and sequential accesses of I/O Port 6 store a byte of data into the low order, then the high order eight bits of the Counter register associated with Timer 2 logic. Since the 0 bit of the Control code is 0, Counter register contents will be interpreted as binary data.

As soon as a byte of data has actually been deposited in the high order eight bits of the Timer 2 logic Counter register, this register will start to decrement. Notice that the clock signal input to Timer 2 logic need not be derived from the microcomputer system clock, nor need it be related to this clock in any way. Very precise interval timing can therefore be difficult to achieve in Mode 2 — if you initiate the time interval by loading the Timer register; it is quite easy to be one or two clock cycles off. Using the gate input signal G to initiate the time interval is very precise — and is recommended for great accuracy.

Mode 3. This is a "rate generator with square wave output" mode; it differs from Mode 2 only in that the output O remains high for half of the Counter time interval instead of remaining high for a single clock pulse. This may be illustrated as follows:


On this pulse, Counter decrements to half of initial value
On this pulse, Counter counts out
C
G
Start decrementing Counter

If the Counter initial contents is not an even number, then the output will remain high for the extra count.

Mode 4. This is a software triggered strobe. Mode 4 differs from Mode 2 only in that the Gate input is not used to initiate counting. When the Gate input is low in Mode 4, the Counter is disabled; but providing the Gate input is high, the Counter starts to decrement as soon as the second byte of the initial Counter value has been loaded.

Mode 5. This is a hardware triggered strobe. Modes 2 and 5 are almost the same. In both cases the rising edge of the Gate input initiates counting. The Gate input in Mode 2 additionally can be used to disable counting; in Mode 5 Gate input level has no effect.

Thus, in Mode 5 you cannot enable or disable counting by the level of the Gate input.

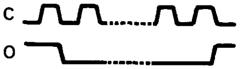
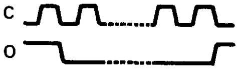
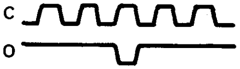
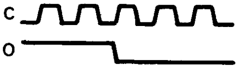
GATE AND OUTPUT SIGNAL SUMMARY

Table 4-14 summarizes the effect of the Gate input in each of the six modes and the shape of the O output which results.

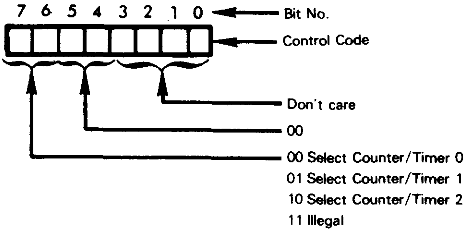
MONITORING 8253 COUNTER/TIMER OPERATION

The 8253 Counter/Timer has no Status register. Operation is monitored by reading the contents of any Counter register, at any time. But two different reading techniques must be used.

Table 4-14. G and O Signal Summary For The 8253 Counter/Timer

MODE	G SIGNAL EFFECT			O SIGNAL SHAPE
	G LOW	G ↑	G HIGH	
0	Disable	None	Enable	
1	None	Initiate counting. Reset O after next clock pulse	None	
2	Disable Set O high	Initiate counting	Enable	
3	Disable Set O high	Initiate counting	Enable	
4	Disable	None	Enable	See Mode 2
5	None	Initiate counting	None	See Mode 2

If you want to read a Counter's contents while it is decrementing, then you must write the following Command code to the Counter/Timer:


7 6 5 4 3 2 1 0
Bit No.
Control Code
Don't care
00
00 Select Counter/Timer 0
01 Select Counter/Timer 1
10 Select Counter/Timer 2
11 Illegal

What this Control code does is load a single, stable count value, from the specified Counter, into a register out of which it may be read. You can now execute two read commands accessing the Counter identified by Control code bits 7 and 6. On the first read, you will access the least significant Counter byte; on the second read, you will access the most significant Counter byte. This may be illustrated by the following instruction sequence, which will read the contents of Counter 1 on the fly:

```
MVI     40H     ;OUTPUT CONTROL CODE TO READ
OUT     7       ;COUNTER 1 CONTENTS
IN      5       ;READ COUNTER 1 CONTENTS AND
MOV     C,A     ;STORE IN BC
IN      5
MOV     B,A
```

Had you not first issued the "Read Counter" command, the Counter would have decremented in the time interval that elapsed between the first and second read; thus you would have obtained low order and high order bytes of different 16-bit values.

You do not need to issue a "Read on the Fly" command to access the contents of the Counter that is not in the process of being decremented. All you have to do is execute two read commands. The first read command will read the least significant byte, while the second read command will read the most significant byte:

```
-
-
-
IN      4       ;READ LOW ORDER BYTE OF COUNTER 0
MOV     C,A     ;SAVE IN C
IN      4       ;READ HIGH ORDER BYTE OF COUNTER 0
MOV     B,A     ;SAVE IN B
```

THE 8259 PRIORITY INTERRUPT CONTROL UNIT (PICU)

This is a very flexible, programmable interrupt handling device; it provides a CALL instruction's object code in response to three interrupt acknowledge ($\overline{\text{INTA}}$) signals; the 8228 System Controller responds to an interrupt acknowledge in this fashion, as described earlier in this chapter. Therefore the 8259 PICU should be looked upon as a companion to the three-chip (8080A, 8224, 8228) microprocessor system.

The 8259 PICU cannot be used with non-8080A systems.

A single 8259 PICU with an 8080A microcomputer system will handle up to eight external interrupts, providing a variety of programmable interrupt priority arbitration schemes.

Alternatively, an 8080A microcomputer system may have a single 8259 PICU designated as a master, controlling up to eight additional 8259 PICUs designated as slaves. This allows a maximum of 64 levels of interrupt priority. Priority arbitration schemes may be set independently for the master and for each slave, resulting in a bewildering profusion of priority arbitration possibilities.

Use extreme caution before including master and slave PICUs within an 8080A microcomputer system. When an application is implemented around a microprocessor with the general speed and performance characteristics of an 8080A, then it is usually more efficient to handle numerous external request lines using multiple CPU configurations and/or programmed polling techniques, rather than interrupts.

The 8259 PICU is fabricated using NMOS technology; it is packaged in a 28-pin plastic DIP. All outputs are TTL compatible.

With reference to the standard logic functions' illustration used throughout this book, the box marked "Direct Memory Access Control Logic" represents the functions implemented by the 8259 PICU. But it is hard to equate the large number of options provided by the 8259 PICU with the interrupt logic provided by other microcomputer systems. An application that needs the 8259 PICU would certainly not be satisfied by Direct Memory Access control logic provided by almost any other device described in this book.

8259 PICU PINS AND SIGNALS

8259 PICU pins and signals are illustrated in Figure 4-53; we will summarize these signals, then discuss how the PICU is used.

From the programmer's point of view, the 8259 PICU will be accessed either as two I/O ports, or as two memory locations. $\overline{\text{CS}}$ is a typical chip select and A0 identifies one of two I/O ports or memory locations. The way you, as a programmer, must interpret the function of each 8259 PICU I/O port or memory location depends on an intricate logical sequence.

The two 8259 addressable locations are accessed via the Data Bus (D0 - D7).

$\overline{\text{IOR}}$ and $\overline{\text{IOW}}$ are standard read and write control signals. If the 8259 PICU is being accessed as two I/O ports, then these two signals will be connected to the $\overline{\text{I/OR}}$ and $\overline{\text{I/OW}}$ controls output by the 8228 System Controller; on the other hand, if the 8259 PICU is being accessed as two memory locations, then IOR and IOW must be connected to the $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ controls output by the 8228 System Controller.

External devices requesting interrupt service have their request signals connected to IR0 - IR7. A high level on any one of these signals will be interpreted as an interrupt request. **An interrupt request is passed on to the CPU via the INT signal.** This is illustrated in Figure 4-54.

In a configuration that includes master and slave 8259 PICUs external logic will connect to the interrupt request signals (IR0 - IR7) of the slave PICUs only. The INT outputs of the slave PICUs will be connected to the interrupt requests (IR0 - IR7) of the master PICU. This is illustrated in Figure 4-55.

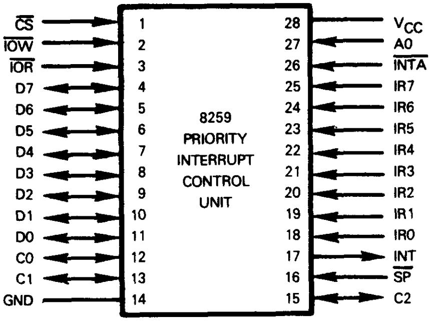
When more than one 8259 PICU is present in a system, **$\overline{\text{SP}}$ identifies the master and slave units.** $\overline{\text{SP}}$ high defines the master, while $\overline{\text{SP}}$ low forces an 8259 PICU to operate as a slave. **$\overline{\text{SP}}$ also determines the sense of the three cascade lines (C0, C1, C2);** these are output lines from the master and input lines to a slave.

The 8080A CPU provides the standard interrupt acknowledge via $\overline{\text{INTA}}$. This interrupt acknowledge will be received by all 8259 PICUs in the system, master or slave.

In a system that includes a master 8259 PICU only, the three bytes of a CALL instruction's object code are output via the Data Bus in response to the three $\overline{\text{INTA}}$ control signals arriving from the 8228 System Controller. The second and third bytes of the CALL instruction's object code provide an address which is unique to the selected interrupt request.

In a configuration that includes master and slave 8259 PICUs, the master PICU outputs the first byte of a CALL instruction's object code; the master also outputs a value between 000 and 111 via the three cascade lines (C0 - C2). This three-bit binary value identifies the interrupt request level being acknowledged — and therefore the slave PICU being selected. The selected slave PICU provides the second and third bytes of the CALL instruction's object code in response to the second and third $\overline{\text{INTA}}$ pulses output by the 8228 System Controller. Thus the slave PICU identifies the interrupt request level it is acknowledging.

The interrupt acknowledge logic of the 8259 PICU is referred to as "Vectoring". Let us examine 8259 vectoring in more detail.



PIN NAME	DESCRIPTION	TYPE
$\overline{\text{CS}}$	Device Select	Input
A0	Identifies PICU as one of two I/O ports or memory locations	Input
D0 - D7	Data Bus	Tristate, Bidirectional
$\overline{\text{IOR}}$	Read control signal	Input
$\overline{\text{IOW}}$	Write control signal	Input
IR0 - IR7	Interrupt request lines to PICU	Input
INT	Interrupt request sent by PICU	Output
$\overline{\text{INTA}}$	Interrupt acknowledge	Input
$\overline{\text{SP}}$	Identifies PICU as either master or slave	Input
C0 - C2	Cascade lines select slave in multiple PICU systems	Output on master Input on slave
V_{CC}, GND	Power and Ground	

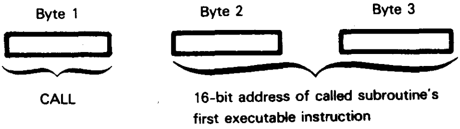
Figure 4-53. 8259 Priority Interrupt Control Unit Signals And Pin Assignments

THE 8259 PICU INTERRUPT ACKNOWLEDGE VECTOR

Vectoring is a general term used to identify an interrupt acknowledge sequence which results in the immediate identification of the interrupting external source. With a non-vectored interrupt acknowledge, the CPU must execute some instruction sequence whose sole purpose is to identify the source of the interrupt — and that assumes more than one possible external interrupting source.

8080A INTERRUPT RESPONSE USING CALL INSTRUCTION

Recall that when an interrupt request is acknowledged by a three-device 8080A microprocessor system, the 8228 System Controller outputs a low pulse on the $\overline{\text{INTA}}$ control line. External logic must interpret the low $\overline{\text{INTA}}$ pulse as a signal to bypass normal instruction fetch logic, and provide the object code for the first instruction to be executed following the interrupt acknowledge. (If this is new to you, refer back to our discussion of the 8080A and 8228 devices.) If a CALL instruction's object code (CD_{16}) is returned to the 8228 System Controller, then low $\overline{\text{INTA}}$ pulses are output for the next two machine cycles — thus making it easy for external logic to fetch all three bytes of a CALL instruction's object code. **The 8259 PICU uses this 8228 logic to supply a three-byte CALL instruction's object code as the first instruction executed following an interrupt acknowledge. But a CALL instruction's object code is interpreted thus:**


Byte 1
Byte 2
Byte 3
CALL
16-bit address of called subroutine's first executable instruction

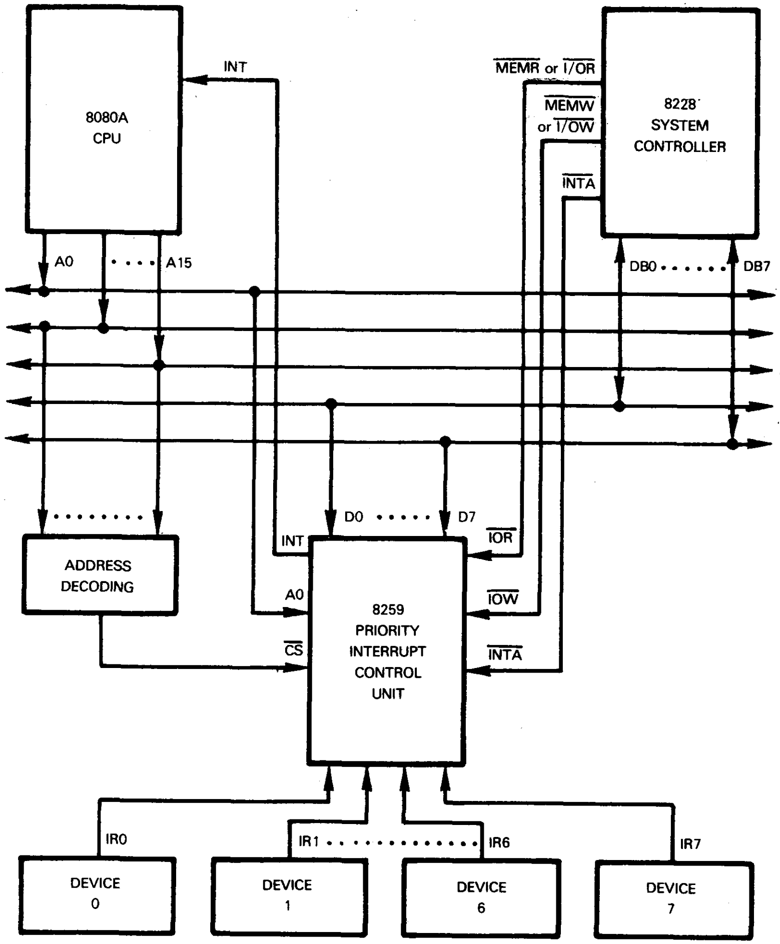


Figure 4-54. A System With One PICU

There are two ways in which the 8259 PICU can compute the address portion of the CALL instruction object code (bytes 2 and 3). These are the two options:

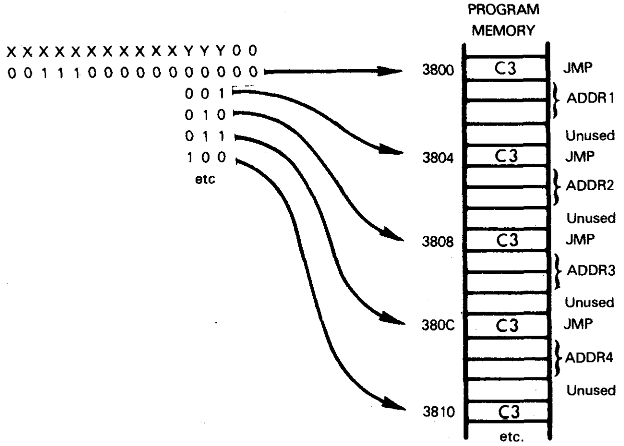
Option 1	Option 2
XXXXXXXXXXXYYY00	XXXXXXXXXXYYY000

X represents binary digits which are defined, under program control, to be a constant portion of the Call address.

Y represents binary digits which identify the interrupt priority level (000 through 111).

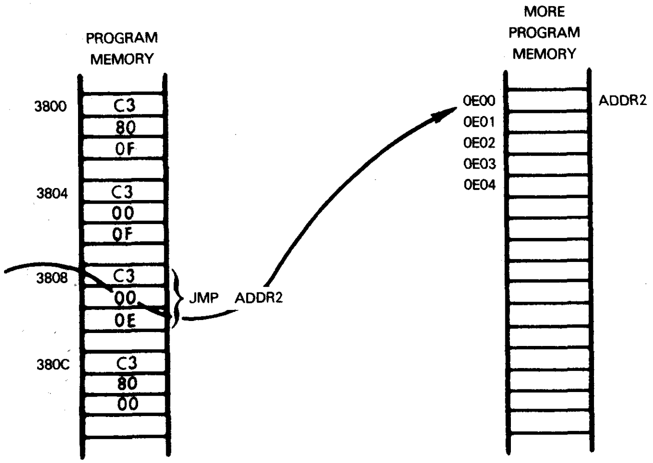
Since the CALL is the first instruction executed following an interrupt acknowledge, it causes program logic to branch to a memory location which is uniquely set aside for a single external interrupting source. **Suppose you have selected CALL instruction**

Option 1, as illustrated above. You would then set aside an area of memory for a jump table, as follows:



Memory addresses have been selected arbitrarily in the illustration above.

Program logic does not have to determine the source of an interrupt. You simply origin separate interrupt service routines at starting addresses specified by the Jump instructions in the jump table. This may be illustrated as follows:



The illustration above arbitrarily assumes that the interrupt request arriving at IR2 has its service routine origined at $0E00_{16}$. In this example, the address vector provided by the 8259 is 3808_{16}:

$$\begin{array}{c} X\,X\,X\,X\,X\,X\,X\,X\,X\,X\,X\,\overbrace{Y\,Y\,Y}^{2}\,0\,0 \\ \underbrace{0\,0\,1\,1}_{3}\,\underbrace{1\,0\,0\,0}_{8}\,\underbrace{0\,0\,0\,0}_{0}\,\underbrace{1\,0\,0\,0}_{8} \end{array}$$

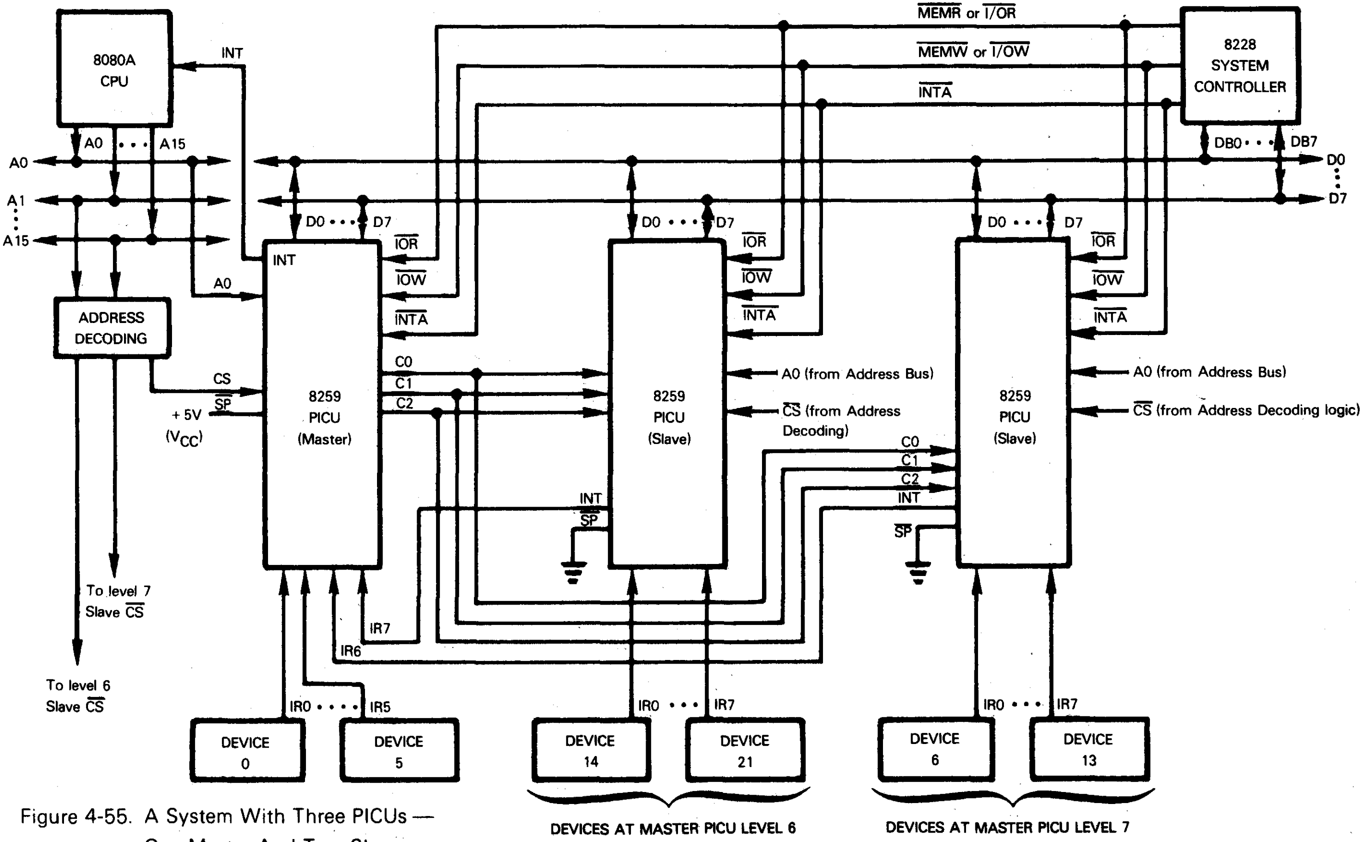


Figure 4-55. A System With Three PICUs — One Master And Two Slaves

At memory location 3808_{16}, the object code for the instruction:

```
        JMP     ADDR2
```

takes us directly to the required interrupt service routine.

8259 PICU PRIORITY ARBITRATION OPTIONS

Priority arbitration logic is used to determine which interrupt request will be acknowledged when two or more interrupt requests exist simultaneously. The 8259 PICU allows interrupt priorities to be specified at two levels — which need to be clearly separated and identified.

As discussed in "Volume I — Basic Concepts", interrupt priority arbitration usually applies to simultaneous interrupt requests; at the instant an interrupt is acknowledged, **if more than one external requesting source is requesting an interrupt, priority arbitration logic decides which single interrupt request will be acknowledged.** Once an interrupt has been acknowledged, priority arbitration has nothing to do with whether the interrupt service routine can itself be interrupted, or by whom.

The 8259 PICU extends interrupt priorities to the service routines themselves. Once an interrupt has been acknowledged, its service routine can only be interrupted by a higher priority interrupt.

If you are unsure of the difference between interrupt priority arbitration at the point when interrupts are acknowledged, as against priority arbitration for the entire duration of an interrupt service routine, then refer back to "Volume I — Basic Concepts", where this subject is covered thoroughly.

Let us now look at the various priority arbitration options provided by the 8259 PICU.

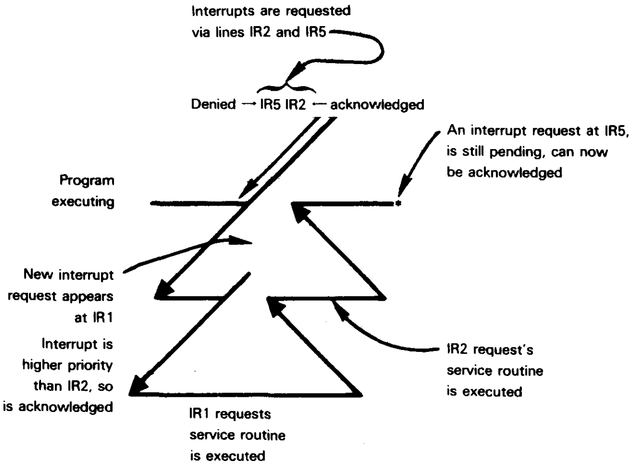
The Fully Nested Mode is the default case. Interrupt priorities are set sequentially from 0 (highest) to 7 (lowest).

As we will describe shortly, **the 8259 PICU must be initialized by an appropriate instruction sequence** before it can be used in any way. **Upon completing programmed initialization, Fully Nested Mode is the priority arbitration option in force.** It takes additional instructions to specify any other priority arbitration option.

8259 PICU INTERRUPT SERVICE ROUTINE PRIORITIES

In Fully Nested Mode, interrupt priorities will never change. An interrupt request arriving at an IR line will never be acknowledged if an interrupt request exists at a higher priority line; or if an interrupt service routine is being executed in response to a higher priority interrupt request. Conversely, once an interrupt has been acknowledged, the interrupt service routine which is subsequently executed may be interrupted only by a higher priority interrupt. It makes no difference whether interrupts have, or have not been disabled, the 8259 PICU will ignore all interrupt requests at priority levels below that of an interrupt service routine currently being executed. For example, suppose interrupts are being requested simultaneously at levels 2 and 5. The level 2 interrupt will be acknowledged and its interrupt service routine will be executed. While the level 2 interrupt is being executed, the level 5 interrupt request will be denied by the 8259 PICU, whether or not interrupts have been disabled at the

CPU. However, if an interrupt request arrives at priority level 1, the PICU will acknowledge this interrupt request, and will allow the level 2 interrupt service routine to be interrupted. This may be illustrated as follows:



It is very important to understand that the 8259 PICU extends interrupt priority logic beyond the interrupt acknowledge, to the interrupt service routine itself. Standard priority arbitration logic does not extend to the interrupt service routine. Thus, in the standard case if interrupts were being requested at priorities 2 and 5, then the priority level 2 request would be acknowledged, but the priority level 2 interrupt service routine could be interrupted by the level 5 interrupt request, unless all interrupts were disabled at the CPU — in which case an interrupt request at level 1 would also be denied.

If you do not want to extend interrupt priorities to the interrupt service routines, you can output a Special Mask Mode command (which we will describe shortly) to selectively enable interrupt requests of lower priority than the currently executing interrupt service routine.

8259 PICU ROTATING INTERRUPT PRIORITIES

Rotating Priority, Mode A is the next option. This differs from the Fully Nested Priority Mode, which we just described, in that after being serviced, a request is immediately relegated to lowest priority. This may be illustrated as follows:

	Priorities assigned to IR lines							
	Lowest							Highest
	7	6	5	4	3	2	1	0
Before first acknowledge	IR7	IR6	IR5*	IR4	IR3	IR2*	IR1	IR0
After first acknowledge	IR2	IR1	IR0	IR7	IR6	IR5*	IR4	IR3
After second acknowledge	IR5	IR4	IR3	IR2	IR1	IR0	IR7	IR6

* *identifies active interrupt requests.*

In a microcomputer system that makes heavy use of interrupts, Rotating Run in Priority Mode A may be a necessary replacement for the default Fully Nested Priority Mode. In the default case, the lowest priority levels may get little or no service if there is heavy interrupt traffic. In an application that does not have a well defined hierarchy of interrupt priorities, a rotation of priorities, as illustrated above, is superior — because it has the effect of giving every priority level equal service.

Rotating Priority Mode A is implemented as a sequence of single programmed events. The microprocessor outputs an appropriate Control code to the 8259 PICU upon completing every interrupt service routine. Thus Rotating Priority Mode A is not a permanently specified PICU condition; each rotation represents a single response to a single Control code — unconnected to previous or future priority selections. For the moment, however, it is not necessary that you understand the programming techniques employed when selecting 8259 interrupt priority modes; that is a subject we will cover after completing the description of all available priority options.

Rotating Priority Mode B gives you some flexibility in determining future priorities. Now under program control you can fix the next division between top and bottom priorities at any time. This may be illustrated as follows:

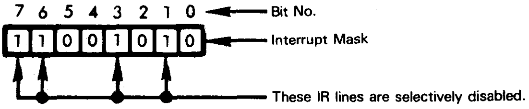
	Priority assigned to IR lines							
	Lowest					Highest		
	7	6	5	4	3	2	1	0
Before first acknowledge	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
After first acknowledge IR5 is defined as lowest priority	IR5	IR4	IR3	IR2	IR1	IR0	IR7	IR6
After next acknowledge IR3 is defined as lowest priority	IR3	IR2	IR1	IR0	IR7	IR6	IR5	IR4
-				-				
-				-				
-				-				
etc.				etc.				

Rotating Priority Mode B allows program logic to determine subsequent interrupt priorities based upon transient system conditions. Rotating Priority Mode B rotates priorities any number of positions to the right, much as you might rotate the bits of an Accumulator.

Like Rotating Priority Mode A, Rotating Priority Mode B depends on the microprocessor outputting an appropriate Control code to the 8259 PICU. However, in Rotating Priority Mode A, rotation can be done only at the conclusion of an interrupt service routine, whereas in Rotating Priority Mode B, priorities can be changed at any time.

Two mask modes allow individual priorities to be selectively disabled. A Simple Mask Mode allows the microprocessor to output an 8-bit mask, where 1 bits will cause corresponding interrupt request lines to be disabled. For example, the mask value CA_{16} will disable interrupt lines IR7, IR6, IR3 and IR1:

8259 PICU INTERRUPT MASKING

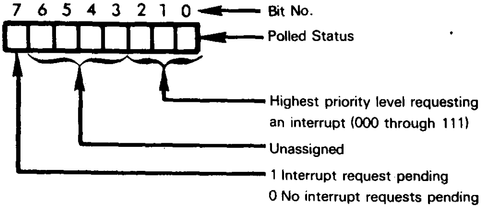


A Special Mask Mode is also provided; it **allows you to enable interrupts at a lower priority level than that of the currently executing interrupt service routine.** By writing a 1 to the appropriate bit of the Mask register, an interrupt level can be disabled while its interrupt service routine is executing. Even though the level is masked, all lower level interrupts will remain disabled until the conclusion of the service routine. Once the current level is masked, however, entering Special Mask Mode will enable all unmasked lower priority interrupt levels. Thus a request can interrupt a service routine operating on a higher priority level.

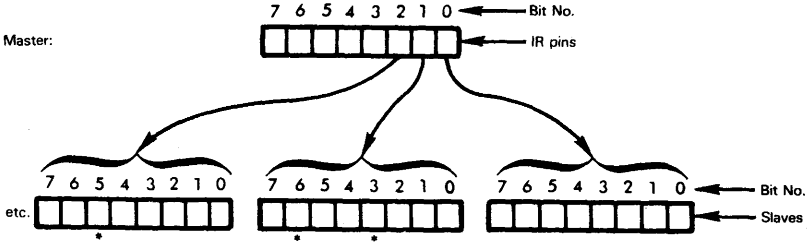
Masks may be superimposed on Rotating Priority Mode A or Mode B without restriction. This allows you to selectively enable and disable individual interrupt request lines, then rotate priorities for the enabled lines. Special Mask Mode also allows you to selectively enable interrupts of lower priority than a currently executed interrupt service routine.

8259 PICU POLLING

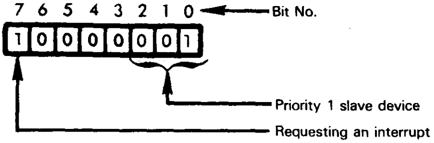
Polled Mode bypasses priority arbitration altogether. If you select Polled Mode, then you must poll the 8259 PICU. You will interpret the polled data as follows:


7 6 5 4 3 2 1 0
Bit No.
Polled Status
Highest priority level requesting an interrupt (000 through 111)
Unassigned
1 Interrupt request pending
0 No interrupt requests pending

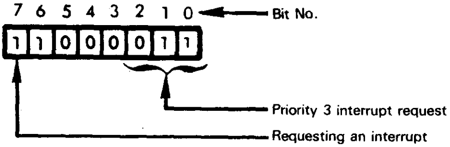
In a configuration that includes master and slave 8259 PICUs, you will first read status from the master PICU. Upon detecting a 1 bit in bit 7, you will poll the slave PICU which is identified by bits 2, 1 and 0 of the master's polled data. The slave poll identifies the highest priority interrupt request. This may be illustrated as follows:



Suppose the * represents interrupt requests. The master poll would return:



The polling program must now poll slave 1; it will read:



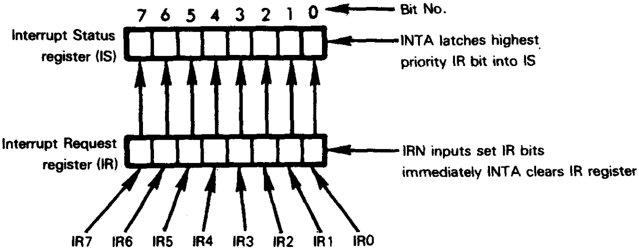
In Polled Mode, the 8259 PICU is not being used as an interrupt processing device at all. In effect, interrupt requests are reduced to status flags, which will be processed by the CPU when it is ready to do so. External logic is no longer able to force the CPU to suspend current program execution; thus the key concept of an interrupt is missing.

While it may not immediately appear obvious, using the 8259 PICU in Polled Mode is possibly one of the most effective ways of utilizing this device. A point we have frequently made, both in Volume I and in Volume II, is that the average microprocessor is simply too slow to efficiently handle random, nested interrupts in a traditional minicomputer fashion. It is faster and more efficient to poll status on a round-robin basis, branching to appropriate subroutines upon detecting a status flag via which external logic has requested service. A detailed discussion of this point may be found in the book "8080 Programming For Logic Design"

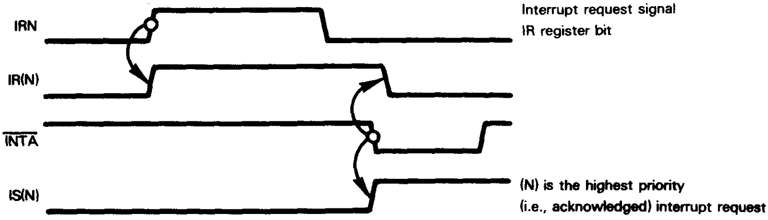
HOW INTERRUPT REQUESTS AND PRIORITY STATUS ARE RECORDED

Internal to the 8259 PICU there are two registers: an Interrupt Request (IR) register and an Interrupt Status (IS) register.

The Interrupt Request and Interrupt Status registers may be looked upon as receiving external interrupt request status in a cascaded fashion as follows:


7 6 5 4 3 2 1 0
Bit No.
Interrupt Status register (IS)
INTA latches highest priority IR bit into IS
Interrupt Request register (IR)
IRN inputs set IR bits immediately INTA clears IR register
IR7 IR6 IR5 IR4 IR3 IR2 IR1 IR0

Any active interrupt request appearing on the interrupt request lines IR0 - IR7 will set corresponding bits of the Interrupt Request register. When any interrupt is acknowledged, the acknowledged interrupt's bit in the Interrupt Status register is set; simultaneously, all bits of the Interrupt Request register are reset. This may be illustrated as follows:


IRN
IR(N)
INTA
IS(N)
Interrupt request signal
IR register bit
(N) is the highest priority
(i.e., acknowledged) interrupt request

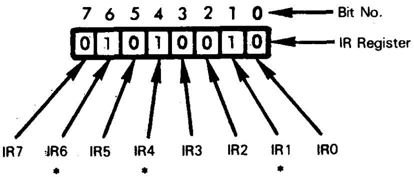
In order to reset any bit of the Interrupt Status register you must issue a specific "End-Of-Interrupt" instruction which we will describe shortly.

You may therefore look upon the Interrupt Request register as identifying active, but unacknowledged interrupt requests. Notice that Interrupt Request status is not preserved across an acknowledge. This means external logic must hold its Interrupt Request true until it has been selected and acknowledged.

You may look upon the Interrupt Status register as identifying the interrupt requests which are currently being serviced. If you do not nest interrupts, then only one bit of the Interrupt Status register will be set at any time. If you do nest interrupts, then more than one bit of the Interrupt Status register may be set — for the interrupt request being serviced currently and for any interrupt requests which were being serviced, but were themselves interrupted. But remember you can misuse the Interrupt Status register. If you do not end interrupt service routines by outputting an "End-Of-Interrupt" command to the 8259 PICU, then bits of the Interrupt Status register will remain set after the appropriate interrupt has been serviced.

If you use a mask to inhibit interrupt levels, then the inhibit logic will prevent bits of the Interrupt Request and Interrupt Status register from being set for the inhibited interrupt levels.

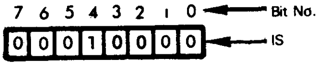
The Interrupt Request (IR) register stores a 1 bit at every requesting level; it may be visualized as a simple reflection of IR input signals:



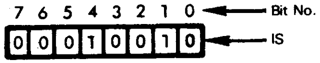
* represents active interrupt requests

The Interrupt Status (IS) register reflects the status of current interrupt priority arbitration logic. Whenever an interrupt is acknowledged, the IS bit corresponding to the interrupt level is set. This bit is reset by the End-Of-Interrupt (EOI) instruction at the end of the interrupt service routine. We will tell you how to issue an EOI instruction shortly.

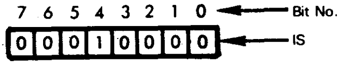
Suppose the 8259 PICU is operating in the default mode: fully nested interrupts, no mask bits set. An interrupt request is made at level 4. When this interrupt is acknowledged, bit 4 of the IS register is set:



and interrupts at levels 5, 6 and 7 are disabled, since they are of lower priority than level 4. While the level 4 request is being serviced, a request is made at level 1. Since level 1 has higher priority, it will be acknowledged, interrupting the level 4 service routine. IS will look like this:



Now interrupt levels 2 through 7 are disabled. At the conclusion of the level 1 service routine, EOI will reset bit 1:



thus enabling interrupt levels 2 and 3 — and level 4, whose service routine can now continue. On the next EOI, assuming no further interruptions, bit 4 of IS will be reset, at which time levels 5, 6 and 7 will again be enabled.

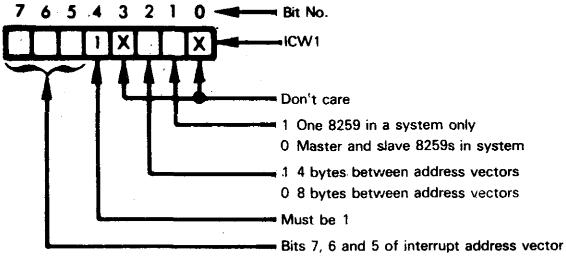
In priority modes other than the Fully Nested Mode (Rotating Priorities A and B and Special Mask Mode) the 8259 PICU cannot be depended on to reset the correct IS bit when it receives the usual EOI. Therefore, it is sent a special EOI which specifies which level's service routine is ending — and therefore which IS bit is to be reset.

PROGRAMMING THE 8259 PICU

As we have already stated, the 8259 PICU appears to the programmer as two I/O ports, or memory locations. However, there are a number of ways in which data

written to, or read from either location may be interpreted. Let us begin by defining these interpretations; then we will explain the sequence in which Co codes should be written, and statuses read, in order to access the capabilities of the 8259 PICU.

Control codes output to the lower I/O port or memory address (A0 ⇒ 0) may terpreted in one of three ways, labeled Initialization Control Word 1 (ICW1) and tion Control Words 2 and 3 (OCW2 and OCW3):



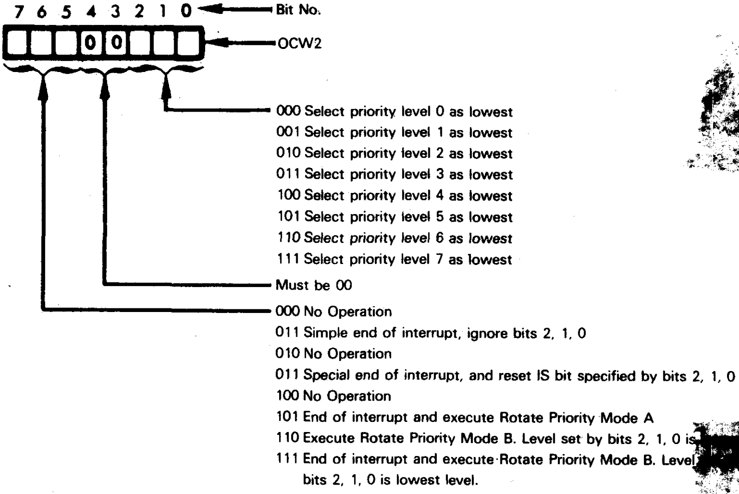
Don't care

1 One 8259 in a system only
0 Master and slave 8259s in system

1 4 bytes between address vectors
0 8 bytes between address vectors

Must be 1

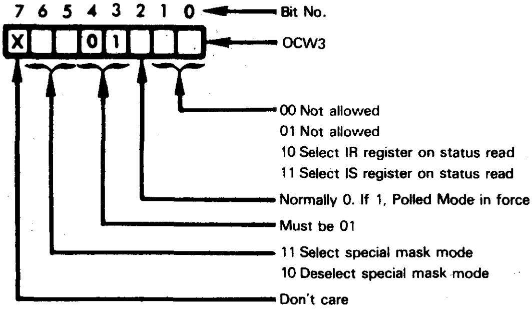
Bits 7, 6 and 5 of interrupt address vector



000 Select priority level 0 as lowest
001 Select priority level 1 as lowest
010 Select priority level 2 as lowest
011 Select priority level 3 as lowest
100 Select priority level 4 as lowest
101 Select priority level 5 as lowest
110 Select priority level 6 as lowest
111 Select priority level 7 as lowest

Must be 00

000 No Operation
011 Simple end of interrupt, ignore bits 2, 1, 0
010 No Operation
011 Special end of interrupt, and reset IS bit specified by bits 2, 1, 0
100 No Operation
101 End of interrupt and execute Rotate Priority Mode A
110 Execute Rotate Priority Mode B. Level set by bits 2, 1, 0 is
111 End of interrupt and execute Rotate Priority Mode B. Level
bits 2, 1, 0 is lowest level.



00 Not allowed
01 Not allowed
10 Select IR register on status read
11 Select IS register on status read

Normally 0. If 1, Polled Mode in force

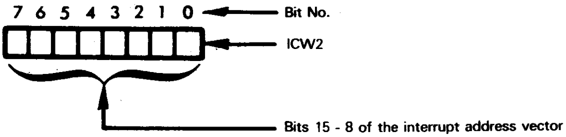
Must be 01

11 Select special mask mode
10 Deselect special mask mode

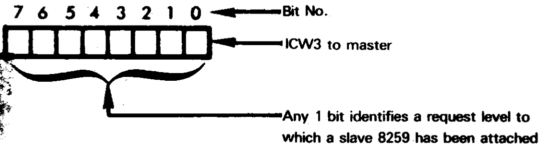
Don't care

When reading from the lower address (A0 = 0), the condition of the most recently issued OCW3 bits 0 and 1 determine what will be read. If these two bits were 01, the Interrupt Request register (IR) is read; if these two bits are 11, the Interrupt Status register [illegible]

[illegible] **output to the higher I/O port or memory address (A0 = 1) may also [illegible]d in one of three ways.** After an ICW1 control has been output to the [illegible]s (A0 = 0), either one, or two Control codes must be output to the higher [illegible]1). If ICW1, bit 1 is 1, a second Control code (ICW2) must be output to the higher address (A0 = 1) of the master 8259 PICU, and to every slave 8259 PICU, that may be present. This is the format of ICW2:

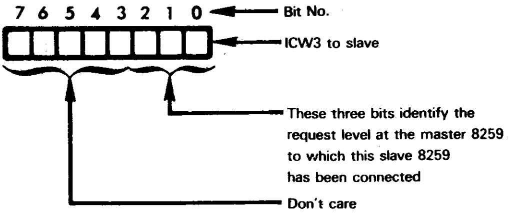


If ICW1, bit 1 is 0, ICW2, as illustrated above, must be output — and it must be followed by a second Control code (ICW3), output to the higher address (A0 = 1) of the master 8259 PICU, and then to each slave 8259 PICU. The master 8259 will interpret ICW3 as follows:



7 6 5 4 3 2 1 0
Bit No.
ICW3 to master
Any 1 bit identifies a request level to which a slave 8259 has been attached

[illegible] will interpret ICW3 as follows:



7 6 5 4 3 2 1 0
Bit No.
ICW3 to slave
These three bits identify the request level at the master 8259 to which this slave 8259 has been connected
Don't care

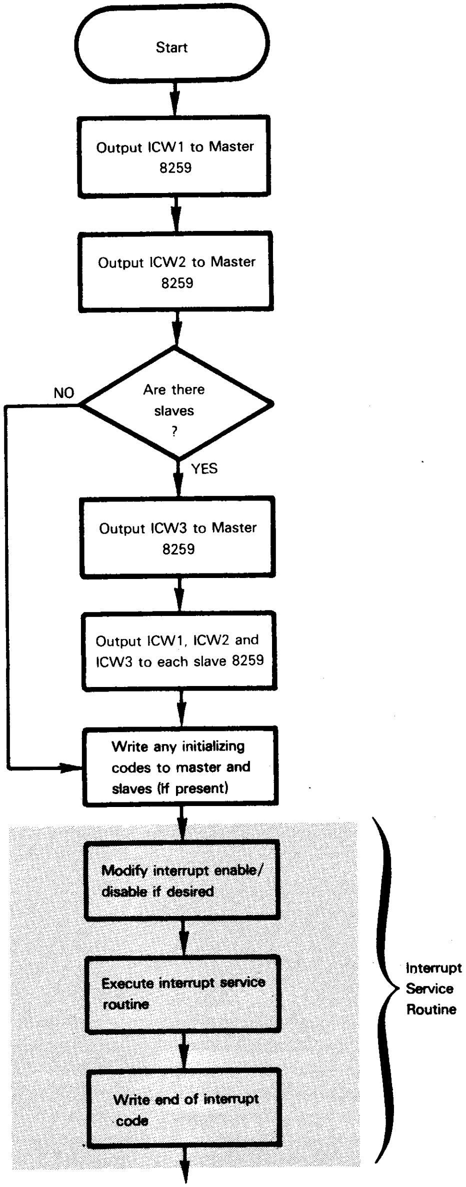
[illegible]th a single 8259, therefore, has ICW1, then ICW2 output to it.

[illegible]h master and slave 8259 devices must have ICW1, ICW2 and ICW3 output [illegible]er, then ICW1, ICW2 and ICW3 output to each slave.

[illegible]nitiation sequence has been completed, when reading or writing to the higher I/O port address (A0 = 1), the Interrupt Mask register is accessed. Writing a 1 into any bit position will disable corresponding IR line requests. 0 bits enable interrupt requests at corresponding IR lines. When you return to the initiation sequence, the higher I/O port address again accesses ICW2 or ICW3.

8259 PICU INTERRUPT MASK

We will now examine the normal sequence in which the 8259 PICU will be programmed. Programming logic may be defined as follows:


Start
Output ICW1 to Master 8259
Output ICW2 to Master 8259
Are there slaves ?
NO
YES
Output ICW3 to Master 8259
Output ICW1, ICW2 and ICW3 to each slave 8259
Write any initializing codes to master and slaves (if present)
Modify interrupt enable/ disable if desired
Execute interrupt service routine
Write end of interrupt code
Interrupt Service Routine

Using arbitrary data, the initiation sequence for a single 8259 PICU system may be illustrated as follows:

```
        MVI        PICUL,12H          ;WRITE OUT ICW1
        MVI        PICUH,40H          ;WRITE OUT ICW2
```

The labels PICUL and PICUH address the lower and higher 8259 PICU addressable locations, respectively.

The two instructions above assume that the 8259 PICU is being addressed as memory. The two immediate data bytes specify an interrupt address vector beginning at location 4000_{16}, incrementing eight bytes with each priority level.

Now consider a configuration where there is a master PICU and three slave PICUs connected to IR0, IR1 and IR2. Here is the initiating instruction sequence required:

```
 INITIALIZE MASTER PICU
        MVI        PICUL,14H          ;WRITE OUT ICW1
        MVI        PICUH,40H          ;WRITE OUT ICW2
        MVI        PICUH,07H          ;IDENTIFY SLAVES TO MASTER
 INITIALIZE FIRST SLAVE PICU
        MVI        SPCL1,10H          ;WRITE OUT ICW1
        MVI        SPCH1,48H          ;WRITE OUT ICW2
        MVI        SPCH1,0            ;IDENTIFY PRIORITY TO SLAVE
 INITIALIZE SECOND SLAVE PICU
        MVI        SPCL2,30H          ;WRITE OUT ICW1
        MVI        SPCH2,48H          ;WRITE OUT ICW2
        MVI        SPCH2,1            ;IDENTIFY PRIORITY TO SLAVE
 INITIALIZE THIRD SLAVE PICU
        MVI        SPCL3,52H          ;WRITE OUT ICW1
        MVI        SPCH3,48H          ;WRITE OUT ICW2
        MVI        SPCH3,2            ;IDENTIFY PRIORITY TO SLAVE
```

Since there is a single master, and three slaves, there must be four sets of initiating instructions.

First, we initiate the master. Again, the interrupt address vector is origined at 4000_{16}. This origin and the specification that four bytes will separate each vector will be used when interrupts are requested on levels to which no slave 8259 PICUs are connected. In this case the value 07_{16} is output indicating that IR0, IR1 and IR2 have connected slaves.

Slave initiation is straightforward. The first slave PICU has labels SPCL1 and SPCH1, representing the lower and higher addressable locations. SPCL2 and SPCH2 are second slave PICU labels, while SPCL3 and SPCH3 are third slave PICU labels.

All three slave PICUs specify a four-byte displacement between interrupt address vectors. Initial origins of 4800_{16}, 4820_{16} and 4840_{16} are specified for slave 1, 2 and 3, respectively. Notice that the second byte written out to the high order address SPCH1, SPCH2 or SPCH3 identifies the slave's priority.

Once 8259 PICUs have been initiated, programmable features are controlled by outputting appropriate Control codes and inputting appropriate status. Every interrupt service program must end by outputting an "End-Of-Interrupt" Control code to the 8259 PICU. Any form of "End-Of-Interrupt" Control code will do. Otherwise, there is no well defined sequence in which controls and status should be used.

Table 4-15. A Summary Of 8259 PICU Operations

OPERATION	INSTRUCTION SEQUENCE
Select Fully Nested Mode	None. This is selected after initiation.
Issue simple End Of Interrupt command	Output 20_{16} (OCW2) to PICUL.
Rotate Priorities Mode A with End Of Interrupt	Output $A0_{16}$ (OCW2) to PICUL.
Rotate Priorities Mode B without End Of Interrupt	Output CN_{16} (OCW2) to PICUL. N is the new lowest priority.
Rotate Priorities Mode B with End Of Interrupt	Output EN_{16} (OCW2) to PICUL. N is the new lowest priority.
Output an interrupt mask	Output mask byte to PICUL any time after initiation sequence.
Read interrupt mask	Input PICUH.
Enter special mask mode	Output OCW3 to PICUL with 68_{16} in lower 7 bits.
Exit special mask mode	Output OCW3 to PICUL with 48_{16} in lower 7 bits.
Specify Polled Mode	Output OCW3 to PICUL with $0C_{16}$ in lower 7 bits.
Poll any PICU	Output OCW3 to PICUL with 011 in bits 4, 3, 2, then immediately read from PICUL.
Read IR Status	Output OCW3 to PICUL with $0A_{16}$ in lower 7 bits. Then read from PICUL.
Read IS Status	Output OCW3 to PICUL with $0B_{16}$ in lower 7 bits. Then read from PICUL.
Reset an IS status bit	Output $6N_{16}$ (OCW2) to PICUL if End Of Interrupt. N is the IS status bit to be reset.

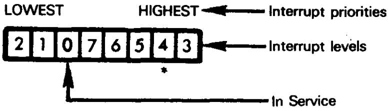
PICUL identifies the PICU lower address (A0 = 0).
PICUH identifies the PICU higher address (A0 = 1).

Here is an example of the end of an interrupt service routine:

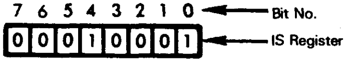
```
        MVI     PICUL,20H       ;SIMPLE END OF INTERRUPT
        RET                     ;RETURN TO INTERRUPTED SEQUENCE
```

The simplest "End-Of-Interrupt" (EOI) is sent as OCW3. This command will reset the highest set bit in the IS register. Notice that we thus assume that this interrupt occurred in Fully Nested Priority Mode, where the highest bit corresponds to the highest priority level.

In other priority schemes, however, the interrupt level being serviced may not correspond to the highest set bit of the IS register. Suppose the interrupt handling scheme is Rotating Priority Mode B with level 2 the lowest priority and a level 0 request being serviced:



A request at level 4 (*) will interrupt the level 0 routine. The IS register would look like this:



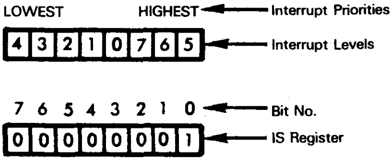
A simple EOI in the level 4 service routine will now reset bit 0 — which is wrong. The following instruction sequence will reset the correct IS bit and return:

```
MVI      PICUL,64H       ;END LEVEL 4 INTERRUPT
RET                      ;RETURN TO INTERRUPTED SEQUENCE
```

Since we are rotating priorities, the following would be preferable:

```
MVI      PICUL,E4        ;END LEVEL 4 INTERRUPT AND MAKE
                         ;LEVEL 4 LOWEST PRIORITY
RET                      ;RETURN TO INTERRUPTED SEQUENCE
```

The priorities and IS register now look like this:



Either of the suggested EOI instructions would allow the level 0 routine to resume.

THE 8214 PRIORITY INTERRUPT CONTROL UNIT (PICU)

The 8214 PICU performs much the same functions as the 8259 PICU, however the 8214 device is far simpler. It has eight lines via which external logic can request an interrupt. The highest priority interrupt request is identified via three output signals which accompany an interrupt request output. The only programmable feature of the 8214 PICU is a Current Status register; into this register you can write an interrupt level which becomes a priority cutoff, below which all interrupt requests will be denied.

Like the 8257, the 8214 PICU is normally used in conjunction with an 8212 I/O port. However, the 8212 I/O port is used in a totally different way. The 8257 DMA Controller needs the 8212 I/O port in order to demultiplex its Data Bus. The 8214 PICU uses the 8212 I/O port in order to create a Restart instruction's object code. Figure 4-58 illustrates a typical configuration.

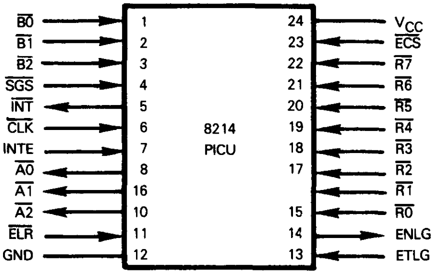
Do not dismiss the 8214 PICU simply because it has fewer capabilities than the 8259 PICU. For most simple applications the 8214 PICU is more than adequate — and being a smaller package it will occupy less space on a PC card.

The 8214 PICU is fabricated using NMOS technology; it is packaged as a 24-pin DIP.

8214 PRIORITY INTERRUPT CONTROL UNIT PINS AND SIGNALS

Figure 4-56 illustrates 8214 PICU pins and signals. Signals are quite elementary and therefore will be covered along with a description of 8214 capabilities.

Figure 4-57 presents a simplified logic diagram for the 8214 PICU.



PIN NAME	DESCRIPTION	TYPE
$\overline{R0}$ - $\overline{R7}$	External interrupt request lines	Input
$\overline{A0}$ - $\overline{A2}$	Identification of highest priority interrupt request	Output
$\overline{INT}$	Interrupt signal	Output
$\overline{B0}$ - $\overline{B2}$	Current status	Input
$\overline{SGS}$	Status Group Select (Master comparator disable)	Input
$\overline{ECS}$	Enable Current Status (Chip Select)	Input
ENLG	Enable Next Level Group on daisy chain	Output
ETLG	Enable This Level Group on daisy chain	Input
$\overline{ELR}$	Enable Level Read	Input
INTE	Master Interrupt Enable	Input
$\overline{CLK}$	System clock (Φ2 TTL)	Input
V_{CC}, GND	Power and Ground	

Figure 4-56. 8214 Priority Interrupt Control Unit Signals And Pin Assignments

External logic requests interrupts by placing a low input at one of the signals $\overline{R0}$ - $\overline{R7}$. Providing conditions allow the 8214 PICU to acknowledge an interrupt request, **outputs $\overline{A0}$ - $\overline{A2}$ will identify the highest priority line on which an interrupt is currently being requested; the output on $\overline{A0}$ - $\overline{A2}$ will be accompanied by a low output at $\overline{INT}$.**

Interrupt request priorities are ordered sequentially from $\overline{R7}$ which has the highest priority, through $\overline{R0}$ which has the lowest priority. The output at $\overline{A0}$, $\overline{A1}$ and $\overline{A2}$ identifies the highest priority interrupt request input as follows:

Priority	Pin	$\overline{A0}$	$\overline{A1}$	$\overline{A2}$
highest	$\overline{R7}$	1	1	1
	$\overline{R6}$	1	1	0
	$\overline{R5}$	1	0	1
	$\overline{R4}$	1	0	0
	$\overline{R3}$	0	1	1
	$\overline{R2}$	0	1	0
	$\overline{R1}$	0	0	1
lowest	$\overline{R0}$	0	0	0

Priority arbitration logic can be modified by writing a four binary digit code to the Current Status register of the 8214 PICU. The chip select $\overline{ECS}$ must be set true in order to write into the Current Status register. This $\overline{ECS}$ may be generated in any fashion, so the 8214 PICU may be addressed as a single I/O port or a single memory location.

The Current Status register is write-only.

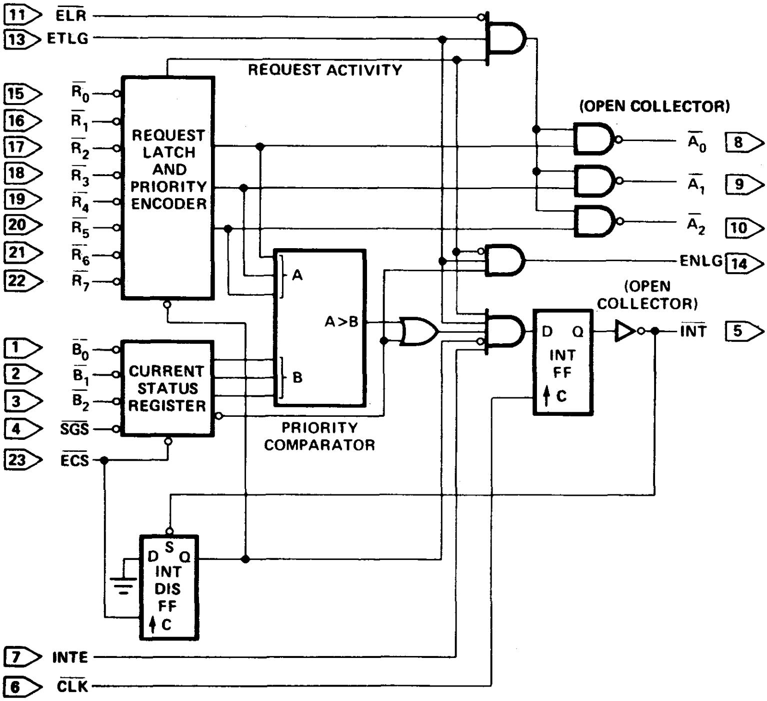
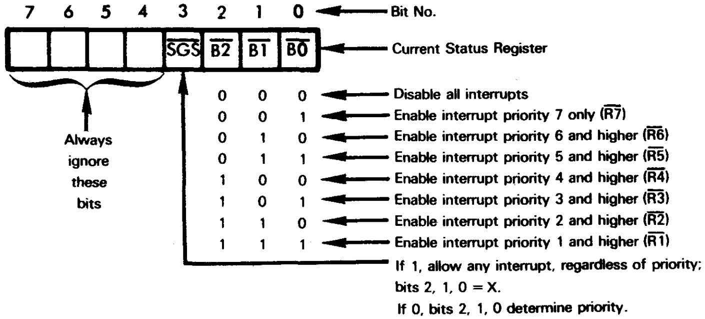


Figure 4-57. Logic Of The 8214 Priority Interrupt Control Unit

Data written into the Current Status register will be interpreted as follows:


7 6 5 4 3 2 1 0
Bit No.
$\overline{\text{SGS}}$ $\overline{\text{B2}}$ $\overline{\text{B1}}$ $\overline{\text{B0}}$
Current Status Register
Always ignore these bits

B2	B1	B0	
0	0	0	Disable all interrupts
0	0	1	Enable interrupt priority 7 only ($\overline{\text{R7}}$)
0	1	0	Enable interrupt priority 6 and higher ($\overline{\text{R6}}$)
0	1	1	Enable interrupt priority 5 and higher ($\overline{\text{R5}}$)
1	0	0	Enable interrupt priority 4 and higher ($\overline{\text{R4}}$)
1	0	1	Enable interrupt priority 3 and higher ($\overline{\text{R3}}$)
1	1	0	Enable interrupt priority 2 and higher ($\overline{\text{R2}}$)
1	1	1	Enable interrupt priority 1 and higher ($\overline{\text{R1}}$)

If 1, allow any interrupt, regardless of priority;
bits 2, 1, 0 = X.
If 0, bits 2, 1, 0 determine priority.

As illustrated above, bit 3 of the Current Status register input is a master comparator disable signal; bits 2, 1 and 0 represent the complement of the cutoff priority level.

8214 devices contain logic which allows them to be cascaded in a daisy chain fashion. To implement daisy chain logic, you must connect the ENLG output of one 8214 device to the ETLG input of the next. The highest priority 8214 device in the daisy chain must have its ETLG input pulled high by connecting it to V_{CC}. If the highest priority 8214 PICU has no active interrupt request, it outputs a high ENLG which enables the next 8214 PICU in the daisy chain. The first 8214 PICU in the daisy chain with an active interrupt request outputs ENLG low and that disables all 8214 PICUs lower in the daisy chain. This is standard daisy chain logic, as described in Volume I.

***You can selectively disable 8214 PICUs in a daisy chain by inputting a high level* signal at $\overline{\text{ELR}}$.** This will force $\overline{\text{A0}}$, $\overline{\text{A1}}$ and $\overline{\text{A2}}$ to all output 111.

There is also a master interrupt enable; it is INTE. A low level at this input will unconditionally disable the 8214 device. INTE will normally be connected to the INTE output from the 8080A CPU *in order* to ensure that 8214 PICUs do not attempt to request interrupts while interrupts have been disabled.

Once an interrupt request appears at an 8214 device, the interrupt request is passed on via $\overline{\text{INT}}$; the request level is identified via $\overline{\text{A0}}$, $\overline{\text{A1}}$ and $\overline{\text{A2}}$. As soon as the 8214 PICU requests *an interrupt, all further interrupts are inhibited, and remain inhibited* until the CPU writes to the Current Status register. **The device select line $\overline{\text{ECS}}$ also acts as an interrupt acknowledge line, enabling 8214 logic to process another interrupt request.** This has two implications:

1) External *logic must keep an* interrupt request active at its appropriate input pin until the interrupt has been acknowledged. The 8214 PICU has no control output which can be used by external logic to determine that its interrupt has been acknowledged. Acknowledge logic must be based on the $\overline{\text{A0}}$ - $\overline{\text{A2}}$ outputs from the 8214, plus the INTA acknowledge signal output by the 8080A CPU.

2) Just as external logic must maintain an interrupt request until the request has been acknowledged, so it must also remove the interrupt request once the interrupt request has been acknowledged. If it does not do so, the same external interrupt request could be acknowledged again and again.

A TYPICAL 8214 PRIORITY INTERRUPT CONTROL CONFIGURATION

Figure 4-58 illustrates a typical configuration in which an 8212 I/O port front-ends an 8214 PICU *in order to generate a Restart instruction's object code.* Observe that the 8212 is configured as an input port with handshaking. By tying the MD input to ground, the STB input clocks the 8212 latches and device select logic enables the output buffers.

The STB input is the complement of the interrupt request output ($\overline{\text{INT}}$) from the 8214 PICU; therefore the instant at which the 8214 requests an interrupt, the current Restart instruction's object code will be latched into the 8212 latches. Device select logic is tied to the interrupt acknowledge signal output by the 8080A CPU ($\overline{\text{INTA}}$); therefore as soon as the interrupt is acknowledged, the Restart instruction's object code held in the 8212 latches will be output via DO0 - DO7.

The interrupt request itself will be passed on by the 8212 I/O Port on the trailing edge of the STB strobe. Thus $\overline{\text{INT}}$ output by the 8214 PICU will be delayed by the STB strobe width before it appears as $\overline{\text{INT}}$ output by the 8212 I/O Port.

THE TMS 5501 MULTIFUNCTION INPUT/OUTPUT CONTROLLER

This is a multifunction peripheral logic device built by Texas Instruments only. It is designed to work with 8080 or 8080A CPUs. The TMS 5501 does not use the 8228 System Controller; it decodes the Data Bus during the SYNC pulse.

The TMS 5501 provides many of the functions provided by the 8255 PPI, 8251 USART, 8253 Programmable Timer/Counter and 8259 Priority Interrupt Control Unit. In each case, the TMS 5501 has simpler logic, with fewer options; but for a very large number of applications, TMS 5501 features will be more than adequate.

Here are the TMS 5501 features provided:

1) **Two external interrupt request lines.**
2) **An 8-bit, parallel input port.**
3) **An 8-bit, parallel output port.**
4) **A single, asynchronous serial I/O channel without handshaking.**
5) **Five programmable timers, each of which times out with an interrupt request after an interval that may range from 64 microseconds to 16.32 milliseconds.**

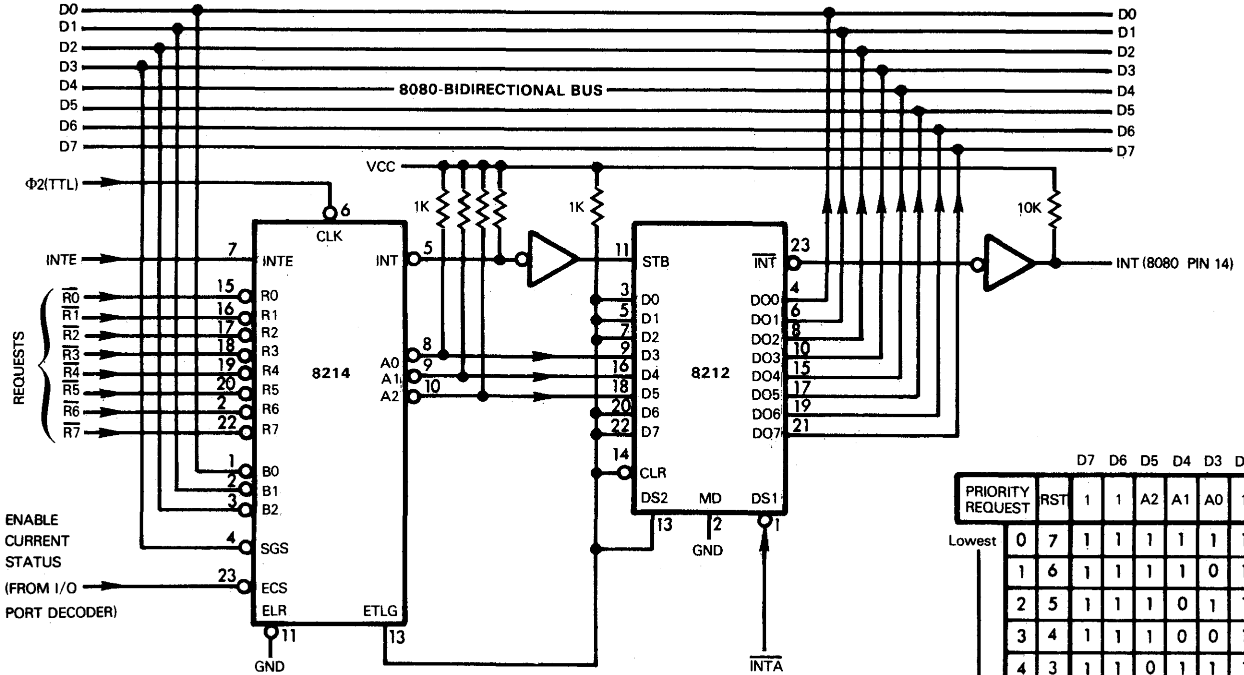
Figure 4-59 illustrates those logic functions in our standard microcomputer system illustration which have been implemented by the TMS 5501.

The TMS 5501 is fabricated using N-channel silicon gate technology and is packaged as a 40-pin DIP.

TMS 5501 DEVICE PINS AND SIGNALS

Figure 4-60 illustrates TMS 5501 device pins and signals. We will begin by summarizing these signals.

There are three data busses. D0 - D7 are the bidirectional Data Bus pins via which data is transferred between the TMS 5501 and the CPU. XI0 - XI7 are the pins via which external logic inputs 8-bit parallel data to the TMS 5501. $\overline{\text{XO0}}$ - $\overline{\text{XO7}}$ are the eight pins via which the TMS 5501 outputs 8-bit parallel data to external logic. Notice that $\overline{\text{XO}}$ lines are negative-true whereas XI lines are positive-true. Optionally XI7 may be used for low priority external interrupt requests.



	PRIORITY REQUEST	RST	D7 1	D6 1	D5 A2	D4 A1	D3 A0	D2 1	D1 1	D0 1
Lowest	0	7	1	1	1	1	1	1	1	1
	1	6	1	1	1	1	0	1	1	1
	2	5	1	1	1	0	1	1	1	1
	3	4	1	1	1	0	0	1	1	1
	4	3	1	1	0	1	1	1	1	1
	5	2	1	1	0	1	0	1	1	1
	6	1	1	1	0	0	1	1	1	1
Highest	7	0	1	1	0	0	0	1	1	1

* RST0 will vector Program Counter to location 0 (zero) and invoke the same routine as "RESET" input to 8080. This could reinitialize the system based on the routine invoked.

(A caution to system programmers.)

Figure 4-58. Generation Of The Restart Instruction Code Following An Interrupt, For An 8080A Microcomputer System

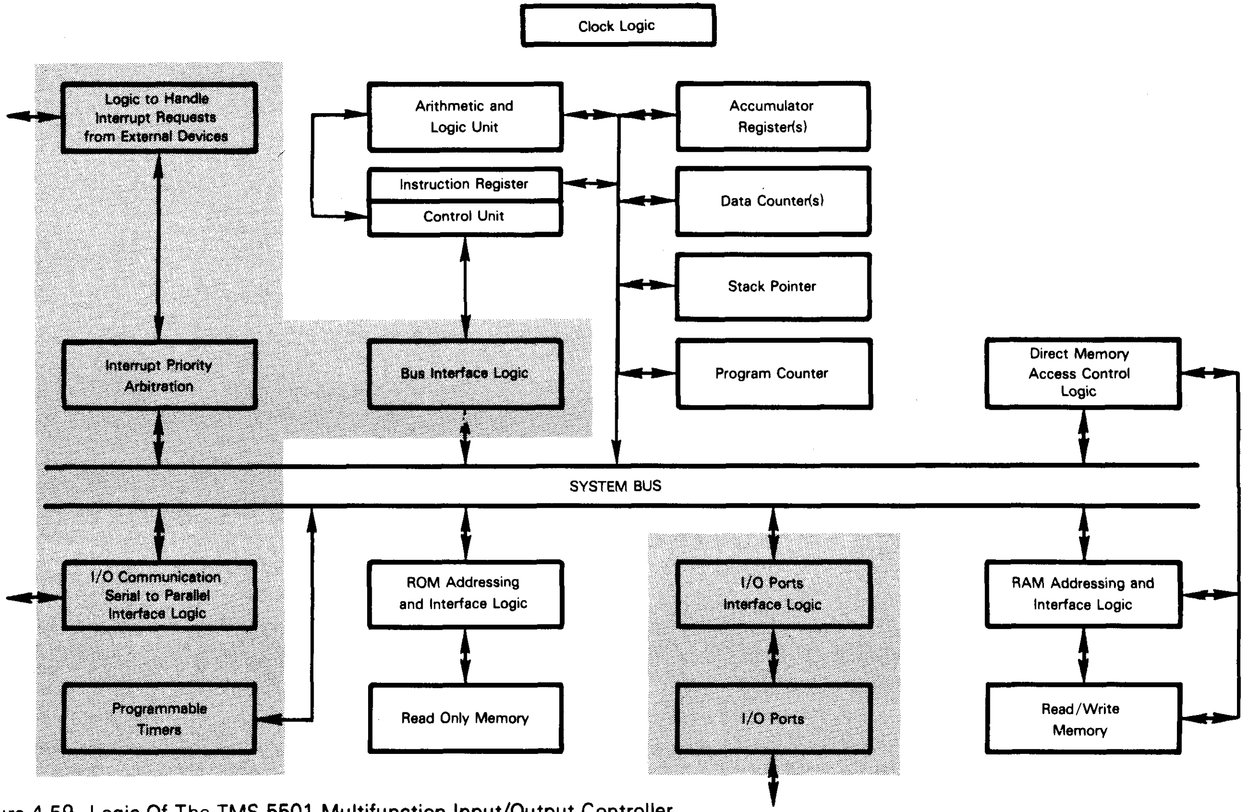
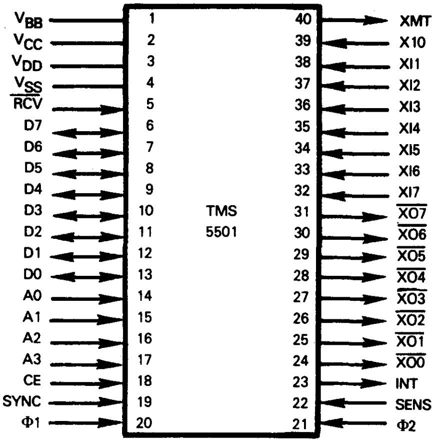

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 4-59. Logic Of The TMS 5501 Multifunction Input/Output Controller



TMS
5501
V_{BB} 1
V_{CC} 2
V_{DD} 3
V_{SS} 4
$\overline{\text{RCV}}$ 5
D7 6
D6 7
D5 8
D4 9
D3 10
D2 11
D1 12
D0 13
A0 14
A1 15
A2 16
A3 17
CE 18
SYNC 19
Φ1 20
40 XMT
39 XI0
38 XI1
37 XI2
36 XI3
35 XI4
34 XI5
33 XI6
32 XI7
31 $\overline{\text{XO7}}$
30 $\overline{\text{XO6}}$
29 $\overline{\text{XO5}}$
28 $\overline{\text{XO4}}$
27 $\overline{\text{XO3}}$
26 $\overline{\text{XO2}}$
25 $\overline{\text{XO1}}$
24 $\overline{\text{XO0}}$
23 INT
22 SENS
21 Φ2

PIN NAME	DESCRIPTION	TYPE
D0 - D7	Data Bus to CPU	Bidirectional
XI0 - XI7	Data Bus from external logic	Input
$\overline{\text{XO0}}$ - $\overline{\text{XO7}}$	Data Bus to external logic	Output
XMT	Transmit serial data line	Output
$\overline{\text{RCV}}$	Receive serial data line	Input
SENS	External interrupt sense	Input
INT	Interrupt request	Output
CE	Chip Select	Input
A0 - A3	Address Select	Input
SYNC	Synchronizing signal (SYNC) from 8080A	Input
Φ1, Φ2	Clock inputs, same as to 8080A	Input
V_{BB},V_{CC},V_{DD},V_{SS}	Power supply (-5V, +5V, +12V) and Ground	

Figure 4-60. TMS 5501 Multifunction Input/Output Controller Signals And Pin Assignments

TMS 5501 OUTPUT SIGNAL INVERSION

Do not miss the significance of $\overline{\text{XO}}$ negative logic; whatever you write to the TMS 5501 for parallel output will be complemented. $\overline{\text{XO}}$ signals are the inverse of the output buffer contents.

Serial I/O data uses the XMT and $\overline{\text{RCV}}$ pins. XMT is used to transmit serial data, whereas $\overline{\text{RCV}}$ is used to receive serial data. Note that $\overline{\text{RCV}}$ is a negative-true signal, whereas XMT is a positive-true signal.

External logic may request interrupt service either via the SENS input or via the XI7 input. A low-to-high transition on either signal constitutes an interrupt request. SENS is always part of external interrupt request logic; XI7 must be programmed for this purpose — in which case the eight XI pins cannot be used to input 8-bit parallel data.

Logic internal to the TMS 5501 may also generate interrupt requests. Whatever the source of the interrupt requests, it is passed on to the CPU via the INT interrupt request signal.

The TMS 5501 is accessed either as 16 I/O ports or 16 memory locations. Addressing logic consists of a chip select (CE) and four address select inputs (A0, A1, A2 and A3).

The TMS 5501 receives the SYNC timing pulse, and this requires special mention. While SYNC is high, the TMS 5501 decodes status off the Data Bus, therefore the 8228 System Controller is not needed.

Additional signals required by the TMS 5501 are the identical two 8080A clock signals Φ1 and Φ2. Slight clock signal variations will confuse serial I/O logic which computes baud rates internally.

A feature of the TMS 5501 which you must note carefully is that it cannot handle Wait states. Any T_W clock periods in a machine cycle will cause the TMS 5501 to malfunction.

TMS 5501 WAIT STATE

There is a further unlikely ramification of the TMS 5501 inability to handle Wait states. **If you are accessing the TMS 5501 as 16 memory locations, then you cannot have a Halt instruction's object code in the memory location immediately preceding the 16 TMS 5501 addresses.** If you do, the Halt instruction will execute, following which the Address Bus will contain the address of the next sequential memory location — which now is a TMS 5501 address. Thus, the TMS 5501 becomes selected. But the TMS 5501 logic cannot cope with a sequence of undefined clock periods, which is exactly what will happen following a Halt instruction's execution. The net effect is that following a Halt, the TMS 5501 receiver buffer loaded flag will be inadvertently cleared.

Always make sure that the memory address directly preceding the 16 addresses assigned to a TMS 5501 remains unused.

TMS 5501 DEVICE ACCESS

Some of the 16 I/O port or memory addresses via which the TMS 5501 device is accessed are equivalent to memory locations, but others are command identifiers. Table 4-16 defines the manner in which addresses are interpreted.

You will find the TMS 5501 far easier to use if you address it as 16 memory locations, because that will give you access to memory referencing instructions.

When creating TMS 5501 select logic, any of the select schemes described earlier in this chapter will do — with one addition. **Include READY as part of the select logic;** if READY is low, a Wait state will follow, and that will cause the TMS 5501 to malfunction. By making READY high a necessary component of device select logic, you can avoid this problem.

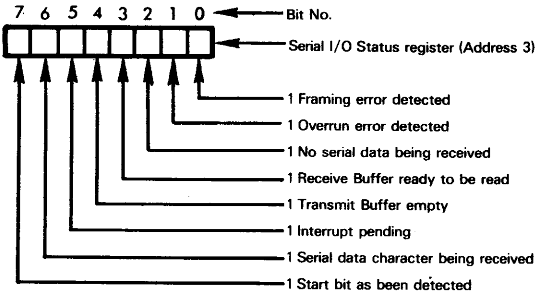
In the following discussion of individual TMS 5501 capabilities, we will use programming examples to show the effectiveness of including the TMS 5501 device within your memory rather than I/O space.

Table 4-16. TMS 5501 Address Interpretations

A3	A2	A1	A0	FUNCTION
0	0	0	0	Read assembled serial input data byte out of Receiver Buffer
0	0	0	1	Read parallel data input via XI0 - XI7
0	0	1	0	Read RST instruction code, as a data byte, when polling interrupt requests
0	0	1	1	Read Status register contents to the CPU
0	1	0	0	Write command code to the TMS 5501
0	1	0	1	Load serial I/O Control register, specifying baud rate and stop bits
0	1	1	0	Write data byte to serial transmit logic
0	1	1	1	Write data byte to parallel output port
1	0	0	0	Write out interrupt mask byte to selectively enable and disable interrupts
1	0	0	1	Write initial count to Interval Timer 1
1	0	1	0	Write initial count to Interval Timer 2
1	0	1	1	Write initial count to Interval Timer 3
1	1	0	0	Write initial count to Interval Timer 4
1	1	0	1	Write initial count to Interval Timer 5
1	1	1	0	No Operation
1	1	1	1	No Operation

TMS 5501 addressable locations 3, 4 and 5 are used for status and controls which generally apply to serial I/O and interrupt processing. We will define how these ports are used now, in advance of our discussion of TMS 5501 serial I/O and interrupt processing capabilities.

Locations 3 and 5 apply to serial I/O logic. Location 3 is a Status register whose bits are interpreted as follows:

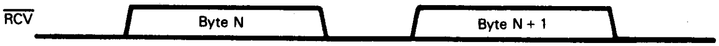

7 6 5 4 3 2 1 0
Bit No.
Serial I/O Status register (Address 3)
1 Framing error detected
1 Overrun error detected
1 No serial data being received
1 Receive Buffer ready to be read
1 Transmit Buffer empty
1 Interrupt pending
1 Serial data character being received
1 Start bit as been detected

Bits 0 and 1 are standard framing and overrun error indicators.

If a framing error is detected, Status register bit 0 will be set to 1 and will remain 1 until assembly of the next complete serial data character has been completed.

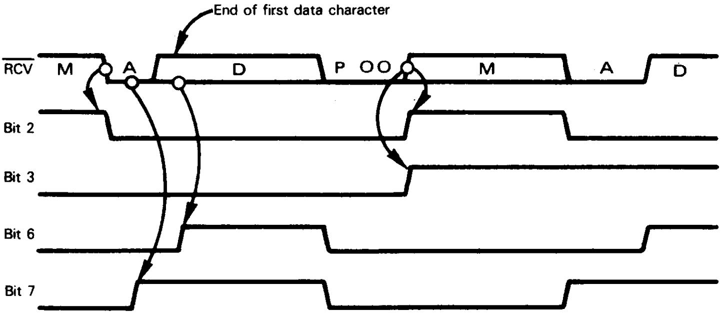
If Receiver Buffer contents are not read while the next serial character is being input and assembled, an overrun error will be reported in bit 1 of the Status register. This error indicator will be cleared as soon as the Status register contents are read, or when a reset command is output. Remember, you have the time it takes to receive and assemble one character in which to read the previous character out of the Receiver Buffer. This is because receive logic includes a double buffer. A character is assembled in a

Receiver register; when completely assembled, it is shifted to a Receiver Buffer and the next character is assembled in the Receiver register:



Receiver Register Contents	Byte N being assembled	Byte N + 1 being assembled
Receiver Buffer Contents	Assembled Byte N-1, waiting to be read	Assembled Byte N, waiting to be read

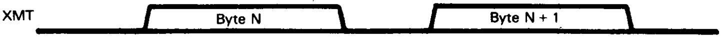
Status bits 2, 3, 6 and 7 monitor the condition of the serial data input signal. During a break, that is, when no valid serial data is being input, status bit 2 will be high. As soon as a start bit has been detected, status bit 2 will be reset low and status bit 7 will be set high. When the first valid data bit is detected, status bit 6 is also set high. When the received character has been assembled in the Receiver Buffer, and may be read by the CPU, status bits 7 and 6 are reset and status bit 3 is set. This may be illustrated as follows:



M Marking
A Start bit
D Data bits
P Parity bit
O Stop bits

Status bit 4 applies to serial transmit logic. As soon as the Transmit Buffer is ready to receive another byte of data, status bit 4 will be set high. It will remain high until new data has been loaded into the Transmit Buffer.

Transmit logic, like receive logic, is double-buffered. A byte of data is held in a Transmitter register while being output serially; meanwhile, the next data byte may be loaded into a Transmitter Buffer. Transmitter Buffer contents are automatically shifted to the Transmitter register when serial output of a data byte is complete. This may be illustrated as follows:

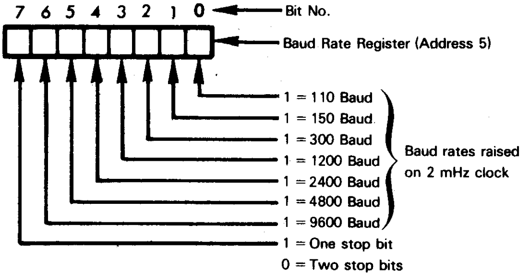


Transmitter Register Contents	Byte N being output serially	Byte N + 1 being output serially
Transmitter Buffer Contents	Write Byte N + 1 into Transmitter Buffer during this time	Write Byte N + 2 into Transmitter Buffer during this time

Status bit 4 is high from the instant Transmitter Buffer contents are shifted into the Transmitter register, until a new data byte is written into the Transmitter buffer.

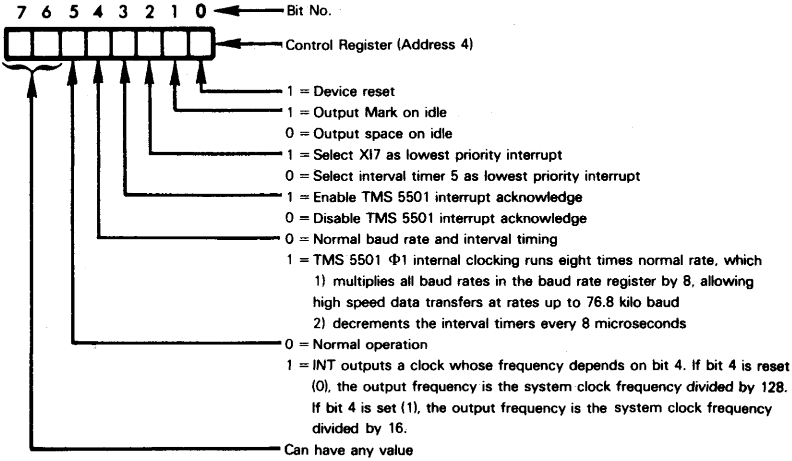
Status bit 5 is set whenever the TMS 5501 has an unacknowledged interrupt request. While this status bit is very important in serial I/O operations, it also may have application elsewhere; this bit therefore may be looked upon as an exception within the Status register, in that it is the only status flag that does not apply strictly to serial I/O operations.

TMS 5501 addressable location 5 is also dedicated to serial I/O. Into this location you must load a control byte which selects baud rate, and the number of stop bits. Register contents will be interpreted as follows:



If more than one of bits 0 through 6 are high, then the highest indicated baud rate will be selected. If no baud rate bit is high, then all serial transmit and receive logic will be inhibited.

TMS 5501 addressable location 4 is a general command register. Its contents will be interpreted as follows:


7 6 5 4 3 2 1 0
Bit No.
Control Register (Address 4)
1 = Device reset
1 = Output Mark on idle
0 = Output space on idle
1 = Select XI7 as lowest priority interrupt
0 = Select interval timer 5 as lowest priority interrupt
1 = Enable TMS 5501 interrupt acknowledge
0 = Disable TMS 5501 interrupt acknowledge
0 = Normal baud rate and interval timing
1 = TMS 5501 Φ1 internal clocking runs eight times normal rate, which 1) multiplies all baud rates in the baud rate register by 8, allowing high speed data transfers at rates up to 76.8 kilo baud 2) decrements the interval timers every 8 microseconds
0 = Normal operation
1 = INT outputs a clock whose frequency depends on bit 4. If bit 4 is reset (0), the output frequency is the system clock frequency divided by 128. If bit 4 is set (1), the output frequency is the system clock frequency divided by 16.
Can have any value

If your system does not require interrupts from the TMS 5501, you can set bit 5 high to derive a TTL compatible clock from the INT output.

TMS 5501 RESET

If the TMS 5501 device is reset by outputting 1 to bit 0, then the following events will occur:

1) Serial receive logic enters the Hunt mode. Status bits 2, 3, 6 and 7 are all reset; however, reset will not clear the Receive Buffer contents.
2) Serial transmit logic will output a high marking signal. Status bit 4 will be set high indicating that transmit logic is ready to receive another data byte.
3) The interrupt mask register is cleared with the exception of the Transmit Buffer interrupt, which is enabled. (Interrupt levels and interrupt masking are described shortly.)
4) All interval timers are halted.

The Reset has no effect on any of the following:

- Parallel input and output port contents
- Interrupt acknowledge enable
- Interrupt Mask register contents
- Baud rate register contents
- Serial Transmit or Receive Buffer contents

Control command bit 1 determines whether serial transmit logic will mark or space when not transmitting data. A 1 in bit 1 will cause serial transmit logic to mark (output high) while a 0 in bit 1 will cause transmit logic to space (output low).

If Reset conflicts with the break specification, then Reset will override and transmit logic will mark, irrespective of the break bit specification.

The TMS 5501 can receive an interrupt request from one of nine different sources. Using the eight Restart instructions, each interrupt request is assigned one of eight priorities. For this to be possible, two interrupt sources share the lowest priority interrupt level (RST 7); these two sources are an external request arriving via XI7 and the Interval Timer 5 time out interrupt request. **You use bit 2 of the control command to select which requesting source will be active at any time as the lowest priority interrupt.**

Bit 3 of the control command is a master enable/disable for TMS 5501 interrupt logic. If this bit is output as 0, then TMS 5501 interrupt acknowledge logic is disabled — and that effectively disables the entire interrupt processing system. Observe that with interrupt acknowledge logic disabled you can still use polling techniques in lieu of interrupt processing.

Table 4-17. TMS 5501 Interrupt Logic And Priorities

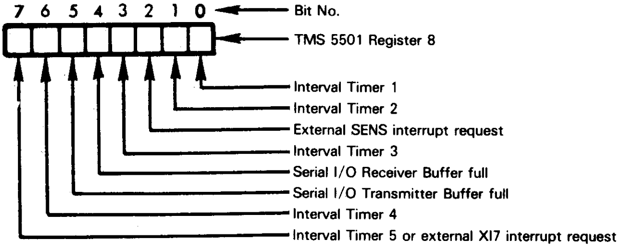
Interrupt and Mask Bit	Data Bus Status			RST Instruction	Interrupting Source
	D5	D4	D3		
0 (highest)	0	0	0	RST 0	Interval Timer 1
1	0	0	1	RST 1	Interval Timer 2
2	0	1	0	RST 2	External SENS interrupt request
3	0	1	1	RST 3	Interval Timer 3
4	1	0	0	RST 4	Serial I/O Receiver Buffer full
5	1	0	1	RST 5	Serial I/O Transmitter Buffer full
6	1	1	0	RST 6	Interval Timer 4
7 (lowest)	1	1	1	RST 7	Interval Timer 5, or external XI7 interrupt request, whichever has been selected by command code

TMS 5501 INTERRUPT HANDLING

The TMS 5501 responds to nine different interrupt requests, with priorities as defined in Table 4-17.

When an interrupt is acknowledged, INT is output high by the TMS 5501. If the TMS 5501 INT output is connected to the 8080A INT input, then the 8080A will acknowledge the interrupt by outputting D1 high at SYNC high. The TMS 5501 responds to this acknowledge by placing an RST instruction's object code on the Data Bus, as required by standard 8080A timing. **This is an utterly standard 8080A interrupt request/acknowledge sequence.**

Interrupts may be selectively disabled by writing a mask to TMS 5501 Register 8; see Table 4-16. A 0 bit will disable an interrupt; mask bits are related to priorities as follows:


7 6 5 4 3 2 1 0
Bit No.
TMS 5501 Register 8
Interval Timer 1
Interval Timer 2
External SENS interrupt request
Interval Timer 3
Serial I/O Receiver Buffer full
Serial I/O Transmitter Buffer full
Interval Timer 4
Interval Timer 5 or external XI7 interrupt request

Note that TMS 5501 interrupt priorities apply to the request/acknowledge sequence only—which is the standard passive interrupt priority arbitration sequence used in most microcomputer applications. Once an interrupt is acknowledged and is being serviced by an interrupt service routine, it is up to the programmer to disable all interrupts, or selected interrupts, if the interrupt service routine is not itself to get interrupted. If, for example, an interrupt were to be acknowledged at priority 3 (Interval Timer 3), in the normal course of events the 8080A CPU will disable all interrupts upon acknowledging any interrupt. Therefore the Interval Timer 3 interrupt service routine will deny any other interrupt request, whatever its priority, until the Interval Timer 3 service routine completes execution. If the Interval Timer 3 interrupt service routine were to immediately enable all interrupts, then any other interrupt request would be acknowledged, irrespective of priority.

If you want to ensure that only higher priority requests interrupt the Timer 3 service routine, then the Timer 3 service routine must begin by outputting a mask to disable all lower level interrupts at the TMS 5501; then it must enable all interrupts at the CPU. Here is the necessary instruction sequence:

```
        MVI     TMS8,07H        ;OUTPUT MASK TO REGISTER 8 OF TMS 5501
        EI                      ;ENABLE INTERRUPTS
```

The mask output in this case has the value 07, since mask bits 0, 1 and 2 only must be set to 1, enabling the highest three interrupt priority levels.

TMS 5501 NONSTANDARD FEATURES

Let us now look at the nonstandard features associated with TMS 5501 interrupt handling logic. First of all, so long as there is an unacknowledged interrupt request, Status register bit 5 is set to 1; next the RST instruction object code for the highest level interrupt request is stored in TMS 5501 Register 2. This allows you to bypass normal interrupt processing logic and poll the TMS 5501 instead.

In order to bypass interrupt logic, simply disconnect the TMS 5501 INT output from the 8080A INT input. You can still identify interrupt requests occurring within the TMS 5501 by reading the TMS 5501 Status register. If bit 5 of the Status register is 1, then one or more interrupt requests are active within the TMS 5501. In order to determine which is the highest level active interrupt request, read the contents of TMS 5501 memory location 2. The RST instruction object code corresponding to the highest priority interrupt request will have been assembled in this location. Bits 3, 4 and 5 of the RST instruction object code identify the priority level. Thus you can determine which of the eight priority levels was the highest active interrupt request. Here is a typical polling sequence:

```
;ASSUME THAT THE TMS 5501 ADDRESS SPACE CONSISTS OF 16 MEMORY
;LOCATIONS FROM 8000 THROUGH 800F. TMS5 IS THE SYMBOL ASSIGNED
;TO THE BASE ADDRESS
TMS5     EQU      8000H
         -
         -
         -
;TEST STATUS REGISTER FOR INTERRUPT PENDING
         LDA      TMS5+3     ;LOAD STATUS TO ACCUMULATOR
         ANI      20H        ;ISOLATE BIT 5
         JNZ      TMS5+2     ;IF NOT ZERO, AN INTERRUPT HAS BEEN
                             ;REQUESTED
         -
         -
         -
```

It is worth spending a minute looking at the three-instruction sequence illustrated above. The TMS 5501 Status register contents are loaded into the Accumulator by the LDA instruction. The next instruction isolates bit 5. If bit 5 is 1, then an interrupt has been requested, and the next instruction, a JNZ, branches program execution to a memory location within the TMS 5501 itself. Will that work? Indeed, it will. The label TMS5+2 addresses TMS 5501 Register 2, which contains an RST instruction's object code; this is the object code which would have been output in response to a normal interrupt acknowledge. What the JNZ instruction does is cause this RST instruction's object code to be executed next; and that is precisely the logic sequence which a normal interrupt response would have implemented.

Notice that the very simple method we have illustrated for polling on status only works if the TMS 5501 can be addressed as memory locations rather than I/O ports.

TMS 5501 PARALLEL I/O OPERATIONS

It is very easy to handle simple parallel I/O, without handshaking, using the TMS 5501. This is equivalent to 8255 Mode 0 operation. TMS 5501 address 1 accesses the parallel 8-bit input port, while address 7 accesses a parallel 8-bit output port (see Table 4-16). Assuming that the TMS 5501 is addressed as memory, input and output operations are handled using any memory reference instructions.

A very limited amount of parallel I/O handshaking is available. The SENS interrupt input signal can be used by external logic either to indicate that it has read output data, or to indicate that it has transmitted input data. However, the TMS 5501 device itself has no control signals which can be used to prompt external logic; that is to say, the TMS 5501 has no signal equivalent to the 8255 $\overline{\text{OBF}}$ control. When comparing the parallel I/O capabilities of the TMS 5501 with the 8255, therefore, we conclude that 8255 Mode 0 operations can be duplicated without problems, but neither Mode 1 nor Mode 2 parallel I/O operations with handshaking can be duplicated. Only a primitive level of parallel I/O with handshaking exists within the TMS 5501 and even this exists at the expense of external interrupt logic.

TMS 5501 SERIAL I/O OPERATION

A significant asynchronous, serial I/O capability is provided by the TMS 5501. Synchronous serial I/O is not supported.

There are very significant differences between the implementation of asynchronous serial I/O by the TMS 5501, as compared to the 8251 USART.

The TMS 5501 has separate serial transmit and receive pins (XMT and $\overline{\text{RCV}}$), but it has no accompanying handshaking control signals; instead 5th and 6th priority interrupts identify Receiver Buffer full and Transmit Buffer full, respectively. Bits 2, 3, 6 and 7 of the Status register (addressable location 3) identify the condition of a serial receive data stream.

When using the TMS 5501, you have to continuously read in the contents of the Status register and test the condition of appropriate status bits in order to implement standard serial receive logic; however, in the end you can implement the same serial receive logic as is provided automatically by the 8251 USART. Here is the relationship between the TMS 5501 and the 8251 USART controls:

8251 USART	TMS 5501 EQUIVALENT
TxRDY	Status register bit 4
TxE	None
TxC	Baud Rate register
RxRDY	Status register bit 3
RxC	Baud Rate register
SYNDET	None

Probably the most significant difference between TMS 5501 and 8251 USART control is the fact that TMS 5501 baud rate is programmed by outputting an appropriate Control code, while it is clocked by rate signals input to the 8251 USART. The TMS 5501 advantage is that the TMS 5501 does not need external baud rate clock generation logic; however there must be a very precise synchronization between the TMS 5501 and whatever external logic it is communicating with. Minor timing differences are no problem when using an 8251 USART since a clock signal can accompany the serial data stream. Minor timing differences can be intolerable when using the TMS 5501; a small difference between TMS 5501 baud rate and external clock signals can generate very significant errors.

TMS 5501 INTERVAL TIMERS

The TMS 5501 has five programmable Interval Timers. Each timer can be loaded with an initial count ranging from 01 (lowest) through FF_{16} (highest). Each Timer will decrement one count every 64 microseconds. As soon as a programmable timer counts out to zero, it requests an interrupt. In our discussion of TMS 5501 interrupt logic, we have defined the priority levels assigned to the various Interval Timers. Notice that Interval Timer priorities have been spread across the range of priority levels. By using Interval Timer 1 or 2, you can be sure of precise time intervals, since an interrupt request will be acknowledged with little or no delay. Timers 4 and 5, being the lowest priority, can be used to generate less precise time intervals. It is conceivable that interrupt requests originating at these two timers might have to wait a significant amount of time before being serviced — if there is any degree of interrupt traffic within the microcomputer system.

Loading a 0 value into an Interval Timer causes an immediate interrupt request.

When a nonzero value is loaded into an Interval Timer, it starts to count down immediately. If a new value is loaded into an Interval Timer while it is halfway through counting out, then the new value will be accepted; it will override the previous value and subsequently will be decremented. Therefore the Interval Timers are retriggerable.

Once an Interval Timer counts out, it halts.

MISCELLANEOUS 8080A SUPPORT DEVICES

A few devices need to be identified as members of the 8080A microcomputer family, not because these devices are in any way specific to microcomputer system logic, but rather because they are frequently used within microcomputer configurations. These devices include the 8205, which is a 1-of-8 decoder, and various bidirectional bus drivers.

THE 8205 1-OF-8 DECODER

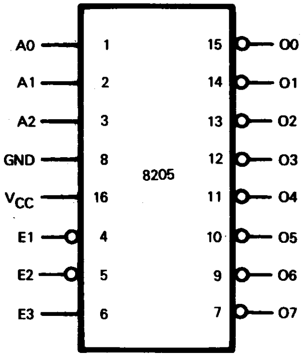
This device, along with its truth table, is illustrated in Figure 4-61. A three binary digit octal value, arriving at inputs A0, A1 and A2, selects one of the eight outputs, O0 - O7. The selected output is low, while all other outputs are high. There are three device enable signals, $\overline{\text{E1}}$, $\overline{\text{E2}}$ and E3. The device is enabled by low inputs at $\overline{\text{E1}}$ and $\overline{\text{E2}}$, with a high input at E3. Any type of synchronizing logic can be used as part of the enable logic in order to synchronize outputs with the microcomputer system.

BIDIRECTIONAL BUS DRIVERS

Bidirectional bus drivers are very important in microcomputer systems, because in most cases devices can handle very limited loads. The purpose of the bidirectional bus driver is to guarantee signal integrity during normal operating loads. **Figure 4-62 illustrates typical bidirectional bus driver logic.** Each buffered line consists of two separate buffers. On one side of the driver the output of one buffer and the input of another are tied together to provide the System Bus interface (DB). On the other side of the driver, inputs and outputs are separated. Two inputs control data flow within the device. $\overline{\text{DIEN}}$ determines the direction of data flow; when low, data flows from DI0 - DI3 to DB0 - DB3; when high, data flows from DB0 - DB3 to DO0 - DO3.

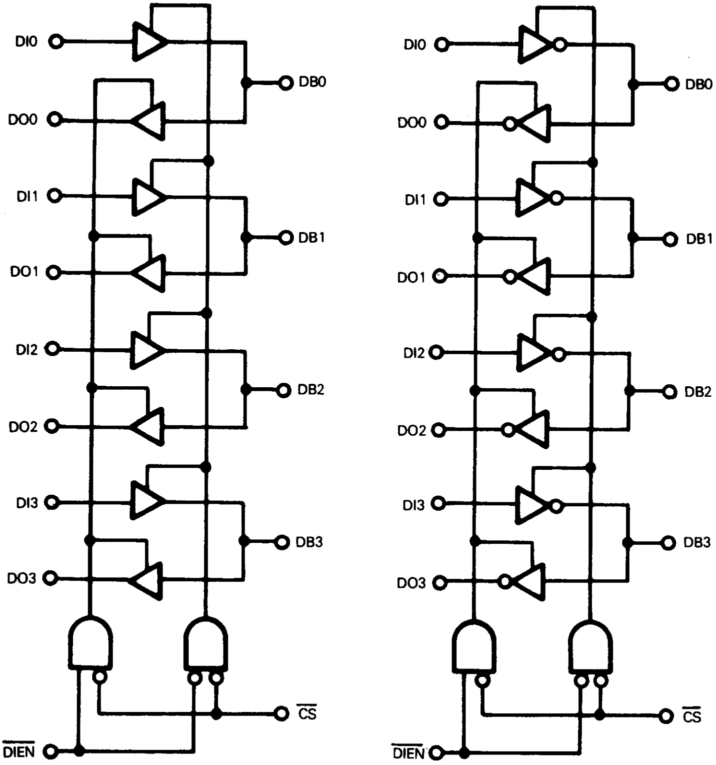
$\overline{\text{CS}}$ is a standard device select which must be enabled in the usual way.

In addition to the 8216 and 8226 bidirectional bus drivers, a number of products are available from Texas Instruments; device numbers include the following: SN543240, SN54S241, SN74S240, SN74S241, SN54LS240, SN54LS241, SN54LS242, SN54LS243, SN74LS240, SN74LS241, SN74LS242, SN74LS243.



ADDRESS			ENABLE			OUTPUTS							
A_0	A_1	A_2	E_1	E_2	E_3	0	1	2	3	4	5	6	7
L	L	L	L	L	H	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
L	H	L	L	L	H	H	H	L	H	H	H	H	H
H	H	L	L	L	H	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
H	L	H	L	L	H	H	H	H	H	H	L	H	H
L	H	H	L	L	H	H	H	H	H	H	H	L	H
H	H	H	L	L	H	H	H	H	H	H	H	H	L
X	X	X	L	L	L	H	H	H	H	H	H	H	H
X	X	X	H	L	L	H	H	H	H	H	H	H	H
X	X	X	L	H	L	H	H	H	H	H	H	H	H
X	X	X	H	H	L	H	H	H	H	H	H	H	H
X	X	X	H	L	H	H	H	H	H	H	H	H	H
X	X	X	L	H	H	H	H	H	H	H	H	H	H
X	X	X	H	H	H	H	H	H	H	H	H	H	H

Figure 4-61. The 8205 1-Of-8 Decoder



The 8216 Device

The 8226 Device

Figure 4-62. Bidirectional Bus Drivers

DATA SHEETS

In this section you will find the electrical characteristics and specific delay times for the 8080A and all related devices discussed in this chapter.

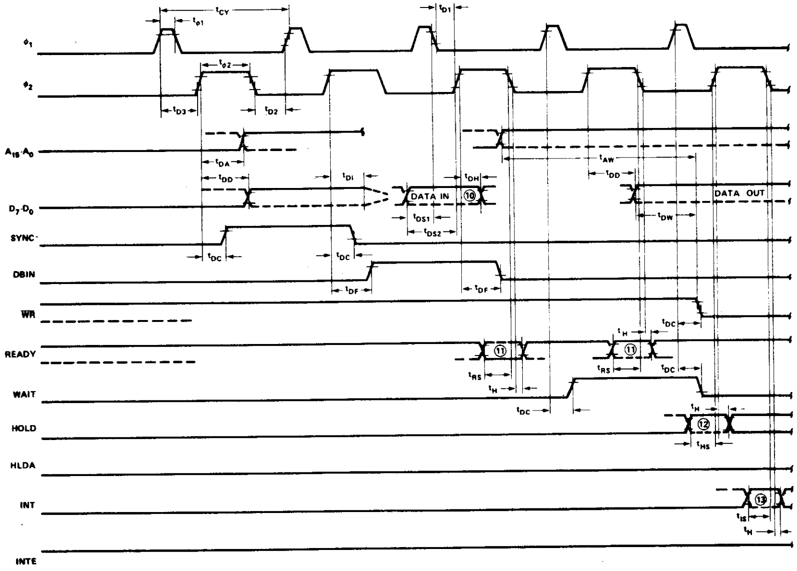
A.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{DD} = +12V ± 5%, V_{CC} = +5V ± 5%, V_{BB} = -5V ± 5%, V_{SS} = 0V, Unless Otherwise Noted

Symbol	Parameter	Min.	Max.	Unit	Test Condition
t_{CY}[3]	Clock Period	0.48	2.0	μsec	
t_r, t_f	Clock Rise and Fall Time	0	50	nsec	
$t_{\phi 1}$	ϕ_1 Pulse Width	60		nsec	
$t_{\phi 2}$	ϕ_2 Pulse Width	220		nsec	
t_{D1}	Delay ϕ_1 to ϕ_2	0		nsec	
t_{D2}	Delay ϕ_2 to ϕ_1	70		nsec	
t_{D3}	Delay ϕ_1 to ϕ_2 Leading Edges	80		nsec	
t_{DA} [2]	Address Output Delay From ϕ_2		200	nsec	C_L = 100pf
t_{DD} [2]	Data Output Delay From ϕ_2		220	nsec	
t_{DC} [2]	Signal Output Delay From ϕ_1, or ϕ_2 (SYNC, $\overline{WR}$,WAIT, HLDA)		120	nsec	C_L = 50pf
t_{DF} [2]	DBIN Delay From ϕ_2	25	140	nsec	
t_{DI}[1]	Delay for Input Bus to Enter Input Mode		t_{DF}	nsec	
t_{DS1}	Data Setup Time During ϕ_1 and DBIN	30		nsec	

TIMING WAVEFORMS [14]

(Note: Timing measurements are made at the following reference voltages: CLOCK "1" = 8.0V "0" = 1.0V; INPUTS "1" = 3.3V, "0" = 0.8V; OUTPUTS "1" = 2.0V, "0" = 0.8V.)



ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias	0°C to +70° C
Storage Temperature	-65°C to +150°C
All Input or Output Voltages With Respect to V_{BB}	-0.3V to +20V
V_{CC}, V_{DD} and V_{SS} With Respect to V_{BB}	-0.3V to +20V
Power Dissipation	1.5W

**COMMENT*: *Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{DD} = +12V ± 5%, V_{CC} = +5V ± 5%, V_{BB} = -5V ± 5%, V_{SS} = 0V, Unless Otherwise Noted.

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	$V_{SS}-1$		$V_{SS}+0.8$	V	
V_{IHC}	Clock Input High Voltage	9.0		$V_{DD}+1$	V	
V_{IL}	Input Low Voltage	$V_{SS}-1$		$V_{SS}+0.8$	V	
V_{IH}	Input High Voltage	3.3		$V_{CC}+1$	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 1.9mA on all outputs, I_{OH} = -150µA.
V_{OH}	Output High Voltage	3.7			V	
$I_{DD\,(AV)}$	Avg. Power Supply Current (V_{DD})		40	70	mA	Operation T_{CY} = .48 µsec
$I_{CC\,(AV)}$	Avg. Power Supply Current (V_{CC})		60	80	mA	
$I_{BB\,(AV)}$	Avg. Power Supply Current (V_{BB})		.01	1	mA	
I_{IL}	Input Leakage			±10	µA	$V_{SS} \leqslant V_{IN} \leqslant V_{CC}$
I_{CL}	Clock Leakage			±10	µA	$V_{SS} \leqslant V_{CLOCK} \leqslant V_{DD}$
I_{DL} [2]	Data Bus Leakage in Input Mode			-100 -2.0	µA mA	$V_{SS} \leqslant V_{IN} \leqslant V_{SS}+0.8V$ $V_{SS}+0.8V \leqslant V_{IN} \leqslant V_{CC}$
I_{FL}	Address and Data Bus Leakage During HOLD			+10 -100	µA	$V_{ADDR/DATA} = V_{CC}$ $V_{ADDR/DATA} = V_{SS} + 0.45V$

CAPACITANCE

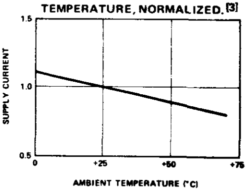
T_A = 25°C $V_{CC} = V_{DD} = V_{SS}$ = 0V, V_{BB} = -5V

Symbol	Parameter	Typ.	Max.	Unit	Test Condition
C_ϕ	Clock Capacitance	17	25	pf	f_c = 1 MHz
C_{IN}	Input Capacitance	6	10	pf	Unmeasured Pins
C_{OUT}	Output Capacitance	10	20	pf	Returned to V_{SS}

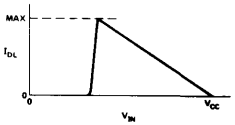
NOTES:

1. The RESET signal must be active for a minimum of 3 clock cycles..
2. When DBIN is high and $V_{IN} > V_{IH}$ an internal active pull up will be switched onto the Data Bus.
3. ΔI supply / ΔT_A = -0.45%/°C.

TYPICAL SUPPLY CURRENT VS. TEMPERATURE, NORMALIZED.[3]



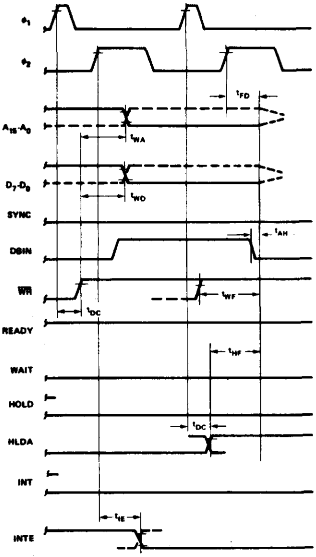
DATA BUS CHARACTERISTIC DURING DBIN



A.C. CHARACTERISTICS (Continued)

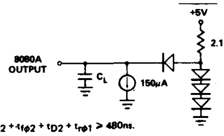
T_A = 0°C to 70°C, V_{DD} = +12V ± 5%, V_{CC} = +5V ± 5%, V_{BB} = -5V ± 5%, V_{SS} = 0V, Unless Otherwise Noted

Symbol	Parameter	Min.	Max.	Unit	Test Condition
t_{DS2}	Data Setup Time to ϕ_2 During DBIN	150		nsec	C_L = 50pf
t_{DH} [1]	Data Hold Time From ϕ_2 During DBIN	[1]		nsec	
t_{IE} [2]	INTE Output Delay From ϕ_2		200	nsec	
t_{RS}	READY Setup Time During ϕ_2	120		nsec	
t_{HS}	HOLD Setup Time to ϕ_2	140		nsec	
t_{IS}	INT Setup Time During ϕ_2 (During ϕ_1 in Halt Mode)	120		nsec	
t_H	Hold Time From ϕ_2 (READY, INT, HOLD)	0		nsec	
t_{FD}	Delay to Float During Hold (Address and Data Bus)		120	nsec	
t_{AW} [2]	Address Stable Prior to $\overline{WR}$	[5]		nsec	C_L=100pf: Address, Data C_L=50pf: $\overline{WR}$, HLDA, DBIN
t_{DW} [2]	Output Data Stable Prior to $\overline{WR}$	[6]		nsec	
t_{WD} [2]	Output Data Stable From $\overline{WR}$	[7]		nsec	
t_{WA} [2]	Address Stable From $\overline{WR}$	[7]		nsec	
t_{HF} [2]	HLDA to Float Delay	[8]		nsec	
t_{WF} [2]	$\overline{WR}$ to Float Delay	[9]		nsec	
t_{AH} [2]	Address Hold Time After DBIN During HLDA	-20		nsec	



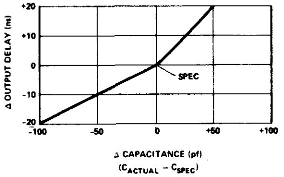
NOTES:

1. Data input should be enabled with DBIN status. No bus conflict can then occur and data hold time is assured. t_{DH} = 50 ns or t_{DF}, whichever is less.
2. Load Circuit.



3. $t_{CY} = t_{D3} + t_{r\phi2} + t_{\phi2} + t_{f\phi2} + t_{D2} + t_{r\phi1} \geq 480ns.$

TYPICAL Δ OUTPUT DELAY VS. Δ CAPACITANCE



4. The following are relevant when interfacing the 8080A to devices having V_{IH} = 3.3V:
   a) Maximum output rise time from .8V to 3.3V = 100ns @ C_L = SPEC.
   b) Output delay when measured to 3.0V = SPEC +60ns @ C_L = SPEC.
   c) If $C_L \neq$ SPEC, add .6ns/pF if $C_L > C_{SPEC}$, subtract .3ns/pF (from modified delay) if $C_L < C_{SPEC}$.
5. $t_{AW} = 2\,t_{CY} - t_{D3} - t_{r\phi2}$ -140nsec.
6. $t_{DW} = t_{CY} - t_{D3} - t_{r\phi2}$ -170nsec.
7. If not HLDA, $t_{WD} = t_{WA} = t_{D3} + t_{r\phi2}$ +10ns. If HLDA, $t_{WD} = t_{WA} = t_{WF}$.
8. $t_{HF} = t_{D3} + t_{r\phi2}$ -50ns.
9. $t_{WF} = t_{D3} + t_{r\phi2}$ -10ns
10. Data in must be stable for this period during DBIN $\cdot T_3$. Both t_{DS1} and t_{DS2} must be satisfied.
11. Ready signal must be stable for this period during T_2 or T_W. (Must be externally synchronized.)
12. Hold signal must be stable for this period during T_2 or T_W when entering hold mode, and during T_3, T_4, T_5 and T_{WH} when in hold mode. (External synchronization is not required.)
13. Interrupt signal must be stable during this period of the last clock cycle of any instruction in order to be recognized on the following instruction. (External synchronization is not required.)
14. This timing diagram shows timing relationships only; it does not represent any specific machine cycle.

D.C. Characteristics

T_A = 0°C to 70°C; V_{CC} = +5.0V ±5%; V_{DD} = +12V ±5%.

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
I_F	Input Current Loading			-.25	mA	V_F = .45V
I_R	Input Leakage Current			10	μA	V_R = 5.25V
V_C	Input Forward Clamp Voltage			1.0	V	I_C = -5mA
V_{IL}	Input "Low" Voltage			.8	V	V_{CC} = 5.0V
V_{IH}	Input "High" Voltage	2.6 2.0			V	Reset Input All Other Inputs
V_{IH}-V_{IL}	REDIN Input Hysteresis	.25			mV	V_{CC} = 5.0V
V_{OL}	Output "Low" Voltage			.45 .45	V V	(ϕ_1,ϕ_2), Ready, Reset, $\overline{\text{STSTB}}$ I_{OL} =2.5mA All Other Outputs I_{OL} = 15mA
V_{OH}	Output "High" Voltage ϕ_1 , ϕ_2 READY, RESET All Other Outputs	9.4 3.6 2.4			V V V	I_{OH} = -100μA I_{OH} = -100μA I_{OH} = -1mA
I_{SC}[1]	Output Short Circuit Current (All Low Voltage Outputs Only)	-10		-60	mA	V_O = 0V V_{CC} = 5.0V
I_{CC}	Power Supply Current			115	mA	
I_{DD}	Power Supply Current			12	mA	

Note: 1. Caution, ϕ_1 and ϕ_2 output drivers do not have short circuit protection

CRYSTAL REQUIREMENTS

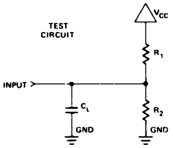
Tolerance: .005% at 0°C -70°C
Resonance: Series (Fundamental)*
Load Capacitance: 20-35pF
Equivalent Resistance: 75-20 ohms
Power Dissipation (Min): 4mW

*With tank circuit use 3rd overtone mode.

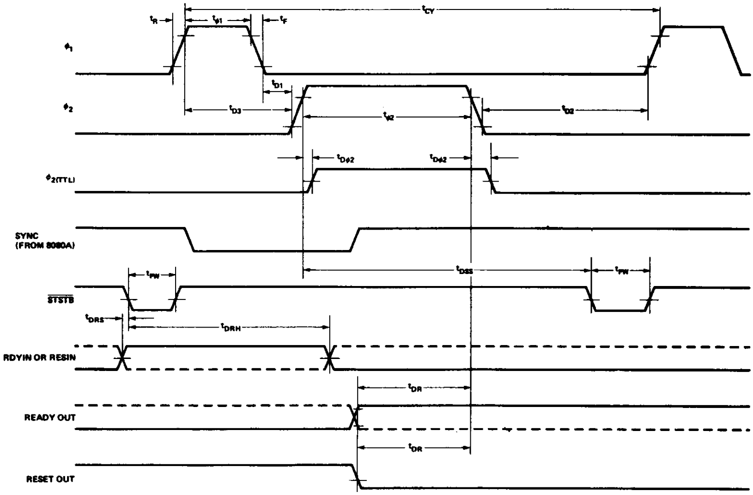
A.C. Characteristics

V_{CC} = +5.0V ± 5%; V_{DD} = +12.0V ± 5%; T_A = 0°C to 70°C

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
$t_{\phi 1}$	ϕ_1 Pulse Width	$\frac{2tcy}{9}$ - 20ns			ns	C_L = 20pF to 50pF
$t_{\phi 2}$	ϕ_2 Pulse Width	$\frac{5tcy}{9}$ - 35ns				
t_{D1}	ϕ_1 to ϕ_2 Delay	0				
t_{D2}	ϕ_2 to ϕ_1 Delay	$\frac{2tcy}{9}$ - 14ns				
t_{D3}	ϕ_1 to ϕ_2 Delay	$\frac{2tcy}{9}$		$\frac{2tcy}{9}$ + 20ns		
t_R	ϕ_1 and ϕ_2 Rise Time			20		
t_F	ϕ_1 and ϕ_2 Fall Time			20		
$t_{D\phi 2}$	ϕ_2 to ϕ_2 (TTL) Delay	-5		+15	ns	ϕ_2TTL,CL=30 R_1=300Ω R_2=600Ω
t_{DSS}	ϕ_2 to $\overline{STSTB}$ Delay	$\frac{6tcy}{9}$ - 30ns		$\frac{6tcy}{9}$		$\overline{STSTB}$, CL=15pF R_1 = 2K R_2 = 4K
t_{PW}	$\overline{STSTB}$ Pulse Width	$\frac{tcy}{9}$ - 15ns				
t_{DRS}	RDYIN Setup Time to Status Strobe	50ns - $\frac{4tcy}{9}$				
t_{DRH}	RDYIN Hold Time After $\overline{STSTB}$	$\frac{4tcy}{9}$				
t_{DR}	RDYIN or RESiN to ϕ_2 Delay	$\frac{4tcy}{9}$ - 25ns				Ready & Reset CL=10pF R_1=2K R_2=4K
t_{CLK}	CLK Period		$\frac{tcy}{9}$			
f_{max}	Maximum Oscillating Frequency	27			MHz	
C_{in}	Input Capacitance			8	pF	V_{CC}=+5.0V V_{DD}=+12V V_{BIAS}=2.5V f=1MHz



WAVEFORMS



VOLTAGE MEASUREMENT POINTS: ϕ_1, ϕ_2 Logic "0" = 1.0V, Logic "1" = 8.0V. All other signals measured at 1.5V.

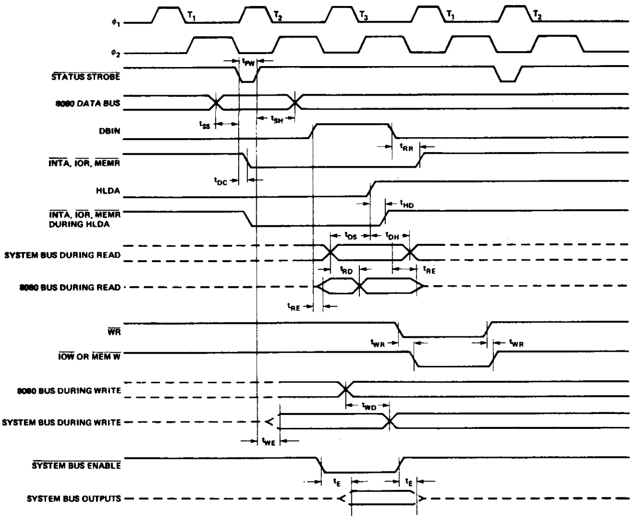
EXAMPLE:

A.C. Characteristics (For t_{CY} = 488.28 ns)

T_A = 0°C to 70°C; V_{DD} = +5V ±5%; V_{DD} = +12V ±5%.

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
$t_{\phi 1}$	ϕ_1 Pulse Width	89			ns	t_{CY}=488.28ns
$t_{\phi 2}$	ϕ_2 Pulse Width	236			ns	
t_{D1}	Delay ϕ_1 to ϕ_2	0			ns	
t_{D2}	Delay ϕ_2 to ϕ_1	95			ns	ϕ_1 & ϕ_2 Loaded to C_L = 20 to 50pF
t_{D3}	Delay ϕ_1 to ϕ_2 Leading Edges	109		129	ns	
t_r	Output Rise Time			20	ns	
t_f	Output Fall Time			20	ns	
t_{DSS}	ϕ_2 to $\overline{STSTB}$ Delay	296		326	ns	
$t_{D\phi 2}$	ϕ_2 to ϕ_2 (TTL) Delay	-5		+15	ns	
t_{PW}	Status Strobe Pulse Width	40			ns	Ready & Reset Loaded to 2mA/10pF All measurements referenced to 1.5V unless specified otherwise.
t_{DRS}	RDYIN Setup Time to $\overline{STSTB}$	-167			ns	
t_{DRH}	RDYIN Hold Time after $\overline{STSTB}$	217			ns	
t_{DR}	READY or RESET to ϕ_2 Delay	192			ns	
f_{MAX}	Oscillator Frequency			18.432	MHz	

WAVEFORMS



VOLTAGE MEASUREMENT POINTS: D_0-D_7 (when outputs) Logic "0" = 0.8V, Logic "1" = 3.0V. All other signals measured at 1.5V.

A.C. Characteristics T_A = 0°C to 70°C; V_{CC} = 5V ±5%.

Symbol	Parameter	Limits		Units	Condition
		Min.	Max.		
t_{PW}	Width of Status Strobe	22		ns	
t_{SS}	Setup Time, Status Inputs D_0-D_7	8		ns	
t_{SH}	Hold Time, Status Inputs D_0-D_7	5		ns	
t_{DC}	Delay from $\overline{\text{STSTB}}$ to any Control Signal	20	60	ns	C_L = 100pF
t_{RR}	Delay from DBIN to Control Outputs		30	ns	C_L = 100pF
t_{RE}	Delay from DBIN to Enable/Disable 8080 Bus		45	ns	C_L = 25pF
t_{RD}	Delay from System Bus to 8080 Bus during Read		30	ns	C_L = 25pF
t_{WR}	Delay from $\overline{\text{WR}}$ to Control Outputs	5	45	ns	C_L = 100pF
t_{WE}	Delay to Enable System Bus DB_0-DB_7 after $\overline{\text{STSTB}}$		30	ns	C_L = 100pF
t_{WD}	Delay from 8080 Bus D_0-D_7 to System Bus DB_0-DB_7 during Write	5	40	ns	C_L = 100pF
t_E	Delay from $\overline{\text{System Bus Enable}}$ to System Bus DB_0-DB_7		30	ns	C_L = 100pF
t_{HD}	HLDA to Read Status Outputs		25	ns	
t_{DS}	Setup Time, System Bus Inputs to HLDA	10		ns	
t_{DH}	Hold Time, System Bus Inputs to HLDA	20		ns	C_L = 100pF

D.C. Characteristics T_A = 0°C to 70°C; V_{CC} = 5V ±5%.

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.[1]	Max.		
V_C	Input Clamp Voltage, All Inputs		.75	-1.0	V	V_{CC}=4.75V; I_C=-5mA
I_F	Input Load Current, $\overline{STSTB}$			500	µA	V_{CC} = 5.25V
	D_2 & D_6			750	µA	V_F = 0.45V
	D_0, D_1, D_4, D_5, & D_7			250	µA	
	All Other Inputs			250	µA	
I_R	Input Leakage Current $\overline{STSTB}$			100	µA	V_{CC} = 5.25V
	DB_0-DB_7			20	µA	V_R = 5.25V
	All Other Inputs			100	µA	
V_{TH}	Input Threshold Voltage, All Inputs	0.8		2.0	V	V_{CC} = 5V
I_{CC}	Power Supply Current		140	190	mA	V_{CC}=5.25V
V_{OL}	Output Low Voltage, D_0-D_7			.45	V	V_{CC}=4.75V; I_{OL}=2mA
	All Other Outputs			.45	V	I_{OL} = 10mA
V_{OH}	Output High Voltage, D_0-D_7	3.6	3.8		V	V_{CC}=4.75V; I_{OH}=-10µA
	All Other Outputs	2.4			V	I_{OH} = -1mA
I_{OS}	Short Circuit Current, All Outputs	15		90	mA	V_{CC}=5V
$I_{O(off)}$	Off State Output Current, All Control Outputs			100	µA	V_{CC}=5.25V; V_O=5.25
				-100	µA	V_O=.45V
I_{INT}	INTA Current			5	mA	(See Figure below)

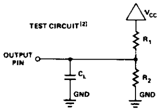
Note 1: Typical values are for T_A = 25°C and nominal supply voltages.

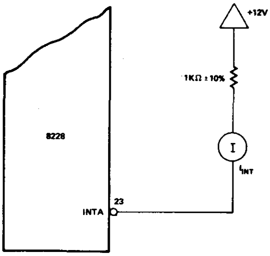
Capacitance This parameter is periodically sampled and not 100% tested.

Symbol	Parameter	Limits			Unit
		Min.	Typ.[1]	Max.	
C_{IN}	Input Capacitance		8	12	pF
C_{OUT}	Output Capacitance Control Signals		7	15	pF
I/O	I/O Capacitance (D or DB)		8	15	pF

TEST CONDITIONS: V_{BIAS} = 2.5V, V_{CC}=5.0V, T_A = 25°C, f = 1MHz.

Note 2: For D_0-D_7: R_1 = 4KΩ, R_2 = ∞Ω, C_L = 25pF. For all other outputs: R_1 = 500Ω, R_2 = 1KΩ, C_L = 100pF.





INTA Test Circuit (for RST 7)

Absolute Maximum Ratings*

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to +150°C
Voltage On Any Pin With Respect to Ground	−0.5V to +7V
Power Dissipation	1 Watt

**COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. Characteristics:

T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; GND = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 1.6mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -100μA
I_{DL}	Data Bus Leakage			−50 10	μA μA	V_{OUT} = .45V V_{OUT} = V_{CC}
I_{IL}	Input Leakage			10	μA	V_{IN} = V_{CC}
I_{CC}	Power Supply Current		45	80	mA	

Capacitance:

T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND.

TEST LOAD CIRCUIT:

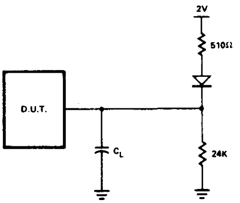
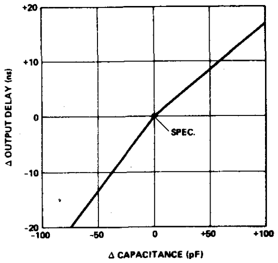


Figure 1.

TYPICAL Δ OUTPUT DELAY VS. Δ CAPACITANCE (dB)



A.C. Characteristics:

T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; GND = 0V

BUS PARAMETERS: (Note 1)

READ CYCLE

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AR}	Address Stable Before $\overline{\text{READ}}$ ($\overline{\text{CS}}$, C/D)	50		ns	
t_{RA}	Address Hold Time for $\overline{\text{READ}}$ ($\overline{\text{CS}}$, C/D)	5		ns	
t_{RR}	$\overline{\text{READ}}$ Pulse Width	430		ns	
t_{RD}	Data Delay from $\overline{\text{READ}}$		350	ns	C_L = 100 pF
t_{DF}	$\overline{\text{READ}}$ to Data Floating		200	ns	C_L = 100 pF
		25		ns	C_L = 15 pF
t_{RV}	Recovery Time Between WRITES (Note 2)	6		t_{CY}	

WRITE CYCLE

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AW}	Address Stable Before $\overline{\text{WRITE}}$	20		ns	
t_{WA}	Address Hold Time for $\overline{\text{WRITE}}$	20		ns	
t_{WW}	$\overline{\text{WRITE}}$ Pulse Width	400		ns	
t_{DW}	Data Set Up Time for $\overline{\text{WRITE}}$	200		ns	
t_{WD}	Data Hold Time for $\overline{\text{WRITE}}$	40		ns	

NOTES: 1. AC timings measured at V_{OH} = 2.0, V_{OL} = .8, and with load circuit of Figure 1.

2. This recovery time is for initialization only, when MODE, SYNC1, SYNC2, COMMAND and first DATA BYTES are written into the USART. Subsequent writing of both COMMAND and DATA are only allowed when TxRDY = 1.

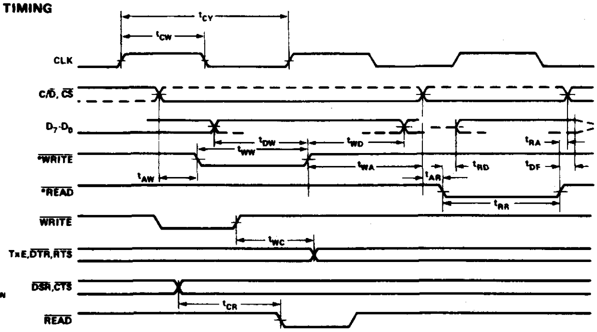
OTHER TIMINGS:

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{CY}	Clock Period (Note 3)	.420	1.35	µs	
$t_{\phi W}$	Clock Pulse Width	220	.7 t_{CY}	ns	
t_R, t_F	Clock Rise and Fall Time	0	50	ns	
t_{DTx}	TxD Delay from Falling Edge of TxC		1	µs	C_L = 100 pF
t_{SRx}	Rx Data Set-Up Time to Sampling Pulse	2		µs	C_L = 100 pF
t_{HRx}	Rx Data Hold Time to Sampling Pulse	2		µs	C_L = 100 pF
f_{Tx}	Transmitter Input Clock Frequency				
	1x Baud Rate	DC	56	KHz	
	16x and 64x Baud Rate	DC	520	KHz	
t_{TPW}	Transmitter Input Clock Pulse Width				
	1x Baud Rate	12		t_{CY}	
	16x and 64x Baud Rate	1		t_{CY}	
t_{TPD}	Transmitter Input Clock Pulse Delay				
	1x Baud Rate	15		t_{CY}	
	16x and 64x Baud Rate	3		t_{CY}	
f_{Rx}	Receiver Input Clock Frequency				
	1x Baud Rate	DC	56	KHz	
	16x and 64x Baud Rate	DC	520	KHz	
t_{RPW}	Receiver Input Clock Pulse Width				
	1x Baud Rate	12		t_{CY}	
	16x and 64x Baud Rate	1		t_{CY}	
t_{RPD}	Receiver Input Clock Pulse Delay				
	1x Baud Rate	15		t_{CY}	
	16x and 64x Baud Rate	3		t_{CY}	
t_{Tx}	TxRDY Delay from Center of Data Bit		16	t_{CY}	C_L = 50 pF
t_{Rx}	RxRDY Delay from Center of Data Bit		20	t_{CY}	
t_{IS}	Internal SYNDET Delay from Center of Data Bit		25	t_{CY}	
t_{ES}	Internal SYNDET Set-Up Time Before Falling Edge of RxC		16	t_{CY}	
t_{TxE}	TxEMPTY Delay from Center of Data Bit		16	t_{CY}	C_L = 50 pF
t_{WC}	Control Delay from Rising Edge of WRITE (TxE,$\overline{DTR}$,$\overline{RTS}$)		16	t_{CY}	
t_{CR}	Control to READ Set-Up Time ($\overline{DSR}$,$\overline{CTS}$)		16	t_{CY}	

3. The TxC and RxC frequencies have the following limitations with respect to CLK.
   For 1x Baud Rate , f_{Tx} or $f_{Rx} < 1/(30\ t_{CY})$
   For 16x and 64x Baud Rate, f_{Tx} or $f_{Rx} < 1/(4.5\ t_{CY})$

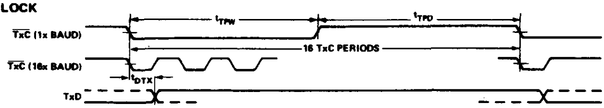
4. Reset Pulse Width = 6 t_{CY} minimum.

READ AND WRITE TIMING

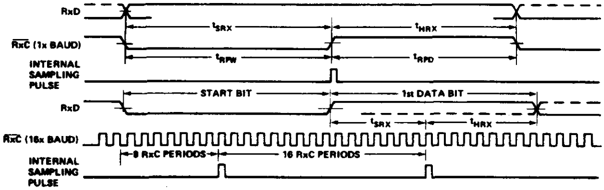


*WRITE AND READ PULSES HAVE NO TIMING LIMITATION WITH RESPECT TO CLK.

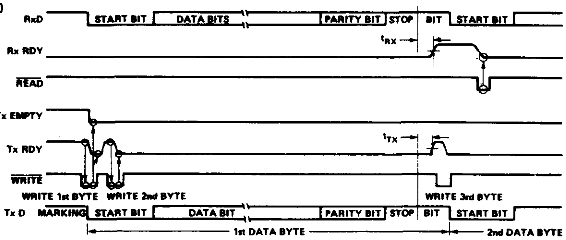
TRANSMITTER CLOCK AND DATA



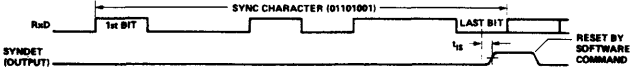
RECEIVER CLOCK AND DATA



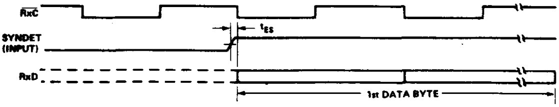
Tx RDY AND Rx RDY TIMING (ASYNC MODE)



INTERNAL SYNC DETECT



EXTERNAL SYNC DETECT



ABSOLUTE MAXIMUM RATINGS

PARAMETER	SYMBOL	MIN.	MAX.	UNIT	TEST CONDITIONS
V_{DD} Supply Voltage (Pin 22)	V_{DD}	0	+16	V	(1) (2)
V_{CC} Supply Voltage (Pin 1)	V_{CC}	0	+8	V	(1) (2)
V_{BB} Supply Voltage (Pin 21)	V_{BB}	0	−10	V	(1)
Input Voltage	V_I	0	+8	V	(1) (2)
Output Voltage	V_O	0	+8	V	(1) (2)
Operating Temp.	T_{opt}	0	+70	°C	
Storage Temp.	T_{stg}	−40	+125	°C	

Notice: Stress beyond levels listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Operating Conditions" of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Notes:

(1) All voltages measured with respect to GND (Pin 3).

(2) $V_{BB} = -5V \pm 5\%$.

DC CHARACTERISTICS

Ta = 0° to 70°C, V_{DD} = +12V +5%, V_{CC} = +5V ± 5%, V_{BB} = −5V ± 5%

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT	TEST CONDITIONS
Output High Level	V_{OH}	3.5			V	$I_{OH} = -100\ \mu A$
Output Low Level	V_{OL}			0.4	V	$I_{OL} = 1.6$ mA
Input Low Current	I_{IL}		−0.28	−1.4	mA	$V_{IL} = 0.4V$
Output Leakage Current	I_{OLK}	−20		20	μA	V_{OUT}=0.4V to 3.5V SFD = RRD = 3V
V_{DD} Supply Current	I_{DD}		15	20	mA	$V_{IL} = 0.4V$
V_{CC} Supply Current	I_{CC}		35	50	mA	$V_{IL} = 0.4V$
V_{BB} Supply Current	I_{BB}		−0.2	−1.0	mA	

Note:

Minus (−) designates current flow out of the device.

OPERATING CONDITIONS

Ta = 0° to 70°C, V_{DD} = +12V ± 5%, V_{CC} = +5V ± 5%, V_{BB} = –5V ± 5%

CHARACTERISTIC		SYMBOL	MIN.	TYP.	MAX.	UNIT	TEST CONDITIONS
V_{DD} Supply Voltage (Pin 22)		V_{DD}	11.4	12.0	12.6	V	(1)
V_{CC} Supply Voltage (Pin 1)		V_{CC}	4.75	5.0	5.25	V	(1)
V_{BB} Supply Voltage (Pin 21)		V_{BB}	–4.75	–5.0	–5.25	V	(1)
Input High Level Voltage		V_{IH}	3.0			V	(1) (3)
Input Low Level Voltage		V_{IL}			0.8	V	(1) (3)
Clock Frequency		f_{clk}	DC		800	KHz	(2) (RRC, TRC)
t_{pw} – Pulse Width	Control Register Load	CRL_{pw}	500			ns	(CRL)
	Transmitter Holding Register Load	$THRL_{pw}$	200			ns	(THRL)
	Master Reset	MR_{pw}	500			ns	(MR)
	Data Receiver Reset	DRR_{pw}	200			ns	(DRR)
Setup Time		t_{setup}	0			ns	Refer Timing Chart (pg 12)
Hold Time		t_{hold}	20			ns	Refer Timing Chart (pg 12)

(1) All voltages measured with respect to GND (Pin 3).

(2) Clock Duty Cycle is 50%.

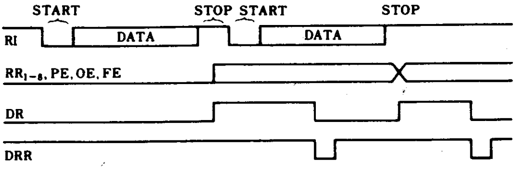
(3) Internal 18K Pull-Up Resistor.

AC CHARACTERISTICS

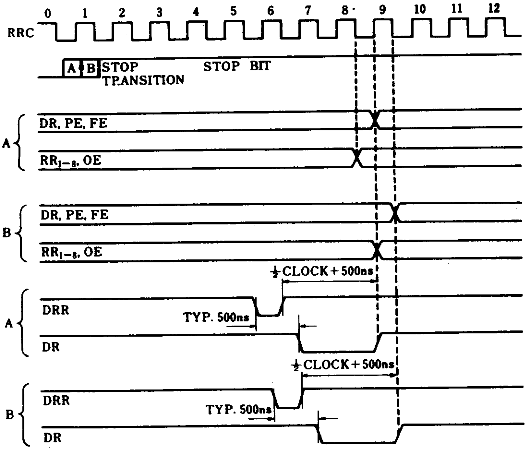
Ta = 0° to 70°C, V_{DD} = +12V ± 5%, V_{CC} = +5V ± 5%, V_{BB} = –5V ± 5%

PARAMETER	SYMBOL	MIN.	TYP.	MAX.	UNIT	TEST CONDITIONS
Input Capacitance	C_{IN}			20	pF	$V_{IN} = V_{CC}$ f = 1 MHz
Output Capacitance	C_{OUT}			20	pF	$V_{IN} = V_{CC}$ f = 1 MHz
Output Delay (Low to High)	t_{pd1}			500	ns	C_L = 20 pF 1TTL Load
Output Delay (High to Low)	t_{pd0}			500	ns	C_L = 20 pF 1TTL Load

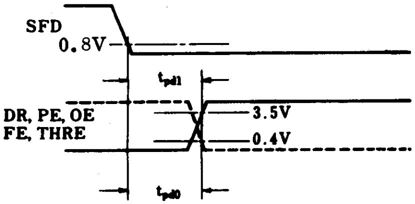
RECEIVER TIMING SEQUENCE



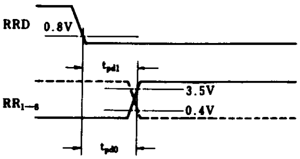
DETAILED TIMING OF RECEIVER STATUS INFORMATION



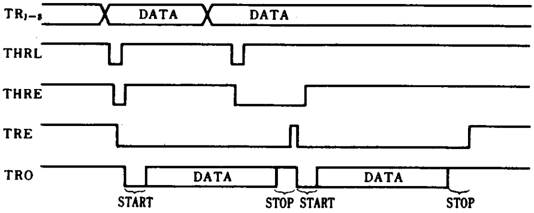
STATUS FLAG OUTPUT DELAYS



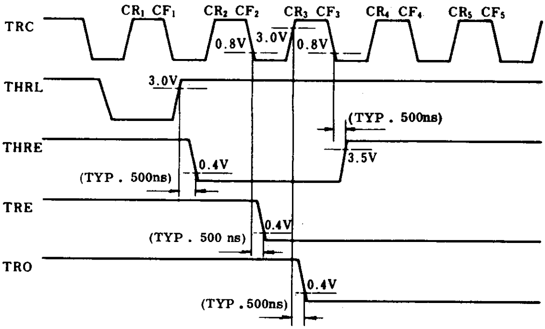
DATA OUTPUT DELAYS



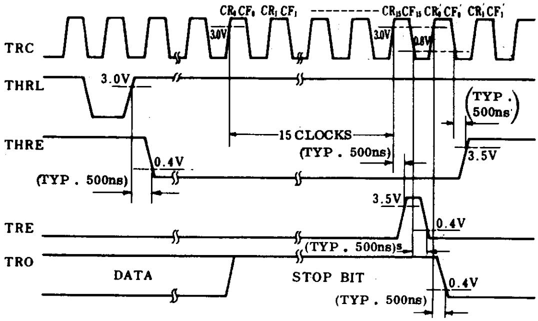
TRANSMITTER TIMING SEQUENCE



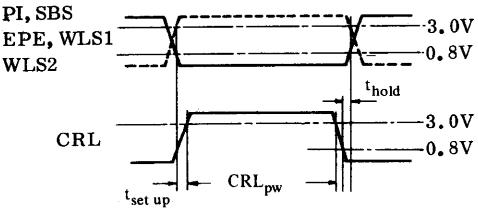
DETAILED TIMING OF TRANSMITTER BUFFER CONTROL WHEN TRE IS LOW



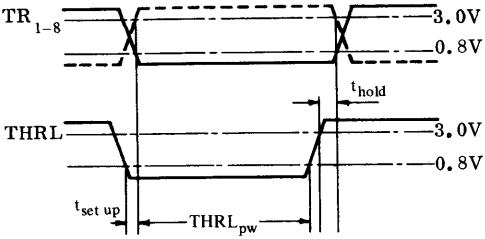
DETAILED TIMING OF TRANSMITTER BUFFER CONTROL WHEN TRE IS HIGH



CONTROL REGISTER LOAD CYCLE



DATA INPUT LOAD CYCLE



ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias	0°C to +70°C
Storage Temperature	−40°C to +125°C
All Output Voltages	0V to +8.0V
All Input Voltages	0V to +8.0V
Supply Voltage V_{CC}	0V to +8.0V
Supply Voltage V_{DD}	0V to +16.0V
Supply Voltage V_{BB}	−10.0V to 0V

COMMENT: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

*T_a = 25°C

AC CHARACTERISTICS

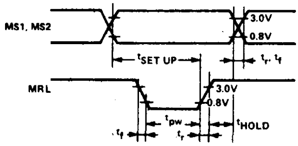
T_a = 0°C to +70°C, V_{DD} = 12V ± 5%, V_{CC} = 5V + 5%, V_{BB} = −5V ± 5%

Parameter	Symbol	Limits			Unit	Test Conditions
		Min	Typ	Max		
Clock Frequency	f_c	DC		800	KHz	TC, RC
Pulse Width	t_{PW}		*			TC, RC
		250			ns	MRL
		250			ns	$\overline{\text{TCBL}}$
		250			ns	$\overline{\text{SNTR}}/\overline{\text{CFT}}$
		250			ns	$\overline{\text{ZIP}}$
		400			ns	$\overline{\text{RR}}$
		250			ns	$\overline{\text{DRR}}$
Setup Time	$t_{SET\ UP}$	250			ns	
Hold Time	t_{HOLD}	150			ns	
Rise Time	t_r			150	ns	
Fall Time	t_f			150	ns	
Pulse Interval	t_{cc}	100			ns	
Output Delay Time	t_{pd1}		180	270	ns	C_L = 20 pf
	t_{pd2}		410	600	ns	1 TTL Load

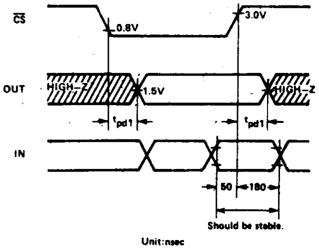
*50% Duty Cycle

TIMING WAVEFORMS

MODE SELECT



CHIP SELECT



DC CHARACTERISTICS

T_a = 0°C to +70°C, V_{DD} = 12V + 5%, V_{CC} = 5V ± 5%, V_{BB} = –5V ± 5%

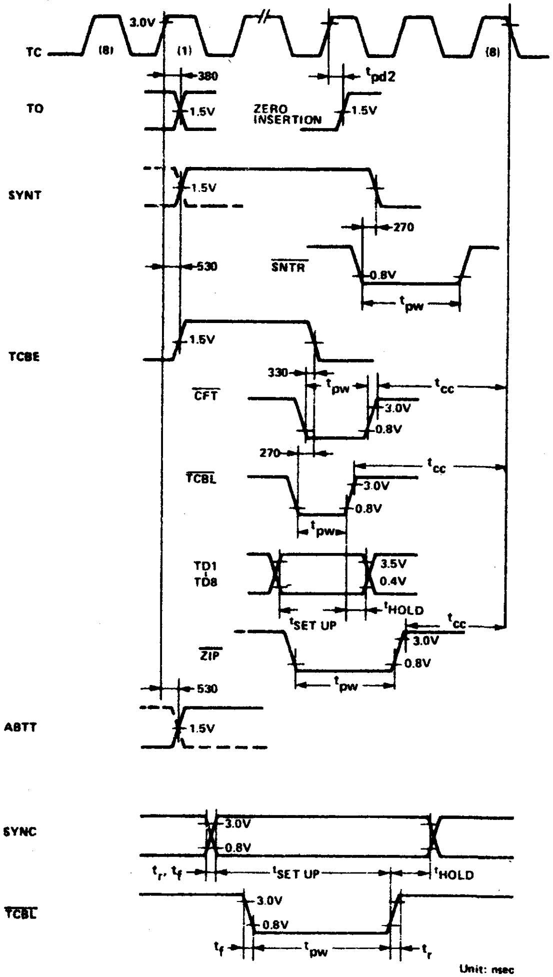
Parameter	Symbol	Limits			Unit	Test Conditions
		Min	Typ	Max		
Input High Voltage	V_{IH}	3.0		V_{DD}	V	With Built-in pull-up resistors
Input Low Voltage	V_{IL}			0.8	V	
Output Leakage Current	I_{OLK}	–20		20	μA	V_O = 0.4 to 3.5V (CS)= 3.5V
Output High Voltage	V_{OH}	3.5			V	I_{OH} = –100μA
Output Low Voltage	V_{OL}			0.4	V	I_{OL} = 1.6mA
Input Low Current	I_{IL}			–1.4	mA	V_{IL} = 0.4V
V_{DD} Supply Current	I_{DD}		15	20	mA	
V_{CC} Supply Current	I_{CC}		40	65	mA	
V_{BB} Supply Current	I_{BB}		–0.2	–2.0	mA	
Fan-out	N			1		Standard TTL Load

CAPACITANCE

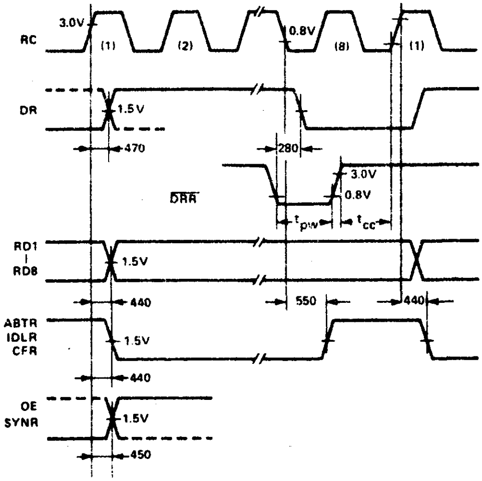
Parameter	Symbol	Limits			Unit	Test Conditions
		Min	Typ	Max		
Input Capacitance	C_{IN}			20	pf	f = 1 MHz
Output Capacitance	C_{OUT}			20	pf	f = 1 MHz

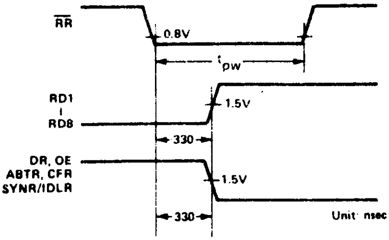
TIMING WAVEFORMS (CONT.)

TRANSMITTER SECTION



RECEIVER SECTION





Absolute Maximum Ratings*

Ambient Temperature Under Bias. . . . . 0°C to 70°C
Storage Temperature . . . . −65°C to +150°C
Voltage on Any Pin
With Respect to Ground . . . . −0.5V to +7V
Power Dissipation . . . . 1 Watt

**COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. Characteristics: T_A = 0°C to 70°C; V_{CC} = +5V ±5%; GND = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	V_{SS}-.5		.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			.45	V	I_{OL} = 1.7 mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -50μA (-100μA for D.B. Port)
I_{OH}[1]	Darlington Drive Current	1	2.0	4	mA	V_{OH} = 1.5V, R_{EXT} = 750Ω
I_{CC}	Power Supply Current		40	120	mA	
I_{IL}	Input Leakage			10	μA	V_{IN} = V_{CC}
I_{OFL}	Output Float Leakage			10	μA	V_{OUT} = V_{SS} + 0.45, V_{CC}

NOTE:
1. Available on 8 pins only.

Capacitance:

T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}.

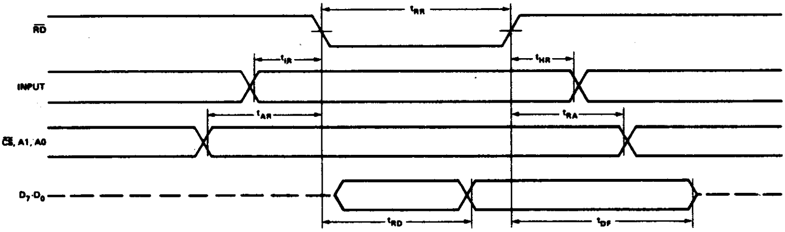
A.C. Characteristics: T_A = 0°C to 70°C; V_{CC} = +5V ±5%; GND = 0V

8080 BUS PARAMETERS:

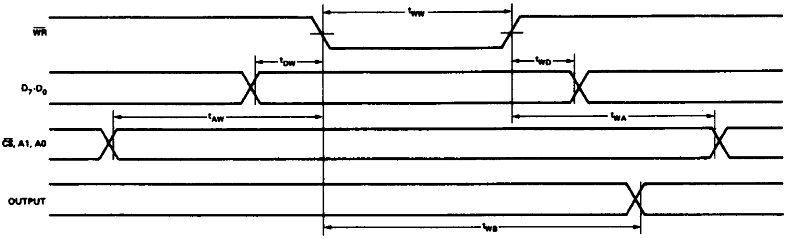
READ:

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AR}	Address Stable Before $\overline{\text{READ}}$	50		ns	
t_{RA}	Address Stable After $\overline{\text{READ}}$	0		ns	
t_{RR}	$\overline{\text{READ}}$ Pulse Width	405		ns	
t_{RD}	Data Valid From $\overline{\text{READ}}$		295	ns	CL = 100 pF
t_{DF}	Data Float After $\overline{\text{READ}}$		150	ns	CL = 100 pF
		10		ns	CL = 15 pF
t_{RV}	Time Between $\overline{\text{READS}}$ and/or $\overline{\text{WRITES}}$	850		ns	
WRITE:					
t_{AW}	Address Stable Before $\overline{\text{WRITE}}$	20		ns	
t_{WA}	Address Stable After $\overline{\text{WRITE}}$	20		ns	
t_{WW}	$\overline{\text{WRITE}}$ Pulse Width	400		ns	
t_{DW}	Data Valid To $\overline{\text{WRITE}}$ (L.E.)	50		ns	
t_{WD}	Data Valid After $\overline{\text{WRITE}}$	35		ns	
OTHER TIMINGS:					
t_{WB}	$\overline{\text{WR}}$=0 To Output		500	ns	CL = 50 pF
t_{IR}	Peripheral Data Before $\overline{\text{RD}}$	0		ns	
t_{HR}	Peripheral Data After $\overline{\text{RD}}$	100		ns	
t_{AK}	$\overline{\text{ACK}}$ Pulse Width	500		ns	
t_{ST}	$\overline{\text{STB}}$ Pulse Width	500		ns	
t_{PS}	Per. Data Before T.E. Of $\overline{\text{STB}}$	60		ns	
t_{PH}	Per. Data After T.E. Of $\overline{\text{STB}}$	180		ns	
t_{AD}	$\overline{\text{ACK}}$=0 To Output		400	ns	CL = 50 pF
t_{KD}	$\overline{\text{ACK}}$=0 To Output Float		480	ns	CL = 50 pF
		20			CL = 15 pF
t_{WOB}	$\overline{\text{WR}}$=1 To $\overline{\text{OBF}}$=0		650	ns	CL = 50 pF
t_{AOB}	$\overline{\text{ACK}}$=0 To $\overline{\text{OBF}}$=1		450	ns	CL = 50 pF
t_{SIB}	$\overline{\text{STB}}$=0 To IBF=1		450	ns	CL = 50 pF
t_{RIB}	$\overline{\text{RD}}$=1 To IBF=0		360	ns	CL = 50 pF
t_{RIT}	$\overline{\text{RD}}$=0 To INTR=0		450	ns	CL = 50 pF
t_{SIT}	$\overline{\text{STB}}$=1 To INTR=1		400	ns	CL = 50 pF
t_{AIT}	$\overline{\text{ACK}}$=1 To INTR=1		400	ns	CL = 50 pF
t_{WIT}	$\overline{\text{WR}}$=0 To INTR=0		850	ns	CL = 50 pF

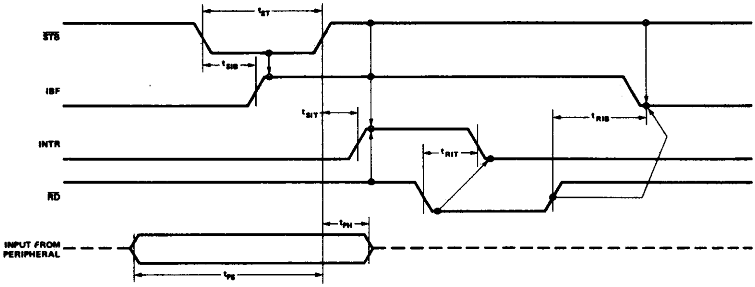
Note: Period of Reset pulse must be at least 50μs during or after power on.
Subsequent Reset pulse can be 500 ns min.



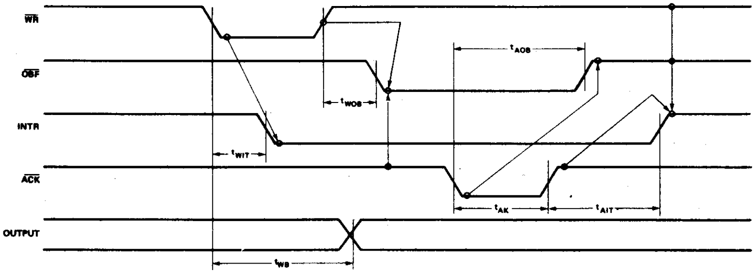
Mode 0 (Basic Input)



Mode 0 (Basic Output)

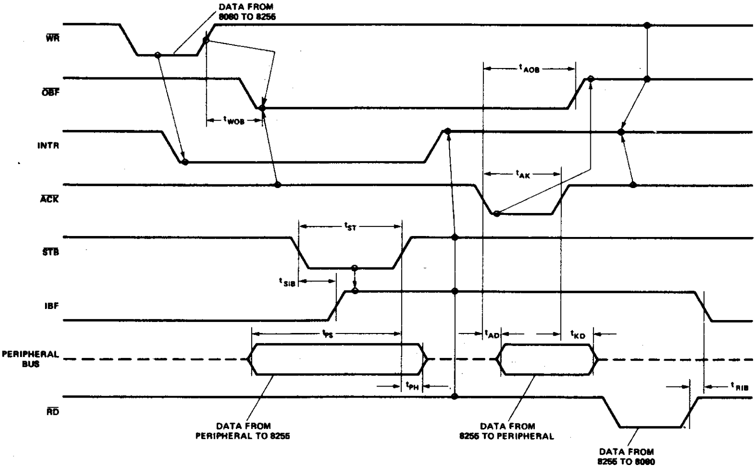


Mode 1 (Strobed Input)



Mode 1 (Strobed Output)

8255



Mode 2 (Bi-directional)

NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
(INTR = IBF · $\overline{MASK}$ · $\overline{STB}$ · $\overline{RD}$ + $\overline{OBF}$ · $\overline{MASK}$ · $\overline{ACK}$ · $\overline{WR}$)

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0°C to 70°C
Storage Temperature . . . . −65°C to +150°C
Voltage on Any Pin
With Respect to Ground . . . . −0.5V to +7V
Power Dissipation . . . . 1 Watt

**COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = +5V ±5%; GND = 0V

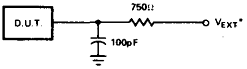
SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
I_{OL}(DB)	Output Low Current (Data Bus)	2.5		mA	V_{OL} = 0.45V
I_{OL}(PER)	Output Low Current (Peripheral Port)	1.7		mA	V_{OL} = 0.45V
I_{OH}(DB)	Output High Current (Data Bus)	-400		μA	V_{OH} = 2.4V
I_{OH}(PER)	Output High Current (Peripheral Port)	-200		μA	V_{OH} = 2.4V
I_{DAR}[1]	Darlington Drive Current	-1.0	-4.0	mA	R_{EXT} = 750Ω; V_{EXT} = 1.5V
I_{CC}	Power Supply Current		120	mA	
I_{IL}	Input Leakage		10	μA	V_{IN} = V_{CC}
I_{OFL}	Output Float Leakage		10	μA	V_{OUT} = GND + 0.45, V_{CC}

Note: 1. Adaptable on any 8 pins from Ports Band C.

CAPACITANCE T_A = 25°C; V_{CC} = GND = 0V

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNIT	TEST CONDITIONS
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND

TEST LOAD CIRCUIT (FOR DB)



* V_{EXT} IS SET AT VARIOUS VOLTAGES DURING TESTING TO GUARANTEE THE SPECIFICATION.

A.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = +5V ±5%; GND = 0V

BUS PARAMETERS:

READ:

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AR}	Address Stable Before $\overline{\text{READ}}$	0		ns	
t_{RA}	Address Stable After $\overline{\text{READ}}$	0		ns	
t_{RR}	$\overline{\text{READ}}$ Pulse Width	300		ns	
t_{RD}	Data Valid From $\overline{\text{READ}}$		250	ns	CL = 100 pF
t_{DF}	Data Float After $\overline{\text{READ}}$		150	ns	CL = 100 pF
		10		ns	CL = 15 pF
t_{RV}	Time Between $\overline{\text{READS}}$ and/or $\overline{\text{WRITES}}$	850		ns	
WRITE:					
t_{AW}	Address Stable Before $\overline{\text{WRITE}}$	0		ns	
t_{WA}	Address Stable After $\overline{\text{WRITE}}$	20		ns	
t_{WW}	$\overline{\text{WRITE}}$ Pulse Width	400		ns	
t_{DW}	Data Valid To $\overline{\text{WRITE}}$ (T.E.)	100		ns	
t_{WD}	Data Valid After $\overline{\text{WRITE}}$	30		ns	
OTHER TIMINGS:					
t_{WB}	$\overline{\text{WR}}$=1 To Output		350	ns	CL = 100 pF
t_{IR}	Peripheral Data Before $\overline{\text{RD}}$	0		ns	
t_{HR}	Peripheral Data After $\overline{\text{RD}}$	0		ns	
t_{AK}	$\overline{\text{ACK}}$ Pulse Width	300		ns	
t_{ST}	$\overline{\text{STB}}$ Pulse Width	500		ns	
t_{PS}	Per. Data Before T.E. Of $\overline{\text{STB}}$	0		ns	
t_{PH}	Per. Data After T.E. Of $\overline{\text{STB}}$	180		ns	
t_{AD}	$\overline{\text{ACK}}$=0 To Output		400	ns	CL = 100 pF
t_{KD}	$\overline{\text{ACK}}$=1 To Output Float		250	ns	CL = 100 pF
		20			CL = 15 pF
t_{WOB}	$\overline{\text{WR}}$=1 To $\overline{\text{OBF}}$=0		650	ns	CL = 100 pF
t_{AOB}	$\overline{\text{ACK}}$=0 To $\overline{\text{OBF}}$=1		350	ns	CL = 100 pF
t_{SIB}	$\overline{\text{STB}}$=0 To IBF=1		300	ns	CL = 100 pF
t_{RIB}	$\overline{\text{RD}}$=1 To IBF=0		300	ns	CL = 100 pF
t_{RIT}	$\overline{\text{RD}}$=0 To INTR=0		400	ns	CL = 100 pF
t_{SIT}	$\overline{\text{STB}}$=1 To INTR=1		300	ns	CL = 100 pF
t_{AIT}	$\overline{\text{ACK}}$=1 To INTR=1		350	ns	CL = 100 pF
t_{WIT}	$\overline{\text{WR}}$=0 To INTR=0		850	ns	CL = 100 pF

Note: Period of Reset pulse must be at least 50μs during or after power on.
Subsequent Reset pulse can be 500 ns min.

Absolute Maximum Ratings*

Temperature Under Bias Plastic	−65°C to +75°C
Storage Temperature	−65°C to +160°C
All Output or Supply Voltages	−0.5 to +7 Volts
All Input Voltages	−1.0 to 5.5 Volts
Output Currents	125 mA

**COMMENT:* *Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.*

D.C. Characteristics

T_A = 0°C to +75°C V_{CC} = +5V ±5%

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
I_F	Input Load Current ACK, DS_2, CR, DI_1-DI_8 Inputs			−.25	mA	V_F = .45V
I_F	Input Load Current MD Input			−.75	mA	V_F = .45V
I_F	Input Load Current DS_1 Input			−1.0	mA	V_F = .45V
I_R	Input Leakage Current ACK, DS, CR, DI_1-DI_8 Inputs			10	μA	V_R = 5.25V
I_R	Input Leakage Current MO Input			30	μA	V_R = 5.25V
I_R	Input Leakage Current DS_1 Input			40	μA	V_R = 5.25V
V_C	Input Forward Voltage Clamp			−1	V	I_C = −5 mA
V_{IL}	Input "Low" Voltage			.85	V	
V_{IH}	Input "High" Voltage	2.0			V	
V_{OL}	Output "Low" Voltage			.45	V	I_{OL} = 15 mA
V_{OH}	Output "High" Voltage	3.65	4.0		V	I_{OH} = −1 mA
I_{SC}	Short Circuit Output Current	-15		-75	mA	V_O = 0 V
$\|I_O\|$	Output Leakage Current High Impedance State			20	μA	V_O = .45V/5.25V
I_{CC}	Power Supply Current		90	130	mA	

A.C. Characteristics

T_A = 0°C to +75°C V_{CC} = +5V ± 5%

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
t_{pw}	Pulse Width	30			ns	
t_{pd}	Data To Output Delay			30	ns	
t_{we}	Write Enable To Output Delay			40	ns	
t_{set}	Data Setup Time	15			ns	
t_h	Data Hold Time	20			ns	
t_r	Reset To Output Delay			40	ns	
t_s	Set To Output Delay			30	ns	
t_e	Output Enable/Disable Time			45	ns	
t_c	Clear To Output Delay			55	ns	

CAPACITANCE* F = 1 MHz V_{BIAS} = 2.5V V_{CC} = +5V T_A = 25°C

Symbol	Test	LIMITS	
		Typ.	Max.
C_{IN}	DS_1, MD Input Capacitance	9 pF	12 pF
C_{IN}	DS_2, CK, ACK, DI_1-DI_8 Input Capacitance	5 pF	9 pF
C_{OUT}	DO_1-DO_8 Output Capacitance	8 pF	12 pF

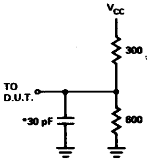
*This parameter is sampled and not 100% tested.

Switching Characteristics

CONDITIONS OF TEST

Input Pulse Amplitude = 2.5 V
Input Rise and Fall Times 5 ns
Between 1V and 2V Measurements made at 1.5V
with 15 mA & 30 pF Test Load

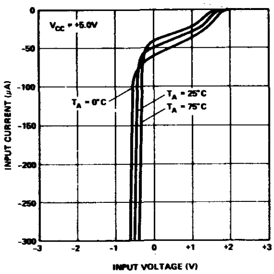
TEST LOAD
15mA & 30pF



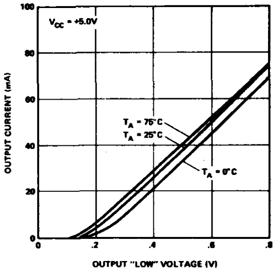
* INCLUDING JIG & PROBE CAPACITANCE

Typical Characteristics

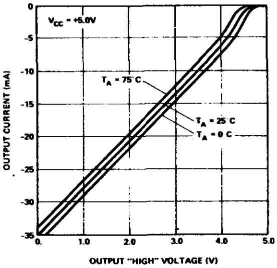
INPUT CURRENT VS. INPUT VOLTAGE



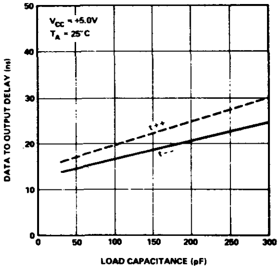
OUTPUT CURRENT VS. OUTPUT "LOW" VOLTAGE



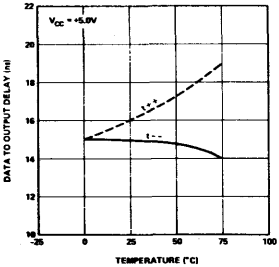
OUTPUT CURRENT VS. OUTPUT "HIGH" VOLTAGE



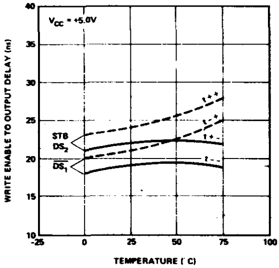
DATA TO OUTPUT DELAY VS. LOAD CAPACITANCE



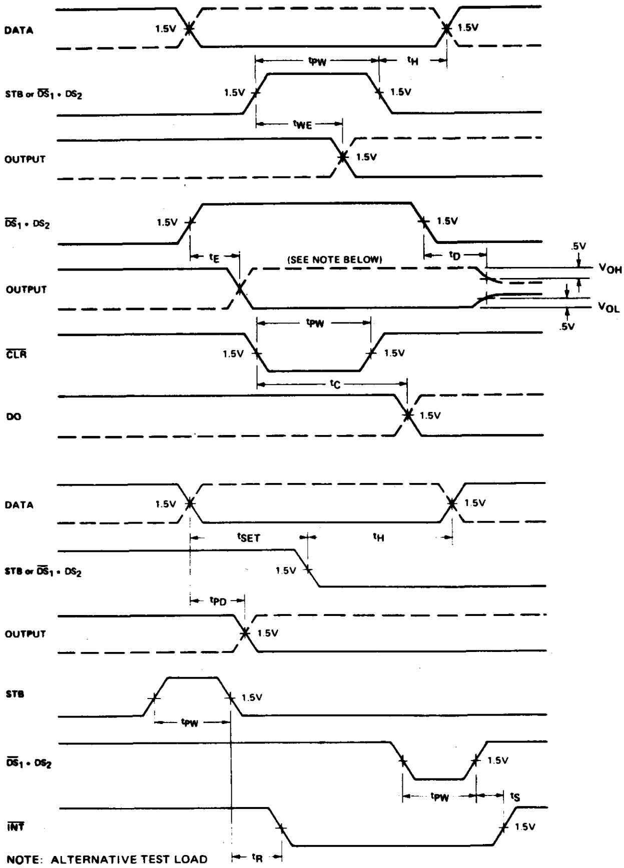
DATA TO OUTPUT DELAY VS. TEMPERATURE



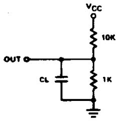
WRITE ENABLE TO OUTPUT DELAY VS. TEMPERATURE



Timing Diagram



NOTE: ALTERNATIVE TEST LOAD



ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias. . . . . 0°C to 70°C
Storage Temperature . . . . –65°C to +150°C
Voltage on Any Pin
With Respect to Ground . . . . –0.5V to +7V
Power Dissipation . . . . 1 Watt

**COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. Characteristics

T_A = 0°C to 70°C, V_{CC} = +5V ± 5%, GND = 0V

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
V_{IL}	Input Low Voltage	-.5	0.8	Volts	
V_{IH}	Input High Voltage	2.0	V_{CC}+.5	Volts	
V_{OL}	Output Low Voltage		0.45	Volts	I_{OL} = 1.6 mA
V_{OH}	Output High Voltage	2.4	V_{CC}	Volts	I_{OH}=-150μA for AB, DB and AEN I_{OH}=-80μA for others
V_{HH}	HRQ Output High Voltage	3.3	V_{CC}	Volts	I_{OH} = -80μA
I_{CC}	V_{CC} Current Drain		120	mA	
I_{IL}	Input Leakage		10	μA	V_{IN} = V_{CC}
I_{OFL}	Output Leakage During Float		10	μA	V_{OUT} [1]

Note 1: $V_{CC} > V_{OUT} >$ GND + .45V.

Capacitance

T_A = 25°C; V_{CC} = GND = 0V

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNIT	TEST CONDITIONS
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS: PERIPHERAL (SLAVE) MODE

T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%; GND = 0V (Note 1).

8080 BUS PARAMETERS:

READ CYCLE

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
T_{AR}	Adr or $\overline{\text{CS}}$↓ Setup to $\overline{\text{Rd}}$↓	50		ns	
T_{RA}	Adr or $\overline{\text{CS}}$↑ Hold from $\overline{\text{Rd}}$↑	0		ns	
T_{RDE}	Data Access from $\overline{\text{Rd}}$↓	0	300	ns	C_L = 100pF
T_{RDF}	DB→Float Delay from $\overline{\text{Rd}}$↑		150	ns	C_L = 100pF
		20		ns	C_L = 15pF
T_{RW}	$\overline{\text{Rd}}$ Width	300		ns	

WRITE CYCLE:

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
T_{CW}	$\overline{\text{CS}}$↓ Setup to $\overline{\text{Wr}}$↓	300		ns	
T_{WC}	$\overline{\text{CS}}$↑ Hold from $\overline{\text{Wr}}$↑	20		ns	
T_{AW}	Adr Setup to $\overline{\text{Wr}}$↓	20		ns	
T_{WA}	Adr Hold from $\overline{\text{Wr}}$↑	20		ns	
T_{DW}	Data Setup to $\overline{\text{Wr}}$↓	200		ns	
T_{WD}	Data Hold from $\overline{\text{Wr}}$↑	35		ns	
T_{WWS}	$\overline{\text{Wr}}$ Width	200		ns	

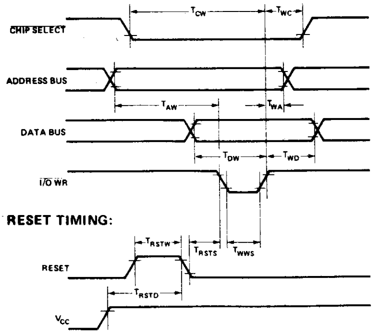
OTHER TIMING:

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
T_{RSTW}	Reset Pulse Width	300		ns	
T_{RSTD}	Power Supply↑(V_{CC}) Setup to Reset↓	500		μs	
T_r	Signal Rise Time		20	ns	
T_f	Signal Fall Time		20	ns	
T_{RSTS}	Rese to First $\overline{\text{IOWR}}$	2		t_{CY}	

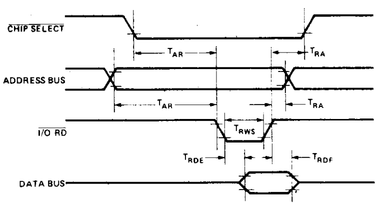
Note 1: All timing measurements are made at the following reference voltages unless specified otherwise: Input "1" at 2.0V, "0" at 0.8V
Output "1" at 2.0V, "0" at 0.8V

8257 PERIPHERAL MODE TIMING DIAGRAM

WRITE TIMING:



READ TIMING:

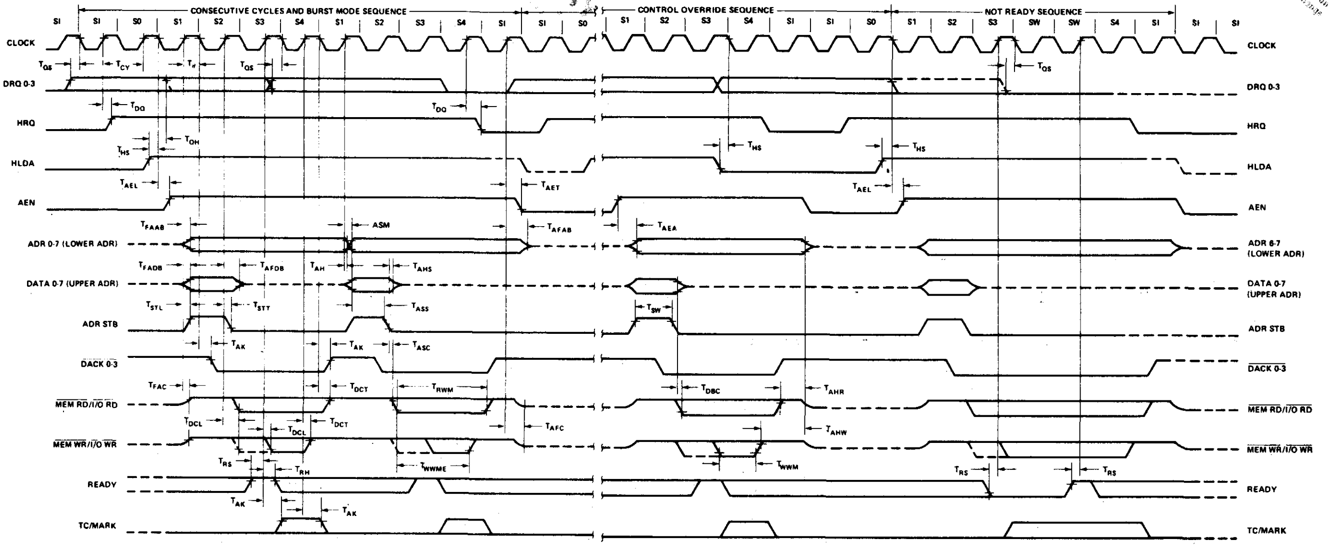


A.C. CHARACTERISTICS: DMA (MASTER) MODE T_A = 0°C to 70°C, V_{CC} = +5V ±5%, GND = 0V

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	NOTES
T_{CY}	Cycle Time (Period)	0.330	4	μs	
T_θ	Clock Active (High)	150	$.8T_{CY}$	ns	
T_{QS}	DRQ↑ Setup to θ↓(SI,S4)	120			
T_{QH}	DRQ↓ Hold from HLDA↑	0			4
T_{DQ}	HRQ↑ or ↓Delay from θ↑(SI,S4) (measured at 2.0V)		160	ns	1
T_{DQ1}	HRQ↑ or ↓Delay from θ↑(SI,S4) (measured at 3.3V)		250	ns	3
T_{HS}	HLDA↑ or ↓Setup to θ↓(SI,S4)	100		ns	
T_{AEL}	AEN↑ Delay from θ↓(S1)		300	ns	1
T_{AET}	AEN↓ Delay from θ↑(SI)		200	ns	1
T_{AEA}	Adr(AB)(Active) Delay from AEN↑(S1)	20		ns	4
T_{FAAB}	Adr(AB)(Active) Delay from θ↑(S1)		250	ns	2
T_{AFAB}	Adr(AB)(Float) Delay from θ↑(SI)		150	ns	2
T_{ASM}	Adr(AB)(Stable) Delay from θ↑(S1)		250	ns	2
T_{AH}	Adr(AB)(Stable) Hold from θ↑(S1)	T_{ASM}-50			2
T_{AHR}	Adr(AB)(Valid) Hold from $\overline{Rd}$↑(S1,SI)	60		ns	4
T_{AHW}	Adr(AB)(Valid) Hold from $\overline{Wr}$↑(S1,SI)	300		ns	4
T_{FADB}	Adr(DB)(Active) Delay from θ↑(S1)		300	ns	2
T_{AFDB}	Adr(DB)(Float) Delay from θ↑(S2)	T_{STT}+20	250	ns	2
T_{ASS}	Adr(DB) Setup to AdrStb↓(S1-S2)	100		ns	4
T_{AHS}	Adr(DB)(Valid) Hold from AdrStb↓(S2)	50		ns	4
T_{STL}	AdrStb↑ Delay from θ↑(S1)		200	ns	1
T_{STT}	AdrStb↓ Delay from θ↑(S2)		140	ns	1
T_{SW}	AdrStb Width (S1-S2)	T_{CY}-100		ns	4
T_{ASC}	$\overline{Rd}$↓ or $\overline{Wr}$(Ext)↓ Delay from AdrStb↓(S2)	70		ns	4
T_{DBC}	$\overline{Rd}$↓ or $\overline{Wr}$(Ext)↓ Delay from Adr(DB) (Float)(S2)	20		ns	4
T_{AK}	DACK↑ or ↓Delay from θ↓(S2,S1) and TC/Mark↑ Delay from θ↑(S3) and TC/Mark↓ Delay from θ↑ (S4)		250	ns	1,5
T_{DCL}	$\overline{Rd}$↓ or $\overline{Wr}$(Ext)↓ Delay from θ↑(S2) and $\overline{Wr}$↓ Delay from θ↑(S3)		200	ns	2,6
T_{DCT}	$\overline{Rd}$↑ Delay from θ↓(S1,SI) and $\overline{Wr}$↑ Delay from θ↑(S4)		200	ns	2,7
T_{FAC}	$\overline{Rd}$ or $\overline{Wr}$ (Active) from θ↑(S1)		300	ns	2
T_{AFC}	$\overline{Rd}$ or $\overline{Wr}$ (Float) from θ↑(SI)		150	ns	2
T_{RWM}	$\overline{Rd}$ Width (S2-S1 or SI)	$2T_{CY} + T_\theta$-50		ns	4
T_{WWM}	$\overline{Wr}$ Width (S3-S4)	T_{CY}-50		ns	4
T_{WWME}	$\overline{Wr}$(Ext) Width (S2-S4)	$2T_{CY}$-50		ns	4
T_{RS}	READY Set Up Time to θ↑ (S3, Sw)	30		ns	
T_{RH}	READY Hold Time from θ↑ (S3, Sw)	20		ns	

Notes: 1. Load = 1 TTL. 2. Load = 1 TTL + 50pF. 3. Load = 1 TTL + (R_L = 3.3K), V_{OH} = 3.3V. 4. Tracking Specification. 5. $\Delta T_{AK} < 50$ ns. 6. $\Delta T_{DCL} < 50$ ns. 7. $\Delta T_{DCT} < 50$ ns.

DMA MODE WAVEFORMS



Absolute Maximum Ratings

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to +150°C
Voltage On Any Pin With Respect to Ground	−0.5 V to +7 V
Power Dissipation	1 Watt

**COMMENT:*

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. Characteristics: (T_A = 0°C to 70°C; V_{CC} = 5V ±5%)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	-.5	.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+.5V	V	
V_{OL}	Output Low Voltage		.45	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400 μA
I_{LI}	Input Load Current		10	μA	V_{IN} = V_{CC} to 0V
I_{LOL}	Output Leakage Current		-10	μA	V_{OUT} = 0.45V
I_{LOH}	Output Leakage Current		10	μA	V_{OUT} = V_{CC}
I_{CC}	V_{CC} Supply Current		85	mA	

Capacitance T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. Characteristics: T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; GND = 0V

BUS PARAMETERS: (Note 1)

READ CYCLE

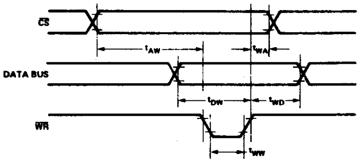
SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AR}	Address Stable Before $\overline{\text{READ}}$	50		ns	
t_{RA}	Address Hold Time for $\overline{\text{READ}}$	5		ns	
t_{RR}	$\overline{\text{READ}}$ Pulse Width	430		ns	
t_{RD}	Data Delay from $\overline{\text{READ}}$		350	ns	C_L = 100 pF
t_{DF}	$\overline{\text{READ}}$ to Data Floating		200	ns	C_L = 100 pF
		25		ns	C_L = 15 pF

WRITE CYCLE

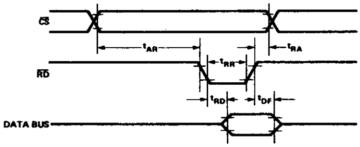
SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AW}	Address Stable Before $\overline{\text{WRITE}}$	20		ns	
t_{WA}	Address Hold Time for $\overline{\text{WRITE}}$	20		ns	
t_{WW}	$\overline{\text{WRITE}}$ Pulse Width	400		ns	
t_{DW}	Data Set Up Time for $\overline{\text{WRITE}}$	200		ns	
t_{WD}	Data Hold Time for $\overline{\text{WRITE}}$	40		ns	
t_{RV}	Recovery Time Between $\overline{\text{WRITES}}$	1		µs	

Note 1: AC timings measured at V_{OH} = 2.0, V_{OL} = .8, and with load circuit of Figure 1.

WRITE TIMING



READ TIMING



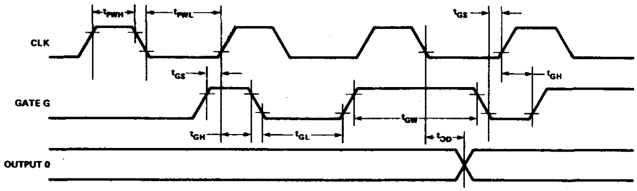
8253

PRELIMINARY
Notice: This is not a final specification. Some parametric limits are subject to change.

A.C. CHARACTERISTICS (Cont'd): T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; GND = 0V

CLOCK AND GATE TIMING

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{CLK}	Clock Period	300	dc	ns	
t_{PWH}	High Pulse Width	200		ns	
t_{PWL}	Low Pulse Width	100		ns	
t_{GW}	Trigger Pulse Width	200		ns	
t_{GS}	Gate Set Up Time To CLK↑	150		ns	
t_{GH}	Gate Hold Time After CLK↑	100		ns	
t_{GL}	Low Gate Width	100		ns	
t_{OD}	Output Delay From CLK↓		300	ns	C_L = 50 pF



8259

PRELIMINARY
Notice: This is not a final specification. Some parametric limits are subject to change.

Absolute Maximum Ratings

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to +150°C
Voltage On Any Pin With Respect to Ground	−0.5 V to +7 V
Power Dissipation	1 Watt

**COMMENT:*
Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. Characteristics: (T_A = 0°C to 70°C; V_{CC} = 5V ±5%)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	-.5	.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+.5V	V	
V_{OL}	Output Low Voltage		.45	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
$V_{OH\text{-}INT}$	Interrupt Output High Voltage	2.4 3.5		V V	I_{OH} = −400 μA I_{OH} = -50 μA
$I_{IL(IR_{0\text{-}7})}$	Input Leakage Current for $IR_{0\text{-}7}$		-300 10	μA μA	V_{IN} = 0V V_{IN} = V_{CC}
I_{IL}	Input Leakage Current for Other Inputs		10	μA	V_{IN} = V_{CC} to 0V
I_{LOL}	Output Leakage Current		-10	μA	V_{OUT} = 0.45V
I_{LOH}	Output Leakage Current		10	μA	V_{OUT} = V_{CC}
I_{CC}	V_{CC} Supply Current		85	mA	

Capacitance T_A = 25°C; V_{CC} = GND = 0V

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNIT	TEST CONDITIONS
C_{IN}	Input Capacitance			10	pF	fc = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. Characteristics: (T_A = 0°C to 70°C; V_{CC} = +5V ±5%, GND = 0V)

BUS PARAMETERS

READ

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AR}	$\overline{CS}$/A_0 Stable before $\overline{RD}$ or $\overline{INTA}$	0		ns	
t_{RA}	$\overline{CS}$/A_0 Stable after $\overline{RD}$ or $\overline{INTA}$	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	300		ns	
t_{RD}	Data Valid from $\overline{RD}$/$\overline{INTA}$		300	ns	CL = 100 pF
t_{DF}	Data Float after $\overline{RD}$/$\overline{INTA}$	 20	120	ns	CL = 100 pF CL = 20 pF

WRITE

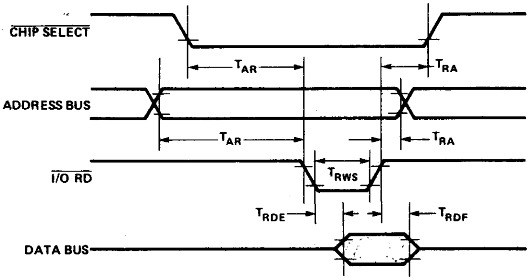
SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AW}	A_0 Stable before $\overline{WR}$	0		ns	
t_{WA}	A_0 Stable after $\overline{WR}$	220		ns	
t_{CW}	$\overline{CS}$ Stable before $\overline{WR}$	0		ns	
t_{WC}	$\overline{CS}$ Stable after $\overline{WR}$	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	300		ns	
t_{DW}	Data Valid to $\overline{WR}$ (T.E.)	200		ns	
t_{WD}	Data Valid after $\overline{WR}$	-20		ns	

OTHER TIMINGS

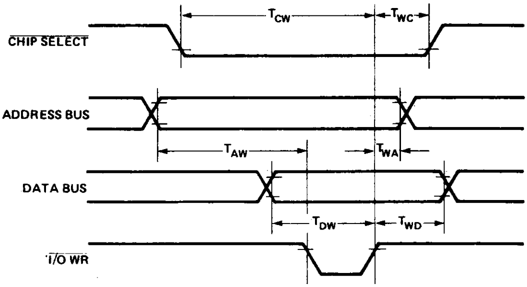
SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{IW}	Width of Interrupt Request Pulse	130		ns	
t_{INT}	INT ↑ after IR ↑	1.1		µs	
t_{IC}	Cascade Line Stable after $\overline{INTA}$ ↑	500		ns	

Waveforms

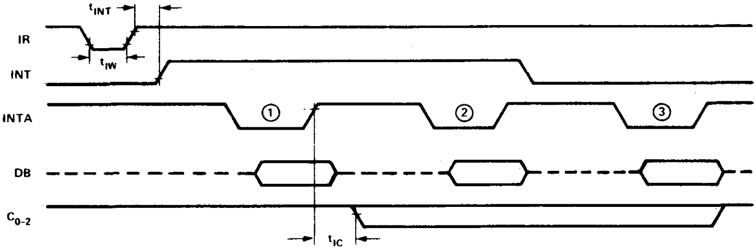
READ TIMING



WRITE TIMING



OTHER TIMING



Note:

Interrupt acknowledge $\overline{\text{INTA}}$ sequence must remain "HIGH" (at least) until leading edge of first $\overline{\text{INTA}}$.

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +150°C
All Output and Supply Voltages	-0.5V to +7V
All Input Voltages	-1.0V to +5.5V
Output Currents	100 mA

***COMMENT: Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specifications is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.**

T_A = 0°C to +70°C, V_{CC} = 5V ±5%.

Symbol	Parameter	Limits			Unit	Conditions
		Min.	Typ.[1]	Max.		
V_C	Input Clamp Voltage (all inputs)			-1.0	V	I_C=-5mA
I_F	Input Forward Current: ETLG input all other inputs		-.15 -.08	-0.5 -0.25	mA mA	V_F=0.45V
I_R	Input Reverse Current: ETLG input all other inputs			80 40	μA μA	V_R=5.25V
V_{IL}	Input LOW Voltage: all inputs			0.8	V	V_{CC}=5.0V
V_{IH}	Input HIGH Voltage: all inputs	2.0			V	V_{CC}=5.0V
I_{CC}	Power Supply Current		90	130	mA	See Note 2.
V_{OL}	Output LOW Voltage: all outputs		.3	.45	V	I_{OL}=15mA
V_{OH}	Output HIGH Voltage: ENLG output	2.4	3.0		V	I_{OH}=-1mA
I_{OS}	Short Circuit Output Current: ENLG output	-20	-35	-55	mA	V_{OS}=0V, V_{CC}=5.0V
I_{CEX}	Output Leakage Current: $\overline{INT}$ and $\overline{A_0}$-$\overline{A_2}$			100	μA	V_{CEX}=5.25V

NOTES:

[1] Typical values are for T_A = 25°C, V_{CC} = 5.0V.

[2] B_0-B_2, $\overline{SGS}$, CLK, $\overline{R_0}$-$\overline{R_4}$ grounded, all other inputs and all outputs open.

A.C. CHARACTERISTICS AND WAVEFORMS $T_A = 0°C$ to $+70°C$, $V_{CC} = +5V \pm 5\%$

Symbol	Parameter	Limits			Unit
		Min.	Typ.[1]	Max.	
t_{CY}	$\overline{\text{CLK}}$ Cycle Time	80	50		ns
t_{PW}	$\overline{\text{CLK}}$, $\overline{\text{ECS}}$, $\overline{\text{INT}}$ Pulse Width	25	15		ns
t_{ISS}	INTE Setup Time to $\overline{\text{CLK}}$	16	12		ns
t_{ISH}	INTE Hold Time after $\overline{\text{CLK}}$	20	10		ns
t_{ETCS}[2]	ETLG Setup Time to $\overline{\text{CLK}}$	25	12		ns
t_{ETCH}[2]	ETLG Hold Time After $\overline{\text{CLK}}$	20	10		ns
t_{ECCS}[2]	$\overline{\text{ECS}}$ Setup Time to $\overline{\text{CLK}}$	80	50		ns
t_{ECCH}[3]	$\overline{\text{ECS}}$ Hold Time After $\overline{\text{CLK}}$	0			ns
t_{ECRS}[3]	$\overline{\text{ECS}}$ Setup Time to $\overline{\text{CLK}}$	110	70		ns
t_{ECRH}[3]	$\overline{\text{ECS}}$ Hold Time After $\overline{\text{CLK}}$	0			
t_{ECSS}[2]	$\overline{\text{ECS}}$ Setup Time to $\overline{\text{CLK}}$	75	70		ns
t_{ECSH}[2]	$\overline{\text{ECS}}$ Hold Time After $\overline{\text{CLK}}$	0			ns
t_{DCS}[2]	$\overline{\text{SGS}}$ and $\overline{B_0}$-$\overline{B_2}$ Setup Time to $\overline{\text{CLK}}$	70	50		ns
t_{DCH}[2]	$\overline{\text{SGS}}$ and $\overline{B_0}$-$\overline{B_2}$ Hold Time After $\overline{\text{CLK}}$	0			ns
t_{RCS}[3]	$\overline{R_0}$-$\overline{R_7}$ Setup Time to $\overline{\text{CLK}}$	90	55		ns
t_{RCH}[3]	$\overline{R_0}$-$\overline{R_7}$ Hold Time After $\overline{\text{CLK}}$	0			ns
t_{ICS}	$\overline{\text{INT}}$ Setup Time to $\overline{\text{CLK}}$	55	35		ns
t_{CI}	$\overline{\text{CLK}}$ to $\overline{\text{INT}}$ Propagation Delay		15	25	ns
t_{RIS}[4]	$\overline{R_0}$-$\overline{R_7}$ Setup Time to $\overline{\text{INT}}$	10	0		ns
t_{RIH}[4]	$\overline{R_0}$-$\overline{R_7}$ Hold Time After $\overline{\text{INT}}$	35	20		ns
t_{RA}	$\overline{R_0}$-$\overline{R_7}$ to $\overline{A_0}$-$\overline{A_2}$ Propagation Delay		80	100	ns
t_{ELA}	$\overline{\text{ELR}}$ to $\overline{A_0}$-$\overline{A_2}$ Propagation Delay		40	55	ns
t_{ECA}	$\overline{\text{ECS}}$ to $\overline{A_0}$-$\overline{A_2}$ Propagation Delay		100	120	ns
t_{ETA}	ETLG to $\overline{A_0}$-$\overline{A_2}$ Propagation Delay		35	70	ns
t_{DECS}[4]	$\overline{\text{SGS}}$ and $\overline{B_0}$-$\overline{B_2}$ Setup Time to $\overline{\text{ECS}}$	15	10		ns
t_{DECH}[4]	$\overline{\text{SGS}}$ and $\overline{B_0}$-$\overline{B_2}$ Hold Time After $\overline{\text{ECS}}$	15	10		ns
t_{REN}	$\overline{R_0}$-$\overline{R_7}$ to ENLG Propagation Delay		45	70	ns
t_{ETEN}	ETLG to ENLG Propagation Delay		20	25	ns
t_{ECRN}	$\overline{\text{ECS}}$ to ENLG Propagation Delay		85	90	ns
t_{ECSN}	$\overline{\text{ECS}}$ to ENLG Propagation Delay		35	55	ns

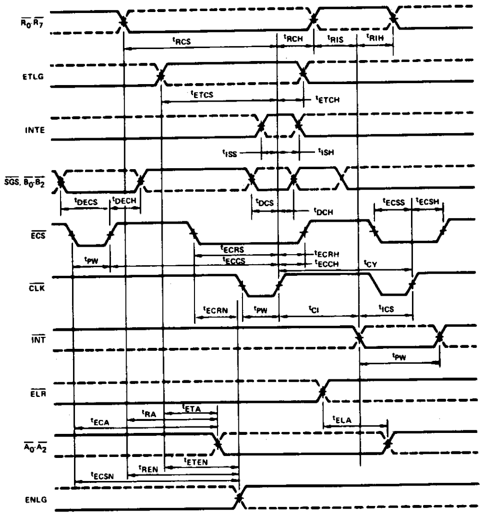
CAPACITANCE [5]

Symbol	Parameter	Limits			Unit
		Min.	Typ.[1]	Max	
C_{IN}	Input Capacitance		5	10	pF
C_{OUT}	Output Capacitance		7	12	pF

TEST CONDITIONS: $V_{BIAS} = 2.5V$, $V_{CC} = 5V$, $T_A = 25°C$, $f = 1$ MHz

NOTE 5. This parameter is periodically sampled and not 100% tested.

WAVEFORMS



NOTES:

[1] Typical values are for $T_A = 25°C, V_{CC} = 5.0V$.

[2] Required for proper operation if ISE is enabled during next clock pulse.

[3] These times are not required for proper operation but for desired change in interrupt flip-flop.

[4] Required for new request or status to be properly loaded.

TEST CONDITIONS:

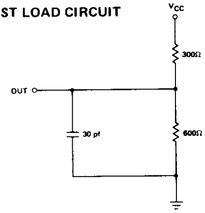
Input pulse amplitude: 2.5 volts.

Input rise and fall times: 5 ns between 1 and 2 volts.

Output loading of 15 mA and 30 pf.

Speed measurements taken at the 1.5V levels.

TEST LOAD CIRCUIT



TMS 5501 ELECTRICAL AND MECHANICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} (see Note 1)	−0.3 V to 20 V
Supply voltage, V_{DD} (see Note 1	−0.3 V to 20 V
Supply voltage, V_{SS} (see Note 1)	−0.3 V to 20 V
All input and output voltages (see Note 1)	−0.3 V to 20 V
Continuous power dissipation	1.1 W
Operating free-air temperature range	0°C to 70°C
Storage temperature range	−65°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: Under absolute maximum-ratings voltage values are with respect to the normally most negative supply voltage, V_{BB} (substrate). Throughout the remainder of this data sheet, voltage values are with respect to V_{SS} unless otherwise noted.

RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB}	−4.75	−5	−5.25	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except clocks)	3.3		V_{CC}+1	V
High-level clock input voltage, $V_{IH(\phi)}$	V_{DD}−1		V_{DD}+1	V
Low-level input voltage, V_{IL} (all inputs except clocks) (see Note 2)	−1		0.8	V
Low-level clock input voltage, $V_{IL(\phi)}$ (see Note 2)	−1		0.6	V
Operating free-air temperature, T_A	0		70	°C

NOTE 2: The algebraic convention where the most negative limit is designated as minimum is used in this specification for logic voltage levels only.

ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

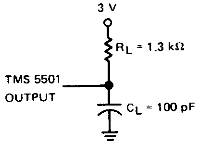
PARAMETER		TEST CONDITIONS		MIN	MAX	UNIT
I_I	Input current (any input except clocks and data bus)	V_I = 0 V to V_{CC}			±10	μA
$I_{I(\phi)}$	Clock input current	$V_{I(\phi)}$ = 0 V to V_{DD}			±10	μA
$I_{I(DB)}$	Input current, data bus	$V_{I(DB)}$ = 0 V to V_{CC},	CE at 0 V		−100	μA
V_{OH}	High-level output voltage	I_{OH} = 400 μA		3.7		V
V_{OL}	Low-level output voltage	I_{OL} = 1.7 mA,			0.45	V
$I_{BB(av)}$	Average supply current from V_{BB}	Operating at $t_{c(\phi)}$ = 480 ns, T_A = 25°C			−1	mA
$I_{CC(av)}$	Average supply current from V_{CC}				100	
$I_{DD(av)}$	Average supply current from V_{DD}				40	
C_i	Capacitance, any input except clock	V_{CC} = V_{DD} = V_{SS} = 0 V, V_{BB} = −4.75 to −5.25 V, All other pins at 0 V	f = 1 MHz,		10	pF
$C_{I(\phi)}$	Clock input capacitance				75	
C_o	Output capacitance				20	

TIMING REQUIREMENTS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURES 5 AND 6)

		MIN	MAX	UNIT
$t_{c(\phi)}$	Clock cycle time	480	2000	ns
$t_{r(\phi)}$	Clock rise time	5	50	ns
$t_{f(\phi)}$	Clock fall time	5	50	ns
$t_{w(\phi 1)}$	Pulse width, clock 1 high	60		ns
$t_{w(\phi 2)}$	Pulse width, clock 2 high	200	300	ns
$t_{d(\phi 1L-\phi 2)}$	Delay time, clock 1 low to clock 2	0		ns
$t_{d(\phi 2-\phi 1)}$	Delay time, clock 2 to clock 1	70		ns
$t_{d(\phi 1H-\phi 2)}$	Delay time, clock 1 high to clock 2 (time between leading edges)	130		ns
$t_{su(ad)}$	Address setup time	50		ns
$t_{su(CE)}$	Chip-enable setup time	50		ns
$t_{su(da)}$	Data setup time	50		ns
$t_{su(sync)}$	Sync setup time	50		ns
$t_{su(XI)}$	External input setup time	50		ns
$t_{h(ad)}$	Address hold time	0		ns
$t_{h(CE)}$	Chip-enable hold time	10		ns
$t_{h(da)}$	Data hold time	10		ns
$t_{h(sync)}$	Sync hold time	10		ns
$t_{h(XI)}$	External input hold time	40		ns
$t_{w(sens\ H)}$	Pulse width, sensor input high	500		ns
$t_{w(sens\ L)}$	Pulse width, sensor input low	500		ns
$t_{d(sens-int)}$	Delay time, sensor to interrupt (time between leading edges)		2000	ns
$t_{d(rst-int)}$	Delay time, RST instruction to interrupt (time between trailing edges)		500	ns

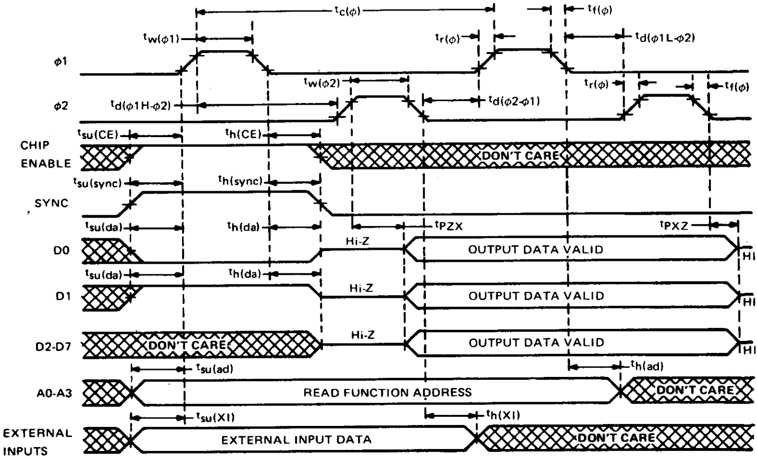
SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURES 6 AND 7)

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
t_{PZX}	Data bus output enable time	C_L = 100 pF, R_L = 1.3 kΩ		200	ns
t_{PXZ}	Data bus output disable time to high-impedance state			180	ns
t_{PD}	External data output propagation delay time from φ2			200	ns



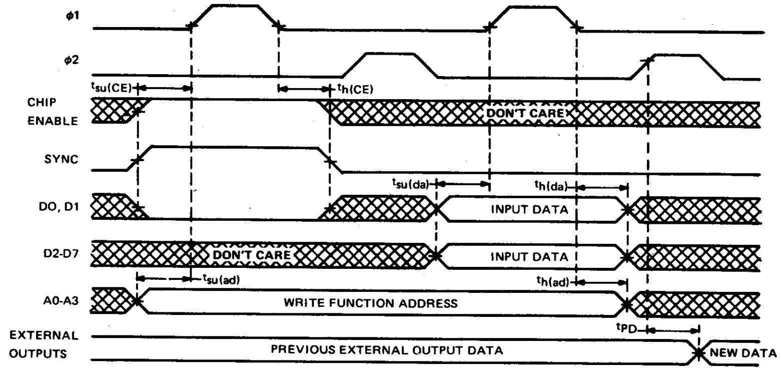
C_L includes probe and jig capacitance

LOAD CIRCUIT



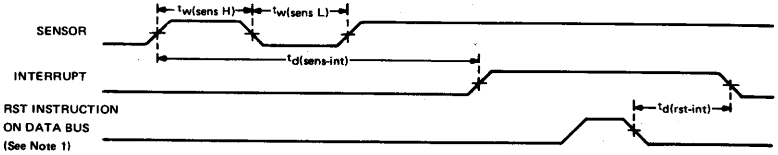
NOTE: For φ1 or φ2 inputs, high and low timing points are 90% and 10% of $V_{IH(\phi)}$. All other timing points are the 50% level.

FIGURE 6–READ CYCLE TIMING



NOTE: For $\phi1$ and $\phi2$ inputs, high and low timing points are 90% and 10% of $V_{IH(\phi)}$. All other timing points are the 50% level.

FIGURE 7–WRITE CYCLE TIMING



NOTES: 1. The RST instruction occurs during the output data valid time of tne read cycle.
2. All timing points are 50% of V_{IH}.

FIGURE 8–SENSOR/INTERRUPT TIMING

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias:	Ceramic	–65°C to +125°C
	Plastic	–65°C to +75°C
Storage Temperature		–65°C to +160°C
All Output or Supply Voltages		–0.5 to +7 Volts
All Input Voltages		–1.0 to +5.5 Volts
Output Currents		125 mA

*COMMENT

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

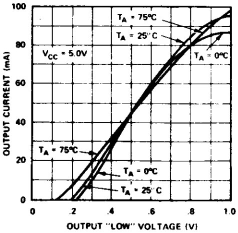
D.C. CHARACTERISTICS T_A = 0°C to +75°C, V_{CC} = 5.0V ±5%

8205

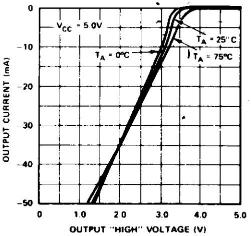
SYMBOL	PARAMETER	LIMIT		UNIT	TEST CONDITIONS
		MIN.	MAX.		
I_F	INPUT LOAD CURRENT		–0.25	mA	V_{CC} = 5.25V, V_F = 0.45V
I_R	INPUT LEAKAGE CURRENT		10	μA	V_{CC} = 5.25V, V_R = 5.25V
V_C	INPUT FORWARD CLAMP VOLTAGE		–1.0	V	V_{CC} = 4.75V, I_C = –5.0 mA
V_{OL}	OUTPUT "LOW" VOLTAGE		0.45	V	V_{CC} = 4.75V, I_{OL} = 10.0 mA
V_{OH}	OUTPUT HIGH VOLTAGE	2.4		V	V_{CC} = 4.75V, I_{OH} = –1.5 mA
V_{IL}	INPUT "LOW" VOLTAGE		0.85	V	V_{CC} = 5.0V
V_{IH}	INPUT "HIGH" VOLTAGE	2.0		V	V_{CC} = 5.0V
I_{SC}	OUTPUT HIGH SHORT CIRCUIT CURRENT	–40	–120	mA	V_{CC} = 5.0V, V_{OUT} = 0V
V_{OX}	OUTPUT "LOW" VOLTAGE @ HIGH CURRENT		0.8	V	V_{CC} = 5.0V, I_{OX} = 40 mA
I_{CC}	POWER SUPPLY CURRENT		70	mA	V_{CC} = 5.25V

TYPICAL CHARACTERISTICS

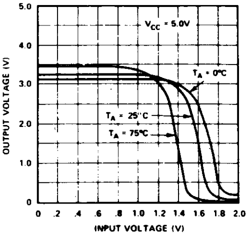
OUTPUT CURRENT VS. OUTPUT "LOW" VOLTAGE



OUTPUT CURRENT VS. OUTPUT "HIGH" VOLTAGE



DATA TRANSFER FUNCTION



8205 SWITCHING CHARACTERISTICS

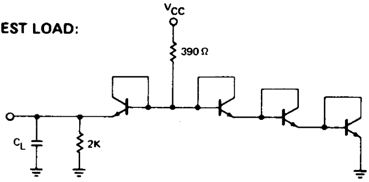
CONDITIONS OF TEST:

Input pulse amplitudes: 2.5V

Input rise and fall times: 5 nsec between 1V and 2V

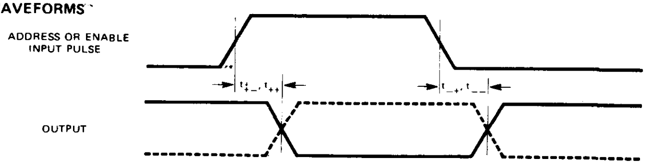
Measurements are made at 1.5V.

TEST LOAD:



All Transistors 2N2369 or Equivalent. C_L = 30 pF,

TEST WAVEFORMS



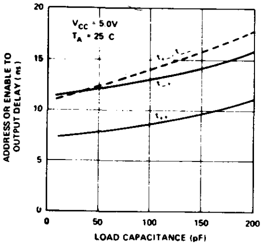
A.C. CHARACTERISTICS T_A = 0°C to +75°C, V_{CC} = 5.0V ±5% unless otherwise specified.

SYMBOL	PARAMETER		MAX. LIMIT	UNIT	TEST CONDITIONS
t_{++}	ADDRESS OR ENABLE TO OUTPUT DELAY		18	ns	
t_{-+}			18	ns	
t_{+-}			18	ns	
t_{--}			18	ns	
C_{IN}[1]	INPUT CAPACITANCE	P8205	4(typ.)	pF	f = 1 MHz, V_{CC} = 0V
		C8205	5(typ.)	pF	V_{BIAS} = 2.0V, T_A = 25°C

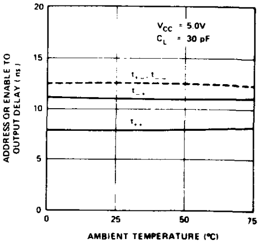
[1] This parameter is periodically sampled and is not 100% tested.

TYPICAL CHARACTERISTICS

ADDRESS OR ENABLE TO OUTPUT DELAY VS. LOAD CAPACITANCE



ADDRESS OR ENABLE TO OUTPUT DELAY VS. AMBIENT TEMPERATURE



D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +150°C
All Output and Supply Voltages	-0.5V to +7V
All Input Voltages	-1.0V to +5.5V
Output Currents	125 mA

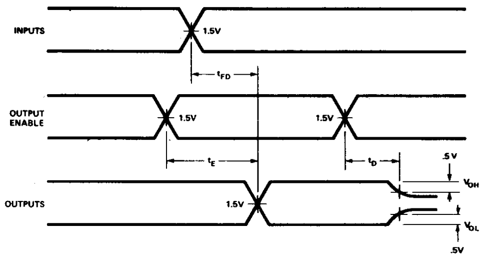
*COMMENT: Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

T_A = 0°C to +70°C, V_{CC}=+5V ±5%

Symbol	Parameter	Limits			Unit	Conditions
		Min.	Typ.	Max.		
I_{F1}	Input Load Current $\overline{DIEN}$, $\overline{CS}$		-0.15	-.5	mA	V_F = 0.45
I_{F2}	Input Load Current All Other Inputs		-0.08	-.25	mA	V_F = 0.45
I_{R1}	Input Leakage Current $\overline{DIEN}$, $\overline{CS}$			20	μA	V_R = 5.25V
I_{R2}	Input Leakage Current DI Inputs			10	μA	V_R = 5.25V
V_C	Input Forward Voltage Clamp			-1	V	I_C = -5mA
V_{IL}	Input "Low" Voltage			.95	V	
V_{IH}	Input "High" Voltage	2.0			V	
$\|I_O\|$	Output Leakage Current (3-State) DO DB			20 100	μA	V_O = 0.45V/5.25V
I_{CC}	Power Supply Current 8216		95	130	mA	
	Power Supply Current 8226		85	120	mA	
V_{OL1}	Output "Low" Voltage		0.3	.45	V	DO Outputs I_{OL}=15mA DB Outputs I_{OL}=25mA
V_{OL2}	Output "Low" Voltage 8216		0.5	.6	V	DB Outputs I_{OL}=55mA
	Output "Low" Voltage 8226		0.5	.6	V	DB Outputs I_{OL}=50mA
V_{OH1}	Output "High" Voltage	3.65	4.0		V	DO Outputs I_{OH} = -1mA
V_{OH2}	Output "High" Voltage	2.4	3.0		V	DB Outputs I_{OH} = -10mA
I_{OS}	Output Short Circuit Current	-15 -30	-35 -75	-65 -120	mA mA	DO Outputs $V_O \cong 0V$, DB Outputs V_{CC}=5.0V

NOTE: Typical values are for T_A = 25°C, V_{CC}=5.0V.

WAVEFORMS



A.C. CHARACTERISTICS

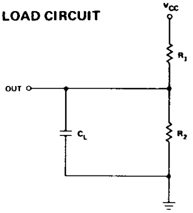
T_A = 0°C to +70°C, V_{CC} = +5V ±5%

Symbol	Parameter	Limits Min.	Limits Typ.[1]	Limits Max.	Unit	Conditions
T_{PD1}	Input to Output Delay DO Outputs		15	25	ns	C_L=30pF, R_1=300Ω R_2=600Ω
T_{PD2}	Input to Output Delay DB Outputs 8216		20	30	ns	C_L=300pF, R_1=90Ω
	8226		16	25	ns	R_2 = 180Ω
T_E	Output Enable Time 8216		45	65	ns	(Note 2)
	8226		35	54	ns	(Note 3)
T_D	Output Disable Time		20	35	ns	(Note 4)

TEST CONDITIONS:

Input pulse amplitude of 2.5V.
Input rise and fall times of 5 ns between 1 and 2 volts.
Output loading is 5 mA and 10 pF.
Speed measurements are made at 1.5 volt levels.

TEST LOAD CIRCUIT



Capacitance[5]

Symbol	Parameter	Limits Min.	Limits Typ.[1]	Limits Max.	Unit
C_{IN}	Input Capacitance		4	8	pF
C_{OUT1}	Output Capacitance		6	10	pF
C_{OUT2}	Output Capacitance		13	18	pF

TEST CONDITIONS: V_{BIAS} = 2.5V, V_{CC} = 5.0V, T_A = 25°C, f = 1 MHz.

NOTES:
1. Typical values are for T_A = 25°C, V_{CC} = 5.0V.
2. DO Outputs, C_L = 30pF, R_1 = 300/10 KΩ, R_2 = 180/1KΩ; DB Outputs, C_L = 300pF, R_1 = 90/10 KΩ, R_2 = 180/1 KΩ.
3. DO Outputs, C_L = 30pF, R_1 = 300/10 KΩ, R_2 = 600/1K; DB Outputs, C_L = 300pF, R_1 = 90/10 KΩ, R_2 = 180/1 KΩ.
4. DO Outputs, C_L = 5pF, R_1 = 300/10 KΩ, R_2 = 600/1 KΩ; DB Outputs, C_L = 5pF, R_1 = 90/10 KΩ, R_2 = 180/1 KΩ.
5. This parameter is periodically sampled and not 100% tested.

Chapter 5
THE 8085

The 8085 is Intel's enhancement of the 8080A — just as the Z80 is Zilog's enhancement of the 8080A. The Z80 is described in Chapter 7.

Intel is the developer of the 8085; Intel is also the principal manufacturer of the 8080A. But the individuals at Zilog who developed the Z80 were previously employed by Intel, at which time they developed the 8080A from the 8008. The Z80 and the 8085 therefore have equal claim to be the legitimate descendent of the 8080A.

The 8085 provides the same logic as the 8080A, 8224 and 8228 three-chip CPU. The 8085 has the following additional enhancements:

1) The 8085 requires a single +5V power supply.
2) The 8085 uses a single clock signal.
3) The 8085 has a primitive on-chip serial I/O capability which may also be used to input status and output control signals.
4) The 8085 has interrupt request pins with hardware-generated interrupt vectoring.
5) The 8085 operates with a standard 320 nanosecond clock as against the standard 500 nanosecond clock of the 8080A. But recall that there are versions of the 8080A that operate with a 250 nanosecond clock.

The 8085 instruction set is almost identical to the 8080A instruction set; in contrast, the Z80 has a massively expanded instruction set. The large Z80 instruction set has been criticized for its complexity, but one could argue that since the Z80 also provides the complete 8080A instruction set, anyone who does not want to use the additional instructions can simply ignore them.

The 8085 multiplexes its Data Bus with the low order Address Bus lines. Such multiplexing demands custom support devices, or external demultiplexing logic.

Figure 5-3 and associated text provide a direct comparison of 8085 and 8080A signal interfaces.

In addition to the 8085 microprocessor, support devices described in this chapter include:

- The 8155/8156 static RAM with I/O ports and timer. This device provides 256 bytes of static read/write memory.
- The 8355 ROM with I/O ports. This device provides 2048 bytes of read-only memory plus I/O logic.
- The 8755 EPROM with I/O ports. This device provides 2048 bytes of erasable programmable read-only memory with I/O logic.

Note that the 8080A support devices described in Chapter 4 may also be used with the 8085.

Currently the only manufacturer of the 8085 is:

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

Intel has a policy of not officially authorizing second sources. Companies currently second sourceing the 8080A are likely to watch and see how the 8085 fares against the Z80 before committing the engineering expense associated with developing an 8085 second source.

The 8085 uses a single +5V power supply; it is packaged as a 40-pin DIP.

Using a 320 nanosecond clock, instruction execution times range from 1.3 microseconds to 5.75 microseconds.

All 8085 devices have TTL compatible signals.

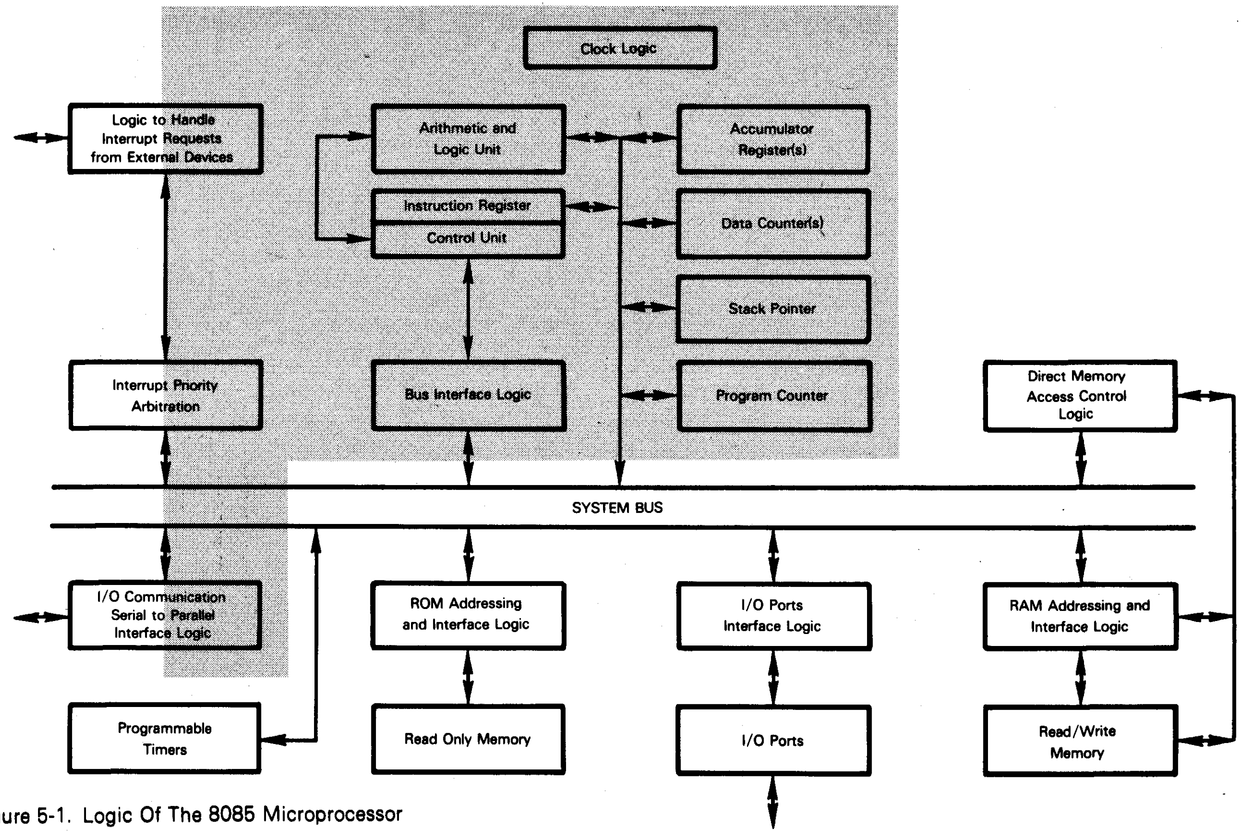

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 5-1. Logic Of The 8085 Microprocessor

THE 8085 CPU

Functions implemented on the 8085 CPU are illustrated in Figure 5-1; they represent typical CPU logic. The 8085 has an Arithmetic and Logic Unit, a Control Unit, Accumulators and registers.

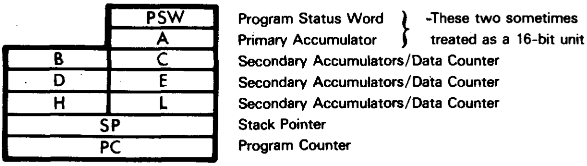
Clock logic is on the 8085 CPU chip; only an external crystal or RC network is needed.

Bus interface logic which was excluded on the 8080A is provided by the 8085.

N-channel silicon gate technology is used by all 8085 devices.

8085 PROGRAMMABLE REGISTERS

The 8085 programmable registers are identical to the 8080A programmable registers. They may be illustrated as follows:


PSW
A
B
C
D
E
H
L
SP
PC
Program Status Word
Primary Accumulator
-These two sometimes treated as a 16-bit unit
Secondary Accumulators/Data Counter
Secondary Accumulators/Data Counter
Secondary Accumulators/Data Counter
Stack Pointer
Program Counter

For a discussion of 8085 programmable registers refer to the 8080A CPU description given in Chapter 4.

8085 ADDRESSING MODES

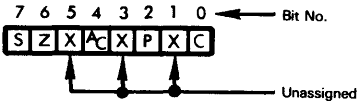
The 8085 uses exactly the same memory addressing modes as the 8080A. Direct and implied memory addressing are available. See the 8080A addressing modes description given in Chapter 4 for details.

8085 STATUS

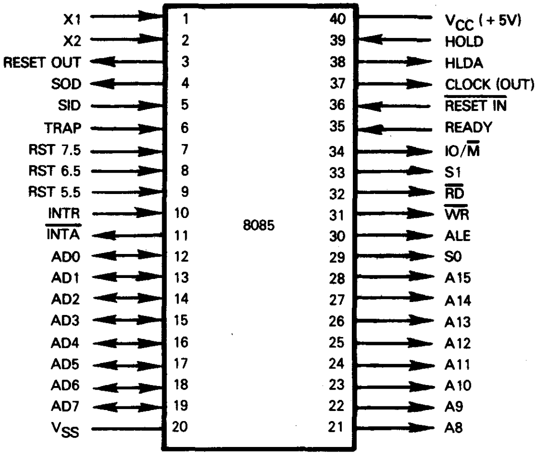
The 8085 has the same set of status flags as the 8080A; status flags are stored in the same bits of the Program Status Words. The five status flags provided are:

Zero (Z)
Sign (S)
Parity (P)
Carry (C)
Auxiliary Carry (AC)

Status flags are assigned to bits of the Program Status Words as follows:



For a discussion of status flags refer to the 8080A status description given in Chapter 4.



PIN NAME	DESCRIPTION	TYPE
AD0 - AD7	Address/Data Bus	Bidirectional, tristate
A8 - A15	Address Bus	Output, tristate
ALE	Address Latch Enable	Output, tristate
$\overline{\text{RD}}$	Read Control	Output, tristate
$\overline{\text{WR}}$	Write Control	Output, tristate
IO/$\overline{\text{M}}$	I/O or Memory Indicator	Output, tristate
S0, S1	Bus State Indicators	Output
READY	Wait State Request	Input
SID	Serial Data Input	Input
SOD	Serial Data Output	Output
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
INTR	Interrupt Request	Input
TRAP	Non-maskable Interrupt Request	Input
RST 5.5 RST 6.5 RST 7.5	Hardware vectored interrupt requests	Input Input Input
$\overline{\text{INTA}}$	Interrupt Acknowledge	Output
$\overline{\text{RESET IN}}$	System Reset	Input
RESET OUT	Peripherals Reset	Output
X1, X2	Crystal or RC Connections	Input
CLK	Clock Signal	Output
V_{CC} V_{SS}	Power, Ground	

Figure 5-2. 8085 CPU Signals And Pin Assignments

8085 CPU PINS AND SIGNALS

8085 CPU pins and signals are illustrated in Figure 5-2.

Whereas the internal architecture and the instruction sets of the 8080A and the 8085 are very similar, pins and signals are not. We will therefore begin by describing 8085 signals without reference to, or comparison with, the 8080A; then we will compare the two interfaces.

The Address and Data Busses of the 8085 are multiplexed. Pins A8 - A15 are output-only lines which carry the high order byte of memory addresses. AD0 - AD7 are bidirectional lines which output the low order byte of memory addresses; AD0 - AD7 also serve as a bidirectional Data Bus.

ALE is an address latch enable signal which pulses high when address data is being output on AD0 - AD7. You may use the falling edge of ALE to strobe the address off AD0 - AD7 into external latches if you are demultiplexing AD0 - AD7 into separate Address and Data Busses.

Five control signals control memory and I/O accesses.

8085 CONTROL SIGNALS

$\overline{\text{RD}}$ is pulsed low for a memory or I/O read operation.

$\overline{\text{WR}}$ is pulsed low for a memory or I/O write operation.

IO/$\overline{\text{M}}$ is output high in conjunction with $\overline{\text{RD}}$ or $\overline{\text{WR}}$ for an I/O access.

IO/$\overline{\text{M}}$ is output low in conjunction with $\overline{\text{RD}}$ or $\overline{\text{WR}}$ for a memory read or write operation.

8085 DATA BUS DEFINITION SIGNALS

The state of the System Bus is further defined by the S0 and S1 status signals as follows:

S1	S0	OPERATION SPECIFIED
0	0	Halt
0	1	Memory or I/O write
1	0	Memory or I/O read
1	1	Instruction fetch

External logic that does not have sufficient time to respond to an access can gain additional time by using the READY input signal. **The READY input can be used to insert Wait state clock periods in any machine cycle.** Timing and logic associated with Wait states is described later in this chapter.

8085 SERIAL I/O

Two signals allow a primitive serial I/O capability. **The high order Accumulator bit may be output via SOD. The signal level at SID may be input to the high order bit of the Accumulator.** SID and SOD may also be used to input status and to output control signals.

Two signals allow external logic to take control of the System Bus.

8085 BUS CONTROL SIGNALS

HOLD, when input high, floats the Address Bus plus the $\overline{\text{RD}}$, $\overline{\text{WR}}$, IO/$\overline{\text{M}}$ and ALE control signals. HLDA is output high to acknowledge this Hold condition.

8085 INTERRUPT SIGNALS

There are six signals associated with interrupt logic. Interrupts may be requested via INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP. An interrupt request made via INTR is acknowledged via the $\overline{\text{INTA}}$ output.

INTR is the general purpose interrupt request used by external logic; it is equivalent to the 8080A INTR signal.

TRAP is a non-maskable, highest priority interrupt request. TRAP is used for catastrophic failure interrupts.

RST 5.5, RST 6.5 and RST 7.5 are three interrupt request signals supported by hardware-implemented vectoring.

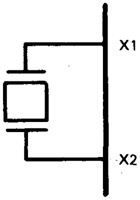
Interrupt capabilities of the 8085 are described in detail later in this chapter.

There are two signals associated with 8085 Reset logic.

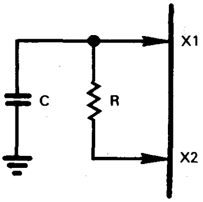
8085 RESET SIGNALS

$\overline{\text{RESET IN}}$ is the Reset input signal. This signal need not be synchronized with the clock. RESET OUT is a Reset signal output by the 8085 for use throughout the rest of the 8085 microcomputer system.

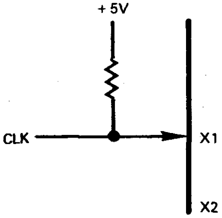
X1 and X2 connect an external crystal or RC network to drive clock logic internal to the 8085. A crystal will be connected as follows:



An RC network will be connected as follows:

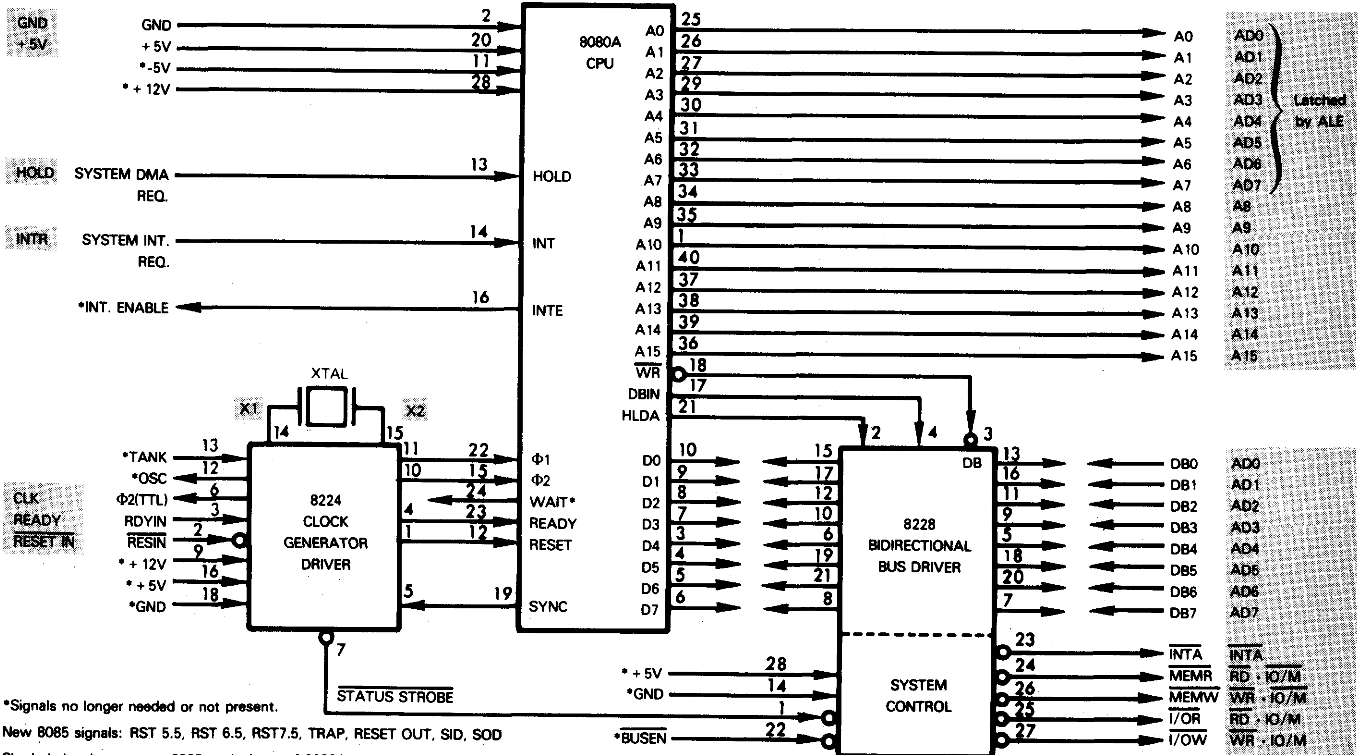


You can apply a clock signal directly to X1:



Slave 8085 devices in a multiple CPU system will usually be driven directly by a clock signal.

A TTL level clock signal CLK is output by the 8085. It may be used to drive slave CPUs, or for any other synchronization purpose within the microcomputer system.



*Signals no longer needed or not present.

New 8085 signals: RST 5.5, RST 6.5, RST7.5, TRAP, RESET OUT, SID, SOD

Shaded signals represent 8085 equivalents of 8080A.

Figure 5-3. A Comparison Of 8085 And 8080A/8224/8228 Signal Interface

A COMPARISON OF 8085 AND 8080A SIGNALS

No attempt has been made to maintain any kind of pin compatibility between the 8085 and the 8080A. Nevertheless, as illustrated in Figure 5-3, it is relatively simple to derive equivalent system busses when using the 8085 or the 8080A. But look at Figure 5-3 with an element of caution. Many logical combinations of 8085 signals are shown reproducing 8080A signals; in reality you will never generate such logical combinations — a point which will become clear as the chapter proceeds. **The purpose of Figure 5-3 is to illustrate the equivalence of the system busses generated by the 8085 and the 8080A without indicating that creation of equivalent busses is desirable.**

The 8080A signals which are shown as having direct 8085 equivalents are either obvious, or will become so after you have read this chapter.

What is more interesting is to look at the 8080A signals which no longer exist and the new 8085 signals which have been added.

Let us first **look at the signals which have been dropped.**

There are the surplus power supplies -5V and +12V, plus the secondary power supplies required by the 8224 Clock Generator and the 8228 System Controller. Elimination of these signals is self-evident.

INTE is an 8080A signal that indicates to external logic when interrupts have or have not been enabled internally by the 8080A. This signal is not very useful since external logic cannot use the information it provides. Apart from illuminating an appropriate indicator on a minicomputer-like control panel, the INTE signal of the 8080A serves little useful purpose.

WAIT is a signal which is output high by the 8080A while Wait states are being inserted within a machine cycle. There is little that external logic can do with this signal, therefore its elimination in the 8085 carries no penalty.

$\overline{\text{BUSEN}}$ is a control input to the 8228 System Controller; it causes the 8228 to float its output signals. This signal is no longer required in the 8085 since the Hold state floats all equivalent 8085 output signals — with the exception of $\overline{\text{INTA}}$, which does not need to be floated.

The 8224 Clock Generator outputs two synchronizing clock signals — OSC and Φ2 (TTL). Φ2 (TTL) is approximately reproduced by CLK; OSC has no equivalent 8085 signal.

The TANK input to the 8224 Clock Generator allows overtones of the external crystal to be used. No such signal exists with the 8085 — which simply means that you have to use the primary frequency of any crystal connected across the X1 and X2 inputs.

Seven new signals have been added to the 8085; it would have been possible to provide separate Data and Address Busses by eliminating these seven signals, plus the ALE control signal whose presence is a direct consequence of having multiplexed Data and Address Busses. Intel has chosen to provide the seven new signals, paying the price of having multiplexed Data and Address Busses.

Let us examine the new signals.

RST 5.5, RST 6.5, RST 7.5 and TRAP represent additional interrupt request inputs. TRAP is a non-maskable, high priority interrupt; the other three interrupt requests are supported by hardware-implemented vectoring.

RESET OUT is a Reset signal output by the 8085; it may be used to reset support devices around the 8085.

SID and SOD are control signals which provide a primitive serial input and output capability. These signals can also be used as a general purpose status input (SID) and a control output (SOD).

8085 TIMING AND INSTRUCTION EXECUTION

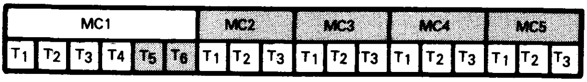
An 8085 instruction's execution is timed by a sequence of machine cycles, each of which is divided into clock periods.

An instruction is executed in from one to five machine cycles labeled MC1, MC2, MC3, MC4 and MC5.

8085 MACHINE CYCLES

The first machine cycle of any instruction's execution will have either four or six clock periods. Subsequent machine cycles will have three clock periods only. This may be illustrated as follows:

8085 CLOCK PERIODS

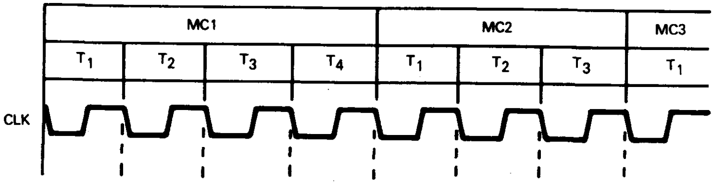


Where MC is shaded, the entire machine cycle is optional. When T is shaded, the clock period is optional within its machine cycle.

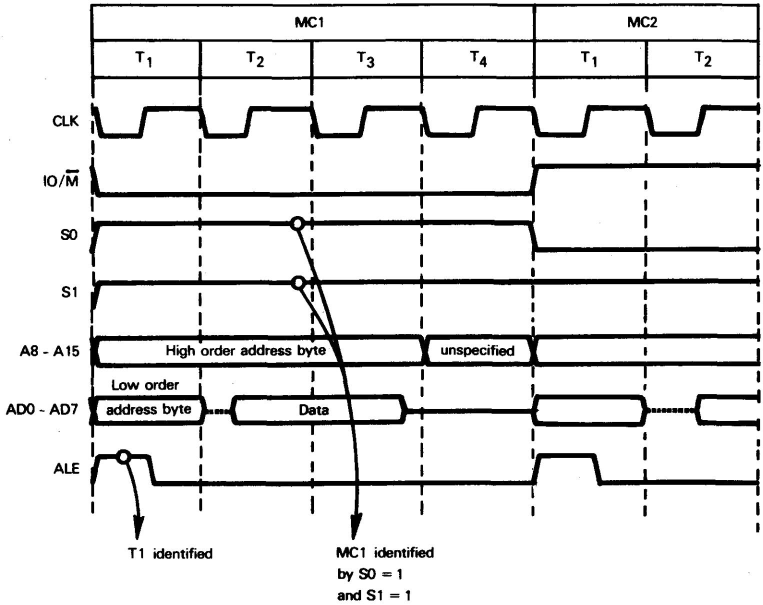
8085 machine cycles and clock periods are very similar to those of the 8080A. You will find in Table 5-1 that the number of clock periods required to execute 8085 instructions are equal to clock periods required by the 8080A to execute the same instruction, or differ by one clock period only.

THE CLOCK SIGNALS

The 8085 times its machine cycles using this simple clock signal:



Although the 8085 has no SYNC signal to identify the start of a new machine cycle, you can use the 8085 ALE signal for the same purpose. This signal is output true during the first clock period of every machine cycle — at which time the AD0 - AD7 lines are outputting address data. In addition you can identify the first (instruction fetch) cycle of any instruction's execution. S0 and S1 will both be output high during an instruction fetch machine cycle. Clock periods and machine cycles may therefore be identified as follows:



MEMORY ACCESS SEQUENCES

So far as external logic is concerned, there is very little difference between an instruction fetch, a memory read, and a memory write. We will therefore examine timing for these operations together.

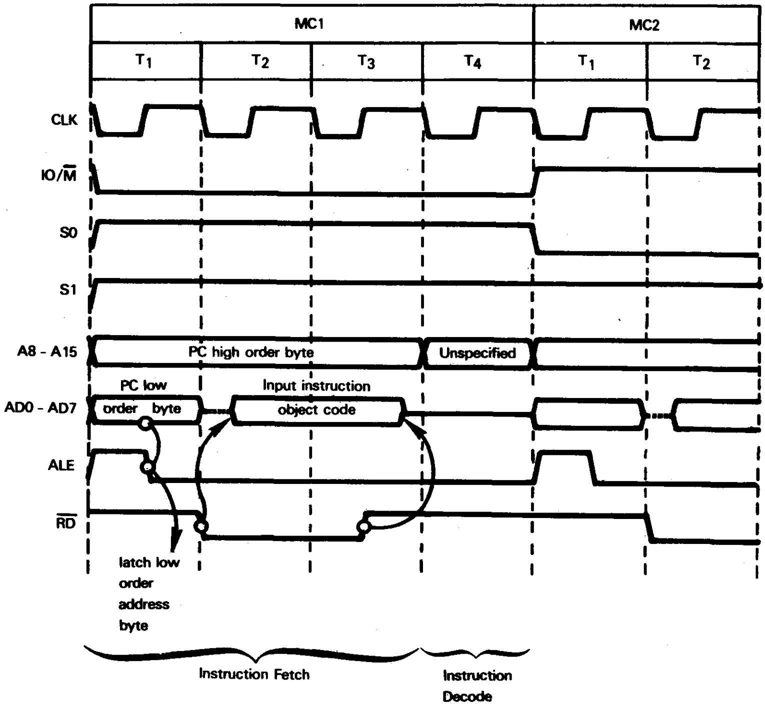


Figure 5-4. A Four Clock Period Instruction Fetch Machine Cycle

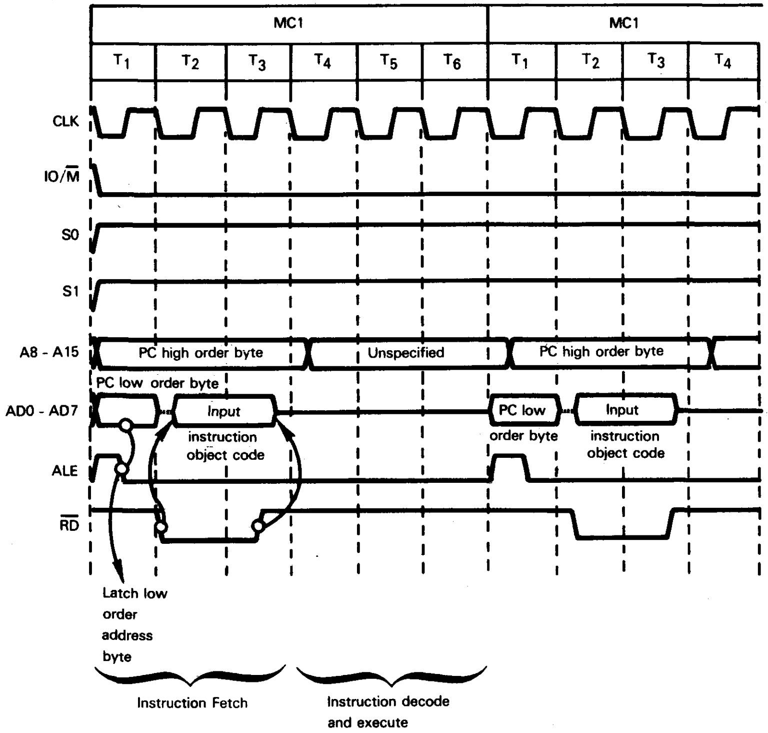


Figure 5-5. A Six Clock Period Instruction Fetch Machine Cycle

Let us first consider an instruction fetch. Timing is illustrated in Figure 5-4 for a four clock period machine cycle, and in Figure 5-5 for a six clock period machine cycle.

The most important aspect of the instruction fetch machine cycle is the fact that it will have either four or six clock periods, as against three for all subsequent machine cycles. The instruction fetch machine cycle must have at least four clock periods since the fourth clock period is needed to decode the instruction object code which has been fetched. If the instruction requires no subsequent memory accesses, then a fifth and sixth clock period may be needed to perform the internal operation specified by the fetched instruction. If additional memory accesses will be required, then the fourth clock period of the first machine cycle is sufficient.

At the end of the first clock period AD0 - AD7 is floated transiently; then it is turned around to act as a Data Input Bus. RD̄ is pulsed low to strobe data onto the Data Bus.

The memory read must occur within three clock periods. Since this is an instruction fetch machine cycle, the CPU will place the input in the Instruction register. If external logic requires more time to respond to the memory access, then it can generate additional Wait clock periods. We will describe the 8085 Wait state shortly.

During the fourth clock period of the instruction fetch machine cycle the instruction object code is interpreted by logic of the 8085 CPU. Fifth and sixth clock periods will be required by some instructions to execute required internal operations.

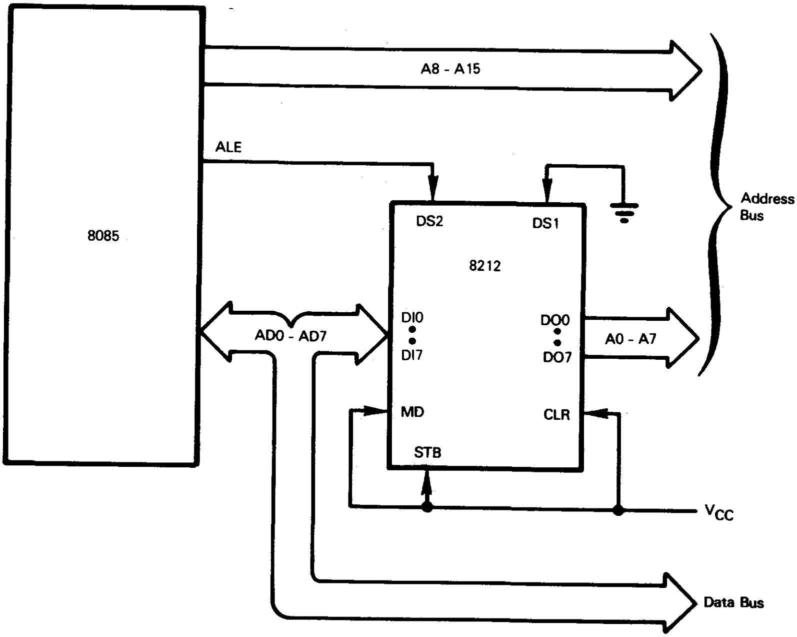
During the fourth and subsequent clock periods AD0 - AD7 is floated and A8 - A15 contains unspecified data.

8085 DEVICE SELECT LOGIC

The fact that AD0 - AD7 and A8 - A15 are unknown data during the fourth and subsequent clock periods of an instruction fetch machine cycle must be taken into account when you create memory select and I/O device select logic.

In Figures 5-4 and 5-5 S0 and S1 are both high, identifying this as an instruction fetch machine cycle. IO/$\overline{\text{M}}$ is low since the instruction object code is to be fetched from memory. An instruction fetch is thus equivalent to a memory read.

The address of the memory location to be accessed is fetched from the Program Counter (PC) and is output on AD0 - AD7 (low order byte) and A8 - A15 (high order byte). **The low order byte of this memory address is stable on AD0 - AD7 during the first clock period.** ALE is pulsed high at this time. **The trailing edge of ALE is designed to** act as a strobe signal which external logic can use to **latch the low order address byte** off AD0 - AD7. **If you are using one of the 8085 support devices (the 8155, the 8355 or the 8755), then the low order byte of the memory address is latched off the AD0 - AD7 lines for you. If you are using standard memory devices, then you must demultiplex AD0 - AD7. Any simple latched buffer can be used for this purpose; here is an example of the 8212 I/O port being used as a demultiplexer:**



MULTIPLE DEVICE SELECTS AND BUS LOADING

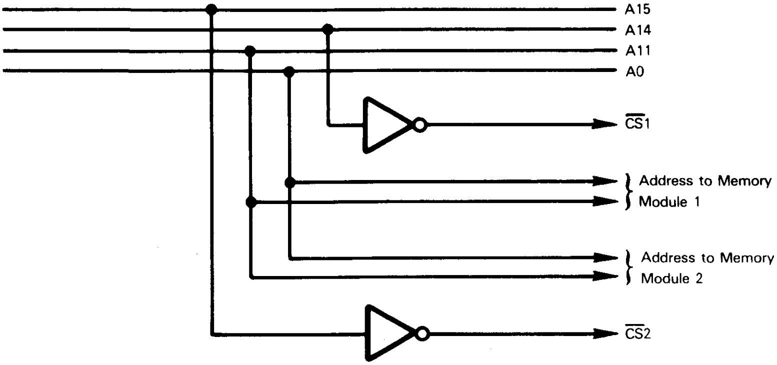
Now you might argue that there is no harm done if memory or I/O devices select themselves when the System Bus is supposed to be idle; if neither the read nor write strobe is present, data transfer between the System Bus and the selected device cannot occur.

Unfortunately, the problem is not so simple.

It is possible for more than one memory or I/O device to consider itself selected while the bus is idle; this may occur under the following conditions:

1) If I/O devices are being selected as I/O ports, then the Address Bus lines may select an I/O port while simultaneously selecting a memory device.

2) In microcomputer systems that use only a small portion of the total allowed memory — and most microcomputer systems fall into this category — memory select logic need not decode unique memory addresses. Here is an example of two 4096-byte memory modules, each of which uses a single line of the Address Bus in order to create device selects:



Memory module 1 will be assigned the address space 8000_{16} through $8FFF_{16}$. Memory module 2 will be assigned the address space 4000_{16} through $4FFF_{16}$. In reality a variety of other addresses will select memory modules 1 or 2. Addresses $C000_{16}$ through $CFFF_{16}$ will select memory modules 1 and 2.

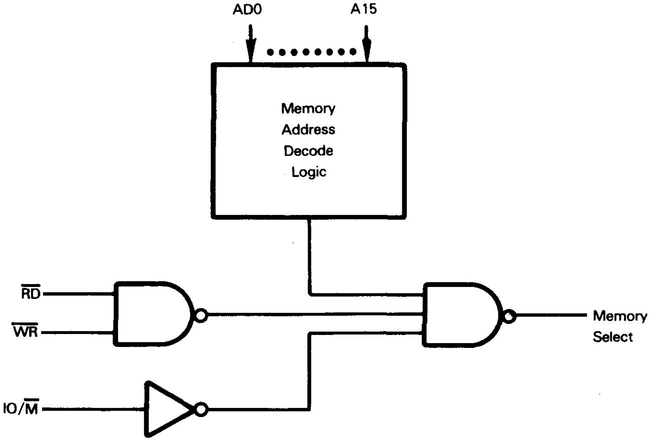
A correctly written program will keep either A15 or A14 low; but while the System Bus is floating, both address lines could be high — in which case both memory modules will become selected.

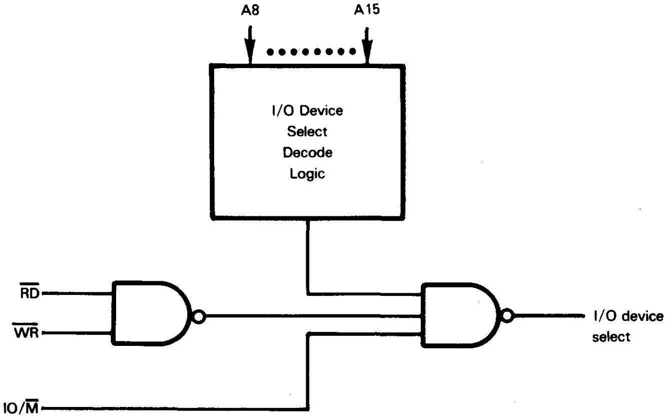
3) While signal levels on the Address Bus are changing state, memory and I/O devices may be transiently selected. Transient selection may occur during T1 as well as during T4, T5 and T6. Transient selection may leave more than one memory or I/O device simultaneously selected for short periods of time.

If more than one memory or I/O device is simultaneously selected, excessive loads may be placed on the System Bus. At best, these excessive loads will cause devices connected to the System Bus to temporarily malfunction; at worst, device failures may result.

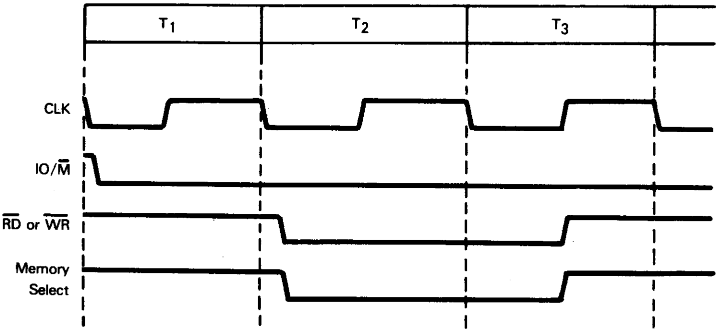
It is very important to prevent devices from being spuriously selected.

The simplest way of preventing memory and I/O device selection is to use IO/$\overline{M}$, $\overline{RD}$ and $\overline{WR}$ as contributors to device select logic:





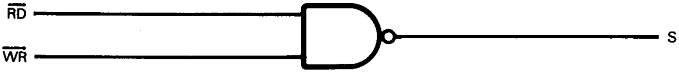
Timing for the memory select illustrated above may be illustrated as follows:



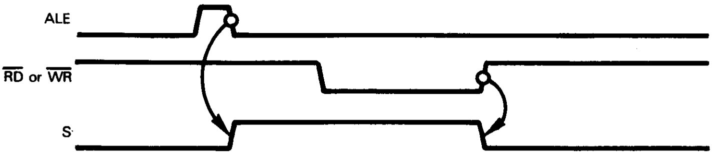
I/O device select logic timing differs only in the level of IO/$\overline{\text{M}}$.

IO/$\overline{\text{M}}$ distinguishes between memory and I/O devices. When $\overline{\text{RD}}$ or $\overline{\text{WR}}$ is low, memory or I/O device addresses must be valid. Thus the logic illustrated above will guarantee that spurious memory and I/O device selects never occur.

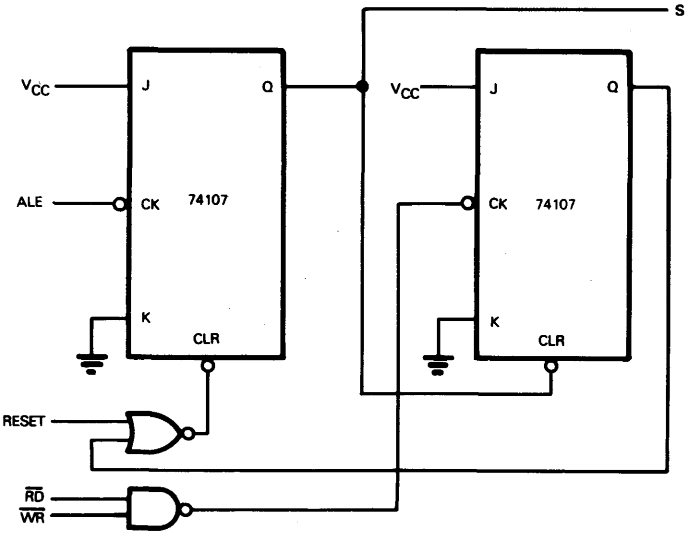
But there is a problem associated with the solution illustrated; memory and I/O devices do not receive a valid select signal until early in the second clock period. This is unfortunate since valid addresses are available early in the first clock period. Delaying memory select logic until the second clock period may require Wait states to be added between clock periods 2 and 3 — and that unnecessarily slows down CPU operations. If execution speed is not a problem to you, then the simple select logic illustrated above will do. **If execution speed is a problem, then you must replace:**



in the simple select logic with alternative logic that may be defined as follows:



The required S output may be generated using two flip-flops as follows:



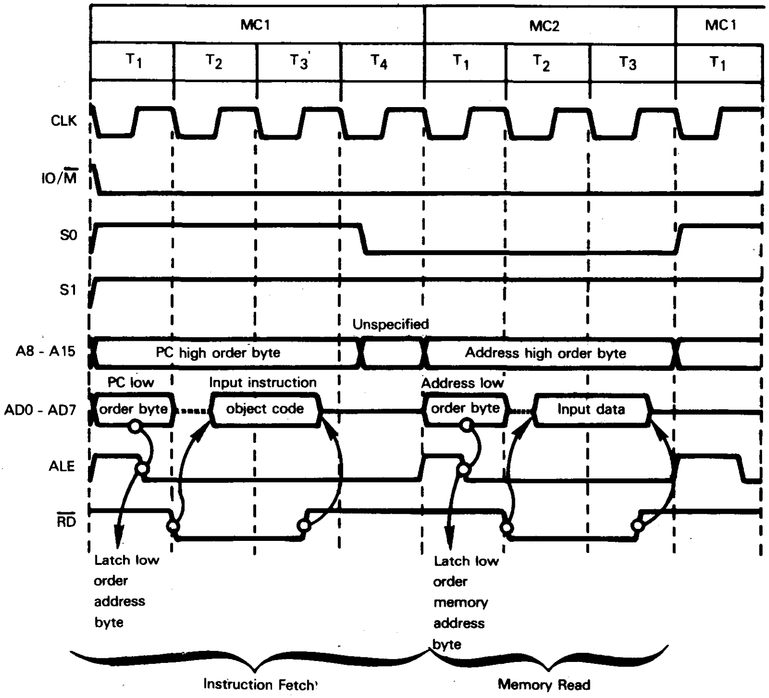


Figure 5-6. A Memory Read Machine Cycle Following An Instruction Fetch

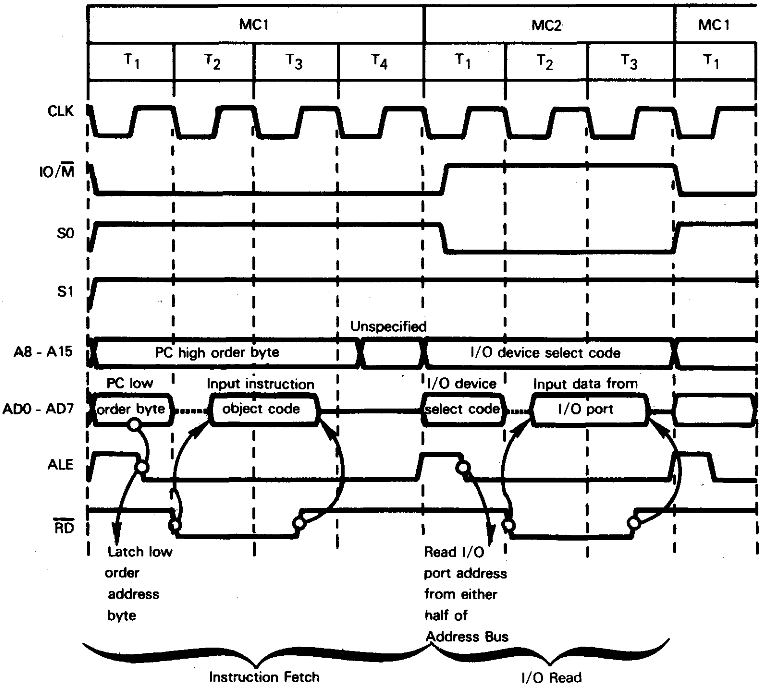


Figure 5-7. An I/O Read Machine Cycle Following An Instruction Fetch

8085 MEMORY READ TIMING

Let us now consider a memory read operation; timing is illustrated in Figure 5-6. So far as external logic is concerned, the only difference between a memory read and an instruction fetch is the S0 and S1 signal levels; they are both high for an instruction fetch but S0 is low during a memory read. Also, the instruction fetch has four or six clock periods while the memory read has three; but the extra instruction fetch clock periods occur after the memory access is completed. Therefore so far as external logic is concerned, the extra clock periods of the instruction fetch machine cycle are irrelevant.

8085 I/O READ TIMING

Figure 5-7 illustrates I/O read timing. Only the IO/M̄ signal level in Figure 5-7 differs from Figure 5-6.

8085 MEMORY WRITE TIMING

Memory write timing, illustrated in Figure 5-8, is very similar to memory read timing. The principal difference is that during a memory write W̄R̄ is output low whereas during a memory read R̄D̄ is output low. Also, during a memory write operation S1 is output low while S0 is output high.

8085 I/O WRITE TIMING

An I/O write operation is illustrated in Figure 5-9. As compared to Figure 5-8, IO/M̄ is high in Figure 5-9 during the write machine cycle; there are no other timing differences.

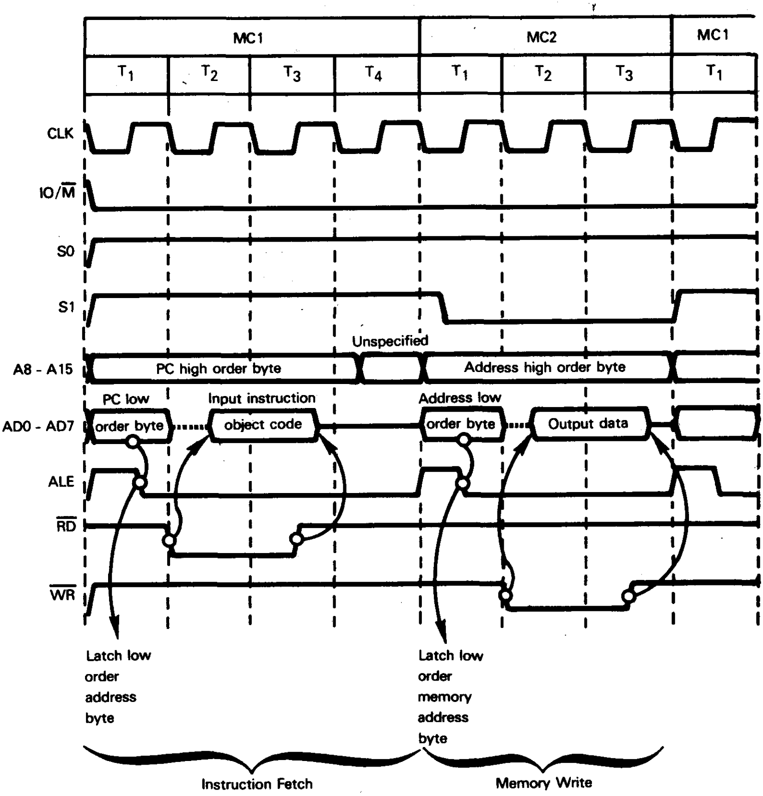


Figure 5-8. A Memory Write Machine Cycle Following An Instruction Fetch

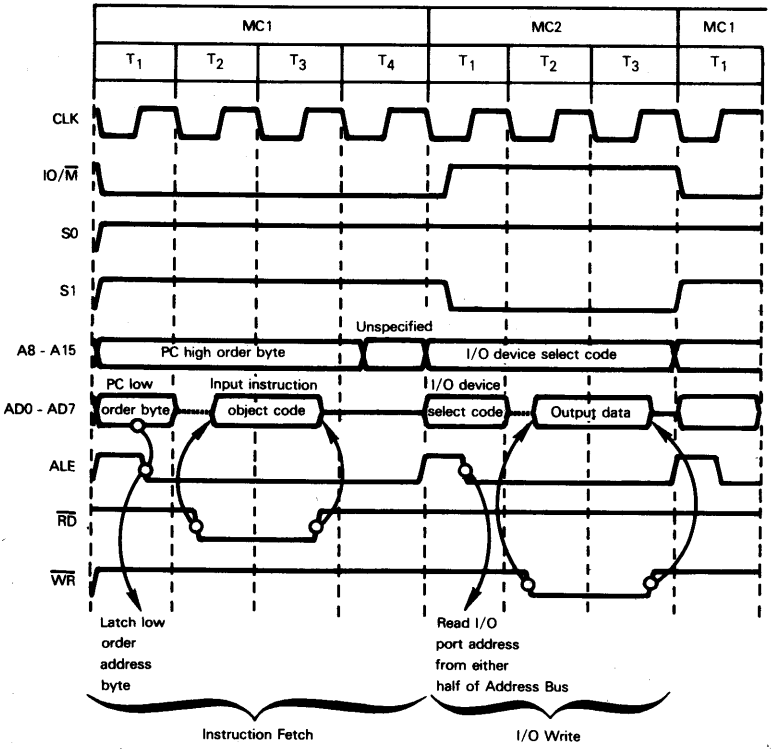


Figure 5-9. An I/O Write Machine Cycle Following An Instruction Fetch

BUS IDLE MACHINE CYCLES

8085 BUS IDLE MACHINE CYCLE

During a Bus Idle machine cycle no control signals change state on the System Bus.

There are three types of Bus Idle machine cycle:

1) An instruction fetch Bus Idle machine cycle. The TRAP, RST 5.5, RST 6.5 and RST 7.5 instructions are interrupt acknowledge instructions whose object codes are created by logic internal to the 8085 CPU chip. No external instruction fetch operations occur; however logic internal to the CPU requires time to create the instruction object code. Therefore a Bus Idle instruction fetch machine cycle is executed. Timing is illustrated in Figure 5-17.

2) The instruction execute Bus Idle machine cycle. Only the DAD instruction uses this machine cycle. The DAD instruction adds the contents of two CPU registers to two other CPU registers. It takes six clock periods for logic internal to the 8085 CPU to complete these operations. The six clock periods are generated via two instruction execute Bus Idle machine cycles. Timing is illustrated in Figure 5-10.

3) The Halt Bus Idle machine cycle. Following execution of a Halt instruction an indeterminate number of Bus Idle machine cycles are executed for the duration of the Halt condition. Timing is illustrated in Figure 5-14.

The condition of the IO/$\overline{M}$, S1 and S2 signals during a Bus Idle machine cycle varies with the type of Bus Idle machine cycle. These three signals will conform to instruction fetch level during an instruction fetch Bus Idle machine cycle. During an instruction execute Bus Idle machine cycle Memory Read signal levels are maintained, but the $\overline{RD}$ control signal is not pulsed low.

During a Halt Bus Idle machine cycle S0 and S1 are both low but IO/$\overline{M}$, along with other tristate signals, is floated.

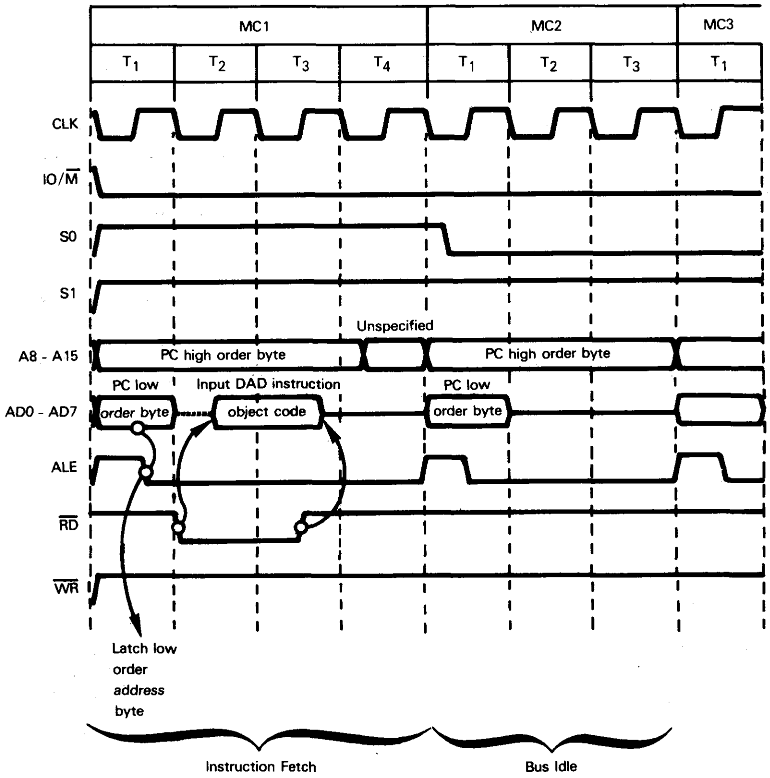


Figure 5-10. A Bus Idle Machine Cycle Following An Instruction Fetch During Execution Of A DAD Instruction

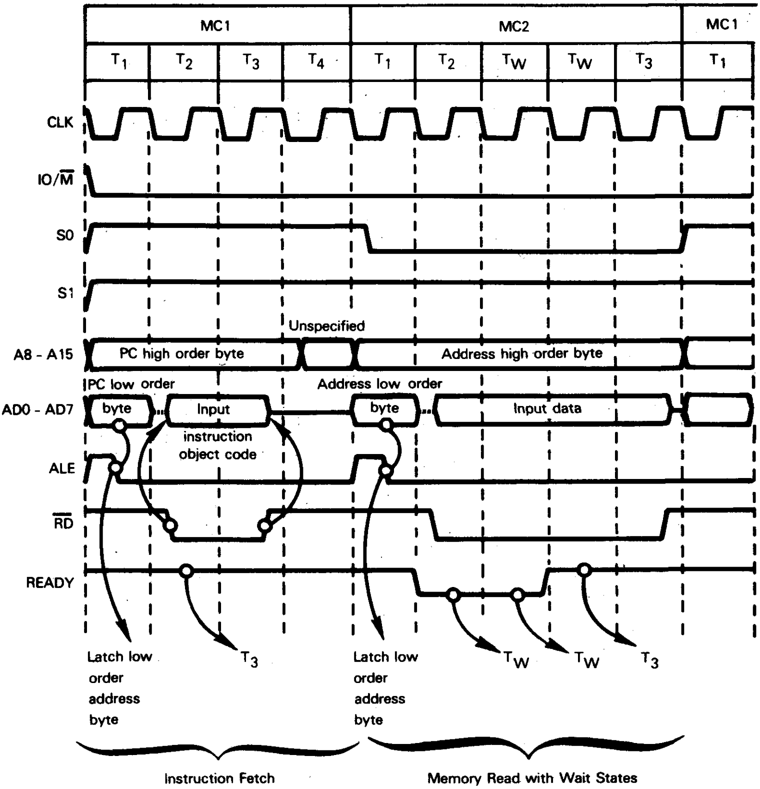
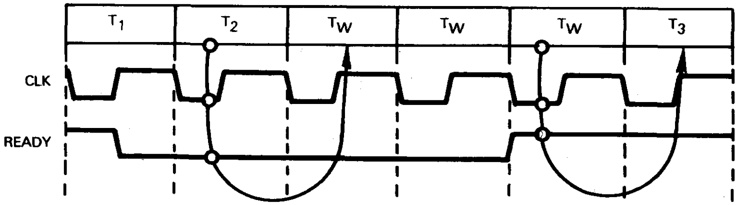


Figure 5-11. Wait States Occurring In A Memory Read Machine Cycle

THE WAIT STATE

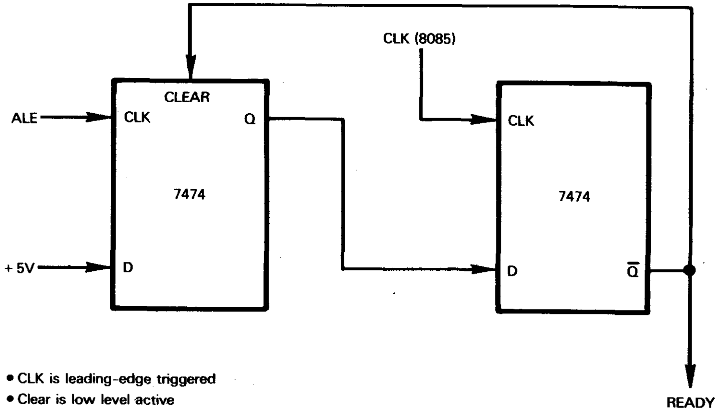
The 8085 will insert Wait states between clock periods T2 and T3 in a manner that is closely analogous to the 8080A. Timing is illustrated in Figure 5-11, which shows Wait states being inserted in a memory read cycle; a Wait state inserted in any other memory reference or I/O machine cycle would differ only in the levels of control signals.

The 8085 samples the READY line during T2. If READY is low during T2, then a Wait clock period will follow T2. The READY line is sampled in the middle of each Wait clock period; Wait clock periods continue to be inserted until READY is sampled high. As soon as READY is sampled high the next clock period will be a T3 clock period — and normal program execution continues. This sampling may be illustrated as follows:



Wait states are used in an 8085 system exactly as described for the 8080A in Chapter 4 — to give slow memories and I/O devices more time in order to respond to an access. Thus the discussion of Wait states provided in Chapter 4 applies equally to the 8085.

In Chapter 4 a pair of 7474 flip-flops are shown creating a low READY pulse that generates a single Wait state in a memory read machine cycle. For the 8085 the following variation applies:



- CLK is leading-edge triggered
- Clear is low level active

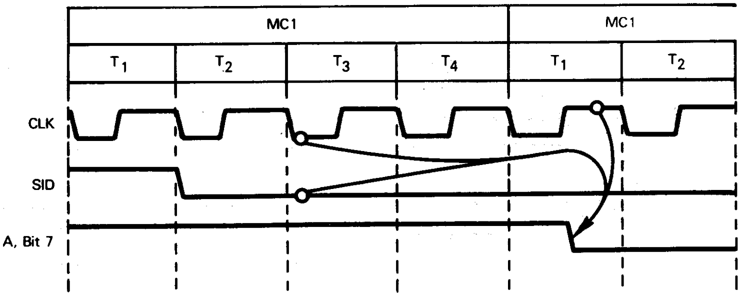
THE SID AND SOD SIGNALS

The 8085 has two instructions which handle single bit data.

The RIM instruction inputs data from the SID pin to the high order bit of the Accumulator. The SIM instruction outputs the high order bit of the Accumulator to the SOD pin.

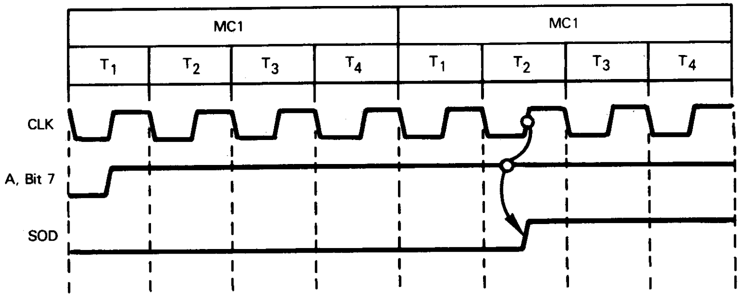
You may use the RIM and SIM instructions in order to implement a primitive serial I/O capability. A more useful application of these instructions is to read single signal status and to output single signal controls.

When the RIM instruction is executed, the SID signal level is sampled on the rising edge of the clock signal during clock period T3 of the instruction fetch machine cycle. The high order bit of the Accumulator is modified while the clock signal is high during T1 of the next instruction fetch machine cycle. Timing may be illustrated as follows:



When an SIM instruction is executed, the actual change in SOD signal level does not occur until T2 of the next instruction fetch machine cycle; that is to say execution of the SIM instruction overlaps with the next instruction fetch.

This may be illustrated as follows:



Following an SIM instruction fetch, the high order bit of the Accumulator is sampled while the clock is low during T1 of the next instruction fetch machine cycle. During the same clock period, the SOD signal level is modified to reflect the contents of the high order Accumulator bit. This overlap is feasible since neither the SOD signal nor the Accumulator contents is modified while an instruction is being fetched. Note that SOD must be enabled before it can be accessed or changed.

Figure 5-12 illustrates SID and SOD signal timing during execution of a RIM instruction followed by a SIM instruction.

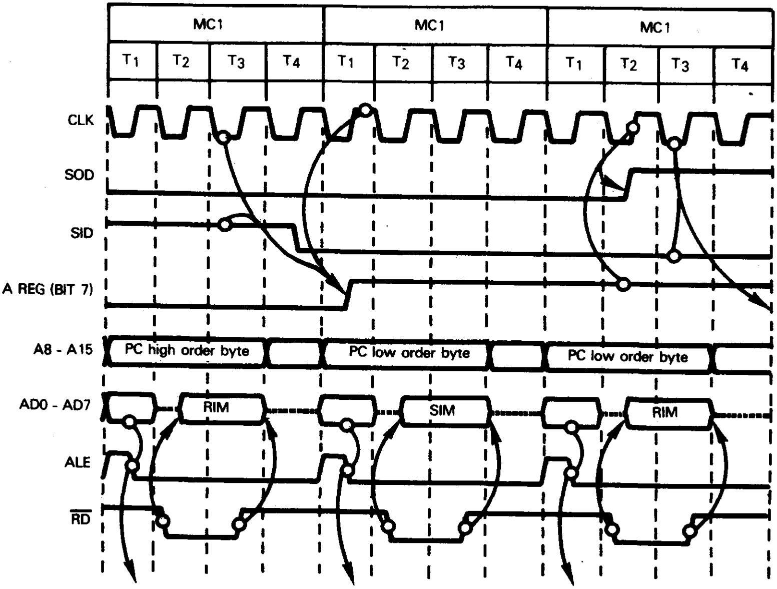


Figure 5-12. A RIM Instruction Followed By A SIM Instruction

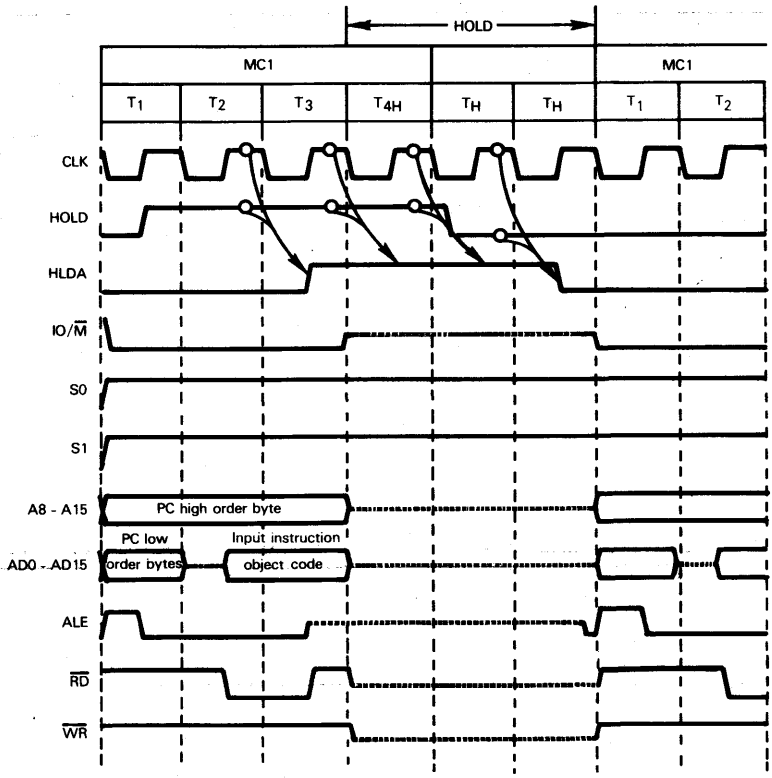


Figure 5-13. A Hold State Following A Single Machine Cycle Instruction Execution

THE HOLD STATE

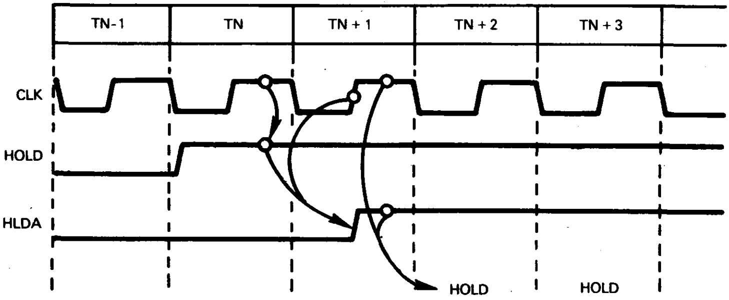
The 8080A and the 8085 both use the Hold state as a means of transiently floating the System Bus. During a Hold external logic gains bus control, usually to perform direct memory access operations.

External logic requests a Hold state by inputting HOLD high. The microprocessor responds by entering the Hold state and outputting HLDA high. During a Hold state the microprocessor floats all tristate signals.

Both microprocessors initiate the Hold state at the conclusion of an instruction's execution. But there are significant differences between Hold state initiation logic for the 8085 as against the 8080A.

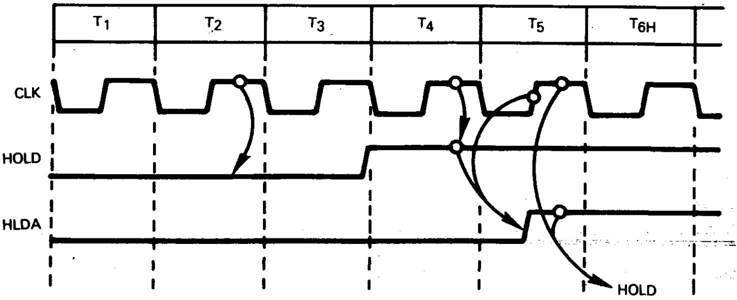
The 8080A initiates a Hold state following T3 for a Read machine cycle, or following T4 for a Write machine cycle. Timing is illustrated in Figures 4-9 and 4-10.

The 8085, in contrast, **has a fixed, two machine cycle sequence for Hold state initiation;** it may be illustrated as follows:



During every machine cycle Hold is sampled during T2; if Hold is high at this time, Hold acknowledge is output high during T3 and the Hold state begins during T4. Timing is illustrated in Figure 5-13.

During a six clock period machine cycle, if Hold is low when sampled during T2, then Hold will be sampled again during T4. If Hold is sampled high during T4, then a Hold state will be initiated during T6. This may be illustrated as follows:



Hold is sampled during every clock period of a Halt state. As soon as Hold is detected high, a two clock period Hold state initiation sequence begins. Figures 5-14 and 5-15 illustrate the onset of Hold states within and before Halt states.

A Hold state terminates two clock periods after the Hold signal goes low.

There are no restrictions placed by 8085 logic on the duration of a Hold state. The Hold state lasts for as long as the HOLD input is high. Here is an example of a one clock period Hold state occurring during T4 and a three clock period Hold state beginning during T6 of a six clock period machine cycle:

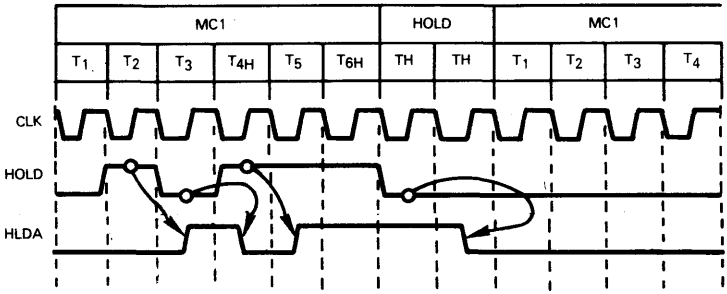


Figure 5-13 illustrates a Hold state lasting three clock periods, beginning during T4 of a four clock period machine cycle.

THE HALT STATE AND INSTRUCTION

When a Halt instruction is executed, the 8085 enters a Halt state. The Halt state consists of an indeterminate number of Halt Bus Idle clock periods during which the S1 and S0 status signals are both output low while the tristate signals are floated.

Halt state timing is illustrated in Figure 5-14.

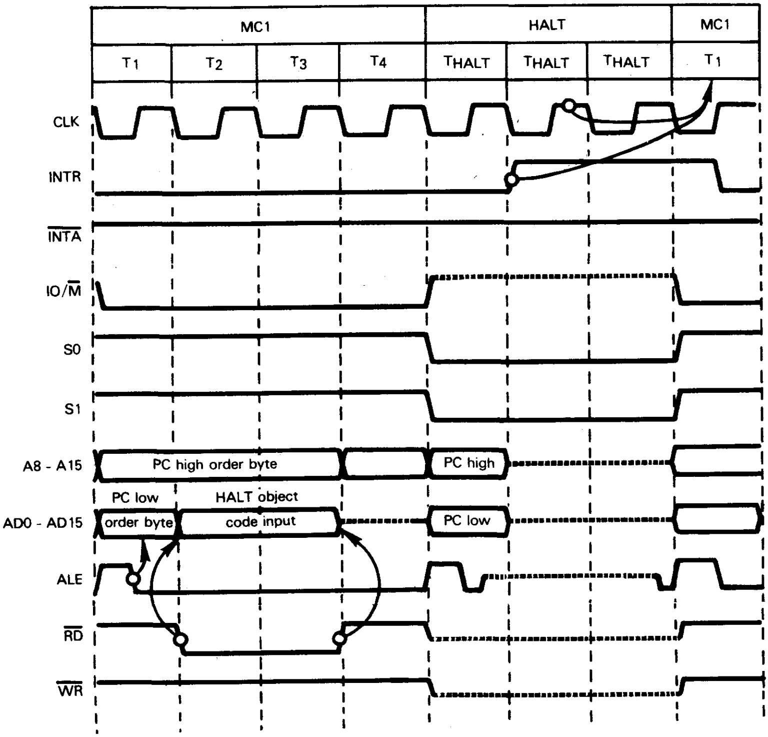


Figure 5-14. A Halt Instruction And A Halt State Terminated By An Interrupt Request

A Halt state may be terminated by a system reset or by an interrupt request. Figure 5-14 shows an interrupt request terminating the Halt state.

Note that the INTR signal, like the HOLD signal, is sampled two clock periods before anything can happen. Thus, as illustrated in Figure 5-14, an additional Halt clock period will occur after the clock period within which INTR goes high.

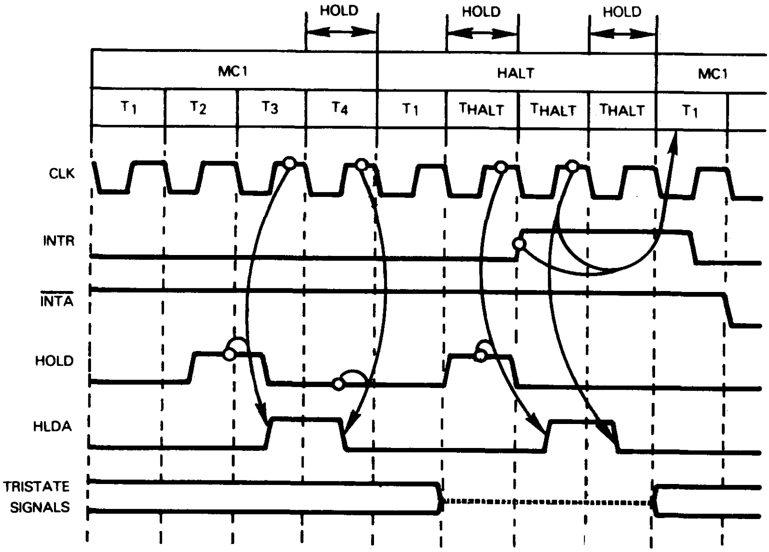


Figure 5-15. Hold States Occurring Within A Halt State

An interrupt request will only be executed if interrupts are enabled; however the 8085 has a TRAP non-maskable interrupt. Thus you can always exit an 8085 Halt state via a TRAP interrupt request, or by resetting the system.

8085 HOLD WITHIN A HALT STATE

While in a Halt state you can enter and exit the Hold state. Figure 5-15 illustrates timing for the Hold state existing within the Halt state. Notice that the Hold state only lasts for as long as the HOLD input is kept high.

Entering a Hold state within a Halt state also prevents you from terminating the 8085 Halt state with an interrupt request; this is because a HOLD request has priority over any interrupt request. Thus if an interrupt request occurs while the 8085 is entering a Hold state, or is in a Hold state, the interrupt request will be ignored until the end of the Hold state. At that time, the interrupt request will be acknowledged — providing interrupts are enabled.

Resetting the 8085 will terminate a Halt state at any time, whether or not you are in a Hold state.

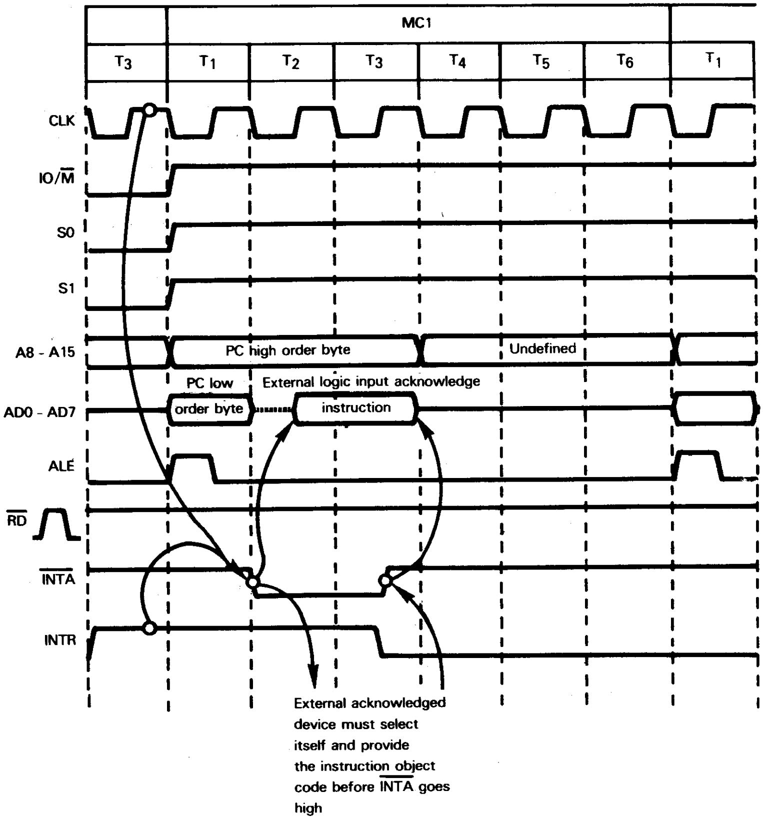

MC1
T3
T1
T2
T3
T4
T5
T6
T1
CLK
IO/M̄
S0
S1
A8 - A15
PC high order byte
Undefined
PC low order byte
External logic input acknowledge instruction
AD0 - AD7
ALE
RD̄
INTA
INTR
External acknowledged device must select itself and provide the instruction object code before INTA goes high

Figure 5-16. An Interrupt Being Acknowledged Using A Single Byte Instruction

EXTERNAL INTERRUPTS

There are some differences between the interrupt acknowledge logic of the 8085 as compared with the 8080A; however the 8080A interrupt acknowledge logic is a subset of 8085 capabilities.

Providing a valid interrupt request has been applied and interrupts are enabled, **the 8085 acknowledges the interrupt request on terminating execution of the current instruction.** The 8085 then executes an interrupt acknowledge machine cycle.

An interrupt acknowledge machine cycle is very similar to a six clock period instruction fetch machine cycle; however during the interrupt acknowledge machine cycle the 8085, like the 8080A, anticipates receiving an instruction object code from an I/O device — presumably the device whose interrupt request is being acknowledged. Since an I/O device is supposed to provide the object code during an interrupt acknowledge instruction fetch, $\overline{\text{INTA}}$ is pulsed low instead of $\overline{\text{RD}}$. Also, IO/$\overline{\text{M}}$ is low. Timing is illustrated in Figure 5-16.

Note that even though memory is not being accessed, Program Counter contents are output on the Address Bus during an interrupt acknowledge instruction fetch; providing memory select logic uses IO/$\overline{\text{M}}$ and $\overline{\text{RD}}$, no harm will be done by having

a valid address on the Address Bus during an interrupt acknowledge instruction fetch.

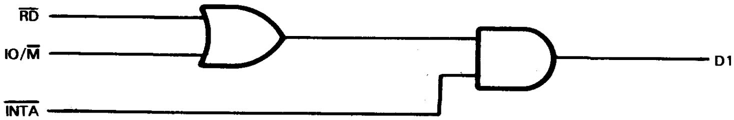
The Program Counter contents are not incremented during the interrupt acknowledge process.

Note that the interrupt acknowledge signal INTA serves both as an interrupt acknowledge and a read strobe. External logic must use INTA both as a device select signal and a strobe signal identifying the time interval during which the interrupt acknowledge instruction code must be placed on the Data Bus. This can cause a timing problem. For any other instruction fetch, the trailing edge of ALE can be used to initiate device select timing; thus during any other instruction fetch you have from the middle of T1 until the middle of T2 to resolve the device select and wait for the read strobe. But you cannot use ALE in this fashion following an interrupt acknowledge, since external logic does not know that the interrupt has been acknowledged until INTA goes low. On the trailing edge of ALE during an interrupt acknowledge instruction fetch machine cycle, the Program Counter contents are being output on the Address Bus even though this address is irrelevant. **You must** therefore **use INTA as a signal which disables all I/O device select logic with the exception of the device whose interrupt request is being acknowledged.**

You may well have to insert Wait states during an interrupt acknowledge instruction fetch machine cycle; the acknowledged external logic has the duration of the low INTA pulse within which it must resolve its select logic and place an instruction object code on the Data Bus.

You can use the low INTA pulse to create a low READY input and thus insert Wait states between clock periods T2 and T3. Using a 320 nanosecond clock, you have 50 nanoseconds within which to generate a low READY input from the low INTA output — and that is sufficient time.

Earlier in this chapter we showed you how you can create a one clock period low READY pulse using two 7474 D-type flip-flops. The OR of RD and IO/M creates the D input to the first flip-flop. You can use these same two flip-flops to create a low READY pulse following an interrupt acknowledge by creating the first D input as follows:



You can respond to an interrupt acknowledge by transmitting any instruction object code to the 8085. Usually a Restart (RST) or a Call instruction object code will be transmitted.

Figure 5-16 illustrates timing for a Restart instruction being transmitted following an interrupt acknowledge. The Restart instruction has been described in detail in Chapter 4 together with circuits which allow a Restart instruction to be created.

The 8085 contains internal logic to cope with multibyte instruction object codes transmitted during the interrupt acknowledge process. During the second and third instruction fetch machine cycles INTA is pulsed low while IO/M is output high. Thus responding to an interrupt acknowledge with a Call instruction simply involves creating a Call instruction's object code.

The 8085 has four interrupt request pins which the 8080A does not have. These are TRAP, RST 5.5, RST 6.5 and RST 7.5. Interrupts requested via these pins cause the 8085 to generate its own internal interrupt acknowledge instruction.

The internal interrupt acknowledge instruction results in subroutine calls to the following addresses:

Interrupt	CALL Address
TRAP	24_{16}
RST 5.5	$2C_{16}$
RST 6.5	34_{16}
RST 7.5	$3C_{16}$

TRAP is a non-maskable interrupt.

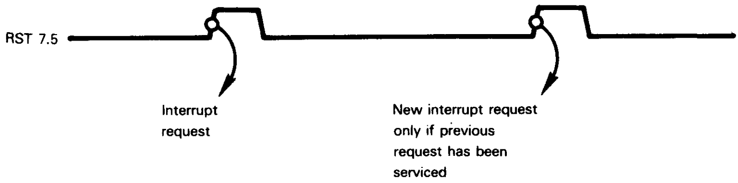
RST 5.5 and RST 6.5 are level sensitive; that means a high level input at these pins generates an interrupt request.

RST 7.5 is edge sensitive; an interrupt request occurs when the input to RST 7.5 makes a low-to-high transition.

TRAP is both level and edge sensitive; the low-to-high transition and the subsequent high level generate an interrupt request.

If an interrupt request is generated at RST 7.5 by a low-to-high transition, the 8085 will remember the interrupt request, whether or not the RST 7.5 input remains high. **You can thus generate an interrupt request via RST 7.5 using a high pulse.**

Since you can request an interrupt via an RST 7.5 low-to-high transition, the RST 7.5 interrupt request signal itself cannot reset the interrupt request. This may be illustrated as follows:


RST 7.5
Interrupt request
New interrupt request only if previous request has been serviced

You need not terminate service of an RST 7.5 interrupt request by executing an SIM instruction with bit 4 of the Accumulator set to 1; this is done automatically.

A low-to-high transition of the TRAP input creates an interrupt request. The interrupt request will only be acknowledged while the TRAP input remains high; however, once a TRAP interrupt request has been acknowledged, **TRAP must go low and then high again before another interrupt request will be acknowledged.**

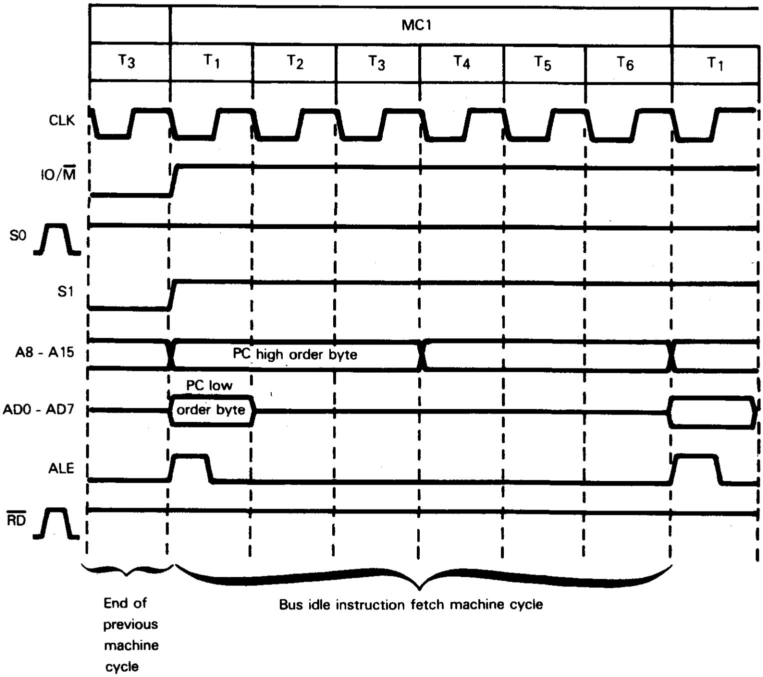


Figure 5-17. A Bus Idle Instruction Fetch Machine Cycle

8085 interrupt priorities are as follows:

Highest	HOLD
	TRAP
	RST 7.5
	RST 6.5
	RST 5.5
Lowest	INTR

The 8085 executes an instruction fetch Bus Idle machine cycle after acknowledging a TRAP, RST 5.5, RST 6.5 or RST 7.5 interrupt request. Timing is given in Figure 5-17.

The TRAP interrupt request cannot be disabled.

The RST 5.5, RST 6.5 and RST 7.5 interrupt requests can be individually enabled and disabled using the SIM instruction. All interrupts bar the TRAP can be enabled and disabled via the EI and DI instructions.

You may at any time examine interrupt enable/disable status by executing the RIM instruction.

The RIM and SIM instructions are described in detail later in this chapter.

You will service interrupts in an 8085 system exactly as described for the 8080A system. For a discussion of an interrupt acknowledge see Chapter 4.

Remember that a HOLD request has priority over an interrupt request. Thus, an interrupt will not be acknowledged while a Hold state exists and the 8085 will respond to a HOLD request following an interrupt acknowledge.

THE RESET OPERATION

You reset an 8085 by inputting a low signal via $\overline{\text{RESET IN}}$.

When power is first turned on, the $\overline{\text{RESET IN}}$ pulse must last at least 500 nanoseconds (3 full clock cycles); no further requirements are imposed on the $\overline{\text{RESET IN}}$ signal. Logic internal to the 8085 will synchronize the $\overline{\text{RESET IN}}$ pulse with the internal clock. Timing for a Reset following a power up is given in Figure 5-18.

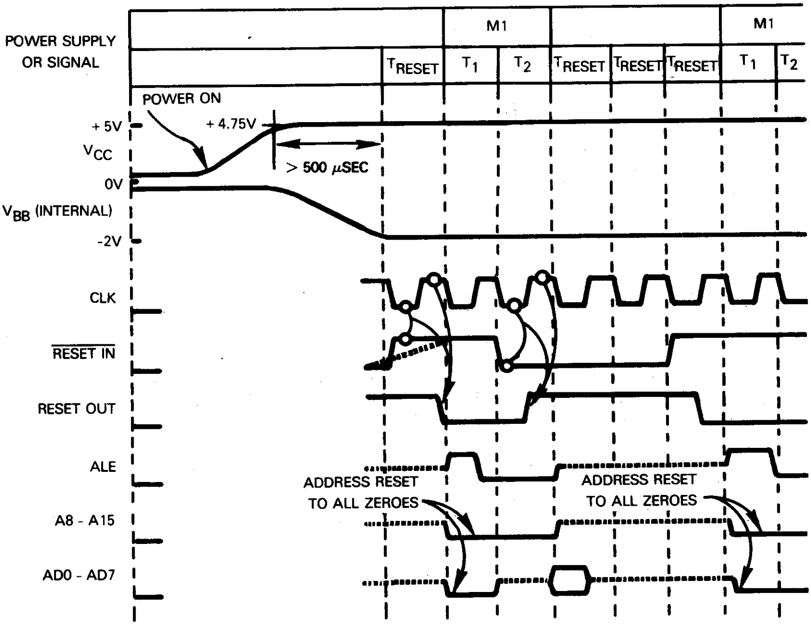


Figure 5-18. Power On And $\overline{\text{RESET IN}}$ Timing For The 8085

Notice that a RESET OUT signal is provided. You can use this signal to reset other devices in the 8085 microcomputer system.

When the 8085 is reset the following events occur:

1) The Program Counter is cleared; thus the first instruction executed following a reset must have its object code stored in memory location 0.
2) The Instruction register is cleared.
3) Interrupts are disabled.
4) The RST 7.5, RST 6.5 and RST 5.5 interrupts are masked out and thus disabled.
5) All tristate bus lines are floated.

Table 5-1. A Summary Of 8085 Instruction Object Codes And Execution Cycles

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS		MACHINE CYCLES
				8080A	8085	
ACI	DATA	CE YY	2	7	7	1 3
ADC	REG	10001XXX	1	4	4	1
ADC	M	8E	1	7	7	1 3
ADD	REG	10000XXX	1	4	4	1
ADD	M	86	1	7	7	1 3
ADI	DATA	C6 YY	2	7	7	1 3
ANA	REG	10100XXX	1	4	4	1
ANA	M	A6	1	7	7	1 3
ANI	DATA	E6 YY	2	7	7	1 3
CALL	LABEL	CD ppqq	3	17	18	2 3 3 5 5
CC	LABEL	DC ppqq	3	11/17	9/18	2 3, 2 3 3 5 5
CM	LABEL	FC ppqq	3	11/17	9/18	2 3, 2 3 3 5 5
CMA		2F	1	4	4	1
CMC		3F	1	4	4	1
CMP	REG	10111XXX	1	4	4	1
CMP	M	BE	1	7	7	1 3
CNC	LABEL	D4 ppqq	3	11/17	9/18	2 3, 2 3 3 5 5
CNZ	LABEL	C4 ppqq	3	11/17	9/18	2 3, 2 3 3 5 5
CP	LABEL	F4 ppqq	3	11/17	9/18	2 3, 2 3 3 5 5
CPE	LABEL	EC ppqq	3	11/17	9/18	2 3, 2 3 3 5 5
CPI	DATA	FE YY	2	7	7	1 3
CPO	LABEL	E4 ppqq	3	11/17	9/18	2 3, 2 3 3 5 5
CZ	LABEL	CC ppqq	3	11/17	9/18	2 3, 2 3 3 5 5
DAA		27	1	4	4	1
DAD	RP	00XX1001	1	10	10	1 7 7
DCR	REG	00XXX101	1	5	4	1
DCR	M	35	1	10	10	1 3 5
DCX	RP	00XX1011	1	5	6	2
DI		F3	1	4	4	1
EI		FB	1	4	4	1
HLT		76	1	4	4	1
IN	PORT	DB YY	2	10	10	1 3 4
INR	REG	00XXX100	1	5	4	1
INR	M	34	1	10	10	1 3 5
INX	RP	00XX0011	1	5	6	2
JC	LABEL	DA ppqq	3	10	7/10	1 3, 1 3 3
JM	LABEL	FA ppqq	3	10	7/10	1 3, 1 3 3
JMP	LABEL	C3 ppqq	3	10	10	1 3 3
JNC	LABEL	D2 ppqq	3	10	7/10	1 3, 1 3 3
JNZ	LABEL	C2 ppqq	3	10	7/10	1 3, 1 3 3
JP	LABEL	F2 ppqq	3	10	7/10	1 3, 1 3 3
JPE	LABEL	EA ppqq	3	10	7/10	1 3, 1 3 3
JPO	LABEL	E2 ppqq	3	10	7/10	1 3, 1 3 3
JZ	LABEL	CA ppqq	3	10	7/10	1 3, 1 3 3
LDA	ADDR	3A ppqq	3	13	13	1 3 3 3
LDAX	RP	000X1010	1	7	7	1 3
LHLD	ADDR	2A ppqq	3	16	16	1 3 3 3 3

Machine cycle types:

1 - Four clock period instruction fetch (Figure 5-4)
2 - Six clock period instruction fetch (Figure 5-5)
3 - Memory read (Figure 5-6)
4 - I/O read (Figure 5-7)
5 - Memory write (Figure 5-8)
6 - I/O write (Figure 5-9)
7 - Bus idle (Figure 5-10)

Table 5-1. A Summary Of 8085 Instruction Object Codes And Execution Cycles (Continued)

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS		MACHINE CYCLES
				8080A	8085	
LXI	RP,DATA16	00XX0001 YYYY	3	10	10	1 3 3
MOV	REG,REG	01dddsss	1	5	4	1
MOV	M,REG	01110sss	1	7	7	1 5
MOV	REG,M	01ddd110	1	7	7	1 3
MVI	REG,DATA	00ddd110 YY	2	7	7	1 3
MVI	M,DATA	36 YY	2	10	10	1 3 5
NOP		00	1	4	4	1
ORA	REG	10110XXX	1	5	4	1
ORA	M	B6	1	7	7	1 3
ORI	DATA	F6 YY	2	7	7	1 3
OUT	PORT	D3 YY	2	10	10	1 3 6
PCHL		E9	1	5	6	2
POP	RP	11XX0001	1	10	10	1 3 3
PUSH	RP	11XX0101	1	11	12	2 5 5
RAL		17	1	4	4	1
RAR		1F	1	4	4	1
RC		D8	1	5/11	6/12	2, 2 3 3
RET		C9	1	10	10	1 3 3
RIM		20	1		4	1
RLC		07	1	4	4	1
RM		F8	1	5/11	6/12	2, 2 3 3
RNC		D0	1	5/11	6/12	2, 2 3 3
RNZ		C0	1	5/11	6/12	2, 2 3 3
RP		F0	1	5/11	6/12	2, 2 3 3
RPE		E8	1	5/11	6/12	2, 2 3 3
RPO		E0	1	5/11	6/12	2, 2 3 3
RCC		0F	1	4	4	1
RST	N	11XXX111	1	11	12	2 3 3
RZ		C8	1	5/11	6/12	2, 2 3 3
SBB	REG	10011XXX	1	4	4	1
SBB	M	9E	1	7	7	1 3
SBI	DATA	DE YY	2	7	7	1 3
SHLD	ADDR	22 ppqq	3	16	16	1 3 3 5 5
SIM		30	1		4	1
SPHL		F9	1	5	6	2
STA	ADDR	32 ppqq	3	13	13	1 3 3 5
STAX	RP	000X0010	1	7	7	1 5
STC		37	1	4	4	1
SUB	REG	10010XXX	1	4	4	1
SUB	M	96	1	7	7	1 3
SUI	DATA	D6 YY	2	7	7	1 3
XCHG		EB	1	4	4	1
XRA	REG	10101XXX	1	4	4	1
XRA	M	AE	1	7	7	1 3
XRI	DATA	EE YY	2	7	7	1 3
XTHL		E3	1	18	16	1 3 3 5 5

ppqq represents four hexadecimal digit memory address
YY represents two hexadecimal data digits
YYYY represents four hexadecimal data digits
X represents an optional binary digit
ddd represents optional binary digits identifying a destination register
sss represents optional binary digits identifying a source register

THE 8085 INSTRUCTION SET

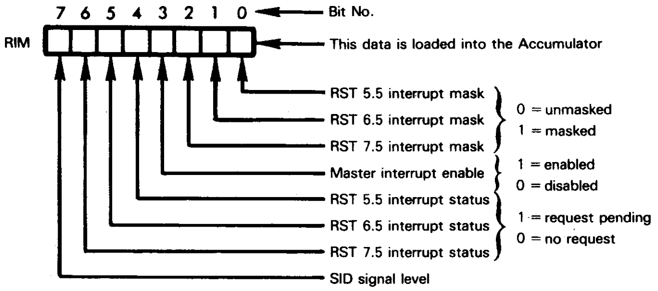
There are just three differences between the 8085 and the 8080A instruction sets:

1) The 8085 has two additional instructions — RIM and SIM.
2) The number of clock periods required to execute instructions differs in some cases; Table 5-1 summarizes these differences.
3) Following a Halt instruction's execution, the 8085 floats tristate bus lines in the ensuing Halt state; the 8080A does not.

Because the 8085 and 8080A instruction sets are so similar, the same benchmark program applies to both microprocessors. Refer to Chapter 4 for a discussion of this benchmark program.

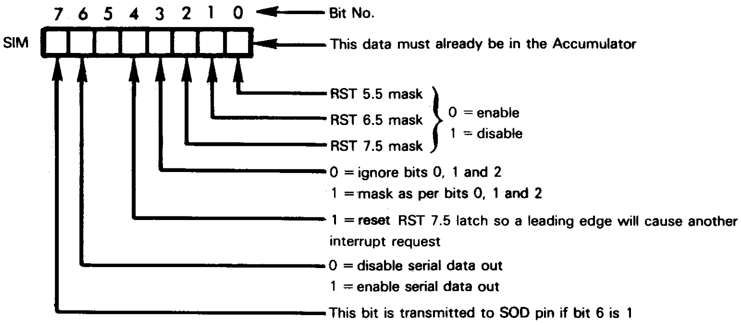
Refer to Table 4-4 for a summary of the 8085 instruction set. The only two 8085 instructions not present in Table 4-4 are the RIM and SIM instructions.

When the RIM instruction is executed, the following data is loaded into the Accumulator:



Thus, the RIM instruction allows you to examine interrupt and external status.

When the SIM instruction is executed the contents of the Accumulator are interpreted as follows:

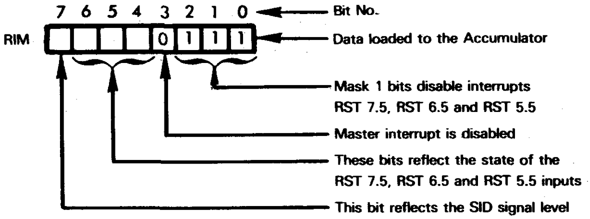


7 6 5 4 3 2 1 0
Bit No.
SIM
This data must already be in the Accumulator
RST 5.5 mask
RST 6.5 mask
RST 7.5 mask
0 = enable
1 = disable
0 = ignore bits 0, 1 and 2
1 = mask as per bits 0, 1 and 2
1 = reset RST 7.5 latch so a leading edge will cause another interrupt request
0 = disable serial data out
1 = enable serial data out
This bit is transmitted to SOD pin if bit 6 is 1

Thus the SIM instruction is used to selectively mask interrupts and to output a control signal via the SOD pin.

Note that if bit 6 of the Accumulator is 0 when the SIM instruction is executed, then the contents of bit 7 will not be transferred to the SOD pin.

From our discussion of the 8085 reset, recall that following a reset RST 5.5, RST 6.5 and RST 7.5 are all disabled; also, reset sets the SOD output to 0. Thus, following a reset an RIM instruction would input the following data to the Accumulator:


7 6 5 4 3 2 1 0
Bit No.
RIM
0 1 1 1
Data loaded to the Accumulator
Mask 1 bits disable interrupts RST 7.5, RST 6.5 and RST 5.5
Master interrupt is disabled
These bits reflect the state of the RST 7.5, RST 6.5 and RST 5.5 inputs
This bit reflects the SID signal level

8085 MICROPROCESSOR SUPPORT DEVICES

The 8085 has three special purpose multifunction support devices; they are described in this chapter.

Providing you demultiplex the Address and Data busses of the 8085, you can use any of the 8080A support devices described in Chapter 4 with the 8085.

THE 8155/8156 STATIC READ/WRITE MEMORY WITH I/O PORTS AND TIMER

The 8155 is a custom circuit designed specifically for the 8085 microprocessor. This device provides 256 bytes of static read/write memory, two or three parallel I/O ports and a programmable timer.

Figure 5-19 illustrates that part of general microcomputer system logic which has been implemented on the 8155 device.

Figure 5-20 provides a functional diagram of 8155 logic.

The 8155 device is packaged as a 40-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL compatible.

8155 DEVICE PINS AND SIGNALS

8155 pins and signals are illustrated in Figure 5-21. Signals may be divided into the following categories:

1) CPU interface and control
2) Parallel I/O
3) Programmable Timer

We will first consider CPU interface and control signals.

AD0 - AD7 connect to a bidirectional, multiplexed Data and Address Bus. As illustrated in Figure 5-22, these pins connect to the AD0 - AD7 bus lines output by the 8085 microprocessor.

ALE is the Address Latch Enable control signal output by the 8085 microprocessor to identify addresses on the multiplexed Data and Address Bus.

The 8155 device has both a memory space and an I/O address space. **When IO/$\overline{\text{M}}$ is**

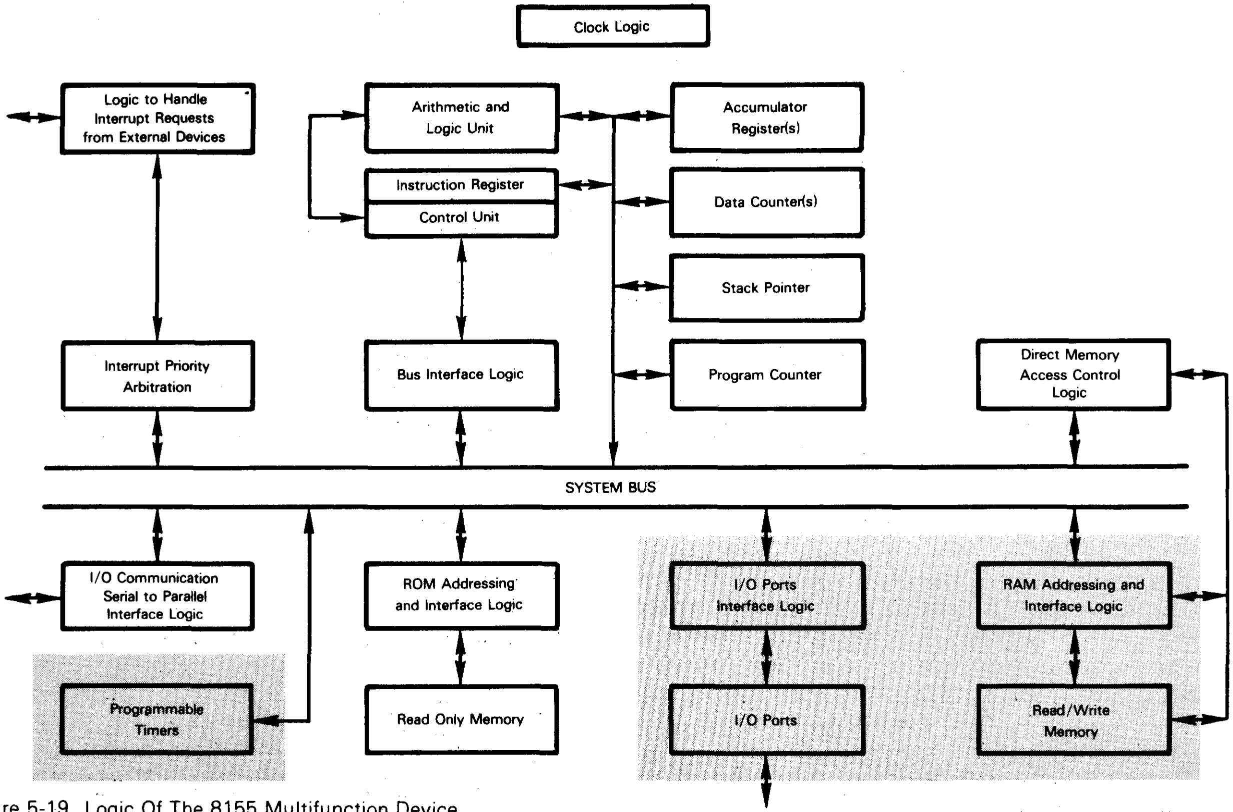
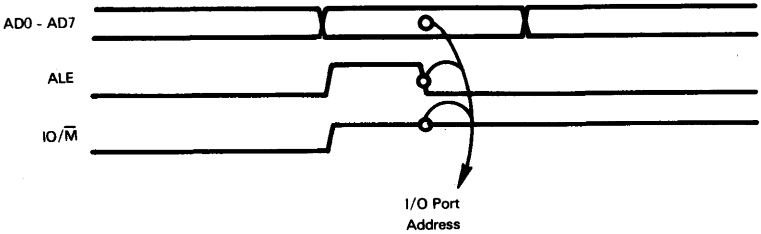

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Instruction Register
Control Unit
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 5-19. Logic Of The 8155 Multifunction Device

high, I/O port addresses are decoded off AD0 - AD7 on the high-to-low transition of ALE; this may be illustrated as follows:



When IO/$\overline{M}$ is low, the address strobed off AD0 - AD7 is interpreted as a memory address.

CE is active high in the 8156 device; it is active low in the 8155. There is no other difference between the 8155 and 8156 devices.

The 8155 device uses standard 8085 control signals on its CPU interface. These signals are $\overline{RD}$, $\overline{WR}$, ALE and IO/$\overline{M}$. Refer to the description of these control signals given in the 8085 section of this chapter.

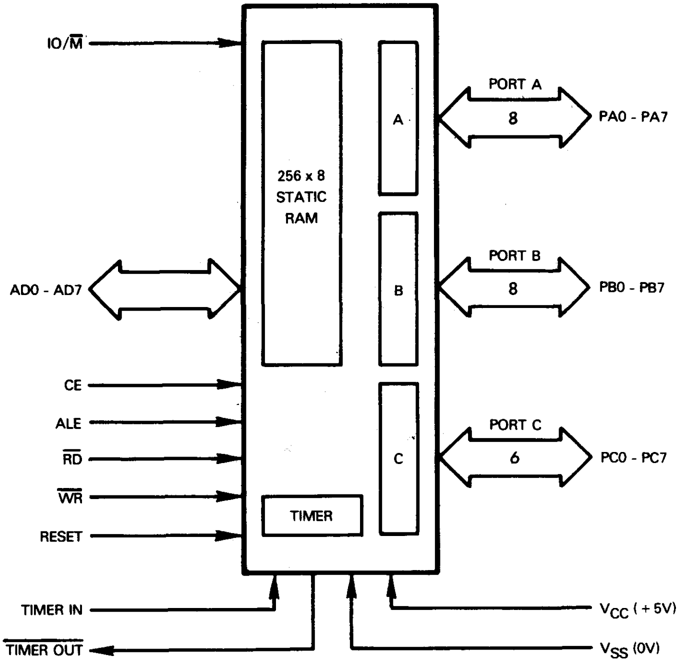
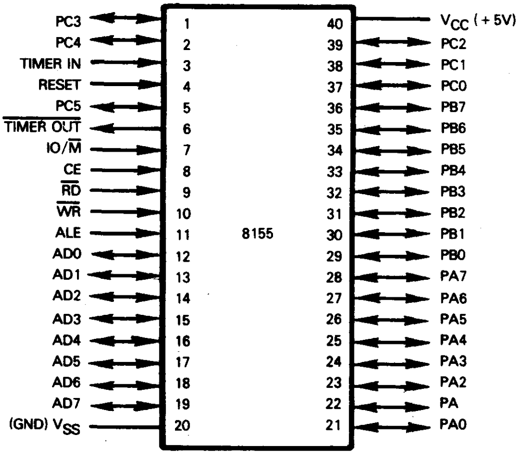


Figure 5-20. Logic Functions Of The 8155 Device



PIN NAME	DESCRIPTION	TYPE
AD0 - AD7	Multiplexed Address and Data Bus	Bidirectional
PA0 - PA7	Eight I/O pins, designated as Port A	Bidirectional
PB0 - PB7	Eight I/O pins, designated as Port B	Bidirectional
PC0 - PC5	Six I/O pins, designated as Port C	Bidirectional
R̄D̄	Read from device control	Input
W̄R̄	Write to device control	Input
IO/M̄	I/O ports or memory select	Input
ALE	Address latch enable	Input
RESET	System reset	Input
CE	Chip enable	Input
TIMER IN	Timer clock	Input
T̄ĪM̄ĒR̄ ŌŪT̄	Timer output signal	Output
V_{SS} V_{CC}	Ground, Power	

Figure 5-21. 8155 Multifunction Device Signals And Pin Assignments

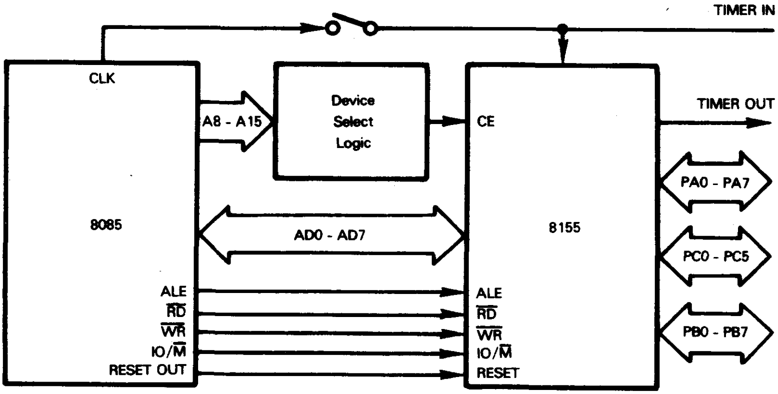


Figure 5-22. An 8155 Device Connected To An 8085 CPU Bus

Table 5-2. 8155 Device Port C Pin Options

Pin	ALT 1	ALT 2	ALT 3	ALT 4
PC0	Input Port	Output Port	A INTR (Port A Interrupt)	A INTR (Port A Interrupt)
PC1	Input Port	Output Port	A BF (Port A Buffer Full)	A BF (Port A Buffer Full)
PC2	Input Port	Output Port	A $\overline{STB}$ (Port A Strobe)	A $\overline{STB}$ (Port A Strobe)
PC3	Input Port	Output Port	Output Port	B INTR (Port B Interrupt)
PC4	Input Port	Output Port	Output Port	B BF (Port B Buffer Full)
PC5	Input Port	Output Port	Output Port	B $\overline{STB}$ (Port B Strobe)

The 8155 device is reset by a high input at the RESET pin. **The Reset operation does not clear memory or I/O locations within the 8155 device.** Thus all memory locations contain zero, I/O ports are assigned to input mode and the Counter/Timer is stopped with an initial zero value.

8155 DEVICE RESET

8155 PARALLEL INPUT/OUTPUT

The interface presented by the 8155 device to external logic consists of three I/O ports and two signals associated with Counter/Timer logic.

We will examine the I/O port logic and then the Counter/Timer logic.

I/O Ports A and B are 8-bit parallel ports; each may be defined as an input port or an output port.

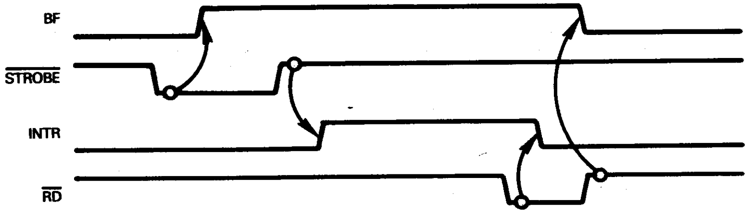
I/O Port C is a 6-bit parallel I/O port; it may be used to input or output parallel data, or Port C pins may support handshaking control signals for Ports A and B. Table 5-2 defines the four ways in which I/O Port C may be used.

When I/O Ports A and B are used for simple parallel input or output, then their operation is identical to Mode 0 as described in Chapter 4 for the 8255 PPI. Handshaking mode is identical to 8255 Mode 1. We will therefore discuss 8155 input and output with handshaking briefly. For a more detailed discussion refer to the 8255 PPI description given in Chapter 4.

8155 I/O MODE 0

8155 I/O MODE 1

Input with handshaking may be illustrated as follows:



An event sequence begins with external logic inputting parallel data to I/O Port A or B; **external logic must pulse $\overline{STROBE}$ low, at which time the parallel data is loaded into the I/O port buffer.** This causes BF, the Buffer Full signal, to go high.

External logic uses the BF signal as an indicator that no more data can be written.

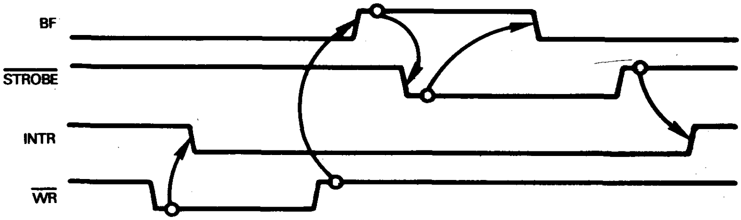
As soon as the externally provided low $\overline{STROBE}$ pulse is over the interrupt request signal INTR goes high. This allows the 8085 to be interrupted once data has been loaded into the input buffer of the I/O port.

BF and INTR remain high until the CPU reads the contents of the I/O port. The read operation will be identified by a low $\overline{RD}$ pulse input to the 8155 device. INTR is reset at the beginning of the $\overline{RD}$ pulse while BF is reset at the end of the $\overline{RD}$ pulse. BF therefore is high while data is waiting to be read and while data is being loaded into the I/O port buffer or read out of the I/O port buffer. INTR is high only while data is waiting to be read.

BF and INTR have associated bits in the Status register of the 8155 device.

You connect INTR to an 8085 interrupt request if you want an interrupt-driven system. You write a program which polls the Status register of the 8155 if you want to operate the system under program control.

Strobed output timing may be illustrated as follows:



In output mode the I/O port buffer is initially empty which means that the CPU must transmit data to the I/O port. Therefore INTR is initially high.

As soon as the CPU writes data to the I/O port the interrupt request signal INTR is reset low; this occurs on the leading edge of the $\overline{WR}$ pulse. On the trailing edge of the $\overline{WR}$ pulse **BF is output high telling external logic that data is in the I/O port buffer and may be read.**

External logic strobes the data out by providing a low pulse at $\overline{STROBE}$. The leading edge of $\overline{STROBE}$ resets BF low while the trailing edge of $\overline{STROBE}$ sets INTR high, causing the CPU to again output parallel data.

You connect INTR to an appropriate 8085 interrupt request pin if you want an interrupt-driven system. You write a program to poll the Status register if you want to operate the 8155 under program control.

A simple method of using the 8155 device parallel input/output with handshaking in interrupt mode would be to connect INTRA and INTRB to RST 5.5 and RST 6.5.

8155 DEVICE ADDRESSING

Having discussed 8155 device memory and I/O ports, we must now look at device addressing.

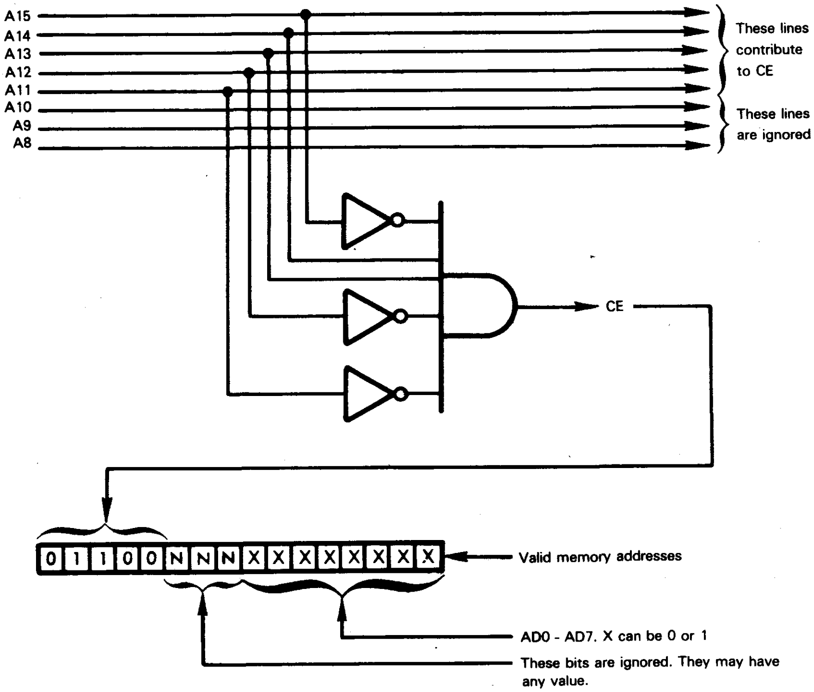
The 8155 has 256 bytes of static read/write memory which are addressed by AD0 - AD7 while CE is true and IO/$\overline{M}$ equals 0.

8155 I/O PORT ADDRESSES

The 8155 has eight addressable I/O ports. AD0, AD1 and AD2 select I/O ports while CE is true and IO/$\overline{M}$ equals 1. **These are the eight addressable I/O ports:**

AD2	AD1	AD0	PORT
0	0	0	Status/Command registers
0	0	1	Port A
0	1	0	Port B
0	1	1	Port C
1	0	0	Counter/Timer register, low order byte
1	0	1	Counter/Timer register, high order byte
1	1	0	Unused
1	1	1	Unused

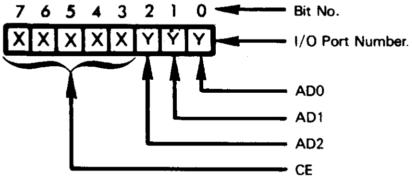
CE is derived from A8 - A15, which holds the high order byte of a memory address, or the I/O device number. **CE** thus **defines the exact address and I/O space for the 8155 device.** Here is one possible configuration:


A15
A14
A13
A12
A11
A10
A9
A8
These lines contribute to CE
These lines are ignored
CE
0 1 1 0 0 N N N X X X X X X X X
Valid memory addresses
AD0 - AD7. X can be 0 or 1
These bits are ignored. They may have any value.

8155 memory bytes will be selected by any memory addresses in the range $6N00_{16}$ through $6NFF_{16}$. N represents any digit in the range 0 through 7. Let us assume that programs access 8155 memory bytes via addresses in the range 6000_{16} through $60FF_{16}$; we must further assume that addresses created by values of N in the range 1 through 7 never occur.

Now **the same chip select that you use to define your memory address space is also going to define your I/O address space.** Recall that the 8-bit I/O device number is output twice following execution of an I/O instruction — once on the high order eight address lines A8 - A15 and again on the low order Address/Data Bus lines AD0 - AD7. Thus the device select code which you generate from the eight high order address lines for a memory address is the same device select code which you generate for the 8155 I/O space.

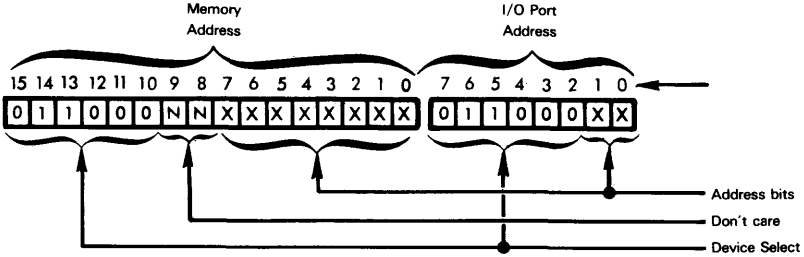
But whereas the 8155 has 256 addressable memory locations, it has eight addressable I/O ports; I/O ports are selected as follows:



If CE is true when A15 - A11 is 01100$_2$, then I/O port addresses will be 60$_{16}$ through 67$_{16}$.

Address lines A15 - A11 represent I/O device number bits 7 through 3. This is because the I/O device number is output on A15 - A8 following execution of an I/O instruction. It is therefore fortunate that we only used address lines A15 - A11 to create CE. Had we used A8, A9 or A10, the low order three I/O device code bits would have served double purpose — with strange results.

Suppose A10 = 0 is a prerequisite for device select logic to be true; these are the memory and I/O port selects which will result:



You can now address only four of the eight 8155 I/O Ports. You cannot include address lines A8, A9 or A10 in the device select logic that you use for any 8155 device; if you do you will limit the I/O capabilities of the device.

If you have an 8155 device present in an 8085 microcomputer system, then unless you include special device select logic you will be limited to a total memory address space of 8192 bytes. For example, the device select code we have been using thus far in our illustrations will cause the 8155 device to respond not only to the addresses 6000$_{16}$ through 60FF$_{16}$, but also to these other addresses:

6100$_{16}$ - 61FF$_{16}$
6200$_{16}$ - 62FF$_{16}$
6300$_{16}$ - 63FF$_{16}$
6400$_{16}$ - 64FF$_{16}$
6500$_{16}$ - 65FF$_{16}$
6600$_{16}$ - 66FF$_{16}$
6700$_{16}$ - 67FF$_{16}$

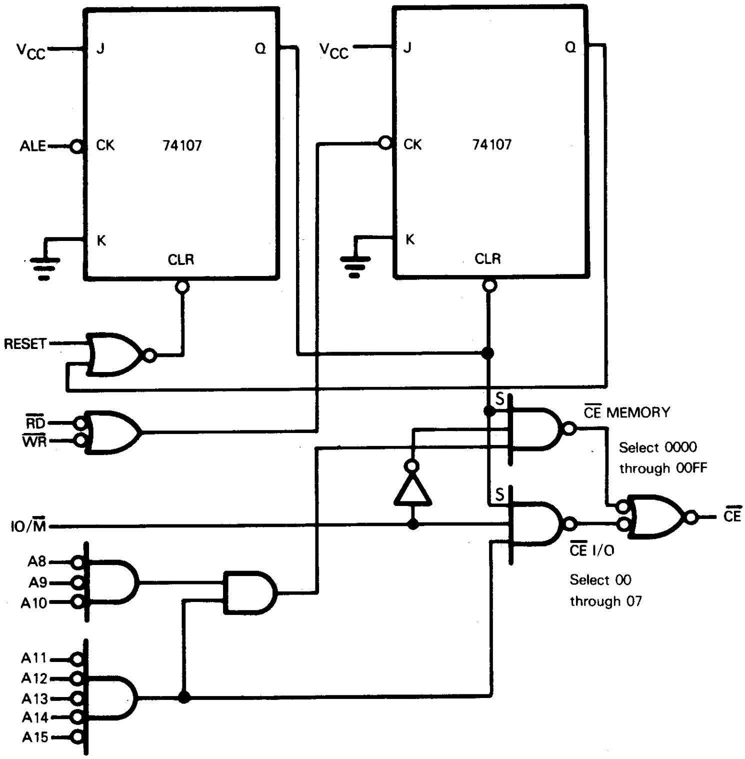


Figure 5-23. Select Logic To Disconnect Memory And I/O Device Spaces In An 8155 Device

Thus the 8155 device will consume eight times as much memory address space as it has memory bytes; you cannot touch three of the address lines when creating your device select logic since these three address lines also select individual I/O ports.

If you wish to use 8155 devices and a maximum of 8192 memory bytes is intolerable, then you must create a chip enable signal which is the OR of separate memory and I/O enables. Figure 5-23 illustrates a chip select circuit which will allow an 8155 device to be selected by memory addresses 0000_{16} through $00FF_{16}$ while I/O ports are selected by addresses 00_{16} through 07_{16}.

The logic in Figure 5-23 creates a low true Master chip select as the AND of separate low true memory and I/O device select. The separate low true memory and I/O device selects are each generated as the NAND of a Master Select (S), IO/$\overline{\text{M}}$ and an Address Bus decode.

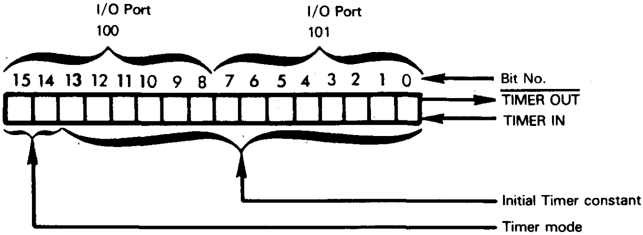
The Master Select (S) has been described earlier in this chapter; its purpose is to enable device selects only between the high-to-low transition of ALE and low-to-high transition of $\overline{\text{RD}}$ or $\overline{\text{WR}}$.

IO/$\overline{\text{M}}$ discriminates between execution of I/O instructions and memory reference instructions.

Address decode logic in this instance is very simple. I/O devices are selected by low levels on the high order five address lines A11 - A15; remember, these are also the five high order I/O device address lines. Memory is selected by low levels on the eight high order address lines A8 - A15.

THE 8155 COUNTER/TIMER

Counter/Timer logic consists of a 16-bit register, addressed as two 8-bit I/O ports, an input clock signal and an output timer signal. This may be illustrated as follows:



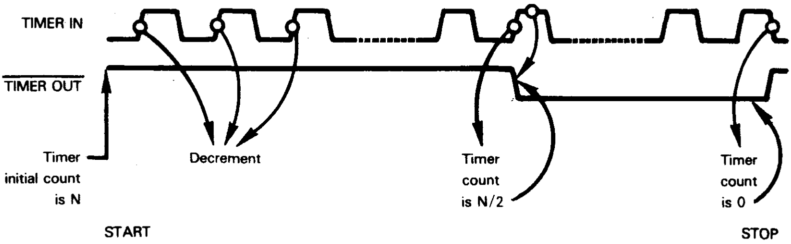
The low order 14 bits of the Counter/Timer register must be initialized with a 14-bit binary value that will decrement on low-to-high transitions of TIMER IN. If TIMER IN is connected to the 8085 clock output signal CLK, then the timer is computing real time. TIMER IN can alternatively be connected to any external logic in which case the timer is counting external events.

The timer times out when it decrements to zero.

The two high order bits of the Counter/Timer register define one of four ways in which the TIMER OUT signal may be created.

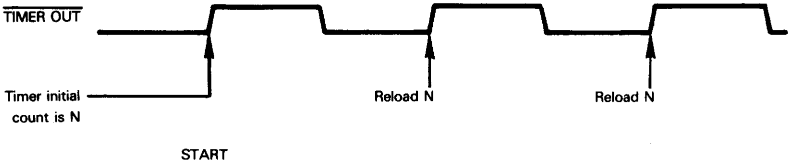
8155 TIMER MODE 0

In **Mode 0** TIMER OUT is high for the first half of the time interval and low for the second half of the time interval. This may be illustrated as follows:

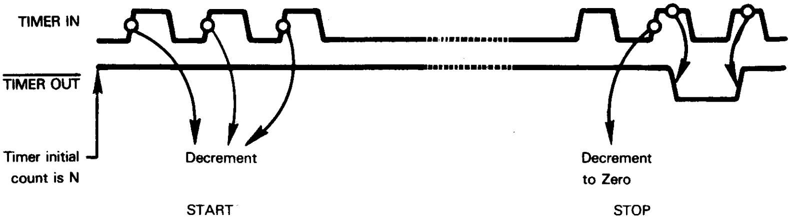


If N is odd, the extra pulse will occur while $\overline{\text{TIMER OUT}}$ is high.

In **Mode 1** as in Mode 0, $\overline{\text{TIMER OUT}}$ is high for the first half of the count and low for the second half. However, the timer is automatically reloaded with the initial value following each time out, creating a square wave which may be illustrated as follows:



Mode 2 outputs a single low clock pulse on the terminal count, then stops the timer. Timing may be illustrated as follows:



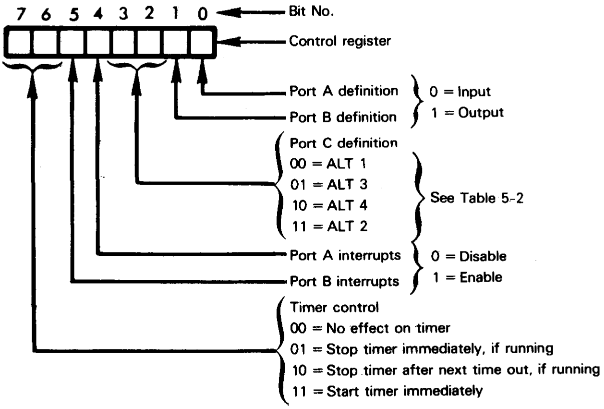
Mode 3 is identical to Mode 2 except that when the timer times out the initial counter value is automatically reloaded.

8155 CONTROL AND STATUS REGISTERS

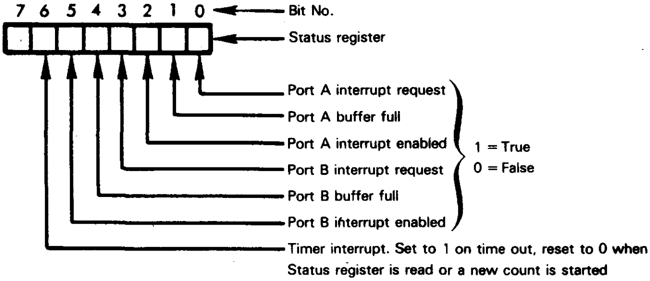
The Control and Status registers of the 8155 are used to control both timer and parallel I/O logic. Let us now examine these registers.

The Control and Status registers of the 8155 device are accessed via a single I/O port address. This is the lowest of the 8155 I/O port addresses. When you write to this address you access the Control register; when you read from this address you access the Status register.

8155 internal logic will interpret Control register bits as follows:



Status register bits are set and reset as follows:



8155 DEVICE PROGRAMMING

Accessing 8155 read/write memory is self-evident. If you execute a memory reference instruction that specifies an address within the 8155 address space, you will access an 8155 memory byte.

Parallel I/O programming is also self-evident; you begin by outputting an appropriate code to the Control register in order to define the modes in which various ports will operate, and to enable or disable Mode 1 interrupts. Your only caution at this time must be to ensure that the two high order bits of the Control code are 0; this prevents initiation of any timer operations.

If you are using I/O ports without handshaking, the Status register is not affected by I/O operations. No control signals or status indicate that new data has been input to, or has been read from I/O ports.

If you are operating the 8155 in handshaking mode under program control, then you must poll the Status register in order to determine whether data is waiting to be read or must be written. Your program will consist of a series of input instructions which read status, followed by conditional branches that read or write data.

If you are operating the 8155 parallel I/O in handshaking mode under interrupt control, then whenever data is waiting to be read or must be written, the high INTR control signal will vector program execution to an appropriate interrupt service routine.

You can at any time read the contents of an I/O port that has been declared an output port. You will simply read back whatever data was most recently written out to that I/O port. Reading the contents of an output port will have no effect on handshaking control signals associated with that port.

Let us now examine programming associated with 8155 Counter/Timer logic.

You must first initialize the 16-bit Counter/Timer register by outputting two bytes that specify timer mode and initial count. The order in which you output these two bytes is unimportant.

Next you output an appropriate Control code in order to start the timer. When you output a Control code remember not to modify any control bits that define parallel I/O operations.

Here is an appropriate initialization instruction sequence:

```
MVI     A,80H     LOAD 2080H AS AN INITIAL COUNTER
OUT     C4H       VALUE. SELECT COUNTER MODE 1
MVI     A,60H
OUT     C5H
MVI     A,FAH     START TIMER
OUT     C0H
```

This instruction sequence assumes that the 8155 I/O port addresses are C0$_{16}$ through C5$_{16}$. The code FA$_{16}$ output to the Control register starts the timer, and defines Port A as an input port, Port B as an output port, both in handshaking mode with interrupts enabled.

You can at any time stop the counter, either immediately or following the next time out. The following instructions will stop the counter immediately:

```
        MVI     A,7AH       STOP THE TIMER IMMEDIATELY
        OUT     C0H
```

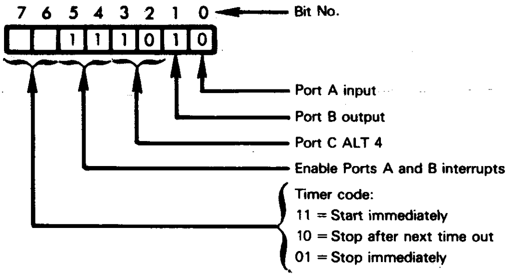
The following instructions will stop the counter after the next time out:

```
        MVI     A,BAH       STOP THE TIMER AFTER THE
        OUT     C0H         NEXT TIME OUT
```

You can read the contents of the Counter/Timer register at any time. Since the counter consists of two bytes, reading "on the fly", that is, while the counter is decrementing, can give you an inaccurate input; the counter will keep on decrementing between the execution of the two instructions needed to read both halves of the Counter/Timer register. In order to obtain an accurate reading of Counter/Timer register contents, you should stop the Counter/Timer, read register contents, then restart the Counter/Timer. Assuming again that the I/O control bits within the Control register must be preserved as 111010, this instruction sequence will stop the timer, read Counter/Timer register contents, then restart the timer:

```
        MVI     A,7AH       STOP THE TIMER
        OUT     C0H
        IN      C5          LOAD COUNTER/TIMER REGISTER
        MOV     B,A         CONTENTS INTO CPU REGISTERS
        IN      C4          B AND C
        MOV     C,A
        MVI     A,FAH       RESTART THE TIMER
```

The Counter/Timer instruction sequences illustrated above contain a non-obvious propensity for programming errors. We start the timer by outputting the code FA$_{16}$ to the Control register; we stop immediately by outputting the code 7A$_{16}$ and we stop the timer after the next time-out by outputting the code BA$_{16}$. In reality this is the code we are outputting:



Whenever you output Control codes to modify 8155 timer operation, you must always remember to output bits 0 through 5 correctly, in order to maintain previously defined parallel I/O options. **A commonly used programming technique that frees you from**

having to remember the condition of irrelevant bits in a control word is to use AND and OR masks. Consider this general purpose instruction sequence:

```
IN      COH       INPUT PRESENT CONTROL CODE
ANI     3FH       CLEAR TIMER BITS
ORI     COH       SET TIMER BITS
OUT     COH       RESTORE CONTROL CODE
```

This technique will not work with the 8155 device since you cannot read the contents of the Control register. If you read from the address of the Control register you will access the Status register. If you want to use a masking technique, you must maintain the Control code in memory. Here is an instruction sequence that will work:

```
LDA     CONTRL    LOAD CONTROL CODE FROM MEMORY
ANI     3FH       CLEAR TIMER BITS
(ORI    COH       SET TIMER BITS)
OUT     COH       OUTPUT CONTROL CODE TO 8155
STA     CONTRL    SAVE CONTROL CODE IN MEMORY.
```

Your instruction sequence will include the ANI mask to clear timer bits, or the ORI mask to set timer bits, but obviously not both.

CONTRL is the label for some read/write memory byte which always holds the current 8155 Control code.

THE 8355 READ ONLY MEMORY WITH I/O

The 8355 provides 2048 bytes of read-only memory and two 8-bit I/O ports. The device has been designed to interface with the 8085 CPU.

Figure 5-24 illustrates that part of our general microcomputer system logic which has been implemented on the 8355 device.

The 8355 is packaged as a 40-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL compatible. The device is implemented using N-channel MOS technology.

Figure 5-25 functionally illustrates logic of the 8355 device. A simple 8085-8155-8355 configuration is illustrated in Figure 5-27.

There are many similarities between the 8155, which we have already described, and the 8355. Where appropriate we will refer back to the 8155 discussion for clarification of concepts.

8355 DEVICE PINS AND SIGNALS

8355 pins and signals are illustrated in Figure 5-26.

The 8355-8085 interface differs somewhat from the 8155-8085 interface in that the 8355 has more memory, fewer addressable I/O ports, plus the ability to address I/O ports within the memory space of the device.

Having 2048 bytes of addressable read-only memory, **the 8355 requires eleven address pins. These are derived from AD0-AD7 and A8-A10.**

Having only four addressable I/O ports, the 8355 I/O address logic decodes AD0 and AD1 only. I/O ports are selected as follows:

AD1	AD0	
0	0	I/O PORT A
0	1	I/O PORT B
1	0	DATA DIRECTION REGISTER A
1	1	DATA DIRECTION REGISTER B

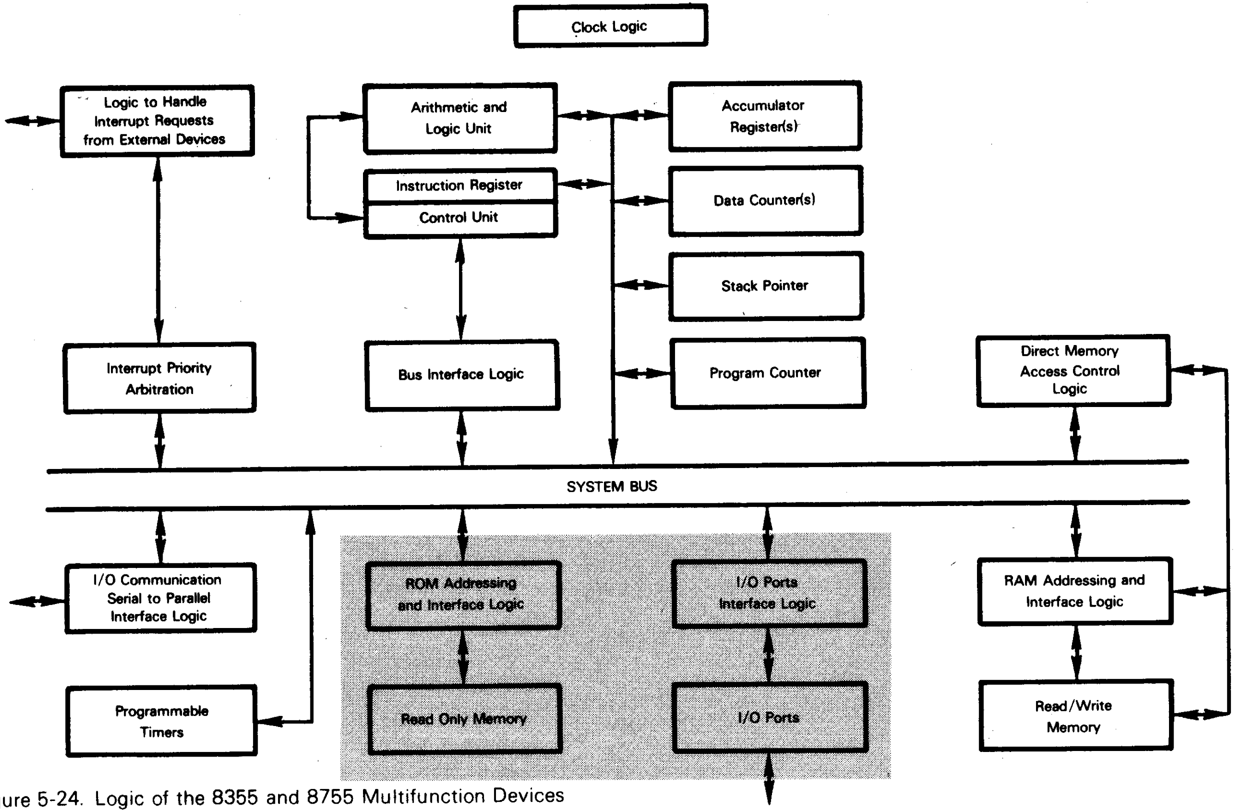

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Instruction Register
Control Unit
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Interrupt Priority Arbitration
Bus Interface Logic
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
Programmable Timers
ROM Addressing and Interface Logic
Read Only Memory
I/O Ports Interface Logic
I/O Ports
RAM Addressing and Interface Logic
Read/Write Memory

Figure 5-24. Logic of the 8355 and 8755 Multifunction Devices

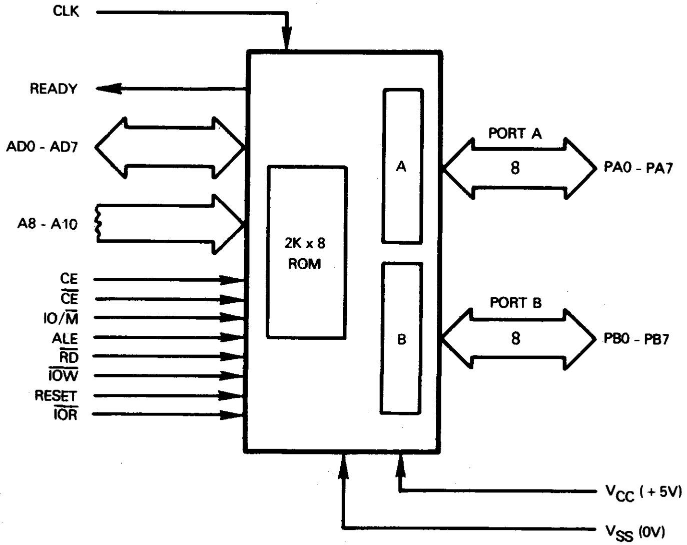


Figure 5-25. Logic Functions of the 8355 Device

8355 device select logic must generate the chip enable signals $\overline{CE}$ and CE from the five address lines A11-A15. The discussion of select logic given for the 8155 device applies also to the 8355.

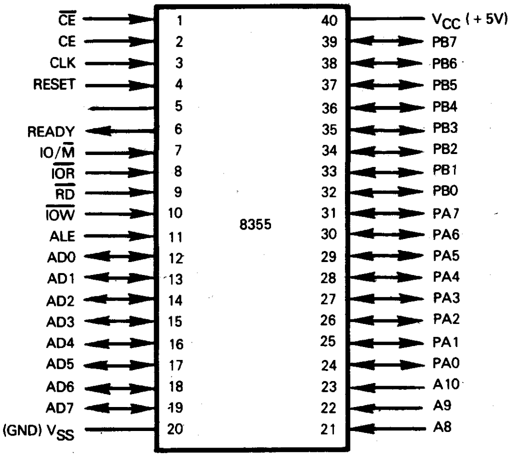
If you select 8355 memory and I/O ports in their respective address spaces, the control signals ALE, $\overline{RD}$, and IO/$\overline{M}$ are used exactly as described for the 8155 device.

But you can also access 8355 I/O ports within the 8355 memory space using control signals $\overline{IOW}$ and $\overline{IOR}$.

$\overline{IOW}$ and $\overline{IOR}$ are control signals which override IO/$\overline{M}$ and $\overline{RD}$ when accessing I/O ports.

Providing CE and $\overline{CE}$ are true, a low input on $\overline{IOW}$ will cause data on the Data Bus to be written into the I/O port selected by AD0 and AD1, irrespective of the IO/$\overline{M}$ level. Similarly, $\overline{IOR}$ low will cause the contents of the I/O port selected by AD0 and AD1 to be output on the Data Bus.

You can connect $\overline{IOW}$ directly to the $\overline{WR}$ control signal and thus write into the four I/O ports of the 8355 device as though they were the four low order memory bytes. But connecting $\overline{IOR}$ to $\overline{RD}$ is not so straightforward. The 8355 device may receive a low input on $\overline{IOR}$ together with low inputs on $\overline{RD}$ and IO/$\overline{M}$; it will then attempt to read the contents of a read only memory byte and an I/O port at the same time. While elaborate schemes could be devised for generating separate selects that map the four I/O ports into a memory space of its own, **it is wisest to ignore the $\overline{IOR}$ signal if you are using 8355 memory and I/O logic. Use $\overline{IOR}$ only when the 8355 is configured as two I/O ports — and the 8355 memory is unused. $\overline{IOR}$ and $\overline{IOW}$ are used in 8048 microcomputer systems; that is the principal reason they were designed into the 8355 device.**



PIN NAME	DESCRIPTION	TYPE
AD0 - AD7	Multiplexed Address and Data Bus	Bidirectional
A8 - A10	Memory Address Lines	Input
PA0 - PA7	Eight I/O pins, designated as Port A	Bidirectional
PB0 - PB7	Eight I/O pins, designated as Port B	Bidirectional
$\overline{RD}$	Read from device control	Input
$\overline{IOR}$	Read from I/O port control	Input
$\overline{IOW}$	Write to I/O port control	Input
IO/$\overline{M}$	I/O ports or memory select	Input
ALE	Address latch enable	Input
RESET	System reset	Input
CE, $\overline{CE}$	Chip enables	Input
READY	Wait state request	Output, tristate
CLK	Timing for Wait state request	Input
V_{SS}, V_{CC}	Ground, Power	

Figure 5-26. 8355 Multifunction Device Signals and Pin Assignments

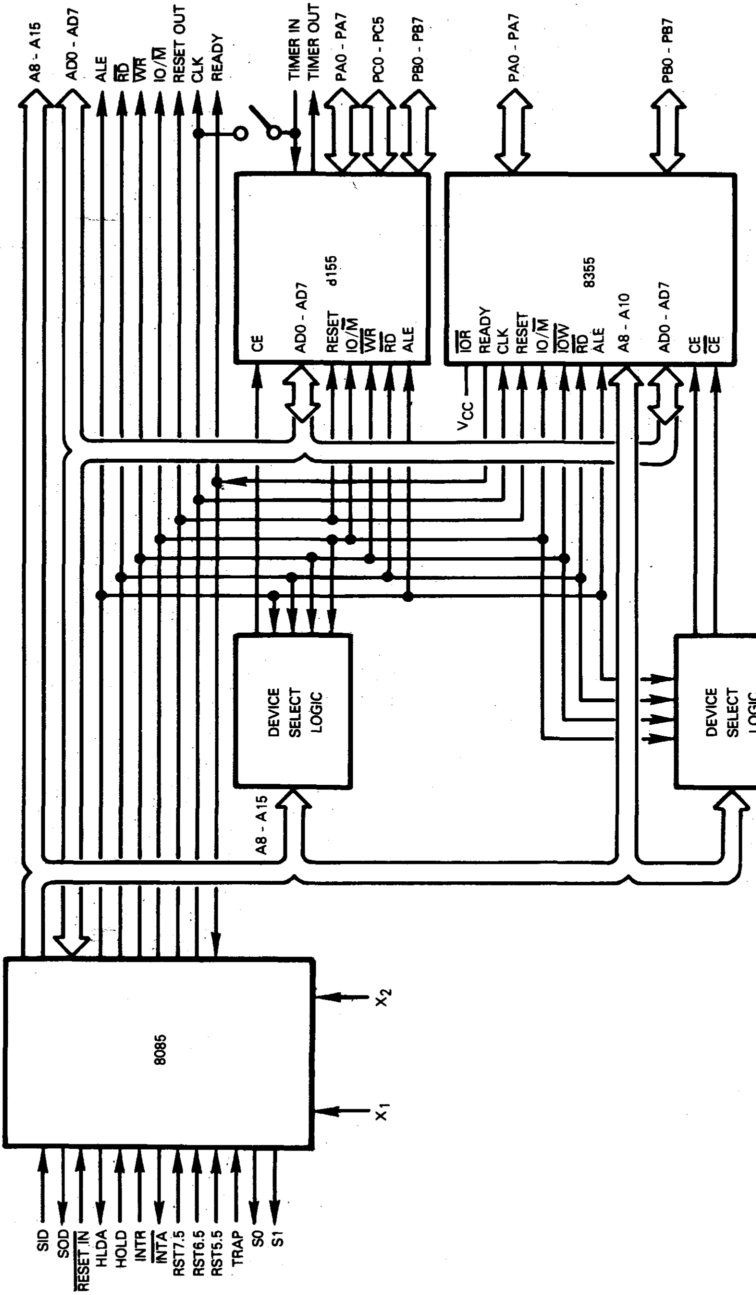
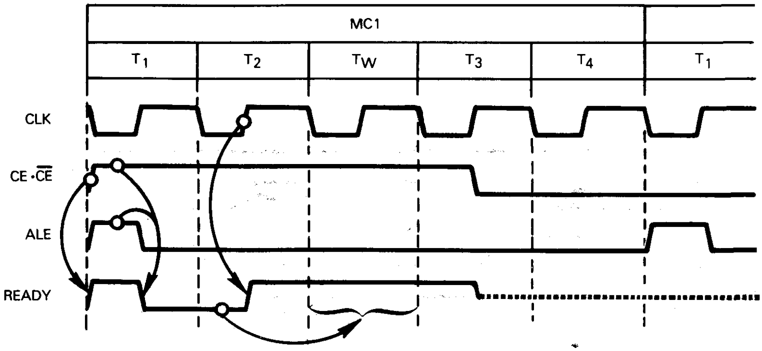


Figure 5-27. An 8085-8155-8355 Microcomputer System

8355 READY LOGIC

The 8355 device has on-chip logic to create a READY signal that will insert one Wait state into the 8085 machine cycle that references the 8355 device. 8355 READY signal timing may be illustrated as follows:

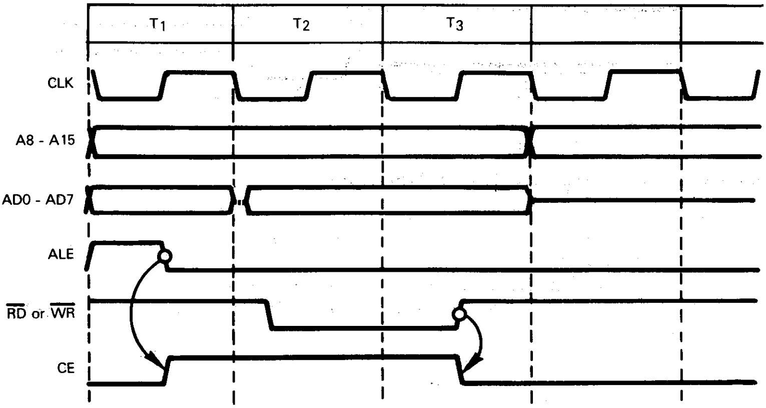


The READY output is floated by the 8355 device while CE•$\overline{CE}$ is false.

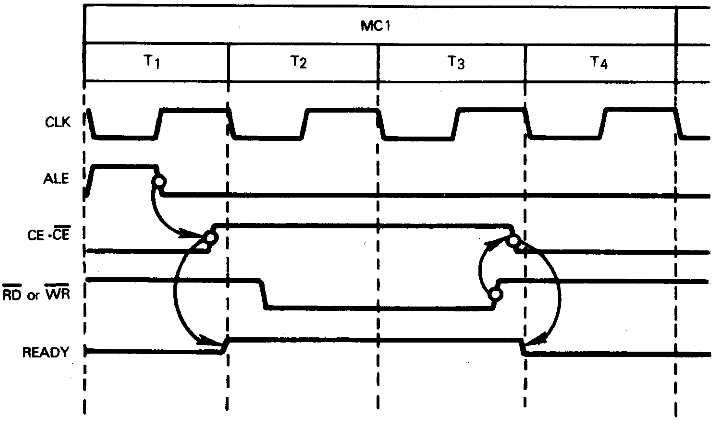
READY is forced low by the combination of Chip Enable true while ALE is high; READY stays low until the first low-to-high transition of CLK following the end of the ALE pulse. If you refer back to Figure 5-11 you will see that this READY logic creates a single Wait state.

The problem with the READY logic illustrated above is that in order to have Chip Enable true while ALE is high, chip enable logic must be tied directly to Address Bus lines. Refer to the timing diagram below and you will see that A0-A15 is stable while ALE is high.

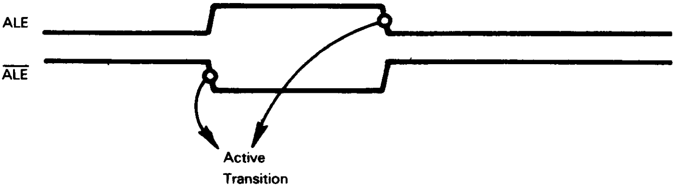
But as we discussed earlier in this chapter, you can derive chip enable logic directly from A8-A15 only in very small 8085 microcomputer systems. When a number of support devices are connected to the System Bus, you must guarantee against spurious device selects by including control signals in the chip enable logic. Logic illustrated earlier in this chapter shows how to create a chip select signal that is true between the trailing edge of ALE and the low-to-high transition of $\overline{RD}$ or $\overline{WR}$. The following chip enable timing results:



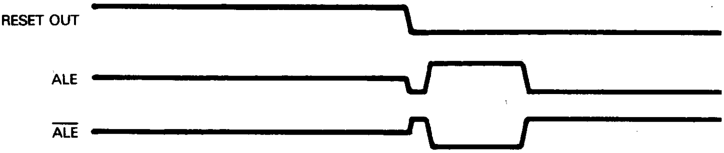
Timing illustrated above is theoretically the best guarantee against spurious selects; but it will not work if you want to create a single Wait state when using an 8355 device. If Chip Enable (CE) goes true on the trailing edge of ALE, READY will never be reset low:



You cannot resolve this problem by simply inverting ALE as a clock input. Normally $\overline{ALE}$ would work:



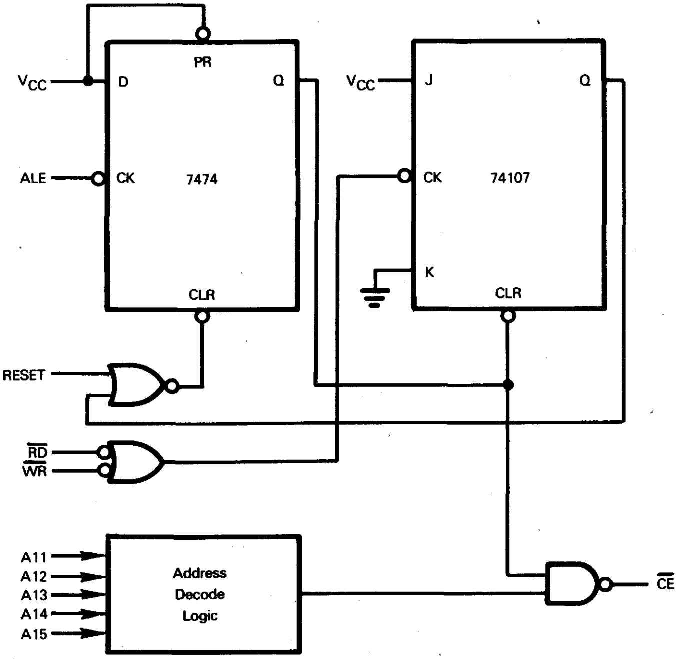
But after a reset, or upon powering up, this is what you get:



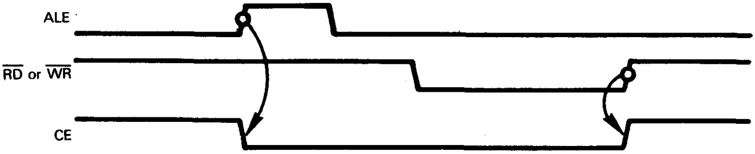
The ALE active transition must be preceded by a clock pulse which outputs Q high. No such pulse will occur following a RESET.

But when do you need to induce a Wait state?

8355 device timing is fast enough to respond to memory and I/O acceses without the inclusion of a Wait state, unless you have buffers on the System Bus and the buffers introduce unacceptably long response delays. Therefore, ignore the READY signal logic of the 8355 in small 8085 systems and derive chip enable logic directly from the high order address lines A11-A15. In larger systems where buffers on the System Bus force the 8355 device to require a Wait state, use READY logic of the 8355 device. Chip enable logic must be derived as follows:



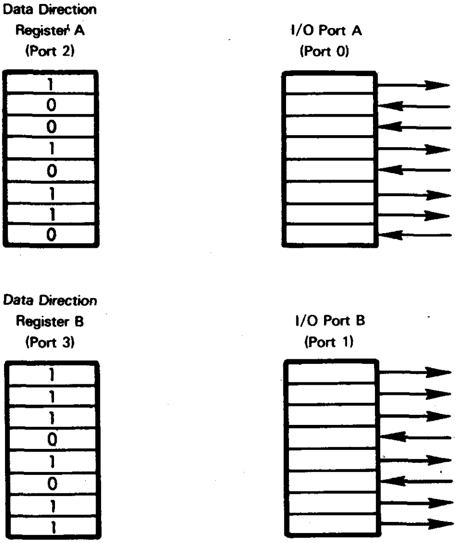
Here is the timing derived:



Your alternative is to create a READY signal externally using two 7474 D type flip-flops, as described earlier in this chapter. If you create the READY signal externally, then the chip enable timing on the 8355 device ceases to be critical.

8355 I/O LOGIC

Let us now look at the I/O logic of the 8355 device. This device has two I/O ports whose pins can be individually assigned to input or output. This assignment is made by loading appropriate Control codes into a Data Direction register associated with each I/O port. A 1 in any bit position of the Data Direction register defines the associated I/O port pin as an output pin. A 0 in any bit position defines the associated I/O port pin as an input pin. This may be illustrated as follows:


Data Direction Register A (Port 2)
1
0
0
1
0
1
1
0
I/O Port A (Port 0)
Data Direction Register B (Port 3)
1
1
1
0
1
0
1
1
I/O Port B (Port 1)

Observe that the 8355 has no I/O with handshaking. For I/O with handshaking you should use the 8155 or the 8255 devices.

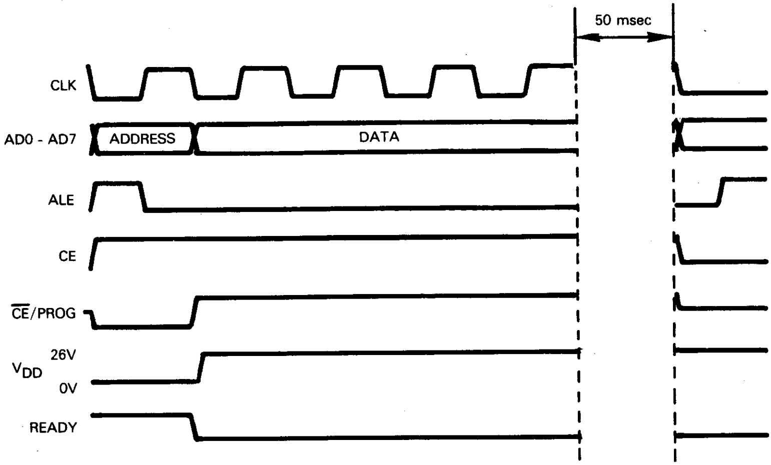
THE 8755 ERASABLE PROGRAMMABLE READ ONLY MEMORY WITH I/O

The 8755 device provides 2048 bytes of erasable programmable read-only memory and two 8-bit I/O ports. The only difference between this device and the 8355, which we have just described, is the fact that the 8755 read-only memory is programmable and erasable. There are minor pin and signal variations supporting the EPROM. These differences are identified in Figure 5-28.

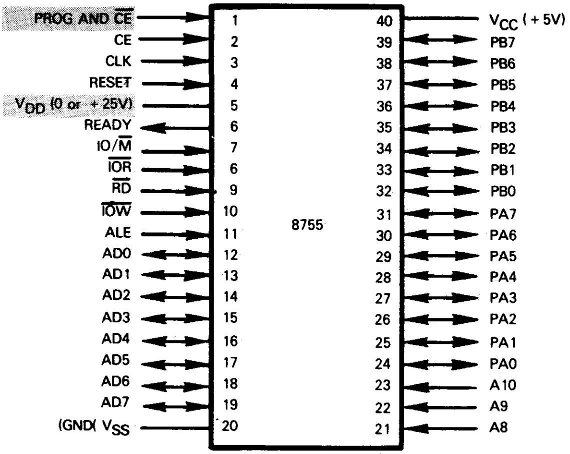
This discussion of the 8755 device is limited to describing how you program the read-only memory. In all other ways, the 8755 device is identical to the 8355.

There are two Chip Enable signals on the 8755 device; CE is the standard chip enable which must be true when the 8755 device is being accessed for any purpose, either in normal operation or when programming the read-only memory. CE is a high true signal.

The second Chip Enable signal $\overline{CE}$ PROG is first held low, then it is pulsed true only when you are programming the read-only memory. You must apply a +26V pulse lasting between 50 and 100 milliseconds, beginning with the leading edge of ALE. At this time data will be written into the addressed read-only memory location. Timing may be illustrated as follows:



You erase the programmable read-only memory by exposing it to ultraviolet light for a minimum of twenty minutes.



PIN NAME	DESCRIPTION	TYPE
AD0 - AD8	Multiplexed Address and Data Bus	Bidirectional
A8 - A10	Memory address lines	Input
PA0 - PA7	Eight I/O pins, designated as Port A	Bidirectional
PB0 - PB7	Eight I/O pins, designated as Port B	Bidirectional
$\overline{RD}$	Read from device control	Input
$\overline{IOR}$	Read from I/O port control	Input
$\overline{IOW}$	Write to I/O port control	Input
IO/$\overline{M}$	I/O ports or memory select	Input
ALE	Address latch enable	Input
RESET	System reset	Input
CE	Chip enable	Input
PROG AND $\overline{CE}$	PROM programming chip enable	Input
READY	Wait state request	Output, tristate
CLK	Timing for Wait state request	Input
V_{SS}, V_{CC}	Ground, Power	

Figure 5-28. 8755 Multifunction Device Signals And Pin Assignments

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin With Respect to Ground	-0.3 to +7V
Power Dissipation	1.5 Watt

***COMMENT:** *Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

(T_A = 0°C to 70°C; V_{CC} = 5V ±5%; V_{SS} = 0V; unless otherwise specified)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{CC}	Power Supply Current		170	mA	
I_{IL}	Input Leakage		±10	μA	$V_{in} = V_{CC}$
I_{LO}	Output Leakage		±10	μA	$0.45V \leqslant V_{out} \leqslant V_{CC}$
V_{ILR}	Input Low Level, RESET	-0.5	+0.8	V	
V_{IHR}	Input High Level, RESET	2.4	V_{CC}	V	
V_{HY}	Hysteresis, RESET	0.25		V	

Bus Timing Specification as a T_{CYC} Dependent

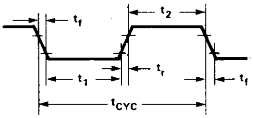
t_{AL}	–	(1/2) T - 50	MIN
t_{LA}	–	(1/2) T - 20	MIN
t_{LL}	–	(1/2) T - 40	MIN
t_{LCK}	–	(1/2) T - 50	MIN
t_{LC}	–	(1/2) T - 30	MIN
t_{AD}	–	(5/2 + N) T - 225	MAX
t_{RD}	–	(3/2 + N) T - 200	MAX
t_{RAE}	–	(1/2) T - 60	MIN
t_{CA}	–	(1/2) T - 40	MIN
t_{DW}	–	(3/2 + N) T - 60	MIN
t_{WD}	–	(1/2) T - 80	MIN
t_{CC}	–	(3/2 + N) T - 80	MIN
t_{CL}	–	(1/2) T - 110	MIN
t_{ARY}	–	(3/2) T - 260	MAX
t_{HACK}	–	(1/2) T - 50	MIN
t_{HABF}	–	(1/2) T + 30	MAX
t_{HABE}	–	(1/2) T + 30	MAX
t_{AC}	–	(2/2) T - 50	MIN
t_1	–	(1/2) T - 80	MIN
t_2	–	(1/2) T - 40	MIN
t_{RV}	–	(3/2) T - 80	MIN
t_{INS}	–	(1/2) T + 200	MIN

NOTE: N is equal to the total WAIT states.
T = t_{CYC}.

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ±5%; V_{SS} = 0V)

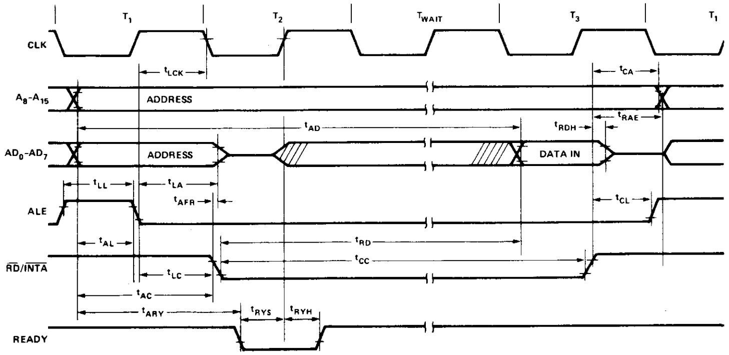
Symbol	Parameter	Min.	Max.	Units	Test Conditions
T_{CYC}	CLK Cycle Period	320	2000	ns	See notes 1, 2, 3, 4, 5
t_1	CLK Low Time	80		ns	
t_2	CLK High Time	120		ns	
t_r, t_f	CLK Rise and Fall Time		30	ns	
t_{AL}	Address Valid Before Trailing Edge of ALE	110		ns	
t_{LA}	Address Hold Time After ALE	100		ns	
t_{LL}	ALE Width	120		ns	
t_{LCK}	ALE Low During CLK High	100		ns	
t_{LC}	Trailing Edge of ALE to Leading Edge of Control	130		ns	
t_{AFR}	Address Float After Leading Edge of $\overline{\text{READ}}$ ($\overline{\text{INTA}}$)		0	ns	
t_{AD}	Valid Address to Valid Data In		575	ns	
t_{RD}	$\overline{\text{READ}}$ (or $\overline{\text{INTA}}$) to Valid Data		280	ns	
t_{RDH}	Data Hold Time After $\overline{\text{READ}}$ ($\overline{\text{INTA}}$)	0		ns	
t_{RAE}	Trailing Edge of $\overline{\text{READ}}$ to Re-Enabling of Address	100		ns	T_{CYC} = 320ns; C_L = 150 pF
t_{CA}	Address (A8-A15) Valid After Control	120		ns	
t_{DW}	Data Valid to Trailing Edge of $\overline{\text{WRITE}}$	420		ns	
t_{WD}	Data Valid After Trailing Edge of $\overline{\text{WRITE}}$	80		ns	
t_{CC}	Width of Control Low ($\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{INTA}}$)	400		ns	
t_{CL}	Trailing Edge of Control to Leading Edge of ALE	50		ns	
t_{ARY}	READY Valid From Address Valid		220	ns	
t_{RYS}	READY Setup Time to Leading Edge of CLK	110		ns	
t_{RYH}	READY Hold Time	0		ns	
t_{HACK}	HLDA Valid to Trailing Edge of CLK	110		ns	
t_{HABF}	Bus Float After HLDA		190	ns	
t_{RV}	Control Trailing Edge to Leading Edge of Next Control	400		ns	
t_{AC}	Address Valid to Leading Edge of Control	270		ns	
t_{HDS}	HOLD Setup Time to Trailing Edge of CLK	170		ns	
t_{HDH}	HOLD Hold Time	0		ns	
t_{INS}	INTR Setup Time to Leading Edge of CLK (M1, T1 only). Also RST and TRAP	360		ns	
t_{INH}	INTR Hold Time	0		ns	

NOTES:
1. A8-15 Address Specs apply to IO/$\overline{M}$, S0 and S1.
2. For all output timing where $C_L \neq$ 150pf use the following correction factors:
   25pf $\leq C_L <$ 150pf : -.10 ns/pf
   150pf $< C_L \leq$ 300pf : +.30 ns/pf
3. Output timings are measured with purely capacitive load.
4. All timings are measured at output voltage V_L = .8V, V_H = 2.0V, and 1.5V with 20ns rise and fall time on inputs.
5. To calculate timing specifications at other values of T_{CYC} use the table in Table 2.
6. L.E. = Leading Edge T.E. = Trailing Edge

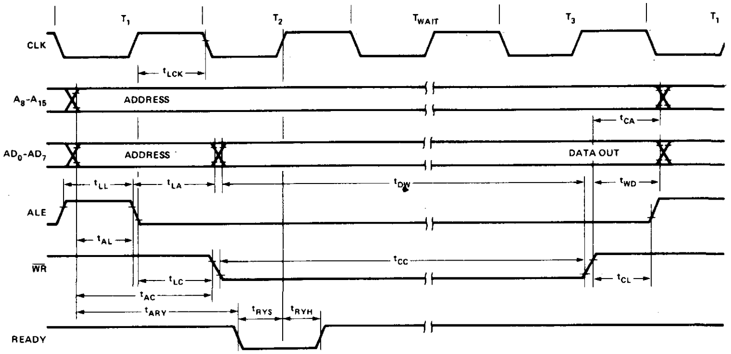


CLOCK TIMING WAVEFORM

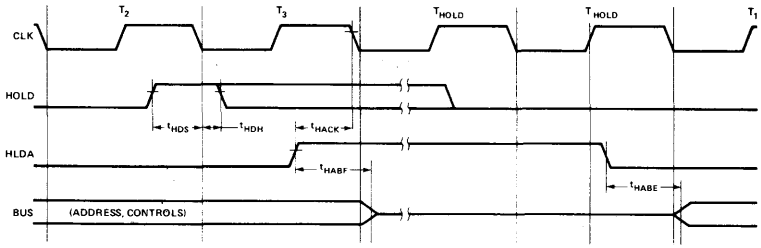
READ OPERATION



WRITE OPERATION



HOLD OPERATION



8755

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... -10°C to +70°C
Storage Temperature .... -65°C to +150°C
Voltage on Any Pin
With Respect to Ground .... -0.5V to +7V
Power Dissipation 1.5W

**COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		10	μA	V_{IN} = V_{CC} to 0V
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤V_{OUT} ≤V_{CC}
I_{CC}	V_{CC} Supply Current		180	mA	

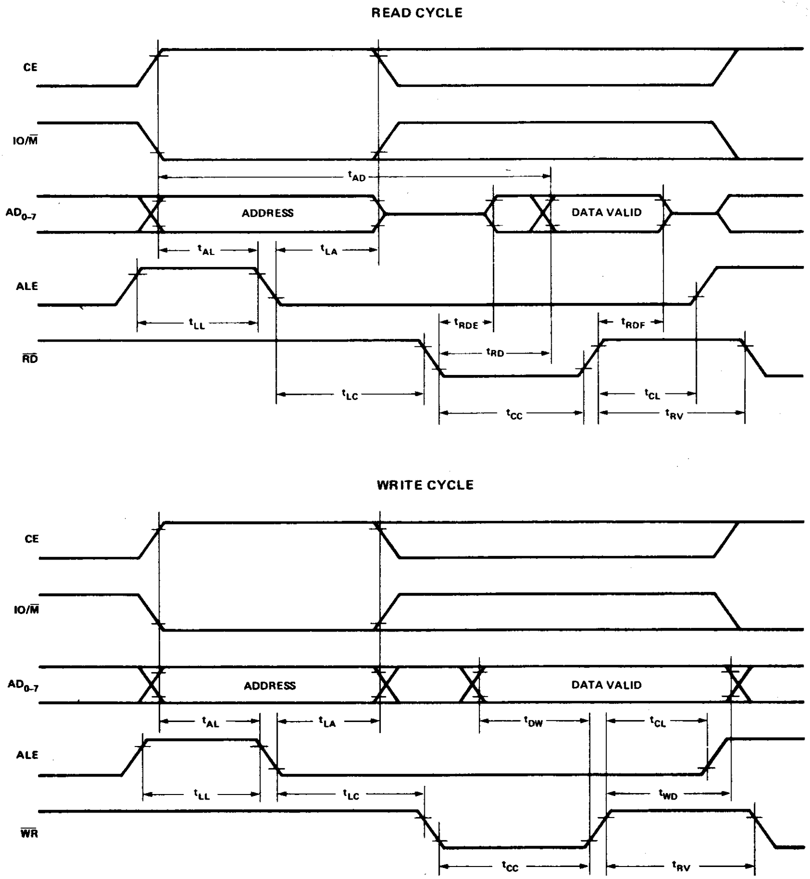
PRELIMINAR

Notice: This is not a final specification. Some parametric limits are subject to change.

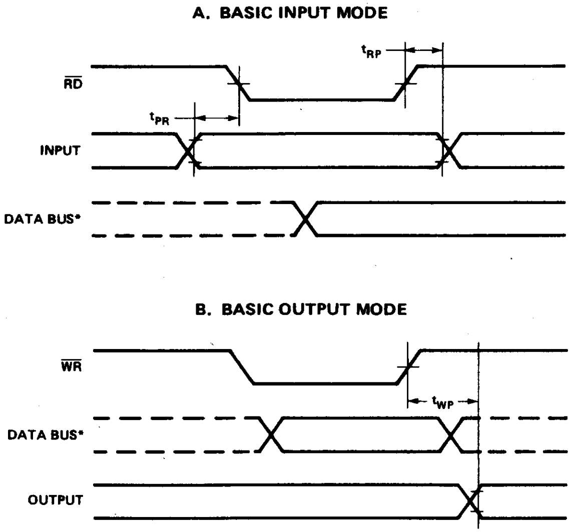
A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
t_{AL}	Address to Latch Set Up Time	50		ns	150 pF Load
t_{LA}	Address Hold Time after Latch	80		ns	
t_{LC}	*Latch to READ/WRITE Control*	100		ns	
t_{RD}	Valid Data Out Delay from READ Control		150	ns	
t_{AD}	Address Stable to Data Out Valid		400	ns	
t_{LL}	Latch Enable Width	100		ns	
t_{RDF}	Data Bus Float After READ	0	100	ns	
t_{CL}	READ/WRITE Control to Latch Enable	20		ns	
t_{CC}	READ/WRITE Control Width	250		ns	
t_{DW}	Data In to WRITE Set Up Time	150		ns	
t_{WD}	Data In Hold Time After WRITE	0		ns	
t_{RV}	Recovery Time Between Controls	300		ns	
t_{WP}	WRITE to Port Output		400	ns	
t_{PR}	Port Input Setup Time	50		ns	
t_{RP}	Port Input Hold Time	50		ns	
t_{SBF}	Strobe to Buffer Full		400	ns	
t_{SS}	Strobe Width	200		ns	
t_{RBE}	READ to Buffer Empty		400	ns	
t_{SI}	Strobe to INTR On		400	ns	
t_{RDI}	READ to INTR Off		400	ns	
t_{PSS}	Port Setup Time to Strobe Strobe	50		ns	
t_{PHS}	Port Hold Time After Strobe	100		ns	
t_{SBE}	Strobe to Buffer Empty		400	ns	
t_{WBF}	WRITE to Buffer Full		400	ns	
t_{WI}	WRITE to INTR Off		400	ns	
t_{TL}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ Low	400		ns	
t_{TH}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ High	400		ns	
t_{RDE}	Data Bus Enable from READ Control	10		ns	

Note: For Timer Input Specification, see Figure 10.

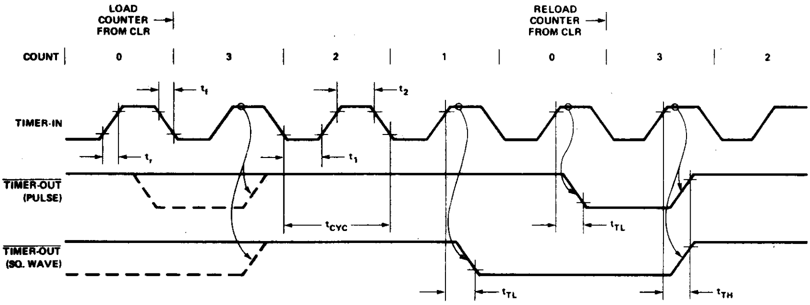


8155 READ/WRITE TIMING DIAGRAM.



*DATA BUS TIMING IS SHOWN IN FIGURE 7.

STROBED I/O TIMING WAVEFORM.



COUNTDOWN FROM 3 TO 0

t_{CYC}	320 ns MIN.
t_{RISE} & t_{FALL}	30 ns MAX.
t_1	80 ns MIN.
t_2	120 ns MIN.
t_{TL}	TIMER-IN TO $\overline{\text{TIMER-OUT}}$ LOW (TO BE DEFINED).
t_{TH}	TIMER-IN TO $\overline{\text{TIMER-OUT}}$ HIGH (TO BE DEFINED).

TIMER OUTPUT WAVEFORM.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to +70°C
Storage Temperature .... -65°C to +150°C
Voltage on Any Pin
With Respect to Ground .... -0.3V to +7V
Power Dissipation 1.5W

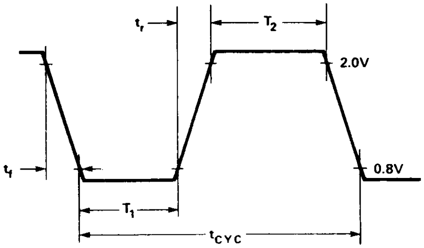
**COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

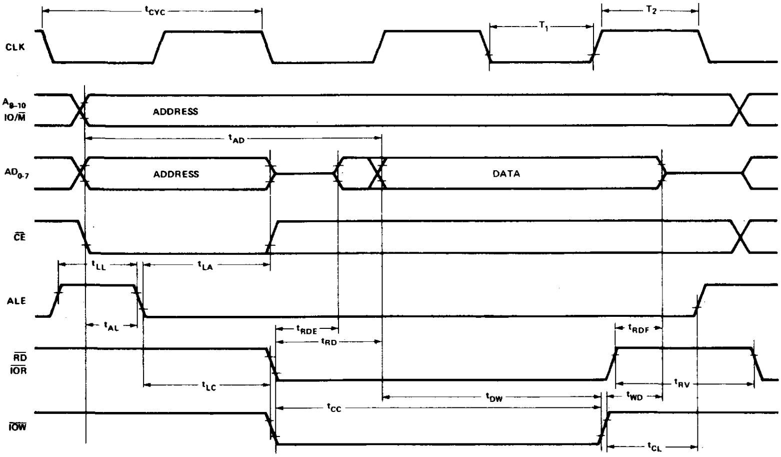
SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		10	μA	V_{IN} = V_{CC} to 0V
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤V_{OUT} ≤V_{CC}
I_{CC}	V_{CC} Supply Current		180	mA	

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
t_{CYC}	Clock Cycle Time	320		ns	C_{LOAD} = 150 pF (See Figure 3)
T_1	CLK Pulse Width	80		ns	
T_2	CLK Pulse Width	120		ns	
t_f,t_r	CLK Rise and Fall Time		30	ns	
t_{AL}	Address to Latch Set Up Time	50		ns	150 pF Load
t_{LA}	Address Hold Time after Latch	80		ns	
t_{LC}	Latch to READ/WRITE Control	100		ns	
t_{RD}	Valid Data Out Delay from READ Control		150	ns	
t_{AD}	Address Stable to Data Out Valid		400	ns	
t_{LL}	Latch Enable Width	100		ns	
t_{RDF}	Data Bus Float after READ	0	100	ns	
t_{CL}	READ/WRITE Control to Latch Enable	20		ns	
t_{CC}	READ/WRITE Control Width	250		ns	
t_{DW}	Data In to WRITE Set Up Time	150		ns	
t_{WD}	Data In Hold Time After WRITE	0		ns	
t_{WP}	WRITE to Port Output		400	ns	
t_{PR}	Port Input Set Up Time	50		ns	
t_{RP}	Port Input Hold Time	50		ns	
t_{RYH}	READY HOLD TIME	0	120	ns	
t_{ARY}	ADDRESS (CE) to READY		160	ns	
t_{RV}	Recovery Time between Controls	300		ns	
t_{RDE}	Data Out Delay from READ Control	10		ns	

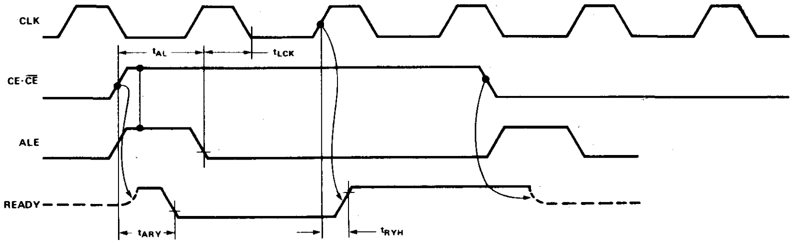


CLOCK SPECIFICATION FOR 8355.

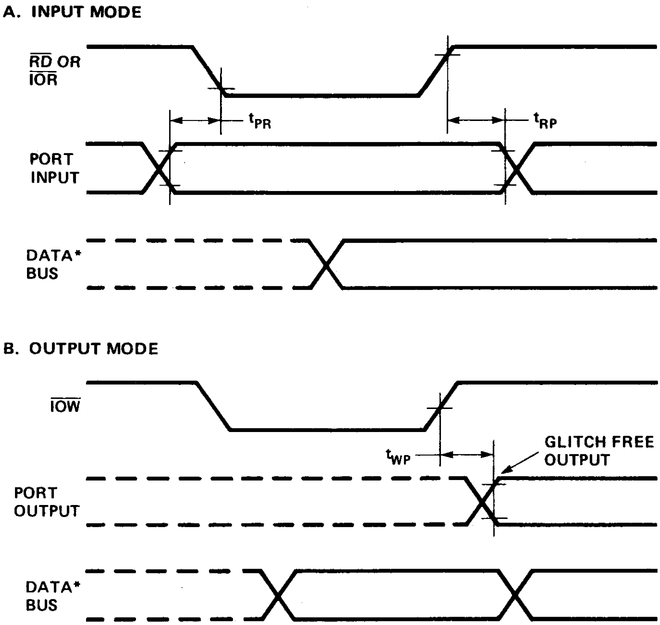


ROM READ AND I/O READ AND WRITE.

8355



WAIT STATE TIMING (READY = 0).



*DATA BUS TIMING IS SHOWN IN FIGURE 3.

I/O PORT TIMING.

8155

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to +70°C
Storage Temperature .... -65°C to +150°C
Voltage on Any Pin
With Respect to Ground .... -0.3V to +7V
Power Dissipation 1.5W

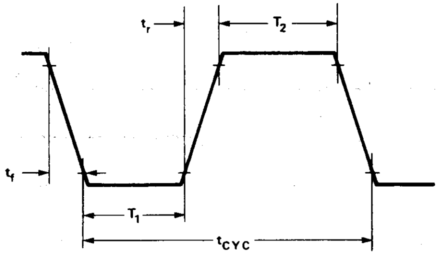
**COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

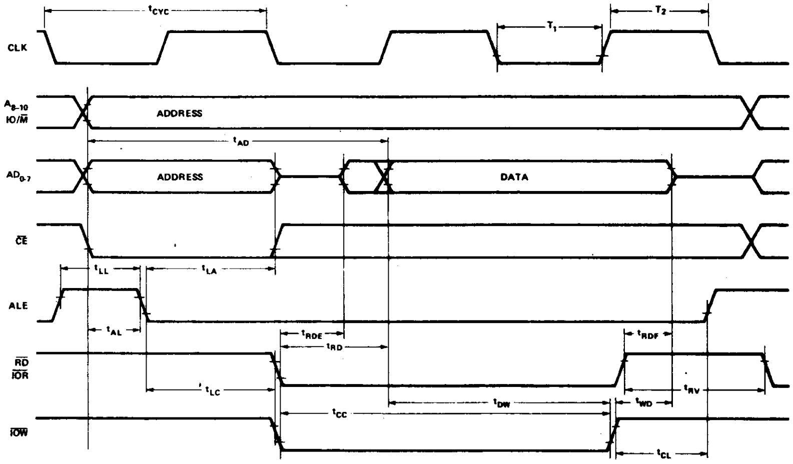
SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		10	μA	V_{IN} = V_{CC} to 0V
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤V_{OUT} ≤V_{CC}
I_{CC}	V_{CC} Supply Current		180	mA	

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

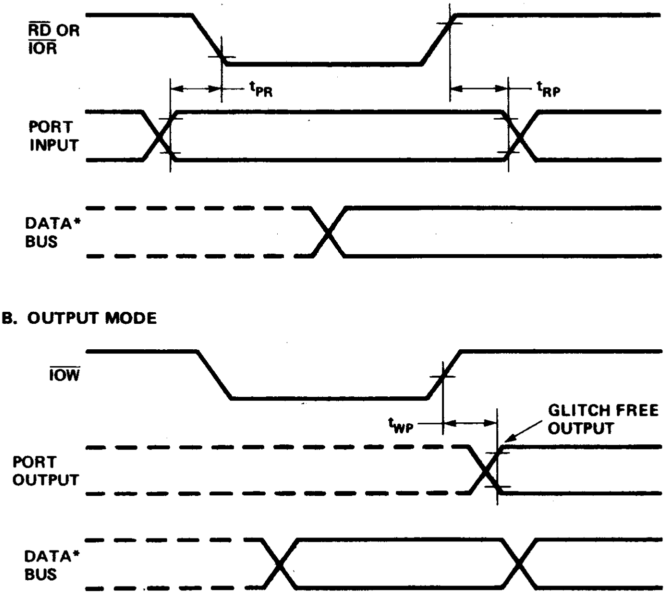
SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
t_{CYC}	Clock Cycle Time	320		ns	C_{LOAD} = 150 pF (See Figure 3)
T_1	CLK Pulse Width	80		ns	
T_2	CLK Pulse Width	120		ns	
t_f, t_r	CLK Rise and Fall Time		30	ns	
t_{AL}	Address to Latch Set Up Time	50		ns	150 pF Load
t_{LA}	Address Hold Time after Latch	80		ns	
t_{LC}	Latch to READ/WRITE Control	100		ns	
t_{RD}	Valid Data Out Delay from READ Control		150	ns	
t_{AD}	Address Stable to Data Out Valid		400	ns	
t_{LL}	Latch Enable Width	100		ns	
t_{RDF}	Data Bus Float after READ	0	100	ns	
t_{CL}	READ/WRITE Control to Latch Enable	20		ns	
t_{CC}	READ/WRITE Control Width	250		ns	
t_{DW}	Data In to WRITE Set Up Time	150		ns	
t_{WD}	Data In Hold Time After WRITE	0		ns	
t_{WP}	WRITE to Port Output		400	ns	
t_{PR}	Port Input Set Up Time	50		ns	
t_{RP}	Port Input Hold Time	50		ns	
t_{RYH}	READY HOLD TIME	0	120	ns	
t_{ARY}	ADDRESS (CE) to READY		160	ns	
t_{RV}	Recovery Time between Controls	300		ns	
t_{RDE}	Data Out Delay from READ Control	10		ns	



CLOCK SPECIFICATION FOR 8755



PROM READ AND I/O WRITE TIMING.

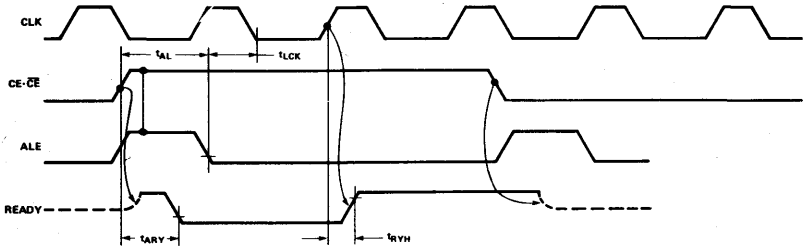


A. INPUT MODE

B. OUTPUT MODE

*DATA BUS TIMING IS SHOWN IN FIGURE 4.

I/O PORT TIMING.



WAIT STATE TIMING (READY = 0).

Chapter 6
THE 8048, 8748 AND 8035

The 8048 series microcomputers are single chip 8-bit devices which have been developed by Intel to compete in the market for low cost, high volume applications. This has been a market where the 8080A, with its high chip counts, does not do well. One version of the 8048, the 8748, is also likely to do exceptionally well in low volume, custom applications because it is very easy to use.

The 8048 looks like a one-chip 8080A with heavy F8 influence. The F8 was the first 8-bit microprocessor to bring the economics of low chip counts to the attention of the semiconductor industry. It is therefore not surprising to find a heavy F8 influence in the 8048.

It is intriguing to note that in terms of general architectural organization, there are striking similarities between the 8048 and the MCS6530 which is described in Chapter 9.

8048 series microcomputers are summarized in Table 6-1.

The only support device described in this chapter is the 8243 I/O Expander. In addition, the 8155, the 8355, and the 8755 multifunction devices which have been described in Chapter 5 can be used with 8048 family microcomputers.

At the present time the only company manufacturing 8048 series microcomputers is:

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

The 8048 series microcomputers use a single +5V power supply. There are two versions of each microcomputer; one uses a 2.5 microsecond clock while the other uses a 5 microsecond clock. 8048 instructions execute in either one or two clock periods.

Table 6-1. A Summary Of 8048 Series Microcomputers

	ON CHIP MEMORY		CYCLE		EXTERNAL	
	ROM/EPROM	RAM	TIME	I/O PORTS	INTERRUPTS	TIMER
8048	1024 ROM	64	2.5μsec	3x8 bits	1	Yes
8035	0	64	2.5μsec	3x8 bits	1	Yes
8035-8	0	64	5.0μsec	3x8 bits	1	Yes
8748	1024 EPROM	64	2.5μsec	3x8 bits	1	Yes
8748-8	1024 EPROM	64	5.0μsec	3x8 bits	1	Yes

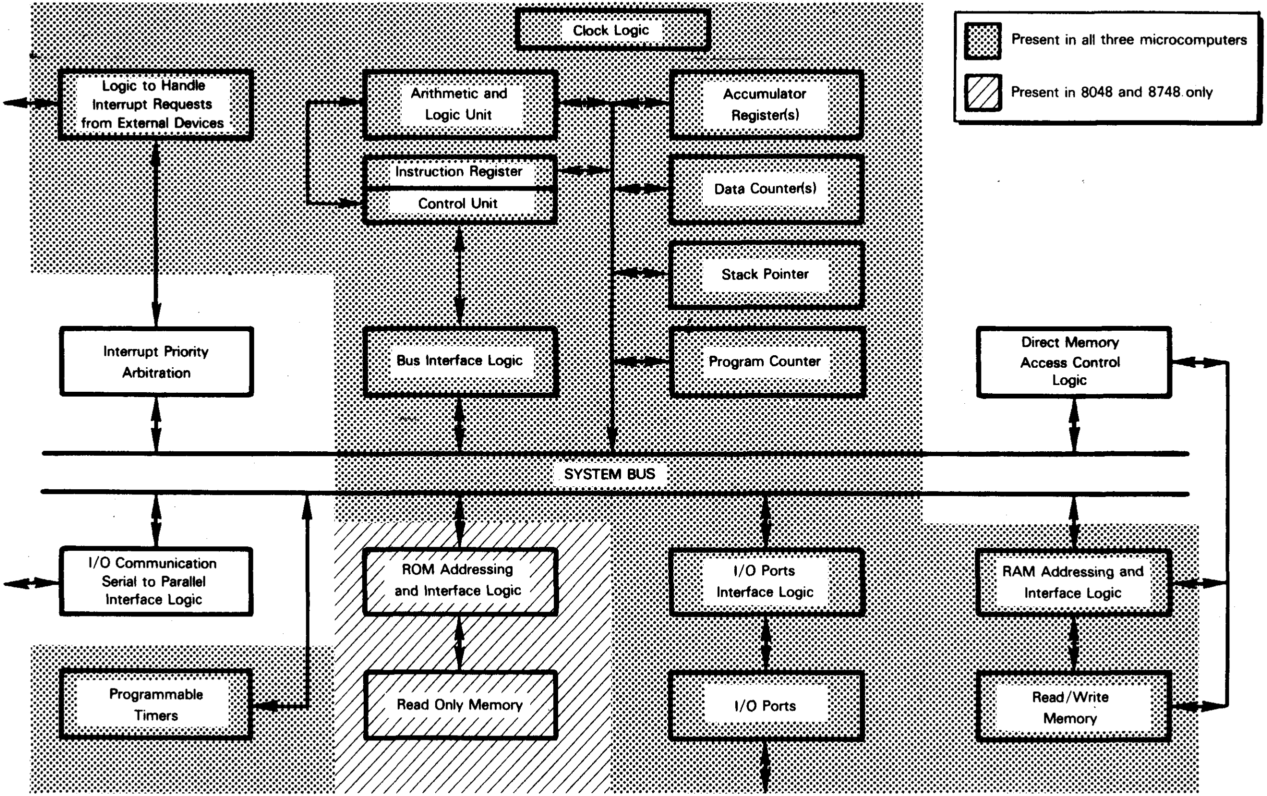

Clock Logic
Present in all three microcomputers
Present in 8048 and 8748 only
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 6-1. Logic Of The 8048, 8748 And 8035 Microcomputers

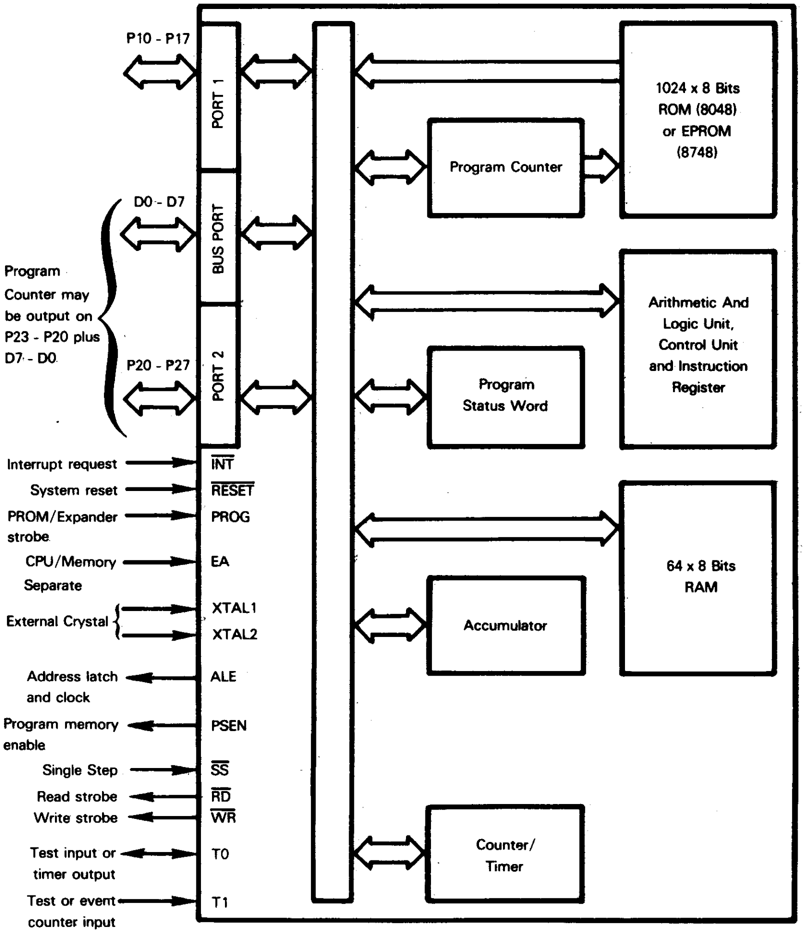


Figure 6-2. Functional Logic Of The 8048, 8748 And 8035 Microcomputers

All 8048 devices are packaged as 40-pin DIPs and have TTL compatible signals.

THE 8048, 8748 AND 8035 MICROCOMPUTERS

Functions implemented on the three versions of the 8048 microcomputer are illustrated in Figure 6-1. With the exception of the 8035 you will see that complete microcomputer logic is provided within a single package. But remember, just because a function is present in Figure 6-1, that does not mean to say it will be sufficient for your application. For example, read/write memory is shown as present; yet there are only 64 bytes of read/write memory on any 8048 series microcomputer chip.

AN 8048 FUNCTIONAL OVERVIEW

Logic of the 8048 series microcomputers is illustrated functionally in Figure 6-2.

The Arithmetic and Logic Unit, the Control Unit and the Instruction register are all inaccessible to you as a user; therefore we will ignore this portion of the microcomputer.

1024 bytes of program memory are provided by the 8048 and 8748 microcomputers; the 8035 has no program memory. The 8048 has 1024 bytes of Read Only Memory (ROM) while the 8748 has 1024 bytes of Erasable Programmable Read Only Memory (EPROM); this is the only difference between the 8048 and 8748.

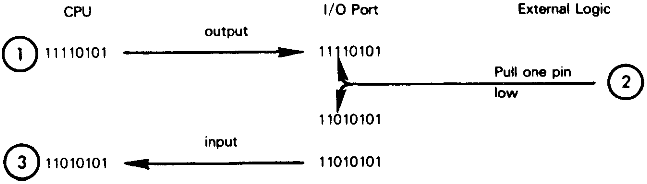
There is a 12-bit Program Counter which allows the 8048 series microcomputers to access 4096 bytes of program memory. Since the 8048 and 8748 microcomputers have only 1024 bytes of program memory on the computer chip, the additional 3072 bytes must be external if you are going to expand program memory to the maximum addressable space. All 8035 microcomputer program memory is external.

8048, 8748 AND 8035 I/O PORTS

All 8048 series microcomputers have three 8-bit I/O ports. One of these, the Bus Port, is a truly bidirectional I/O port with input and output strobes. Outputs can be statically latched while inputs are nonlatching. This means that external logic must hold input data true at Bus Port pins until the data has been read. All eight pins of the Bus Port must be assigned either to input or output; you cannot mix input and output on the Bus Port.

Bus Port is used as the primary I/O port in a single chip microcomputer system. In multiple chip microcomputer systems Bus Port serves as a multiplexed Address and Data Bus.

I/O Ports 1 and 2 are secondary I/O ports with characteristics that differ significantly from Bus Port. If you output parallel data to I/O Port 1 or 2 it is latched and maintained at the I/O port until you next write data. But the only way external logic can input data to I/O Port 1 or 2 is by pulling individual pins from a high to a low level. Thus when a high level is being output at any pin of I/O Port 1 or 2, external logic can pull this level low — and subsequently if the CPU reads back data from the I/O port it will read a 0 bit value. This may be illustrated as follows:



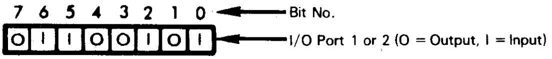
External logic cannot create a high level at any pin of I/O Port 1 or 2 which is outputting a low level.

Here is a summary of I/O Port 1 and 2 capabilities:

1) You can at any time output parallel data to I/O Ports 1 or 2. The data will be latched and held until the next output.

2) Individual pins of I/O Ports 1 and 2 can serve as input or output pins. When you output data to I/O Port 1 or 2 you must output a 1 bit to any input pin. This may be illustrated as follows:

Data Output → X 1 1 X X 1 X 1 (X = 0 or 1)



3) External logic writes to input pins of I/O Ports 1 and 2 by leaving low levels alone, and by pulling high levels low.

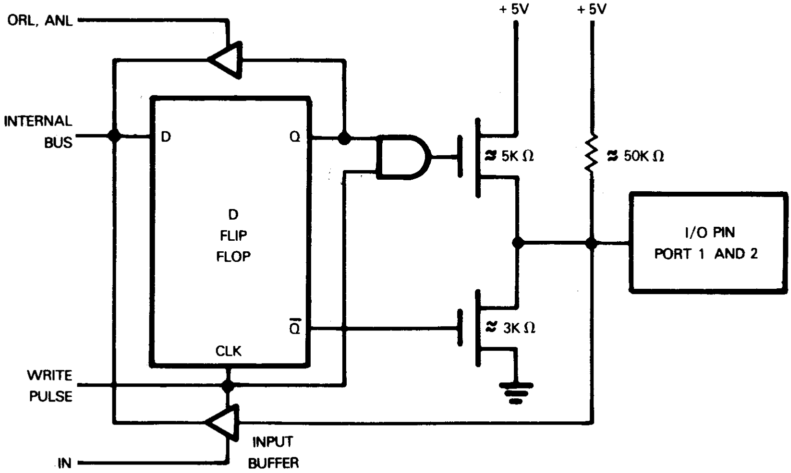


Figure 6-3. 8048 I/O Ports 1 And 2 Pin Logic

8048, 8748 AND 8035 I/O PORT PIN LOGIC

Figure 6-3 illustrates logic associated with each pin of I/O Ports 1 and 2.

Output data is latched by a D-type flip-flop.

The Q and $\overline{Q}$ outputs of the D-type flip-flop control a pair of gates on either side of the pin connection. To provide fast switching times in 0-to-1 transitions, a relatively low impedance (~5K ohms) is switched in for approximately 500 nanoseconds whenever a 1 is output.

Pins are continuously pulled up to +5V through a relatively high impedance (~50K ohms). When a 0 is output to the D-type flip-flop, a low impedance (~3K ohms) overcomes the pull-up and provides TTL current sinking capability.

When a pin of I/O Port 1 or 2 is at a high level, external logic can sink the 50KΩ pull-up. But when the pin is at a low level, external logic cannot overcome the low impedance to ground; thus it cannot pull the pin up to a high level.

By placing an input buffer between the pin and the switching gates, pin logic allows the CPU to read current levels induced by external logic — but only while external logic is connected to the pin.

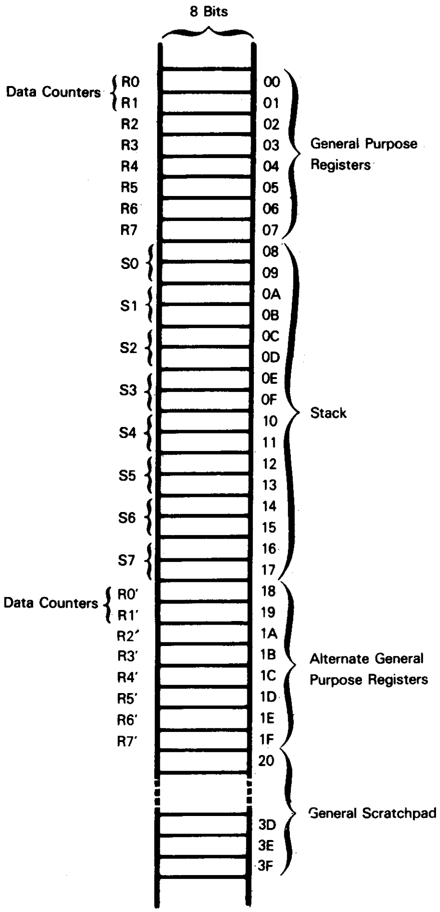
The buffer connecting the Q output of the D-type flip-flop to the D input is present to enable 8048 instructions that mask I/O port data.

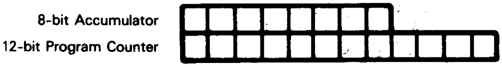
Later in this chapter we will look at I/O ports in more detail, showing programming and design examples.

8048, 8748 AND 8035 MICROCOMPUTER PROGRAMMABLE REGISTERS

The 8048 series microcomputers have an 8-bit Accumulator, a 12-bit Program Counter and 64 bytes of scratchpad memory. Scratchpad memory may be visualized either as read/write memory or as General Purpose registers.

The Accumulator, Program Counter and scratchpad memory may be illustrated as follows:





The Accumulator is the principal conduit for all data transfers. The Accumulator is always one source and the destination for Arithmetic or Boolean operations involving memory or registers.

Two sets of eight scratchpad bytes serve as secondary registers. At any time one set of General Purpose registers is selected while the other set of General Purpose registers is not selected.

The first two General Purpose registers of each set, R0 and R1, act as Data Counters to address scratchpad memory and external data memory. Thus you address scratchpad memory using implied memory addressing via General Purpose Register R0 or R1; you can address any one of the 64 scratchpad bytes, including the General Purpose registers, or even the Data Counter register itself.

In between the two sets of eight General Purpose registers there is a 16-byte stack. The Stack Pointer is maintained in the Program Status Word; therefore we will defer our discussion of stack operations until we look at status.

8048, 8748 AND 8035 ADDRESSING MODES

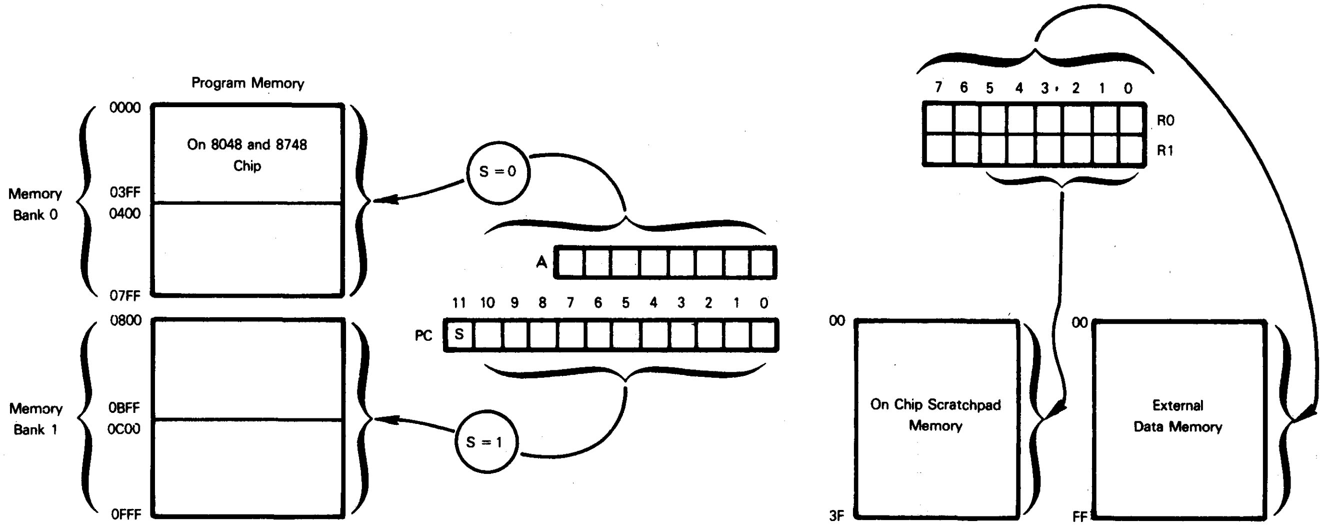
8048, 8748 AND 8035 MEMORY SPACES

The 8048 series microcomputers separate memory into program memory and data memory. Without resorting to complex expansion schemes, you are limited to a maximum of 4096 program memory bytes and 320 data memory bytes.

The 8048 and 8748 microcomputers have 1024 bytes of program memory on the CPU chip; more program memory, if present, must be external to the CPU chip. The 8035 microcomputer has no on-chip program memory; it requires all program memory to be external.

All 8048 series microcomputers provide 64 bytes of read/write data memory on the CPU chip. In addition, 256 bytes of external data memory may be addressed. **The external data memory space must be shared by external data memory and any external I/O ports** — that is to say, I/O ports other than the microcomputer's own three I/O ports or 8243 Expander ports.

8048 series microcomputer address spaces and addressing modes are illustrated in Figure 6-4.



A = Accumulator

PC = Program Counter

R0, R1 are general purpose registers
in scratchpad memory

Figure 6-4. 8048 Series Microcomputers' Memory Addressing

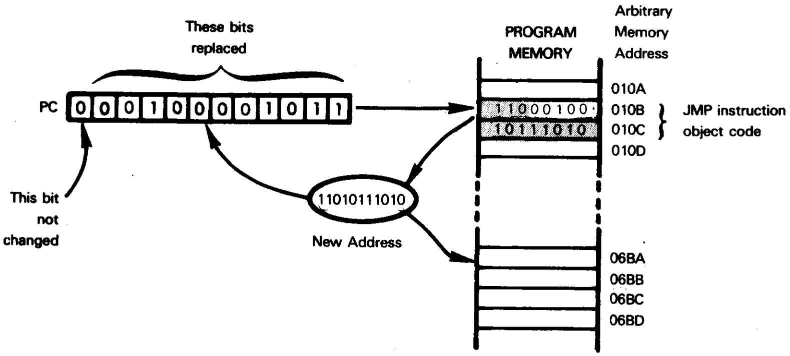
Let us first examine program memory addressing.

8048, 8748 AND 8035 PROGRAM MEMORY ADDRESSING

A single address space is used to access all of program memory. In the normal course of events program memory is addressed via the 12-bit Program Counter. The high order Program Counter bit is isolated in Figure 6-4 because when the Program Counter is incremented only bits 0 through 10 are affected. You must execute special instructions to modify the contents of the high order Program Counter bit. Program memory is therefore effectively divided into two memory banks, each containing up to 2048 bytes of program memory. You cannot branch, via Jump-on-Condition instructions, from one program memory bank to the other, nor can instructions stored in one program memory bank access the other. You can switch completely from one program memory bank to the other, with JMP, CALL or RET, and that is all.

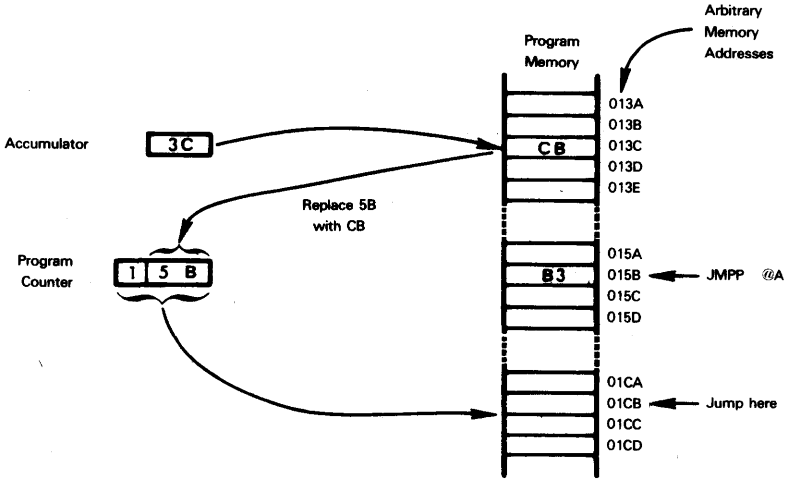
Two types of program memory addressing are available: you can read data from program memory and you can execute Jump instructions.

You can unconditionally jump anywhere within the currently selected program memory bank; this may be illustrated as follows:

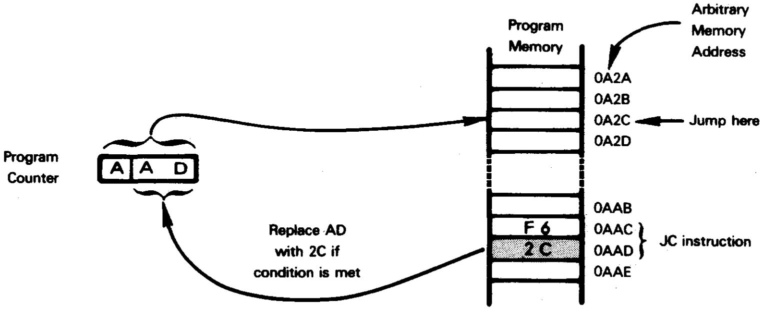


Thus the JMP instruction stored in program memory bytes 010B$_{16}$ and 010C$_{16}$ causes program execution to jump to location 06BA$_{16}$.

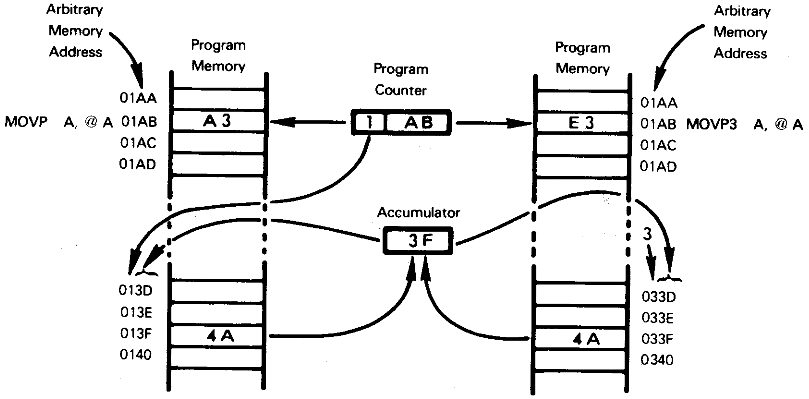
You can also jump using a form of paged, indirect addressing, where the Accumulator points to an indirect address stored in the current page of program memory. This may be illustrated as follows:



All conditional Jump instructions allow you to branch within the current page of program memory only. This may be illustrated as follows:

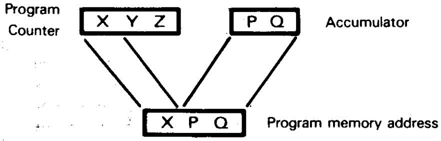


You can read data from program memory, but there are no instructions which allow you to write data to program memory. Instructions, other than data immediate instructions, that read data from program memory use paged, implied addressing. There are two forms of paged implied programming memory addressing; they may be illustrated as follows:

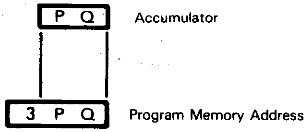


The illustration above compares execution of the MOVP and MOVP3 instructions. These are the two instructions which allow you to read a byte of data from program memory into the Accumulator.

When the MOVP instruction is executed, the program memory address is formed by concatenating the high order four bits of the Program Counter with the contents of the Accumulator:



When the MOVP3 instruction is executed, the program memory address is computed by appending the Accumulator contents to 0011:



Thus the MOVP instruction loads into the Accumulator the contents of a program memory byte within the current program page. The MOVP3 instruction loads into the Accumulator the contents of a byte from program memory page 3.

Note carefully that the extensive use of absolute paged addressing of program memory carries with it the usual page boundary problems. The program memory addressing modes which replace the low order eight Program Counter bits keep the four high order Program Counter bits — after the Program Counter has been incremented.

Refer back to the JMPP @A instruction. This instruction is illustrated as being stored in program memory location $011B_{16}$. But suppose this instruction were stored in memory location $01FF_{16}$; then after the JMPP instruction is fetched, the Program Counter will no longer contain $01FF_{16}$, it would contain 0200_{16}. Now instead of jumping to program memory location $01CB_{16}$, you would jump to program memory location $02CB_{16}$.

This page boundary problem is common to all microcomputers that use absolute paged addressing. For a complete discussion of this problem refer to Volume I — Basic Concepts, Chapter 6.

Note that the 8048 has no instructions which write into program memory. If you want to write into program memory you must have external logic which overlaps external program and data memory.

Let us now look at data memory addressing. First of all, notice that scratchpad memory and external data memory have overlapping address spaces. Separate and distinct instructions access scratchpad memory as against external data memory. External data memory does not represent a continuation of scratchpad memory. For example, there will be memory bytes with addresses in the range 00_{16} through $3F_{16}$ in the scratchpad and in external data memory.

Implied memory addressing is the only addressing mode available to you when accessing data memory.

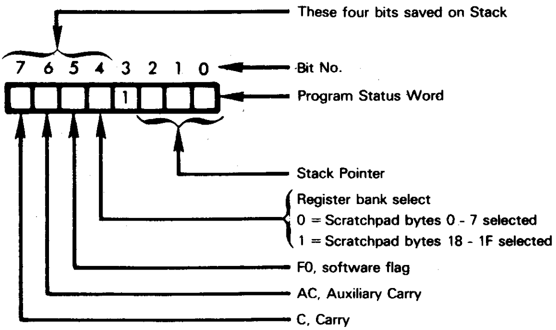
Instructions that access scratchpad memory take the scratchpad memory byte address from the low order six bits of General Purpose Register R0 or R1.

Instructions that access external data memory take the external data memory address from all eight bits of General Purpose Register R0 or R1.

The eight General Purpose registers within scratchpad memory can be addressed directly. We could argue that this constitutes a limited scratchpad memory direct addressing capability; but in order to remain consistent with other microcomputers described in this book, we will classify these direct accesses of General Purpose registers as register-to-register operations rather than direct addressing of data memory.

8048, 8748 AND 8035 STATUS

8048 series microcomputers have an 8-bit Program Status Word which may be illustrated as follows:



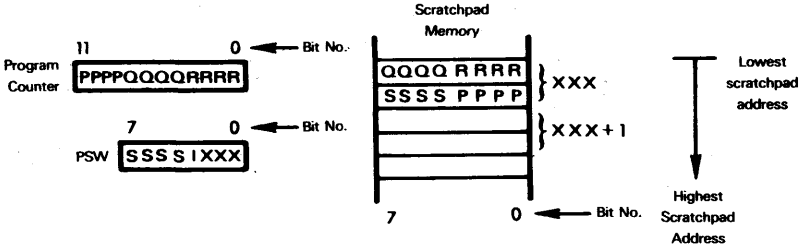
C and AC are the standard Carry and Auxiliary Carry statuses as defined in Volume I and used throughout this book.

FO is a flag which you set or reset using appropriate Status instructions. A conditional Jump instruction tests the level of FO. FO is not connected to external logic and cannot be modified or tested by external logic.

BS identifies which set of General Purpose registers is currently selected. If BS is 0, then scratchpad bytes 0 through 7 are serving as General Purpose registers. If BS is 1, then scratchpad bytes 18_{16} through $1F_{16}$ are serving as General Purpose registers.

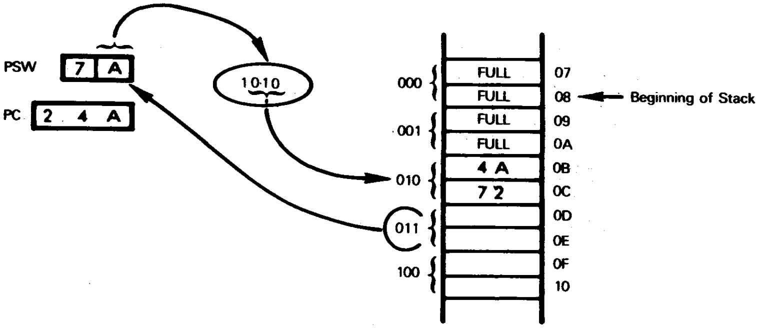
The low order three Program Status Word bits serve as a Stack Pointer. The 16 Stack bytes are treated as eight 16-bit registers, with the current top of Stack identified by the three low order Program Status Word bits.

A subroutine Call instruction pushes the Program Counter contents and the four high order Program Status Word bits onto the Stack as follows:



In the illustration above P, Q, R, S and X represent any binary digits.

Observe that the beginning of the Stack has the lowest scratchpad address. The order in which Program Status Word and Program Counter contents are pushed onto the Stack is illustrated above. Here is a specific case:



You need to know the exact order in which data is stored on the Stack since the Stack is also accessible as general scratchpad memory.

There are two Return-from-Subroutine instructions; one restores Program Counter contents only, the other restores Program Counter and Program Status Word contents.

Since the Stack has eight 16-bit registers, subroutines may be nested eight deep. If you are using interrupts, then the combined total of subroutine nesting levels on either side of the interrupt must sum to 7 or less. For example, if the interrupt service routine nests subroutines to a maximum level of 3, then non-interrupt programs cannot nest subroutines to a level greater than 4. The interrupt itself requires one Stack location.

8048 SERIES MICROCOMPUTER OPERATING MODES

8048 series microcomputers can operate in a variety of modes. Many signals serve more than one function, depending on the operating mode.

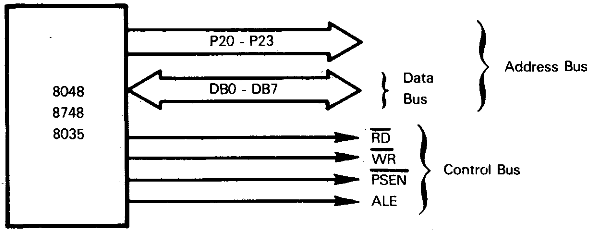
In order to clarify this potentially confusing subject, we will summarize 8048 series operating modes in the paragraphs below, then we will summarize device signals; these two summaries are followed by an in-depth analysis of operating modes, illustrating timing and signal functions.

8048 AND 8748 INTERNAL EXECUTION MODE

Internal execution mode is the simplest case; the 8048 or 8748 microcomputers normally operate in Internal Execution mode, at which time they execute programs without accessing external program memory or data memory. All information transfer with external logic occurs via I/O ports or control signals. The 8035, having no internal program memory, cannot operate in Internal Execution mode.

8048, 8748 AND 8035 EXTERNAL MEMORY ACCESS MODE

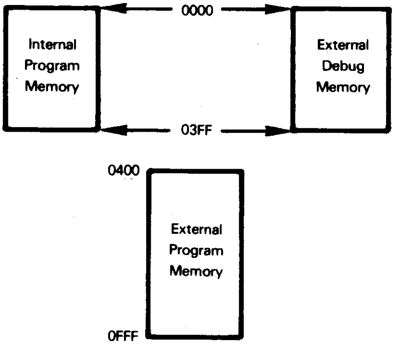
Having external program memory and/or data memory causes the microcomputer to output additional control signals which identify external program and data memory accesses. This is External Memory Access mode. Memory addresses are output via the Bus Port and four pins of I/O Port 2; bidirectional data transfers occur via the Bus Port. This may be illustrated as follows:



External Memory Access mode represents the simplest case for the 8035 microcomputer, which has no on-chip program memory.

8048 AND 8748 DEBUG MODE

The 8048 and 8748 microcomputers can be operated in Debug mode. **In Debug mode the CPU is disconnected from its internal program memory.** All program memory accesses are deflected to external program memory. This may be illustrated as follows:



You will use Debug mode to test microcomputer systems built around an 8048 series microcomputer. Typically special purpose test and verify programs will be maintained in external debug memory.

8048, 8748 AND 8035 SINGLE STEPPING

Single stepping is not really a mode, but is worth mentioning in connection with Debug mode since it is a powerful debugging tool. In any of the operating modes you can apply a Single Step signal ($\overline{SS}$) which halts instruction execution following the next instruction fetch. This allows you to execute programs one instruction at a time in order to locate errors or gain a better understanding of event sequences.

8748 PROGRAMMING MODE

The 8748 microcomputer contains Erasable Programmable Read Only Memory (EPROM). **In Programming mode you can program the 8748 EPROM.**

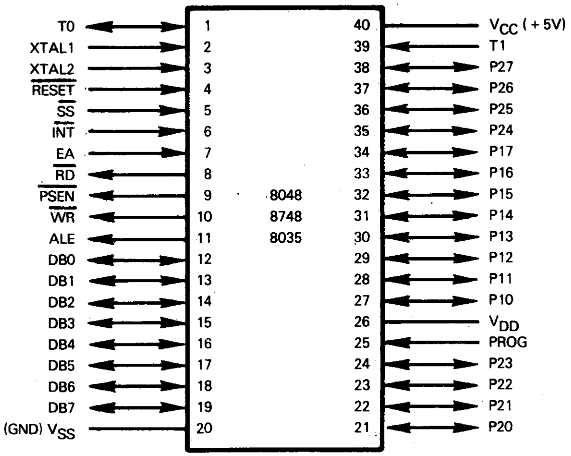
8048, 8748 AND 8035 VERIFY MODE

Finally there is a Verify mode. **In Verify mode you can read the contents of internal or external program memory as data.** Verify mode is used in conjunction with Programming mode to test data written into EPROMs. Verify mode can also be used on its own to examine the contents of program memory for any 8048 series microcomputer.

8048, 8748 AND 8035 CPU PINS AND SIGNALS

Figure 6-5 illustrates pins and signals for the 8048, 8748 and 8035 microcomputers. We will briefly summarize functions performed by signals before discussing how signals are used in different modes.

DB0 - DB7 serves both as a bidirectional I/O port, a static latch and as a multiplexed Address and Data Bus. When no external data or program memory accesses are occurring, DB0 - DB7 serves as a simple bidirectional I/O port or latch. During external program or data memory accesses DB0 - DB7 serves as a bidirectional Data Bus as well as outputting the low order eight bits of all memory addresses. Data outputs are not latched in bidirectional mode. External logic must hold input signal levels until the CPU has read input data.



PIN NAME	DESCRIPTION	TYPE
DB0 - DB7	Bidirectional I/O port, Data Bus and low order eight Address Bus lines	Bidirectional, tristate
P10 - P17	I/O Port 1	Quasibidirectional
P20 - P27	I/O Port 2. P20 - P23 also serves as four high order Address Bus lines	Quasibidirectional
ALE	External clock signal and address latch enable	Output
$\overline{\text{RD}}$	Data memory read control	Output
$\overline{\text{WR}}$	Data memory write control	Output
$\overline{\text{PSEN}}$	External program memory read control	Output
EA	External program memory access	Input
$\overline{\text{SS}}$	Single step control	Input
T0	Test input, optional clock output and Program/Verify mode select	Bidirectional
T1	Test input, optional event counter input	Input
$\overline{\text{RESET}}$	System reset and EPROM address latch	Input
V_{SS}	Ground	
V_{CC}	+5V	
V_{DD}	+25V to program 8748. +5V standby for 8048 RAM	
PROG	+25V input to program 8748. Control output for 4-bit I/O	Bidirectional
XTAL1, XTAL2	External crystal connections	

Figure 6-5. 8048, 8748 And 8035 Microcomputer Pins And Signals

P10 - P17 and P20 - P27 support I/O Ports 1 and 2 respectively. We described the characteristics of these two I/O ports earlier in this chapter. During external accesses of program memory the four high order address lines are output via P20 - P23.

ALE is a control signal which is pulsed high at the beginning of every instruction execution machine cycle. This signal may be used as a clock by external logic. During external memory accesses the trailing edge of ALE strobes memory addresses being output.

$\overline{\text{RD}}$ is a control signal which is **pulsed low to strobe data** from external data memory **onto the Data Bus.**

$\overline{\text{WR}}$ is a control signal which is **strobed low** when external data memory is **to read data off the Data Bus.**

$\overline{\text{PSEN}}$ is a control signal which is **strobed low** when external program memory is **to place data on the Data Bus.**

External logic inputs EA high in order to separate the CPU from internal program memory and force the microcomputer into **Debug mode.**

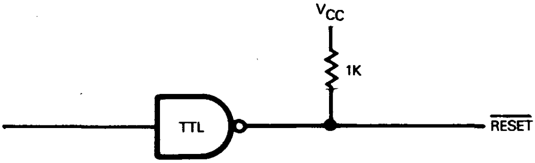
$\overline{\text{SS}}$ is input low in order to stop instruction execution following an instruction fetch; this allows you **to single step through a program.**

T0 is a test input which may be sampled by a conditional Jump instruction. **T0 is also used while selecting Program mode and Verify mode. The internal CPU clock signal can be output via T0.**

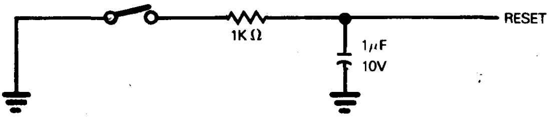
T1 is a test input which can be sampled by a Jump-on-Condition instruction. **T1 can also be used to input a signal to Counter/Timer logic** when it is serving as an event counter.

8048, 8748 AND 8035 RESET

$\overline{\text{RESET}}$ is a standard system reset input signal. The normal $\overline{\text{RESET}}$ signal should be output from an open collector or active pull-up:



The power-on RESET should be generated as follows:



There is an internal pull-up resistor which, in combination with an external 1µF capacitor, generates an adequate internal RESET pulse. If the RESET pulse is generated externally, then it must be held below 0.5V for at least 50 milliseconds.

This is what happens when you reset an 8048 series microcomputer:

1) The Program Counter and the Program Status Word are cleared. This selects register bank 0 and program memory bank 0. Also the first instruction executed following a Reset will be fetched from program memory location 0.
2) The Bus Port is floated.
3) I/O Ports 1 and 2 are set to Input mode.
4) The timer and external interrupts are disabled.
5) The Counter/Timer is stopped and T0 is disconnected from the timer.
6) The timer flag and internal flags F1 and F0 are cleared.

An external crystal, if present, **is connected across XTAL1 and XTAL2.** Typically a 6 MHz crystal will be used. **You can input a clock signal directly to XTAL1.** If you do, the input clock signal should have a frequency in the range of 1 MHz to 6 MHz.

The 8048 series microcomputers use power supplies in a number of interesting ways.

V_{CC} is the standard +5V power supply. V_{SS} is the standard ground connection.

V_{DD} is an additional +5V standby power supply. This standby power supply will maintain the contents of scratchpad memory when all other power has been removed. Typically V_{DD} will be connected to a battery so that when the system is powered down data can be preserved in scratchpad memory (8048 only).

The 8748 microcomputer uses V_{DD} and PROG in order to program the EPROM. While programming the EPROM, a voltage of +25V is input at V_{DD} +25V pulses lasting 50 milliseconds are input at PROG. A single byte of program memory will be written during a single PROG +25V pulse.

PROG serves as a control strobe output to the 8243 Input/Output Expander during the execution of instructions that reference the Expander ports. This function of PROG is described in more detail later in this chapter, when we describe the 8243 I/O Expander.

8048 SERIES TIMING AND INSTRUCTION EXECUTION

Let us begin our detailed analysis of 8048 series microcomputer operations by looking at basic instruction timing.

A master clock signal must be input via XTAL1, or the clock signal may be generated internally by connecting a crystal across XTAL1 or XTAL2. A 6 MHz crystal is recommended. **This clock signal is divided by 3 to generate a master synchronizing 2 MHz signal** which is used throughout the microcomputer system. **You can output this 2 MHz clock signal via the T0 pin.**

All -8 versions of 8048 series microcomputers operate at half speed; they use 3 MHz crystals and generate a 1 MHz master synchronizing signal.

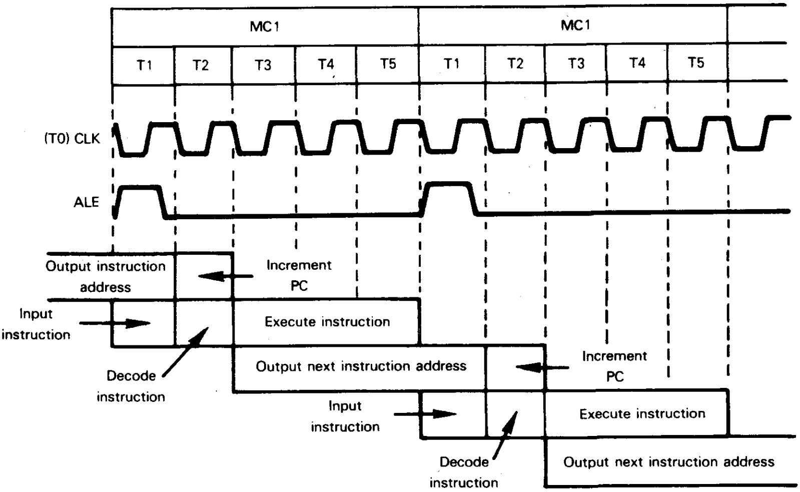


Figure 6-6. Execution Of 8048 Single Machine Cycle Instructions Without Any External Access

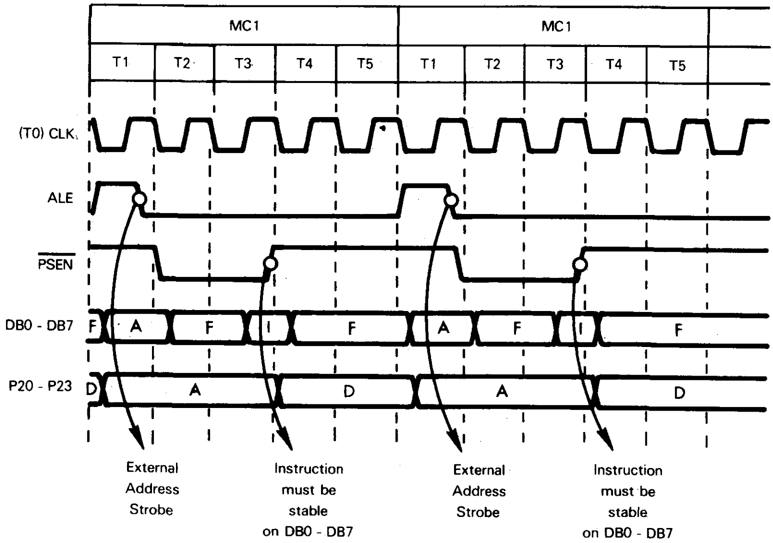
8048, 8748 AND 8035 MACHINE CYCLES AND CLOCK PERIODS

Instructions execute in machine cycles. Every machine cycle has five clock periods. Using a 2 MHz clock signal, therefore, each machine cycle will last 2.5 microseconds. Instructions execute in either one or two machine cycles.

INTERNAL EXECUTION MODE

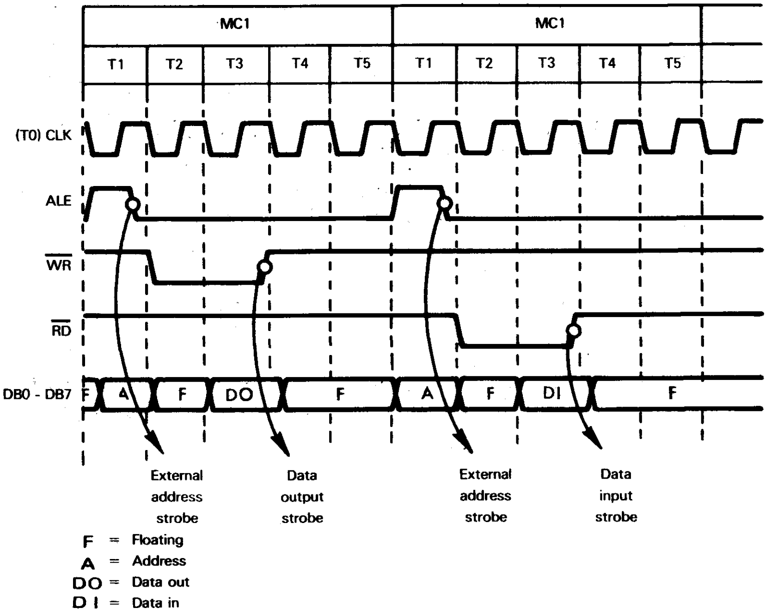
Figure 6-6 illustrates timing for the simplest case — **execution of a** single machine cycle **instruction accessing internal program or data memory only.** The only signal change seen beyond the microcomputer chip itself is the ALE pulse — and the CLK signal, if you elect to output it via T0. The events which occur during each clock period are illustrated in Figure 6-6; but remember, these operations are internal to the microcomputer. They are beyond your access or control.

Figure 6-6 also illustrates timing for instructions that execute in two machine cycles, but access only program and/or data memory internal to the microcomputer chip. Once again external logic sees ALE, and optionally CLK



F = Floating
A = Address
I = Instruction Code
D = I/O Data

Figure 6-7. An 8048, 8748 Or 8035 External Instruction Fetch



These two machine cycles would never occur in the sequence illustrated.
They are shown together for comparison only.

Figure 6-8. An 8048, 8748 Or 8035 External Data Read Or Write

EXTERNAL MEMORY ACCESS MODE

Now consider external program and data memory accesses.

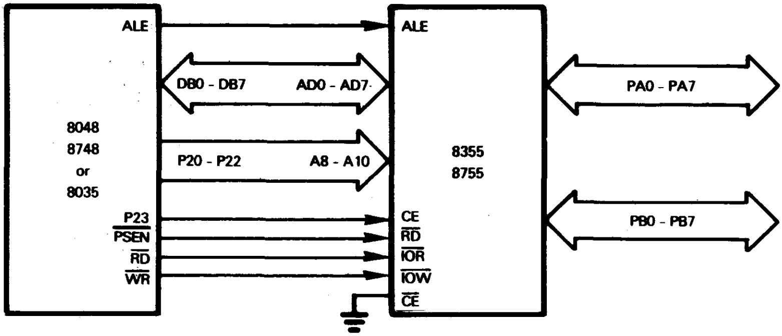
Figure 6-7 illustrates timing for an external program memory read. The external program memory address is output via DB0 - DB7 (low order eight address lines) and P20 - P23 (high order four address lines). The address is maintained stable just long enough for external logic to latch it on the high-to-low transition of ALE.

The low $\overline{\text{PSEN}}$ pulse serves as an external program memory read strobe. While $\overline{\text{PSEN}}$ is low, external program memory must decode the latched address and place the contents of the addressed memory byte on the DB0 - DB7 lines. The microcomputer will read DB0 - DB7 on the trailing (low-to-high) transition of $\overline{\text{PSEN}}$.

Timing associated with reading data from external data memory and writing to external data memory is illustrated in Figure 6-8. Timing is very similar to the external instruction fetch illustrated in Figure 6-7. Instead of $\overline{\text{PSEN}}$ being pulsed low, $\overline{\text{RD}}$ is pulsed low to strobe data input; $\overline{\text{WR}}$ is pulsed low to strobe data output. Since the total external data memory address space is 256 bytes, the complete address is transmitted via DB0 - DB7; thus P20 - P23 is inactive during an access of external data memory.

Note that the 8048 series microcomputers have no Wait state. External memory must therefore respond to read or write operations within the allowed time. This is not much of a problem since 8048 series microcomputers operate relatively slowly; most standard memory devices will have no trouble meeting timing requirements. If you want to use slower memories, use the slower 5 microsecond machine cycle versions of the 48 microcomputers.

8048 WAIT STATE



Signals not directly involved in the 8048-8355 interface are not shown.

Figure 6-9. An 8048-8355 Configuration

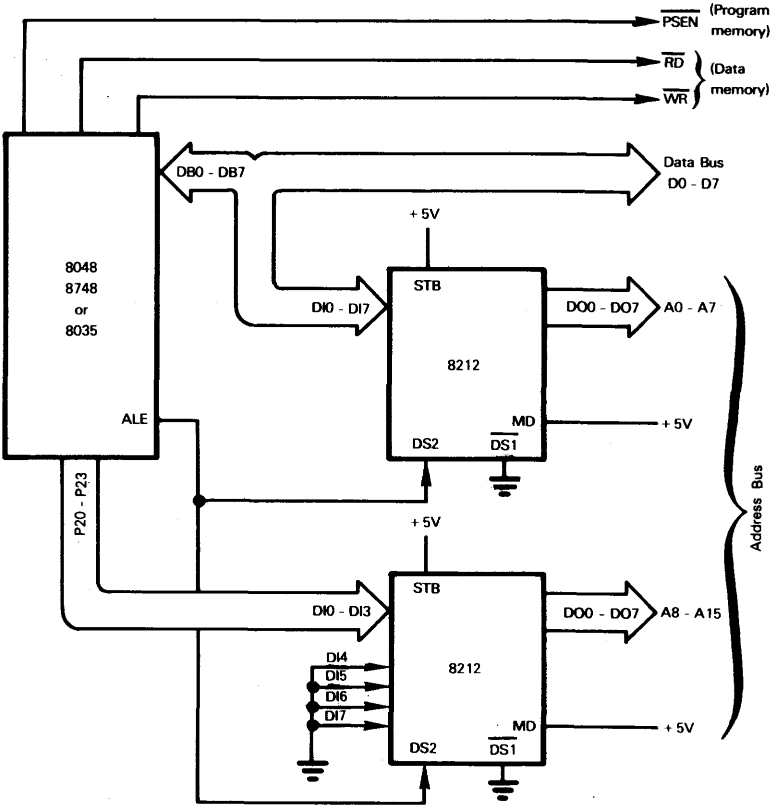


Figure 6-10. Demultiplexing DB0 - DB7 To Create Separate Address And Data Busses

Let us examine microcomputer configurations that include external memory.

Vendor literature illustrates complex microcomputer systems built around 8048 series microcomputers; while such large microcomputer systems are certainly feasible, they are not advisable. If you are going to expand an 8048 series microcomputer system to more than two or three devices, in all probability an 8085 system would be more economical and powerful — not to mention a number of other microcomputers described in this book. We will therefore confine ourselves to illustrating 2- and 3-chip configurations.

Figure 6-9 illustrates an 8048-8355 (or 8755) configuration. The 8355 (or 8755) is a multifunction support device described in Chapter 5.

Figure 6-10 shows how you can connect standard memory devices to an 8048 series microcomputer.

Let us examine Figure 6-9. The 8048 Bus Port is directly compatible with AD0 - AD7 the multiplexed Data and Address Bus of the 8355 device.

The three high order address lines required by the 8355, A8, A9 and A10, are taken directly from P20, P21 and P22. P23, the high order address line output by the 8048, is used to enable the 8355. As shown in Figure 6-9 this means the 8355 will respond to addresses in program memory bank 1. If you are using an 8035 microcomputer, then P23 could be connected to the $\overline{CE}$ enable pin of the 8355; now the 8355 will respond to addresses in program memory bank 0. It would make little sense having the 8355 respond to addresses in program memory bank 0 when using an 8048 or 8748, because the first 1024 bytes of program memory are internal to these microcomputers; that means the first 1024 bytes of 8355 memory would never be accessed.

8355 OR 8755 CONNECTED TO AN 8048, 8748 OR 8035

Control signals needed to read data out of 8355 program memory are easily derived. The 8048 ALE output is exactly what is needed for the 8355 ALE input. The memory strobe $\overline{RD}$ required by the 8355 is adequately generated by the $\overline{PSEN}$ output of the 8048.

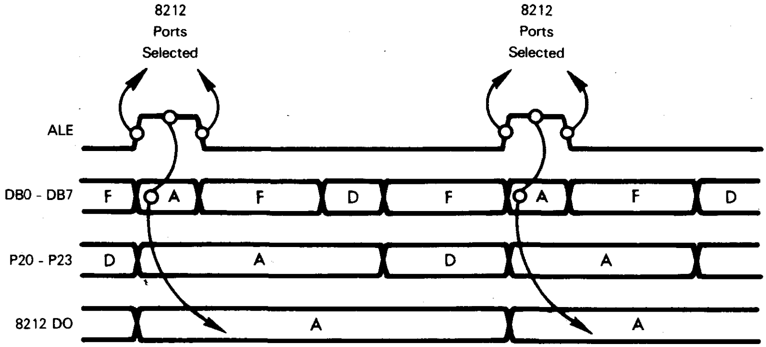
You can also access the 8355 I/O ports by connecting the $\overline{RD}$ and $\overline{WR}$ outputs of the 8048 to the $\overline{IOR}$ and $\overline{IOW}$ inputs of the 8355; the $\overline{IOR}$ and $\overline{IOW}$ control inputs of the 8355 were specifically designed for this purpose. $\overline{RD}$ and $\overline{WR}$ control signals are generated by the 8048 series microcomputers in order to access data memory external to the microcomputer device itself. Thus the I/O ports of the 8355 device must be accessed within the address space of external data memory. In Figure 6-9 external data memory addresses 0, 1, 2 and 3 will access the 8355 I/O ports — and their respective Data Direction registers. Of course, the 8355 I/O ports can be accessed only while the 8355 is selected — via a high CE input.

In order to attach standard memory devices to an 8048 series microcomputer you must demultiplex the DB0 - DB7 lines to create separate Data and Address Busses. Figure 6-10 shows how to do this using two 8212 I/O ports. 8212 I/O port operations are described in Chapter 4. In Figure 6-10 the 8212 I/O ports are being used as simple output ports without handshaking. By tying STB and MD high the 8212 I/O ports will output whatever is

STANDARD MEMORY DEVICES CONNECTED TO AN 8048, 8748 OR 8035

being input while the device is selected. We use the ALE signal to complete selection of the 8212 I/O ports; thus while ALE is high the two ports are selected.

Timing may be illustrated as follows:



Thus the 8212 ports output DB0 - DB7 or P20 - P23 levels latched while ALE is high. Once ALE goes low 8212 port outputs remain constant.

But there are a few subtleties associated with Figure 6-10.

When an 8048 series microcomputer is accessing external program memory, a 12-bit address is output via DB0 - DB7 and P20 - P23; therefore the entire Address Bus is needed as illustrated. A low $\overline{\text{PSEN}}$ pulse serves as the external memory read strobe.

When 8048 series microcomputers access external data memory, however, only DB0 - DB7 is affected. Thus the second 8212 I/O port creates address lines A8 - A15 which will carry the most recent data output to I/O Port 2 — for example, you may set all I/O Port 2 pins to 0 during initialization. If I/O Port 2 is undefined, spurious selection of program memory will result in configurations that include external program and data memory. At the time ALE is output as a high pulse no other signals indicate whether the subsequent memory access will involve program memory or data memory. It is only the separate control strobes — $\overline{\text{PSEN}}$ for program memory, $\overline{\text{WR}}$ and $\overline{\text{RD}}$ for data memory — that insure the correct memory module will be accessed. If your 8048 program uses I/O Port 2 for data output as well as for external memory addressing, you should buffer the System Bus; make sure, in this case, that the System Bus has sufficient capacity to handle two selected memory devices simultaneously.

Even though two memory devices may be selected simultaneously, you will not run into memory access contentions since program memory is strobed by $\overline{\text{PSEN}}$ while data memory is strobed by $\overline{\text{RD}}$ and $\overline{\text{WR}}$. Only one of these signals will be active at any time.

DEBUG MODE

You can bypass program memory internal to the 8048 or 8748 by inputting a high signal at EA. While EA is high, timing for all program memory accesses will conform to external program memory accesses as illustrated in Figure 6-7. You may change the level of EA only when $\overline{\text{RESET}}$ is low; that is, you cannot switch between internal and external memory during program execution.

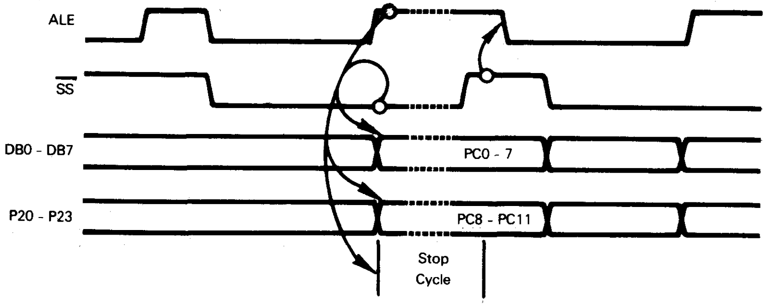
Here is one of the ways in which you may use Debug mode:

In user end products an external memory device may contain test and verify programs. A service representative will execute these test and verify programs by applying a high input at EA. For example, you could connect an 8355 multifunction device to the 8048, selecting it via program memory bank 0. If EA is taken out to a switch, a serviceman will be able to execute programs out of the first 1024 bytes of 8355 program memory, instead of internal 8048 or 8748 memory.

EA is also used by programming and verification modes. This use of EA, however, has nothing to do with Debug mode.

SINGLE STEPPING

If you input a low signal at $\overline{SS}$, then when ALE next pulses high, it will stay high until $\overline{SS}$ returns high. While ALE is high, instruction execution ceases and the current Program Counter contents are output via DB0 - DB7 and P20 - P23. Timing may be illustrated as follows:



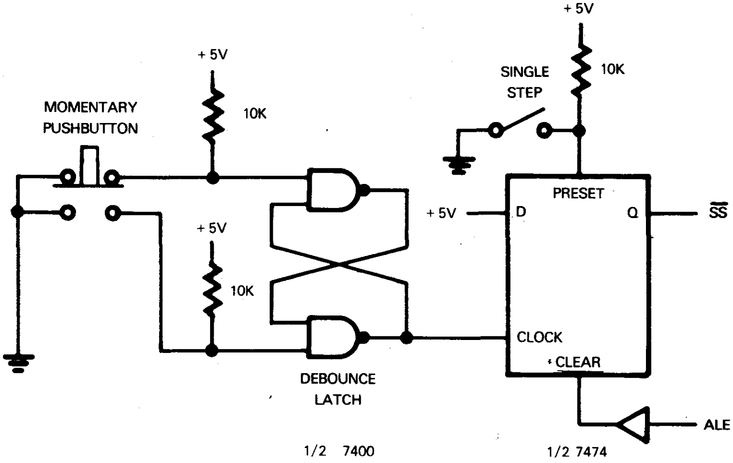


Figure 6-11. An 8048 Single Step Circuit

The CPU only tests $\overline{SS}$ level while ALE is high. At other times $\overline{SS}$ level is irrelevant.

Single stepping is an 8048 program debugging aid. Intel literature suggests the circuit illustrated in Figure 6-11 to create an $\overline{SS}$ signal that is initiated by an ALE pulse and terminated by a push button.

If you do not wish to single step, then connecting the Single Step switch in the Run position will hold PRESET at ground, which forces the Q output high; instructions will execute normally. With the Single Step switch in the Single Step position, PRESET is held high; now the ALE input to $\overline{CLEAR}$ becomes active. As soon as ALE goes low the Q output is also driven low; thus $\overline{SS}$ is low. The low $\overline{SS}$ is detected on the next high ALE pulse, at which time ALE remains high and the cycle is stopped. This condition persists until the push button is depressed. Depressing the push button creates a low-to-high clock transition which forces $\overline{SS}$ high — thus terminating the stopped condition. **You, as a user, will see a program advance one instruction every time you press the push button.**

While the 8048 is stopped in a single step, the current Program Counter contents are output via the Bus Port (DB0 - DB7) and P20 - P23. The Bus Port output presents no problem since you would expect to see address information output at this time. But if I/O Port 2 is being used as a regular I/O port, then prior data present on lines P20 - P23 will not be available during the address output. **Thus if you wish to view I/O data output while single stepping, you must latch I/O Port 2 data externally.**

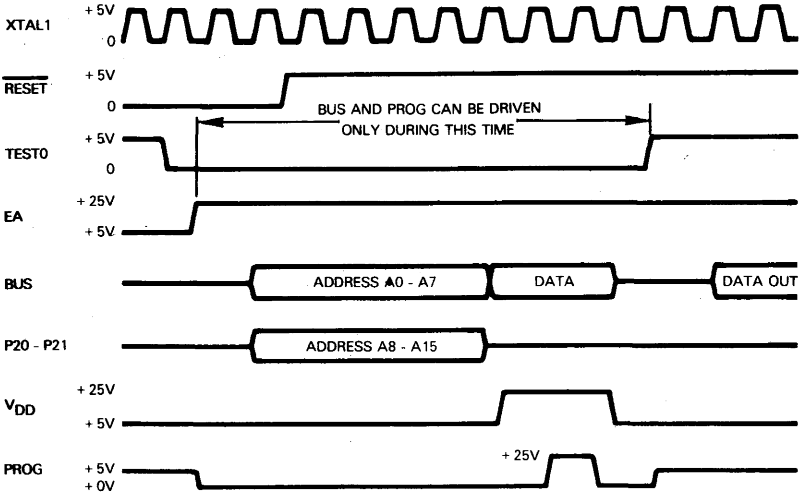


Figure 6-12. 8748 EPROM Programming And Verification Timing

PROGRAMMING MODE

Of the 8048 microcomputer series, only the 8748 program memory can be written into. We will now examine the way in which the 8748 EPROM is programmed and verified.

In all probability, you will program an 8748 memory using a development tool which automates the entire process. That being the case, the event sequence which we are about to describe is not particularly interesting to you, since it is taken care of by the PROM programmer. But if you build your own PROM programmer, or if for any reason you need to understand the PROM programming sequence, then read on.

While programming and verifying the EPROM, you should input a clock signal at XTAL1 with a frequency between 1 and 6 MHz; you can also use the on-chip oscillator at this time.

Operations now proceed one byte at a time; you write a byte into program memory, then you verify that the data has been written correctly.

In the discussion which follows, refer to Figure 6-12 which illustrates timing for the program/verify sequence.

Step 1. Initially +5V is input at V_{DD}, T0 and EA. $\overline{\text{RESET}}$ is held at ground. Under these conditions you insert the 8748 into the programming socket. **You must make certain to insert the 8748 correctly. If you insert the 8748 incorrectly you will destroy it.**

Step 2. T0 is pulled to ground; this selects Programming mode.

Step 3. +25V is applied to EA. This activates Programming mode.

Step 4. A 10-bit memory address is applied via DB0 - DB7 and P20 - P23. Remember there are 1024 bytes of program memory on the 8748 device. The low order eight address bits are input via DB0 - DB7 while the two high order address bits are input via P20 and P21.

Step 5. +5V is applied at $\overline{\text{RESET}}$. This latches the address.

Step 6. The data to be written into the addressed programmed memory byte is input at DB0 - DB7.

Step 7. In order to write the data into the addressed program memory byte apply +25V to V_{DD}, then ground PROG, then apply a +25V pulse at PROG; the +25V pulse at PROG must last at least 50 milliseconds.

Step 8. Now reduce V_{DD} to +5V. Programming is complete and verification is about to begin.

Step 9. In order to verify the data just written, apply +5V to the T0 input. This selects Verify mode.

Step 10. As soon as Verify mode has been selected, the data just written is output on DB0 - DB7. You must read and verify this data using appropriate external circuitry. Verification is now complete.

In order to write into the next memory byte, select Programming mode again by connecting T0 and $\overline{\text{RESET}}$ to ground; then return to Step 3.

Repeat the program/verify sequence, byte-by-byte, until the entire program memory has been written into.

In order to erase the EPROM expose it to ultraviolet light for a minimum of 20 minutes.

VERIFICATION MODE

You can verify the contents of 8048 or 8748 program memory at any time.

When verifying program memory contents for an 8748 microcomputer, you enter the Verify mode by applying +25V to the EA pin and +5V to the T0 pin. $\overline{\text{RESET}}$ must be held at ground while you apply +5V to the T0 pin.

Using an 8048 microcomputer you enter the Verify mode by applying +12V to the EA pin.

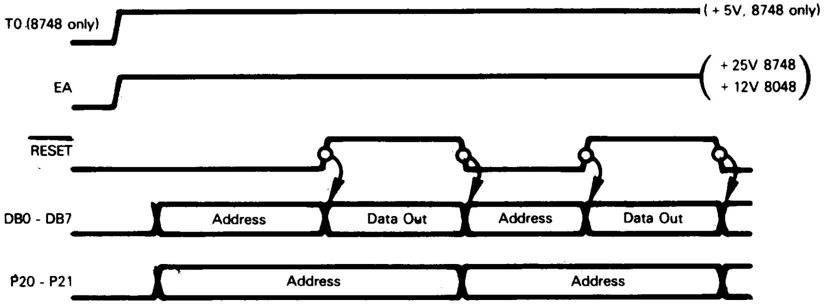
Once in the Verify mode, place the address of the program memory location which is to be read at DB0 - DB7 and P20 - P21.

Latch this address by applying +5V to $\overline{\text{RESET}}$.

While $\overline{\text{RESET}}$ is high the contents of the addressed program memory location are output via DB0 - DB7.

You may repeat the verification process, byte-by-byte.

Verification timing is illustrated as follows:



INPUT/OUTPUT PROGRAMMING

8048 series microcomputers have three I/O ports, the physical characteristics of which we have already described. **Instructions allow you to input or output Accumulator data** via any one of the three I/O ports. **You can also directly mask data** resident at an I/O port using an AND mask or an OR mask.

There are two types of input/output beyond the 8048 series microcomputer chip itself.

The low order four bits of I/O Port 2 may be connected to the 8243 Input/Output Expander which has four individually addressable 4-bit I/O ports. The 8243 Input/Output Expander is described later in this chapter.

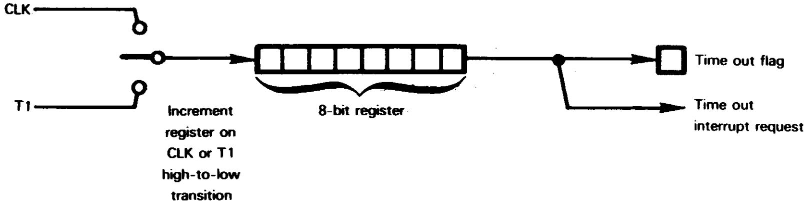
You can also implement I/O ports within the external data memory address space. We have already seen how you do this using an 8355 multifunction device connected to an 8048 series microcomputer. In this particular case the two I/O ports of the 8355 device are addressed as external data memory locations 0 and 1. Any other implementation of external I/O ports is allowed; however in every case the I/O ports must be addressed as external data memory bytes using external data memory access instructions.

HOLD STATE

There is no Hold state that external logic can induce in an 8048 series microcomputer. This is not unreasonable, since the purpose of the Hold state is to enable direct memory access operations — which would make little sense in a microcomputer system as small as an 8048, which has a maximum of 256 external data memory bytes.

COUNTER/TIMER OPERATIONS

All 8048 series microcomputers have an internal Counter/Timer. Counter/Timer logic may be illustrated as follows:


CLK
T1
Increment register on CLK or T1 high-to-low transition
8-bit register
Time out flag
Time out interrupt request

The Counter/Timer register is 8 bits wide; it is accessed via the Accumulator. Instructions move Accumulator contents to the Counter/Timer register, or move Counter/Timer register contents to the Accumulator.

Generally stated, this is how the Counter/Timer works:

You begin by loading an initial value into the Counter/Timer register. Next you start the Counter/Timer by executing the STRT T or STRT CNT instruction. The Counter/Timer will increment continuously until stopped by a Stop Counter/Timer instruction.

Whenever the Counter/Timer increments from FF_{16} to 00_{16}, it activates a Counter/timer interrupt request and sets a time out flag. If the Counter/Timer interrupt has been enabled, then program execution will branch to the appropriate interrupt service routine. If the Counter/Timer interrupt has not been enabled, then you must test for a time out by executing the JTO Branch-on-Condition instruction.

You can operate the Counter/Timer as a Counter or as a Timer. The STRT T instruction operates the Counter/Timer as a Timer, in which case **the internal system clock increments the Timer register every 80 microseconds, assuming a 6 MHz crystal.**

You operate the Counter/Timer as a Counter by executing the STRT CNT instruction. **Now high-to-low transitions of a signal input at T1 increment the Counter.** The minimum time interval between high-to-low T1 transitions is 7.5 microseconds. There is no maximum delay between T1 high-to-low transitions. Once T1 goes high it must remain high for at least 500 nanoseconds.

You operate the Counter/Timer as a Counter by executing the STRT CNT instruction. **Now high-to-low transitions of a signal input at T1 increment the Counter.** The minimum time interval between high-to-low T1 transitions is 7.5 microseconds. There is no maximum delay between T1 high-to-low transitions. Once T1 goes high it must remain high for at least 100 nanoseconds.

You execute the STOP TCNT instruction to stop the Counter/Timer, whether it is operating as a Counter or as a Timer.

Here is an instruction sequence which initiates the Counter/Timer operating as a Timer with interrupts enabled:

```
MOV     A,#TSTART     ;LOAD INITIAL COUNTER/TIMER CONSTANT
MOV     T,A
EN      TCNTI         ;ENABLE TIMER INTERRUPT
STRT    T             ;START THE TIMER
```

The following instruction sequence operates the Counter/Timer as a Counter with interrupts disabled:

```
        DIS     TCNTI          ;DISABLE COUNTER INTERRUPT EARLY IN PRO-
                               GRAM
        -
        -
        -
        MOV     A,#TSTART      ;LOAD INITIAL COUNTER/TIMER CONSTANT
        MOV     T,A
        STRT    CNT            ;START COUNTER
```

INTERNAL AND EXTERNAL INTERRUPTS

The 8048 has a simple interrupt scheme that is effective and adequate for small microcomputer systems. Interrupts can originate from one of three sources:

1) A Reset. This is a nonmaskable interrupt.
2) An external interrupt induced by setting INT low.
3) A Counter/Timer interrupt which is automatically requested every time the Counter/Timer register increments from FF_{16} to 00_{16}.

External interrupts and Counter/Timer interrupts can be enabled and disabled individually.

When any one of the three interrupt requests is acknowledged, the microcomputer executes a Call instruction to one of these three locations:

Reset:	CALL 0
External interrupt:	CALL 3
Counter/Timer interrupt:	CALL 7

The Reset interrupt always has highest priority and cannot be disabled.

If an External interrupt request and a Counter/Timer interrupt request occur simultaneously, the External interrupt will be acknowledged first. **When either an External interrupt or a Counter/Timer interrupt is acknowledged, all interrupts (except Reset) are disabled until an RETR instruction is executed. Within an External or Timer interrupt service routine you cannot enable interrupts under program control.** This may be a problem if you are using the Timer and External interrupts in Timer sensitive applications. If execution time for an External interrupt's service routine extends over more than one Counter/Timer time out, then you will fail to detect one or more time outs. The simplest way of resolving this problem is to make sure that your External interrupt service routines are very short — executing in 75% of the Counter/Timer interval, or less. If this is not feasible, then you must monitor the Counter/Timer by testing its time out flag rather than by using Counter/Timer interrupt logic. You can execute the JTF conditional Jump instruction at frequent intervals within the main program and interrupt service routines, thus catching time outs irrespective of when they occur.

You can re-enable interrupts within an interrupt service routine by executing a dummy RETR instruction. Here is an appropriate instruction sequence:

```
START OF INTERRUPT SERVICE ROUTINE
        -
        -
        -
        CALL    ENAB      ;RE-ENABLE INTERRUPTS
        EN      I
        EN      TCNTI
        -
        -
        -
END OF INTERRUPT SERVICE ROUTINE
ENAB    RETR
```

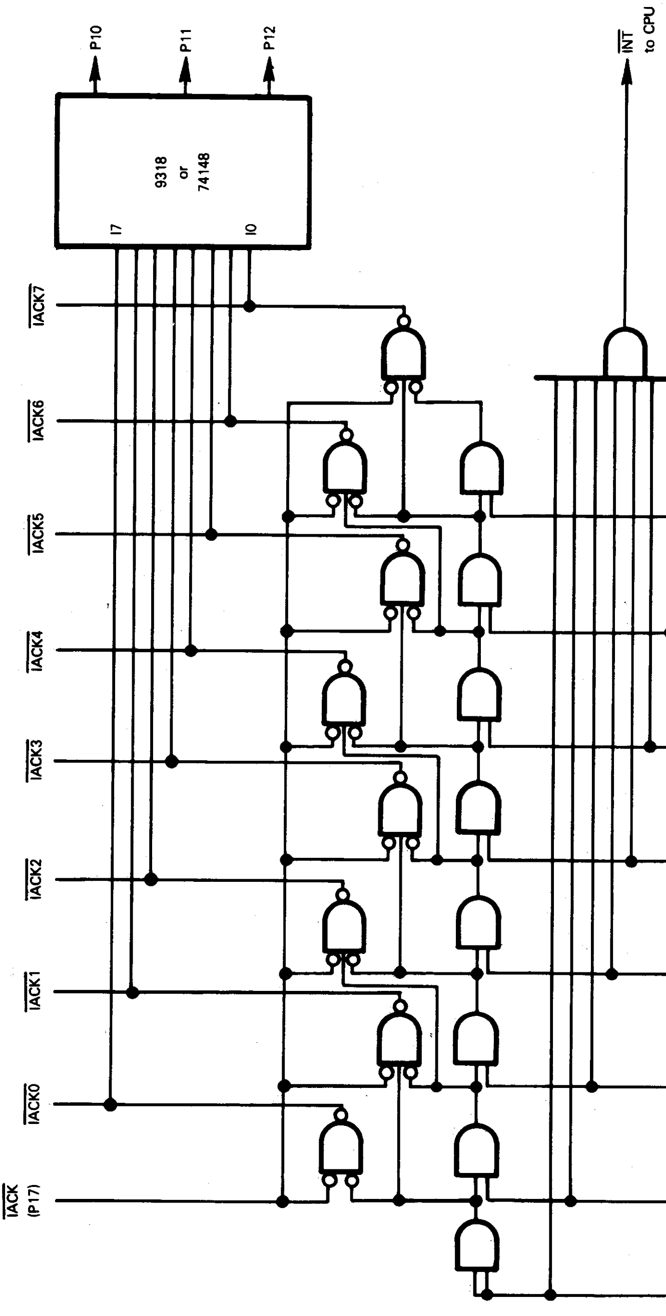


Figure 6-13. An Eight-Device Daisy Chained Interrupt Request/Acknowledge Scheme

Enabling interrupts within a service routine, as illustrated above, is not recommended in an 8048 microcomputer system.

Two problems need to be resolved when using external interrupts in an 8048 series microcomputer system: an interrupt acknowledge must be created and in multiple interrupt configurations we must be able to identify the interrupting source.

8048 series microcomputers have no interrupt acknowledge signal. An interrupt acknowledge signal must be created; otherwise external logic does not know when to remove its interrupt request. And if the interrupt request remains after an RETR instruction executes, the interrupt will be reacknowledged. **The only straightforward way of acknowledging an interrupt is to assign one of the I/O port pins to serve as an interrupt acknowledge signal.** The external interrupt service routine will begin by outputting an appropriate high pin signal. **Here is one possibility:**

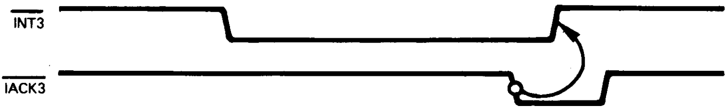
```
        ORL     P1,#80H     ;SET PIN 7 OF I/O PORT 1 HIGH
        ANL     P1,#7FH     ;RESET PIN 7 OF I/O PORT 1 LOW
```

Here, the output at pin 7 of I/O Port 1 is a high pulse with a duration of two machine cycles (5.0 microseconds).

But remember, if you use an I/O port pin as an interrupt acknowledge, you cannot use the same pin to perform standard I/O operations.

If there are many external devices which can request interrupt service, then the most effective way of handling multiple interrupts is via a daisy chain. Daisy chain logic has been discussed in Volume I — Basic Concepts. The acknowledged device in the daisy chain must create a device code that is input to an I/O port. **Figure 6-13 illustrates a scheme whereby eight devices in a daisy chain may request interrupt service, and upon being acknowledged, the selected device will input a unique code to I/O Port 1.** The high order bit of I/O Port 1 serves as an interrupt acknowledge. I/O Port 1, bits 0, 1 and 2 receive as inputs a 3-bit code identifying the acknowledged device.

The daisy chain logic in Figure 6-13 is created using a chain of eight AND gates and eight NAND gates. The AND gates are chained in order of priority, with INT0 having the highest priority and INT7 having the lowest priority. The first NAND gate receives as its inputs INT0 and the acknowledge signal output via pin 7 of I/O Port 1. Subsequent NAND gates receive as their inputs an interrupt request signal, the acknowledge signal and the output of the previous AND gate. The output of each NAND gate becomes an interrupt acknowledge signal which is low true. Thus in Figure 6-13 there are eight low true interrupt requests, represented by signals INT0 through INT7, and there are eight low true interrupt acknowledges, represented by IACK0 through IACK7. Each external device capable of requesting an interrupt must output a low true INTN which it removes upon receiving a low true IACKN. For device 3 this may be illustrated as follows:



The eight interrupt request signals INT0 through INT7 are input to an AND gate. The AND gate generates a master low true interrupt request INT. If any one or more of the INTN signals are low, then the AND gate will output a low INT.

The eight interrupt acknowledge signals IACK0 - IACK7 are input to an 8-to-3 Decoder. The 8-to-3 Decoder will receive seven high signals and one low signal. The one low signal will be identified by the decoder 3-bit output which is transmitted to pins 0, 1 and 2 of I/O Port 1.

This then is the event sequence associated with an interrupt request:

1) $\overline{\text{INT}}$ is input low to the 8048.
2) The interrupt is acknowledged by the CPU which branches to an interrupt service routine.
3) The first instruction of the interrupt service routine outputs a low level via pin 7 of I/O Port 1.
4) The interrupt service routine receives back, via pins 0, 1 and 2 of I/O Port 1, the device code for the acknowledged device. You must make sure that the program being executed gives external logic time to return this code. You may have to insert No Operation instructions to create the necessary time delay.
5) A high level is output via pin 7 of I/O Port 1.
6) Using the code input via pins 0, 1 and 2 of I/O Port 1, branch to the appropriate interrupt service routine.

Here is the initial instruction sequence required by the logic of Figure 6-13:

```
        ORG     3
;START OF INTERRUPT SERVICE ROUTINE
        JMP     EXTINT
        -
        -
        -
        ORG     EXTINT
        ANL     P1,#7FH     ;SET I/O PORT 1 PIN 7 LOW
        NOP                 ;ALLOW SETTLING TIME
        IN      A,P1        ;INPUT PORT 1 CONTENTS
        ORL     P1,#80H     ;SET I/O PORT 1 PIN 7 HIGH
        ANL     A,#7        ;CLEAR ALL ACCUMULATOR BITS BAR 0, 1 AND 2
        JMPP    @A          ;JUMP TO IDENTIFIED INTERRUPT SERVICE ROUTINE
```

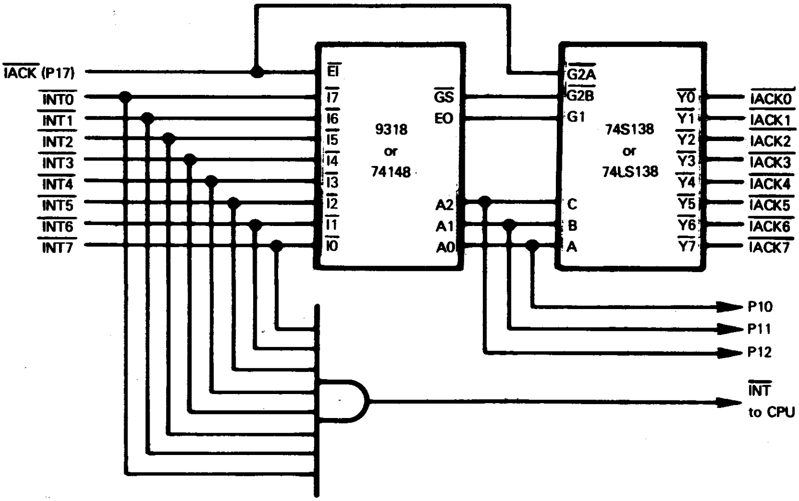
Let us examine the interrupt service routine beginning instruction sequence illustrated above.

When an 8048 series microcomputer is initially reset, all I/O port pins output high levels. Thus you do not have to initialize pin 7 of I/O Port 1 to a high level.

We actually identify one of eight device interrupt service routines by creating a 3-bit code in bits 1, 2 and 3 of the Accumulator. We then perform an indirect Jump. This Jump instruction will branch to a location on the current page of program memory; the address is fetched from the location in the current page addressed by the Accumulator contents. We illustrated this addressing technique earlier in the chapter.

Given the instruction sequence illustrated above, the first eight program memory locations on the same page of the JMPP instruction must be set aside for eight addresses; these are the starting addresses for the interrupt service routines. This may be illustrated as follows:

```
        ORG     #0300H
        DB      IS0         ;ADDRESS OF INTERRUPT SERVICE ROUTINE 0
        DB      IS1         ;ADDRESS OF INTERRUPT SERVICE ROUTINE 1
        DB      IS2         ;ADDRESS OF INTERRUPT SERVICE ROUTINE 2
        DB      IS3         ;ADDRESS OF INTERRUPT SERVICE ROUTINE 3
        DB      IS4         ;ADDRESS OF INTERRUPT SERVICE ROUTINE 4
        DB      IS5         ;ADDRESS OF INTERRUPT SERVICE ROUTINE 5
        DB      IS6         ;ADDRESS OF INTERRUPT SERVICE ROUTINE 6
        DB      IS7         ;ADDRESS OF INTERRUPT SERVICE ROUTINE 7
EXTINT  ANL      #7FH       ;SET I/O PORT 1 PIN 7 LOW
        -
        -
```



9318, 74148 FUNCTION TABLE

INPUTS									OUTPUTS				
$\overline{EI}$	$\overline{I0}$	$\overline{I1}$	$\overline{I2}$	$\overline{I3}$	$\overline{I4}$	$\overline{I5}$	$\overline{I6}$	$\overline{I7}$	A2	A1	A0	$\overline{GS}$	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

74LS138, 74S138 FUNCTION TABLE

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	$\overline{G2}$*	C	B	A	$\overline{Y0}$	$\overline{Y1}$	$\overline{Y2}$	$\overline{Y3}$	$\overline{Y4}$	$\overline{Y5}$	$\overline{Y6}$	$\overline{Y7}$
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

*$\overline{G2} = \overline{G2A} \vee \overline{G2B}$

H = high level, L = low level, X = irrelevant

Figure 6-14. A Low Chip Count Implementation Of An Eight-Device Daisy Chained Interrupt Request/Acknowledge Scheme

The daisy chained interrupt scheme discussed above can also be implemented using the circuit in Figure 6-14. The advantage of this circuit is that it requires fewer chips than the circuit of Figure 6-13. As far as the 8048 program is concerned, however, the two circuits are identical.

The $\overline{\text{INT}}$ and device code inputs are generated in exactly the same way. However, an eight-line-to-three-line priority encoder (9318 or 74148) replaces the network of AND gates. As the function table for the encoder shows, the device code output on lines A2, A1 and A0 is that of the highest priority request. The CPU enables the code outputs by sending the acknowledge signal.

In Figure 6-13, a network of NAND gates generated the low true interrupt acknowledge signal to inform the appropriate device that its interrupt was being serviced. In Figure 6-14, a three-line-to-eight-line decoder (74S138 or 74LS138) translates the device code output by the encoder and sets the corresponding acknowledge line low, as is shown in the function table for the decoder.

Connecting the enable inputs as shown prevents spurious acknowledgements or phantom device codes, provided that the CPU gives the external devices time for response and propagation delay.

THE 8048 MICROCOMPUTER SERIES INSTRUCTION SET

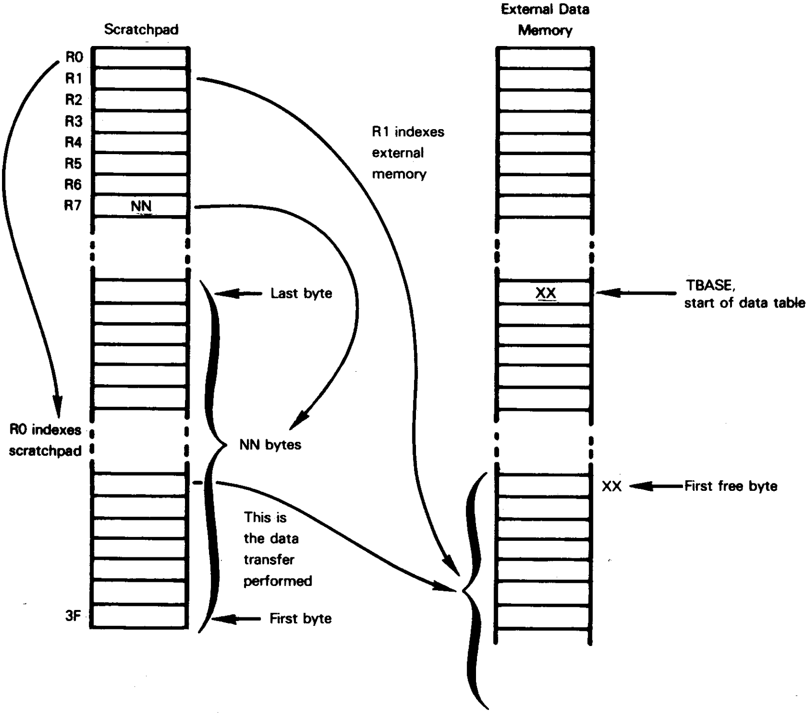
Table 6-2 summarizes the instruction set for the 8048 series microcomputers. Instruction object codes and timing are given in Table 6-3. This instruction set reflects the specific architecture of 8048 series microcomputers. For example, there are separate I/O instructions to access the three on-chip I/O ports as against 8243 Input/Output Expander I/O ports. Also there are separate instructions to access on-chip scratchpad read/write memory as against external data memory.

The 8048 instruction set is probably more versatile than any other one-chip microcomputer instruction set described in this book. The only omission that may cause problems is the lack of an Overflow status; this will make multibyte signed binary arithmetic harder to program.

THE BENCHMARK PROGRAM

The benchmark program we have been using in this book is not realistic for the 8048 with its limited data memory. Using the 8048 you would not load data into some general depository, then transfer it to a specific data table.

In order to provide some illustration of 8048 instructions, however, we will slightly modify the benchmark program and move a number of data bytes from the top of scratchpad memory to a table in external data memory. Since the data in scratchpad memory must have been input from an I/O port, we will assume that the number of scratchpad memory bytes is stored in General Purpose Register R7. The table in external memory begins at a known location and the first table byte addresses the first free table location. Operations performed may be illustrated as follows:



```
        MOV     R0,#TBASE  ;LOAD EXTERNAL TABLE BASE ADDRESS INTO R0
        MOVX    A,@R0      ;LOAD ADDRESS OF FIRST FREE BYTE INTO A
        MOV     R1,A       ;SAVE IN R1
        ADD     A,R7       ;ADD NEW BYTE COUNT TO A
        MOVX    @R0,A      ;RESTORE IN FIRST FREE BYTE OF EXTERNAL TABLE
        MOV     R0,#3FH    ;LOAD SCRATCHPAD ADDRESS INTO R0
LOOP    MOV     A,@R0      ;MOVE DATA FROM SCRATCHPAD TO A
        MOVX    @R1,A      ;STORE IN EXTERNAL DATA TABLE
        DEC     R0         ;DECREMENT R0
        INC     R1         ;INCREMENT R1
        DJNZ    R7,LOOP    ;DECREMENT R7, SKIP IF NOT ZERO
```

These are the abbreviations used in Table 6-2:

A	The Accumulator
A03	Accumulator, bits 0 - 3
R	Register R0 or R1
REG	Accumulator, R0, R1, R2, R3, R4, R5, R6 or R7
RN	Register R0, R1, R2, R3, R4, R5, R6 or R7
T	Timer/Counter
C	Carry status
AC	Auxiliary Carry status
MB0	Program memory bank 0
MB1	Program memory bank 1
MBN	MB0 or MB1
I	The Instruction register
I2	Second object code byte
PC	The Program Counter
PC10	The Program Counter, bits 0 - 10
PCL	The Program Counter, bits 0 - 7
PCH	The Program Counter, bits 8 - 11
SP	Stack Pointer: PSW bits 0, 1 and 2
PSW	The Program Status Word which has bits assigned to status flags as follows:

7	6	5	4	3	2	1	0	← Bit No.
C	AC	F0	F1	1	SP2	SP1	SP0	

S	PSW bit C,F0 or F1
DATA	8-bit immediate data
DEV	An I/O device
PORT	I/O port P1, P2 or BUS
ADDR	An 11-bit address, specifying a data memory byte
ADDR8	The low order eight bits of a memory address
[ ]	Contents of location identified within brackets
[[ ]]	Scratchpad memory byte addressed by location identified within brackets
{[ ]}	External memory byte addressed by location identified within brackets
([ ])	Program memory byte addressed by location identified within brackets
←	Move data in direction of arrow
←→	Exchange contents of locations on either side of arrow
+	Add
-	Subtract
Λ	AND
V	OR
⊻	Exclusive-OR
BUS	Bus I/O port
P1	I/O Port 1
P2	I/O Port 2
EP	8243 Expander Port P4, P5, P6 or P7
PN	P1 or P2

Table 6-2. A Summary Of 8048 Microcomputer Instruction Set

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	AC	
I/O	ANL	PORT,#DATA	2			[PORT]←[PORT] Λ DATA AND immediate data with I/O Port P1, P2 or BUS
	ANLD	EP,A	1			[EP]←[A03] Λ [EP] AND expander port P4, P5, P6 or P7 with Accumulator bits 0 - 3
	IN	A,PN	1			[A]←[PN] Input I/O Port P1 or P2 to Accumulator
	INS	A,BUS	1			[A]←[BUS] Input BUS to Accumulator with strobe
	MOVD	A,EP	1			[A03]←[EP] Input expander port P4, P5, P6 or P7 to Accumulator bits 0 - 3
	MOVD	EP,A	1			[EP]←[A03] Output Accumulator bits 0 - 3 to expander port P4, P5, P6 or P7
	ORL	PORT,#DATA	2			[PORT]←[PORT] V DATA OR immediate data with I/O Port P1, P2 or BUS
	ORLD	EP,A	1			[EP]←[A03] V [EP] OR Accumulator bits 0 - 3 with expander port P4, P5, P6 or P7
	OUTL	PORT,A	1			[PORT]←[A] Output Accumulator contents to I/O Port P1, P2 or BUS
PRIMARY MEMORY REFERENCE	MOV	A,@R	1			[A]←[[R]] Load contents of scratchpad byte addressed by R0 or R1 into Accumulator
	MOV	@R,A	1			[[R]]←[A] Store Accumulator contents in scratchpad byte addressed by R0 or R1
	MOVP	A,@A	1			[A]←([PCH] [A]) Load into the Accumulator the contents of the program memory byte addressed by the Accumulator and Program Counter bits 8 - 11.
	MOVP3	A,@A	1			[A]←(3 [A]) Load into the Accumulator the contents of the program memory byte with binary address 0011XXXXXXXX where XXXXXXXX represents initial Accumulator contents.

Table 6-2. A Summary Of 8048 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	AC	
PRIMARY MEMORY REFERENCE (CONTINUED)	MOVX	A, @R	1			[A]←¦[R]¦ Load contents of external data memory byte addressed by R0 or R1 into Accumulator
	MOVX	@R,A	1			¦[R]¦←[A] Store Accumulator contents in external data memory byte addressed by R0 or R1
	XCH	A, @R	1			[A]←→[[R]] Exchange contents of Accumulator and scratchpad memory byte addressed by R0 or R1
	XCHD	A, @R	1			[A03]←→[[R]03] Exchange contents of Accumulator bits 0 - 3 with bits 0 - 3 of scratchpad memory byte addressed by R0 or R1
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	ADD	A, @R	1	X	X	[A]←[A]+[[R]] Add contents of scratchpad byte addressed by R0 or R1 to Accumulator
	ADDC	A, @R	1	X	X	[A]←[A]+[[R]]+[C] Add contents of scratchpad byte addressed by R0 or R1, plus Carry, to Accumulator
	ANL	A, @R	1			[A]←[A] ∧ [[R]] AND contents of scratchpad byte addressed by R0 or R1 with Accumulator
	ORL	A, @R	1			[A]←[A] ∨ [[R]] OR contents of scratchpad byte addressed by R0 or R1 with Accumulator
	XRL	A, @R	1			[A]←[A] ⊻ [[R]] Exclusive OR contents of scratchpad byte addressed by R0 or R1 with Accumulator
	INC	@R	1			[[R]]←[[R]]+1 Increment the contents of the scratchpad byte addressed by R0 or R1
IMMEDIATE	MOV	REG, #DATA	2			[REG]←DATA Load immediate data into Accumulator, or Register R0, R1, R2, R3, R4, R5, R6 or R7
	MOV	@R, #DATA	2			[[R]]←DATA Load immediate data into scratchpad byte addressed by R0 or R1

Table 6-2. A Summary Of 8048 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	AC	
JUMP	JMP	ADDR	2			[PC10]← ADDR Jump to instruction in current 2K block having label ADDR
	JMPP	@A	1			[PC]←[PCH][A], [PCL]←([PCH][A]) Load into the eight low order Program Counter bits the contents of the program memory byte addressed by the Accumulator and the four hgih order Program Counter bits.
	SEL	MB0	1			With the next JMP or CALL instruction, reset the high order bit of PC to 0, thus selecting first 2K program memory bytes.
	SEL	MB1	1			With the next JMP or CALL instruction, set high order bit of PC to 1, thus selecting second 2K program memory bytes.
SUBROUTINE CALL AND RETURN	CALL	ADDR	2			STACK ←STATUS + [PC], [SP]←[SP] + 1, [PC]← ADDR Call subroutine ADDR
	RET		1			[PC]← STACK, [SP]←[SP]-1 Return from subroutine without restoring status
	RETR		1	X	X	[PC] + STATUS ← STACK, [SP]←[SP]-1 Return from subroutine and restore status
IMMEDIATE OPERATE	ADD	A, #DATA	2	X	X	[A]←[A] + DATA Add immediate data to Accumulator
	ADDC	A, #DATA	2	X	X	[A]←[A] + DATA + [C] Add immediate data plus Carry to Accumulator
	ANL	A, #DATA	2			[A]←[A] ∧ DATA AND immediate data with Accumulator contents
	ORL	A, #DATA	2			[A]←[A] V DATA OR immediate data with Accumulator contents
	XRL	A, #DATA	2			[A]←[A] ⊻DATA Exclusive OR immediate data with Accumulator contents

Table 6-2. A Summary Of 8048 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	AC	
JUMP ON CONDITION	DJNZ	RN,ADDR8	2			[RN]←[RN]-1. If [RN] ≠ 0, [PCL]←ADDR8 Decrement Register R0, R1, R2, R3, R4, R5, R6 or R7. If the result is not 0, branch to ADDR8 on the current program memory page.
	JBb	ADDR8	2			[PCL]←ADDR8 Jump on current page if Accumulator bit b is 1. b must be 0, 1, 2, 3, 4, 5, 6 or 7
	JC	ADDR8	2			[PCL]←ADDR8 Jump on current page if Carry is 1
	JF0	ADDR8	2			[PCL]←ADDR8 Jump on current page if flag F0 is 1
	JF1	ADDR8	2			[PCL]←ADDR8 Jump on current page if flag F1 is 1
	JNC	ADDR8	2			[PCL]←ADDR8 Jump on current page if Carry is 0
	JNI	ADDR8	2			[PCL]←ADDR8 Jump on current page if interrupt request input is 0
	JNT0	ADDR8	2			[PCL]←ADDR8 Jump on current page if T0 input is 0
	JNT1	ADDR8	2			[PCL]←ADDR8 Jump on current page if T1 input is 0
	JNZ	ADDR8	2			[PCL]←ADDR8 Jump on current page if Accumulator contents is nonzero
	JTF	ADDR8	2			[PCL]←ADDR8 Jump on current page if timer has timed out, that is, if timer flag is 1. The timer flag is reset to 0 by this instruction.
	JT0	ADDR8	2			[PCL]←ADDR8 Jump on current page if T0 input is 1
	JT1	ADDR8	2			[PCL]←ADDR8 Jump on current page if T1 input is 1
	JZ	ADDR8	2			[PCL]←ADDR8 Jump on current page if Accumulator contents are zero

Table 6-2. A Summary Of 8048 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	AC	
REGISTER-REGISTER MOVE	MOV	A,RN	1			$[A] \leftarrow [RN]$ Move the contents of a general purpose register to the Accumulator
	MOV	RN,A	1			$[RN] \leftarrow [A]$ Move the Accumulator contents to a general purpose register
	XCH	A,RN	1			$[A] \longleftrightarrow [RN]$ Exchange the Accumulator contents with the contents of a general purpose register
REGISTER-REGISTER OPERATE	ADD	A,RN	1	X	X	$[A] \leftarrow [A] + [RN]$ Add the contents of a general purpose register to the Accumulator
	ADDC	A,RN	1	X	X	$[A] \leftarrow [A] + [RN] + [C]$ Add the contents of a general purpose register, plus Carry, to the Accumulator
	ANL	A,RN	1			$[A] \leftarrow [A] \wedge [RN]$ AND the contents of a general purpose register with the Accumulator
	ORL	A,RN	1			$[A] \leftarrow [A] \vee [RN]$ OR the contents of a general purpose register with the Accumulator
	XRL	A,RN	1			$[A] \leftarrow [A] \veebar [RN]$ Exclusive-OR the contents of a general purpose register with the Accumulator
REGISTER OPERATE	CLR	A	1			$[A] \leftarrow 0$ Zero the Accumulator
	CPL	A	1			$[A] \leftarrow \overline{[A]}$ Complement the Accumulator
	DAA		1			Decimal adjust Accumulator contents
	DEC	REG	1			$[REG] \leftarrow [REG]-1$ Decrement the contents of the Accumulator or general purpose register
	INC	REG	1			$[REG] \leftarrow [REG]+1$ Increment the contents of the Accumulator or general purpose register

Table 6-2. A Summary Of 8048 Microcomputer Instruction Set (Continued)

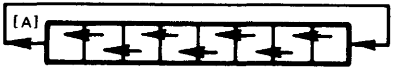
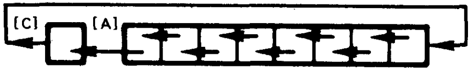
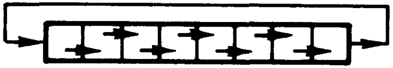
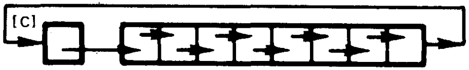
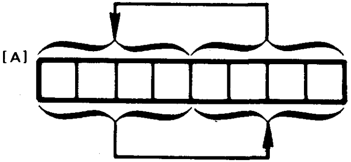
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	AC	
REGISTER OPERATE (CONTINUED)	RL	A	1			Rotate Accumulator left 
	RLC	A	1	X		Rotate Accumulator left through Carry 
	RR	A	1			Rotate Accumulator right 
	RRC	A	1	X		Rotate Accumulator right through Carry 
	SEL	RBO	1			Select register bank 0
	SEL	RB1	1			Select register bank 1
	SWAP	A	1			Swap Accumulator nibbles 

Table 6-2. A Summary Of 8048 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				C	AC	
INTERRUPT	DIS	TCNTI	1			Disable timer interrupt
	EN	TCNTI	1			Enable timer interrupt
	DIS	I	1			Disable external interrupt
	EN	I	1			Enable external interrupts
COUNTER/TIMER	ENT0	CLK	1			Enable timer output on T0 until next system reset
	MOV	A,T	1			[A]←[T] Read timer/counter
	MOV	T,A	1			[T]←[A] Load timer/counter
	STOP	TCNT	1			Stop timer/counter
	STRT	CNT	1			Start counter
	STRT	T	1			Start timer
STATUS	CLR	S	1	0		Clear PSW bit C, F0 or F1
	CPL	S	1	X		Complement PSW bit C, F0 or F1
	MOV	A,PSW	1			[A]←[PSW] Move Program Status Word contents to the Accumulator
	MOV	PSW,A	1	X	X	[PSW]←[A] Move Accumulator contents to the Program Status Word
	NOP		1			No Operation

The following symbols are used in Table 6-3:

bbb Three bits designating which bit of the Accumulator is to be tested.

ee Two bits designating an 8243 Expander port:
00 - P4
01 - P5
10 - P6
11 - P7

k One bit selecting a memory or register bank:
0 MB0 or RB0
1 MB1 or RB1

MM Eight bits of immediate data

nnn Three bits designating one of the eight general purpose registers

pp Two bits designating one of the on-chip I/O ports:
00 - BUS
01 - P1
10 - P2

qq Two bits designating either I/O Port 1 or I/O Port 2:
01 - P1
10 - P2

r One bit selecting a pointer register:
0 - R0
1 - R1

xxx The high order three bits of a program memory address

XX The low order eight bits of a program memory address

Table 6-3 8048 Instruction Set Object Codes

INSTRUCTION		OBJECT CODE	BYTES	MACHINE CYCLES
ADD	A,RN	01101nnn	1	1
ADD	A,@R	0110000r	1	1
ADD	A,#DATA	03 MM	2	2
ADDC	A,RN	01111nnn	1	1
ADDC	A,@R	0111000r	1	1
ADDC	A,#DATA	13 MM	2	2
ANL	A,RN	01011nnn	1	1
ANL	A,@R	0101000r	1	1
ANL	A,#DATA	53 MM	2	2
ANL	PORT,#DATA	100110pp MM	2	2
ANLD	EP,A	100111ee	1	2
CALL	ADDR	xxx10100 XX	2	2
CLR	A	27	1	1
CLR	C	97	1	1
CLR	F1	A5	1	1
CLR	F0	85	1	1
CPL	A	37	1	1
CPL	C	A7	1	1
CPL	F0	95	1	1
CPL	F1	B5	1	1
DA	A	57	1	1
DEC	A	07	1	1
DEC	RN	11001nnn	1	1
DIS	I	15	1	1
DIS	TCNTI	35	1	1
DJNZ	RN,ADDR8	11101rrr XX	2	2
EN	I	05	1	1
EN	TCNTI	25	1	1
ENT0	CLK	75	1	1
IN	A,PN	000010qq	1	2
INC	A	17	1	1
INC	RN	00011nnn	1	1
INC	@R	0001000r	1	1
INS	A,BUS	08	1	2
JBb	ADDR8	bbb10010 XX	2	2
JC	ADDR8	F6 XX	2	2
JF0	ADDR8	B6 XX	2	2
JF1	ADDR8	76 XX	2	2
JMP	ADDR	xxx00100 XX	2	2

Table 6-3. 8048 Instruction Set Oject Codes (Continued)

INSTRUCTION		OBJECT CODE	BYTES	MACHINE CYCLES
JMPP	@A	B3	1	2
JNC	ADDR8	E6 XX	2	2
JNI	ADDR8	86 XX	2	2
JNT0	ADDR8	26 XX	2	2
JNT1	ADDR8	46 XX	2	2
JNZ	ADDR8	96 XX	2	2
JTF	ADDR8	16 XX	2	2
JT0	ADDR8	36 XX	2	2
JT1	ADDR8	56 XX	2	2
JZ	ADDR8	C6 XX	2	2
MOV	A,#DATA	23 MM	2	2
MOV	A,PSW	C7	1	1
MOV	A,RN	11111nnn	1	1
MOV	A,@R	1111000r	1	1
MOV	A,T	42	1	1
MOV	PSW,A	D7	1	1
MOV	RN,A	10101nnn	1	1
MOV	RN,#DATA	10111nn MM	2	2
MOV	@R,A	1010000r	1	1
MOV	@R,#DATA	1011000r MM	2	2
MOV	T,A	62	1	1
MOVD	A,EP	000011ee	1	2
MOVD	EP,A	001111ee	1	2
MOVP	A,@A	A3	1	2
MOVP3	A,@A	E3	1	2
MOVX	A,@R	1000000r	1	2
MOVX	@R,A	1001000r	1	2
NOP		00	1	1
ORL	A,RN	01001nnn	1	1
ORL	A,@R	0100000r	1	1
ORL	A,#DATA	43 MM	2	2
ORL	PORT,#DATA	100010pp MM	2	2
ORLD	EP,A	100011ee	1	2
OUTL	BUS,A	02	1	2
OUTL	PN,A	001110qq	1	2
RET		83	1	2
RETR		93	1	2
RL	A	E7	1	1
RLC	A	F7	1	1
RR	A	77	1	1
RRC	A	67	1	1
SEL	MBk	111k0101	1	1
SEL	RBk	110k0101	1	1
STOP	TCNT	65	1	1
STRT	CNT	45	1	1
STRT	T	55	1	1
SWAP	A	47	1	1
XCH	A,RN	00101nnn	1	1
XCH	A,@R	0010000r	1	1
XCHD	A,@R	0011000r	1	1
XRL	A,RN	11011nnn	1	1
XRL	A,@R	1101000r	1	1
XRL	A,#DATA	D3 MM	2	2

THE 8243 INPUT/OUTPUT EXPANDER

This is the only support device built specifically for the 8048 series microcomputers; it expands I/O Port 2 to four individually addressable 4-bit I/O ports. The 8243 Input/Output Expander is particularly useful in numerical applications where data is transferred in 4-bit nibbles.

Figure 6-15 illustrates that part of our general microcomputer system logic which has been implemented on the 8243 Input/Output Expander.

The 8243 Input/Output Expander is packaged as a 24-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL compatible. The device is implemented using N-channel MOS technology.

8243 INPUT/OUTPUT EXPANDER PINS AND SIGNALS

The 8243 Input/Output Expander pins and signals are illustrated in Figure 6-16. Functional internal architecture is illustrated in Figure 6-17.

P20 - P23 represent the 4-bit bidirectional I/O port or bus **connection between the 8243 Input/Output Expander and the 8048 series microcomputer.** P20 - P23 must be connected to the low order four pins of the microcomputer I/O Port 2. Figure 6-18 illustrates the 8243-8048 interface.

P40 - P43, P50 - P53, P60 - P63 and P70 - P73 provide four bidirectional I/O ports, referred to as Ports 4, 5, 6 and 7, respectively. These are 4-bit ports via which data is transferred to or from external logic.

Data being output via one of these four ports is latched and held in a low impedance state.

Data input is buffered. During a read operation 8243 I/O port pins are sampled — while the read is being executed; then I/O port pins are floated.

$\overline{\text{CS}}$ is the single chip select signal for the 8243 device. $\overline{\text{CS}}$ must be low for the device to be selected. There is no specifically defined manner in which $\overline{\text{CS}}$ has to be created; in Figure 6-18 it is shown being decoded off the four high order pins of I/O Port 2.

PROG is the single control strobe output by the 8048 series microcomputer to time 8243 events. On the falling edge of PROG data input via P20 - P23 is decoded as an I/O port select and operation specification. Resulting 8243 operations are strobed by the rising edge of PROG

8243 RESET

There is no Reset input to the 8243. **The device is reset when power is first applied, or when power input at the V_{CC} pin drops below +1 volt.** Following Reset, Port 2 is in Input mode while Ports 4, 5, 6 and 7 are floated. The 8243 device will exit the Reset mode on the first high-to-low transition of PROG.

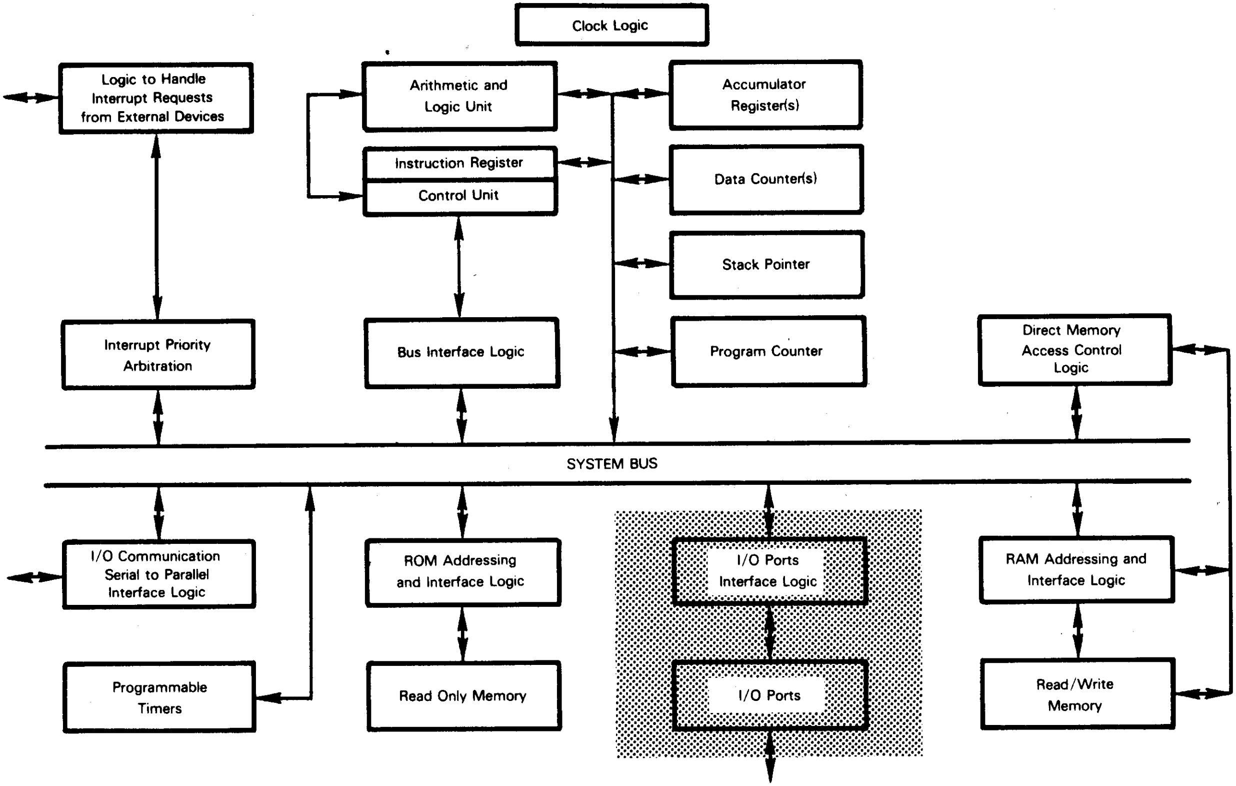
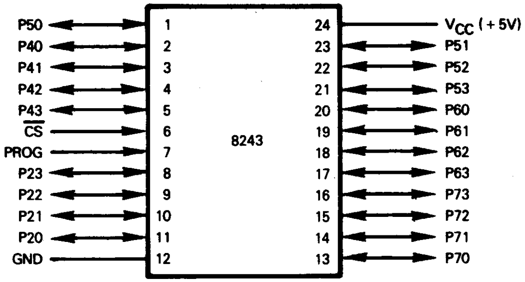

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Instruction Register
Control Unit
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Interrupt Priority Arbitration
Bus Interface Logic
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
Programmable Timers
ROM Addressing and Interface Logic
Read Only Memory
I/O Ports Interface Logic
I/O Ports
RAM Addressing and Interface Logic
Read/Write Memory

Figure 6-15. Logic Of The 8243 Input/Output Expander



PIN NAME	DESCRIPTION	TYPE
P20 - P23	Bidirectional I/O Port to CPU	Bidirectional, tristate
P40 - P43	I/O Port 4	Bidirectional, tristate
P50 - P54	I/O Port 5	Bidirectional, tristate
P60 - P64	I/O Port 6	Bidirectional, tristate
P70 - P74	I/O Port 7	Bidirectional, tristate
PROG	Address/Data Strobe	Input
$\overline{CS}$	Chip Select	Input
V_{CC}, GND	Power, Ground	

Figure 6-16. 8243 Input/Output Expander Pins And Signals

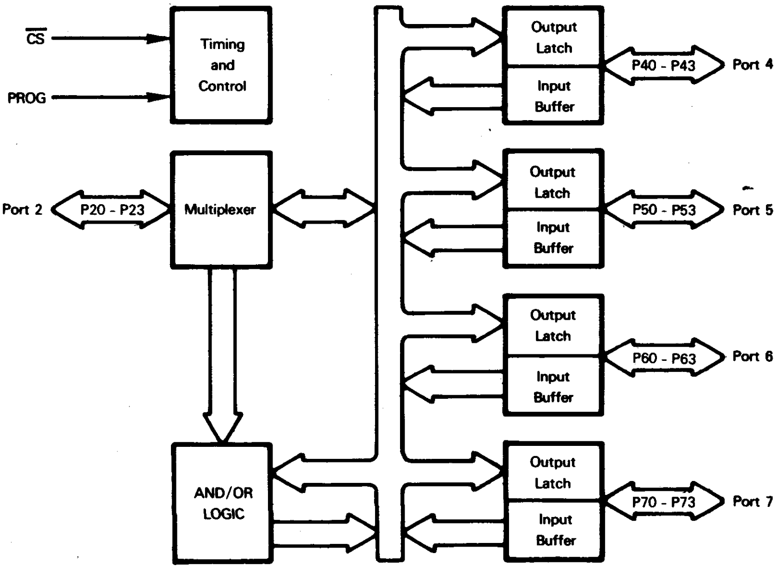
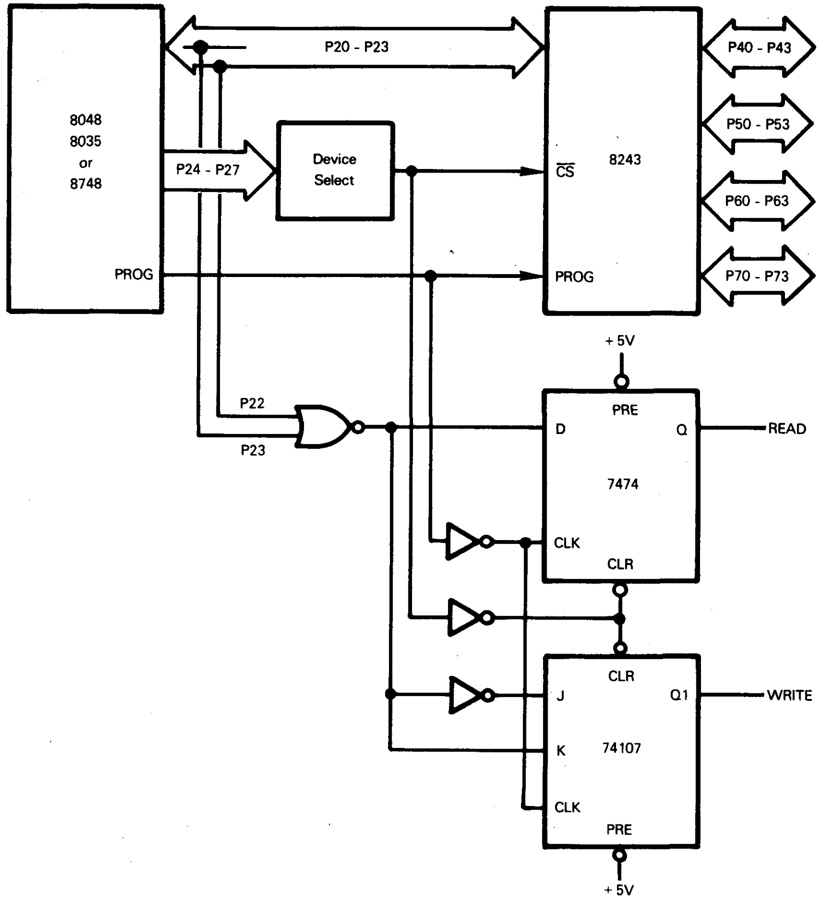


Figure 6-17. Functional Diagram Of The 8243 Input/Output Expander



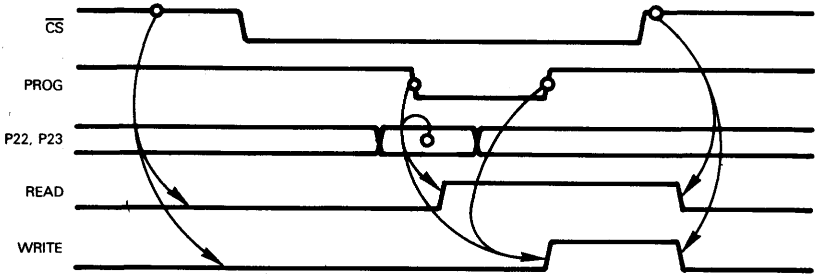


Figure 6-18. An 8243/8048 Configuration With External Logic Read And Write Strobes

8243 INPUT/OUTPUT EXPANDER OPERATIONS

8048 series microcomputers have four instructions designed specifically to access an 8243 Input/Output Expander. These instructions are:

MOVD PN, A
MOVD A, PN
ORLD PN, A
ANLD PN, A

These are the operations performed:

1) **You can output the low order four Accumulator bits** to I/O Expander Port 4, 5, 6 or 7. Following a write operation the four port lines are held in a low impedance state. External logic does not receive any type of "data ready" signal after data has been output; however, as illustrated in Figure 6-18, you can easily create such a signal by combining PROG and device select logic.
2) **You can input data from Port 4, 5, 6 or 7** of the 8243 device to the four low order Accumulator bits. Again Figure 6-18 shows how you can create a strobe signal which tells external logic when to apply data to an I/O port of the 8243 device.
3) **You can** output data from the low order four Accumulator bits to one of the four 8243 device ports, but instead of simply writing to the port, you can **AND or OR with data already in the port output latch.** That is to say, you perform a Boolean operation between the four low order Accumulator bits and the data most recently output to the 8243 port.

You cannot perform a Boolean operation between the low order four Accumulator bits and data input to an 8243 port; the input data is buffered, not latched. You must read the input data to the Accumulator and mask it there.

8243 device Ports 4, 5, 6 and 7 have been designed to operate continuously as input ports or output ports. If you switch a port from input to output, or from output to input, then the first 4-bit data unit written or read will be erroneous and should be discarded.

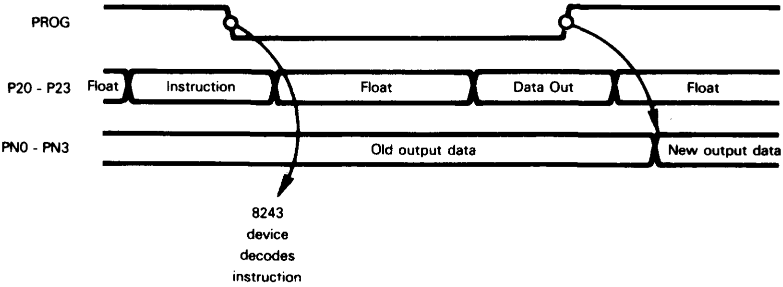


Figure 6-19. Timing For Data Output To An 8243 Port Via A MOVD, ORLD Or ANLD Instruction

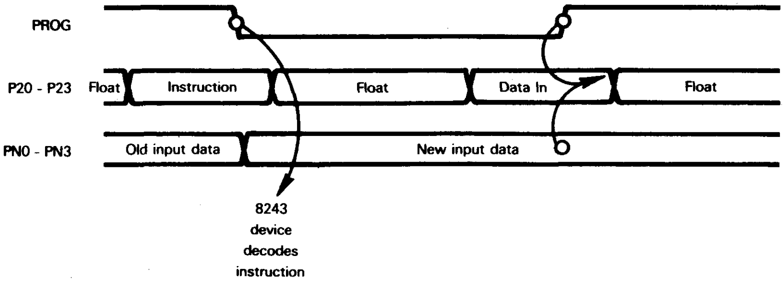


Figure 6-20. Timing For Data Input From An 8243 Port

Timing for 8243 port accesses are illustrated in Figures 6-19 and 6-20.

In each case an instruction is output via P20 - P23 of the 8048 microcomputer on the high-to-low transition of PROG. The instruction is decoded as follows:

P20	P21	8243 Port Selected	P22	P23	Function Defined
0	0	Port 4	0	0	Read from Port
0	1	Port 5	0	1	Write to Port
1	0	Port 6	1	0	OR with Port
1	1	Port 7	1	1	AND with Port

The actual I/O operation within the 8243 device is strobed by the subsequent low-to-high transition of PROG.

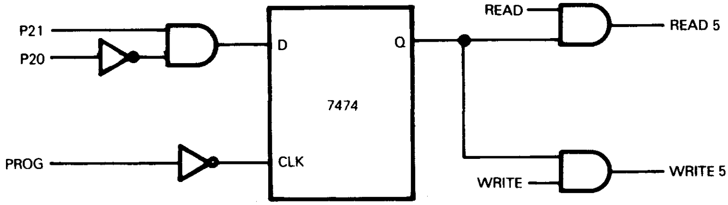
Observe that external logic must transmit data to an 8243 I/O port on the high-to-low transition of PROG. External logic must read data output after the low-to-high transition of PROG. These **signals to external logic are shown in Figure 6-18. Let us take a more careful look at this figure.**

The 8243 device select $\overline{CS}$ is derived in some fashion from the four high order lines of the 8048 I/O Port 2. The manner in which we decode $\overline{CS}$ from these four lines is not relevant; however the fact that we are generating $\overline{CS}$ in this fashion means that any 8243 access instruction must be bracketed by instructions that select and then deselect the 8243 device.

It is not a good idea to leave the 8243 device selected when you are not accessing it; therefore do not leave high order bits of I/O Port 2 in a condition that would select the 8243 device while the device is supposed to be idle.

The PROG signal connecting the 8048 to the 8243 requires no explanation. The signal is output by the 8048 with timing required by the 8243.

The READ and WRITE strobes created in Figure 6-18 identify the time at which external logic must either read data from an I/O port, or write data to an I/O port; however, the I/O port is not itself identified. The READ and WRITE strobes would have to be qualified by P20 and P21 on the high-to-low transition of PROG in order to create READ and WRITE strobes specific to any given I/O port. Here for example is the logic which would make READ and WRITE specific to I/O Port 5:



Referring to the timing in Figure 6-18 let us first look at the READ strobe. This signal must go true on the high-to-low transition of PROG — but only if P22 and P23 are both low. READ can stay high until the device is deselected providing external logic uses the low-to-high transition of READ or timing immediately thereafter, in order to place data at the required I/O port — whence it can be read by the 8048. We obtained the required wave form by using the complement of $\overline{CS}$ as a CLEAR input to the READ 7474 flip-flop. Thus while the 8243 device is not selected READ will be low. The NOR of P22 and P23 becomes the D input to the READ flip-flop; this input will be high only when P22 and P23 are both low — and that specifies a Read operation. On the high-to-low transition of PROG, $\overline{PROG}$ goes low-to-high, and that clocks the READ flip-flop Q output high. READ subsequently stays high until $\overline{CS}$ goes high again, at which point the READ flip-flop is cleared and READ goes low.

A 74107 master-slave flip-flop creates the WRITE pulse. The high-to-low transition of PROG marks the instant at which P22 and P23 must be decoded to determine that a non-read operation is in progress, but the actual low-to-high transition of write must not occur until the subsequent low-to-high transition of PROG.

The 74107 modifies the Q1 output on the trailing edge of CLK, based on the JK inputs at the leading edge of CLK; thus WRITE logic requirements are met.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .... 0°C to 70°C
Storage Temperature .... -65°C to +150°C
Voltage On Any Pin With Respect
to Ground .... -0.5V to +7V
Power Dissipation .... 1.5 Watt

**COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

D.C. AND OPERATING CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = V_{DD} = +5V ±5%, V_{SS} = 0V

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
V_{IL}	Input Low Voltage (All Except XTAL1, XTAL2)	-.5		.8	V	
V_{IH}	Input High Voltage (All Except XTAL1,XTAL2,$\overline{RESET}$)	2.0		V_{CC}	V	
V_{IH1}	Input High Voltage ($\overline{RESET}$,XTAL1)	3.0		V_{CC}	V	
V_{OL}	Output Low Voltage (BUS, $\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)			.45	V	I_{OL} = 2.0mA
V_{OL1}	Output Low Voltage (All Other Outputs Except PROG)			.45	V	I_{OL} = 1.6mA
V_{OH}	Output High Voltage (BUS, $\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	2.4			V	I_{OH} = 100μA
V_{OH1}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = 50μA
I_{IL}	Input Leakage Current (T1, EA, $\overline{INT}$)			±10	μA	$V_{SS} \leqslant V_{IN} \leqslant V_{CC}$
I_{OL}	Output Leakage Current (Bus, T0) (High Impedance State)			-10	μA	$V_{CC} \geqslant V_{IN} \geqslant V_{SS}$ +.45
I_{DD}	V_{DD} Supply Current			30	mA	
I_{CC}	V_{CC} Supply Current			180	mA	

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = V_{DD} = +5V ±5%, V_{SS} = 0V

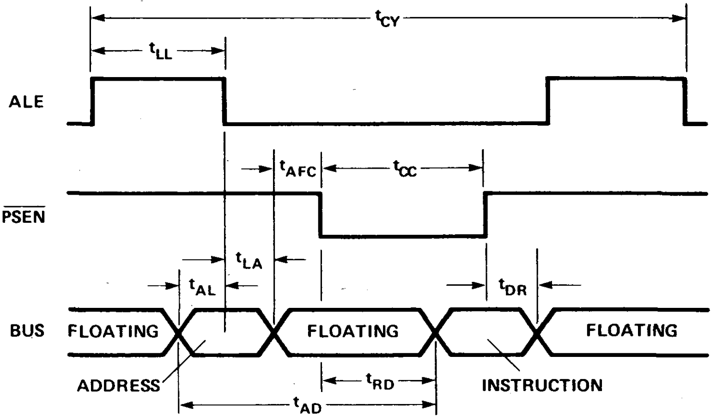
Symbol	Parameter	8048/8748/8035		8048-8 8748-8 8035-8		Unit	Conditions
		Min.	Max.	Min.	Max.		
t_{LL}	ALE Pulse Width	400		800		ns	
t_{AL}	Address Setup to ALE	150		150		ns	
t_{LA}	Address Hold from ALE	80		80		ns	
t_{CC}	Control Pulse Width ($\overline{PSEN}$, $\overline{RD}$, $\overline{WR}$)	900		1800		ns	
t_{DW}	Data Set-Up Before $\overline{WR}$	500		1000		ns	
t_{WD}	Data Hold After $\overline{WR}$	80		80		ns	C_L = 20pF
t_{CY}	Cycle Time	2.5		5.0		μs	6 MHz XTAL (3 MHz XTAL for -8)
t_{DR}	Data Hold	0	130	0	130	ns	
t_{RD}	$\overline{PSEN}$, $\overline{RD}$ to Data In		500		1000	ns	
t_{AW}	Address Setup to $\overline{WR}$	230		260		ns	
t_{AD}	Address Setup to Data In		950		1900	ns	
t_{AFC}	Address Float to $\overline{RD}$, $\overline{PSEN}$	0		0		ns	

A.C. TEST CONDITIONS

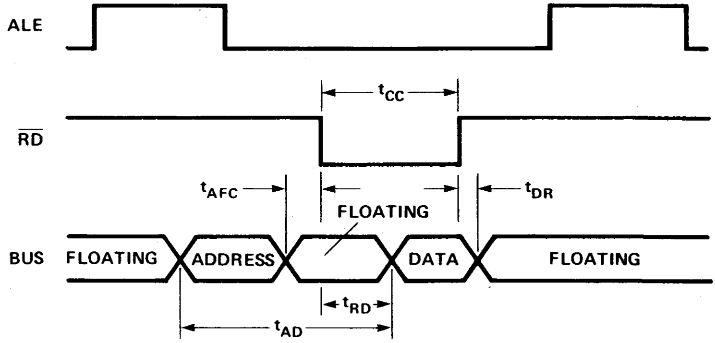
Control Outputs: C_L = 80 pF, 2.2K to V_{SS}, 4.3K to V_{CC}
BUS Outputs: C_L = 150 pF, 2.2K to V_{SS}, 4.3K to V_{CC} t_{CY} = 2.5μs

WAVEFORMS

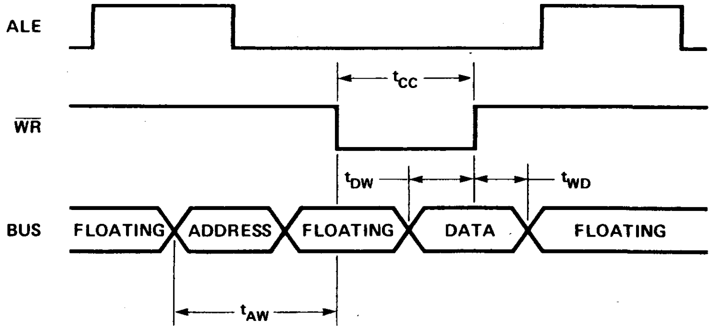
INSTRUCTION FETCH FROM EXTERNAL PROGRAM MEMORY



READ FROM EXTERNAL DATA MEMORY



WRITE TO EXTERNAL DATA MEMORY



Chapter 7
ZILOG Z80

Zilog Z80 microcomputer devices have been designed as 8080A enhancements. In fact, the same individuals responsible for designing the 8080A CPU at Intel designed the Z80 devices at Zilog. The 8085, described in Chapter 5, is Intel's 8080A enhancement.

The Z80 instruction set includes all 8080A instructions as a subset. In deference to rational necessity, however, neither the Z80 CPU, nor any of its support devices attempt to maintain pin-for-pin compatibility with 8080A counterparts. Compatibility is limited to instruction sets and general functional capabilities. A program that has been written to drive an 8080A microcomputer system will also drive the Z80 system — within certain limits; for example, a ROM device that has been created to implement object programs for an 8080A microcomputer system can be physically removed and used in a Z80 system.

But Z80-8080A compatibility does extend somewhat further, since most support devices that have been designed for the 8080A CPU will also work with a Z80 CPU; therefore in many cases you will be able to upgrade an 8080A microcomputer system to a Z80, confining hardware modifications to the CPU and its immediate interface only.

It is interesting to note that the Z80 pins and signal interface is far closer than the 8085 to the three-chip 8080A configuration illustrated in Figure 4-33. Also, whereas the Z80 instruction set is greatly expanded as compared to the 8080A, the 8085 instruction set contains just two new instructions. However, both the Z80 and the 8085 have resolved the two most distressing problems associated with the 8080A — the three-chip 8080A CPU has in both cases been reduced to one chip, and the three 8080A power supplies have in both cases been reduced to a single +5V power supply.

Zilog, Inc., manufacturers of the Z80, are located at:

10460 Bubb Road
Cupertino, California 95014

The official second source for Zilog products is:

Mostek, Inc.
1215 West Crosby Road
Carrollton, Texas 75006

Z80 LSI TECHNOLOGY

N-Channel MOS technology is used for all Z80 devices.

THE Z80 CPU

Functions implemented on the Z80 CPU are illustrated in Figure 7-1. They represent "typical" CPU logic, equivalent to the three devices: 8080A CPU, 8224 Clock and 8228 System Controller.

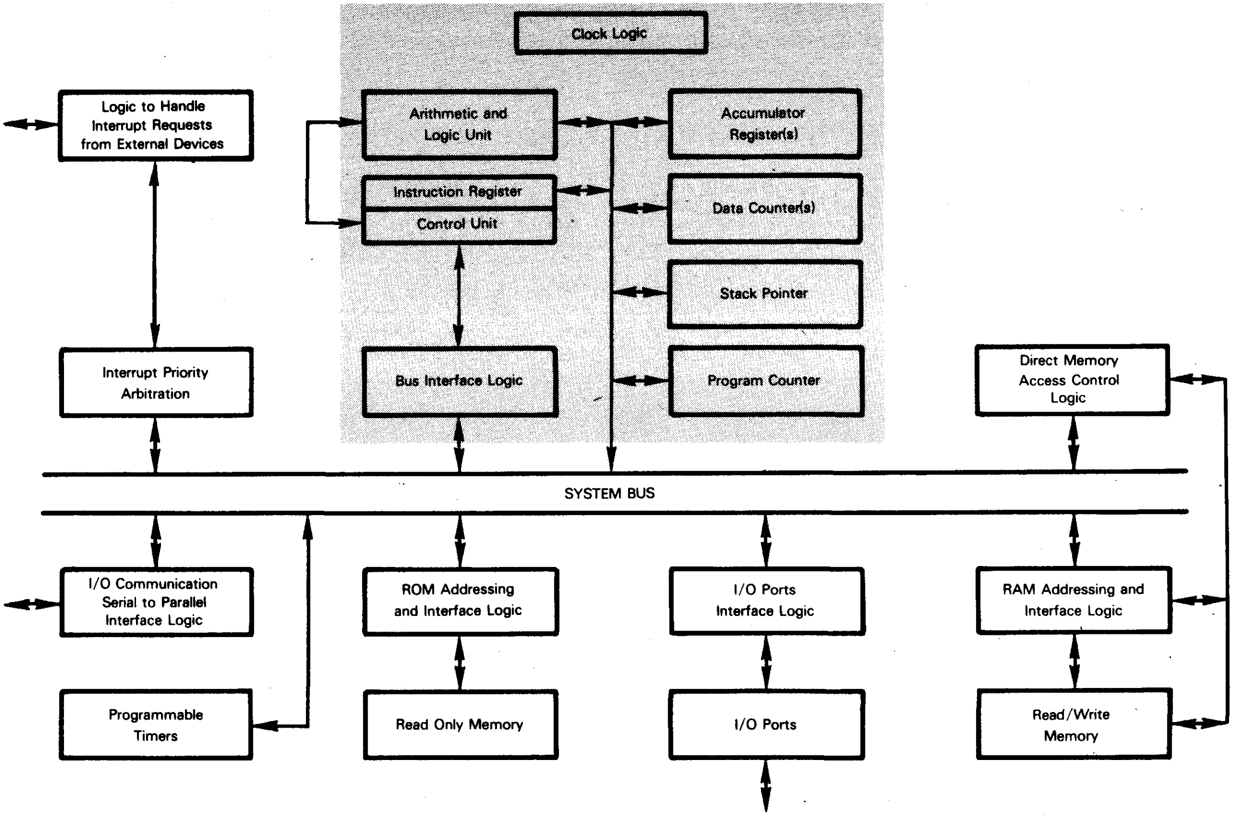

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 7-1. Logic Functions Of The Z80 CPU

A SUMMARY OF Z80/8080A DIFFERENCES

We are going to summarize Z80/8080A differences before describing differences in detail. If you know the 8080A well, read on; if you do not, come back to this summary after reading the rest of the Z80 CPU description. We will also contrast the Z80 and the 8085, where relevant.

For the programmer, the Z80 provides more registers and addressing modes than the 8080A, plus a much larger instruction set.

Significant hardware features are a single power supply (+5V), a single system clock signal, an additional interrupt, and logic to refresh dynamic memories.

The 8085 also has a single power supply and a single system clock signal. The 8085 has three additional interrupts, but lacks logic to refresh dynamic memories.

Is the Z80 CPU indeed the logical next 8080A evolution?

Hardware aspects of the 8080A represent its weakest features, as compared to principal current competitors. Specifically, the fact that the 8080A is really a three-chip CPU is its biggest single problem; three chips are always going to cost more than one. Next, the fact that the 8080A requires three power supplies (+5V, -5V and +12V) is a very negative feature for many users and the desirability of going to a single power supply is self-evident; the Z80 requires a single +5V power supply. This is also true of the 8085.

The problems associated with condensing logic from three chips onto one chip are not so straightforward. Figure 7-2 illustrates the standard three-chip 8080A CPU. Let us assume that the three devices are to be condensed into a single chip. Asterisks (*) have been placed by the signals which must be maintained if the single chip is to be hardware compatible with the three chips it replaces. Forty-three signals are asterisked, therefore the standard 40-pin DIP cannot be used. The problem is compounded by the fact that not all 8080A systems use an 8228 System Controller. Some 8080A systems use an 8212 bidirectional I/O port to create control signals. A few of the earliest 8080 systems use neither the 8228 System Controller, nor an 8212 I/O port; rather external logic decodes the Data Bus when SYNC is true in order to generate control signals; for example, that is how the TMS5501 works. We must therefore conclude that any attempt to reduce three chips to one will create a product that is not pin compatible with the 8080A; and, indeed, the Z80 is not pin compatible. What Zilog has done is include as many hardware enhancements as possible within the confines of a 40-pin DIP that must be philosophically similar to the 8080A, without attempting any form of pin compatibility. Figure 7-2 identifies the correlation between Z80 signals and 8080A signals. Notice that there is a significant similarity.

Figure 5-3 is equivalent to Figure 7-2, comparing 8085 and 8080A signals. Z80 signals are far closer to the 8080A three-chip set than the 8085.

Here is a summary of the hardware differences:

1) The Z80 has reduced three power supplies to a single +5V power supply.

2) Clock logic is entirely within the Z80.

3) The complex, two clock signals of the 8080A have been replaced by a single clock signal.

4) Automatic dynamic memory refresh logic has been included within the CPU.

5) Read and write control signal philosophy has changed. The 8080A uses separate memory read, memory write, I/O read and I/O write signals. The Z80 uses a general read and a general write, coupled with a memory select and an I/O select. This means that if a Z80 CPU is to replace an 8080A CPU then additional logic will be required beyond the Z80 CPU. You will either have to combine the four Z80 control signals to generate 8080A equivalents, or you will have to change the select and strobe logic for every I/O device. We will discuss this in more detail later.

6) Address and Data Bus float timing associated with DMA operations have changed. The 8080A floats these busses at the beginning of the third or fourth time period within the machine cycle during which a bus request occurs; this initiates a Hold state. The Z80 has a more straightforward scheme; a Bus Request input signal causes the Data and Address Busses to float at the beginning of the machine cycle; floating busses are acknowledged with a Bus Acknowledge output signal.

7) The Z80 has an additional interrupt request. In addition to the RESET and normal 8080A interrupt request, the Z80 has a nonmaskable interrupt which is typically used to execute a short program that prepares for power failure, once a power failure has been detected.

Now consider internal organization of the Z80 in terms of instruction set compatibility and enhancement.

As illustrated by Table 7-3 the 8080A instruction set is, indeed, a subset of the Z80 instruction set. Unfortunately, the Z80 uses completely new source program instruction mnemonics, therefore 8080A instructions cannot immediately be identified.

There are very few unused object codes in the 8080A instruction set. The Z80 has therefore taken what few unused object codes there are, and used them to specify that an additional byte of object code follows:

```
11011101 ◄—— Spare 8080A object code
         ◄—— Specifies new Z80 object code follows
```

This results in new Z80 instructions having 16-bit object codes; but simultaneously it means that a very large number of new instructions can be added.

Any enhancement of the 8080A can include major changes within the CPU; providing the 8080A registers and status flags remain as a subset of the new design, instruction compatibility remains. These are the principal enhancements made by the Z80:

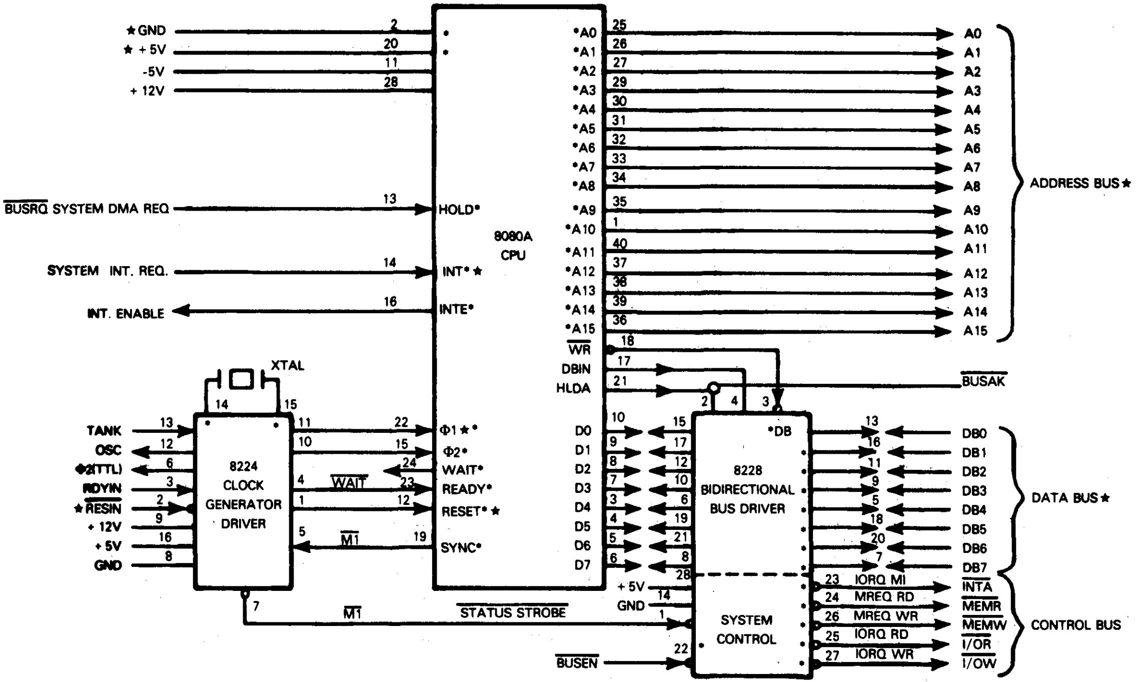
1) The standard general purpose registers and status flags have been duplicated. This makes it very easy to handle single-level interrupts, since general purpose register and Accumulator contents no longer need to be saved on the Stack; instead, the program may simply switch to the alternate register set.

2) Two Index registers have been added. This means that additional Z80 instructions can use indexed memory addressing.

2) An Interrupt Vector register allows external logic the option of responding to an interrupt acknowledge by issuing the equivalent of a Call instruction — which vectors program execution to a memory address which is dedicated to the acknowledged external logic.

4) A single Block Move instruction allows the contents of any number of contiguous memory bytes to be moved from one area of memory to another, or between an area of memory and a single I/O port. You can also scan a block of memory for a defined value by executing a Block Compare instruction.

5) Instructions have been added to test or alter the condition of individual register and memory bits.



$\overline{\text{MI}}$ Z80 equivalent, or new signals.
★ Signals reproduced by the Z80.
• Signals which must be duplicated by a hardware replacement product.

$\overline{\text{RFSH}}$ $\overline{\text{HALT}}$ $\overline{\text{NMI}}$ } These are Z80 signals with no 8080A counterpart

Figure 7-2. The Standard 8080A Three-Chip System And Z80 Signal Equivalents

In contrast to the extensive enhancements of the Z80, the 8085 registers and status architecture are identical to the 8080A. There are only two additional instructions in the 8085 instruction set; however, the 8085, like the Z80, allows Call instructions to be used when acknowledging an interrupt — a particularly useful enhancement.

While on the surface the Z80 instruction set appears to be very powerful, note that instruction sets are very subjective; right and wrong, good and bad are not easily defined. Let us look at some nonobvious features of the Z80 instruction set.

First of all, the execution speed advantage that results from the new Z80 instructions is reduced by the fact that all of these instructions require two bytes of object code. Some examples of Z80 instructions and equivalent 8080A instruction sequences with equivalent cycle times are given in Table 7-1.

Table 7-1. Comparisons Of Z80 And 8080A Instruction Execution Cycles

Z80			8080A		
Instructions		Cycles	Instructions		Cycles
LD	R,(IX + d)	19	LXI DAD MOV	H,d IX R,M	10 10 7 27
LD	RP,ADDR	20	LHLD MOV MOV	ADDR C,L B,H	16 5 5 26
SET	B,(HL)	15	MOV ORI MOV	A,M MASK M,A	7 7 7 21

Also, a novice programmer may find the Z80 instruction set bewilderingly complex. At a time when the majority of potential microcomputer users are terrified by simple assembly language instruction sets, it is possible that users will react negatively to an instruction set whose complexity (if not power) rivals that of many large minicomputers.

Many of the new Z80 instructions use direct, indexed memory addressing to perform operations which are otherwise identical to existing 8080A instructions. Now the Z80 has two new 16-bit Index registers whose contents are added to an 8-bit displacement provided by the instruction code; this is the scheme adopted by the Motorola MC6800. This scheme is inherently weaker than having a 16-bit, instruction-provided displacement, as implemented by the Signetics 2650. When the Index register is larger than the displacement, the Index register, in effect, becomes a base register. When the Index register has the same size, or is smaller than the displacement, it is truly an Index register as described in "Volume I — Basic Concepts". The Signetics 2650 implementation is more powerful.

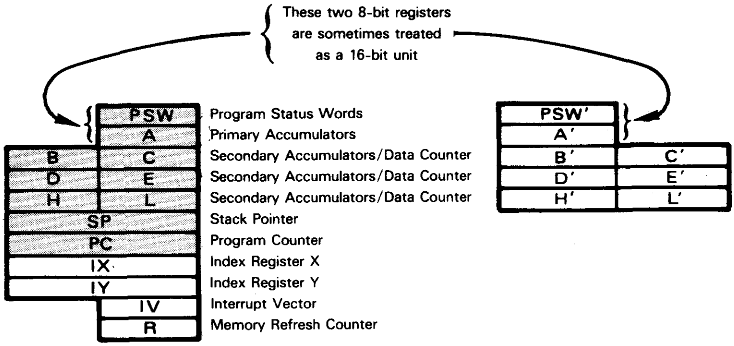
Z80 PROGRAMMABLE REGISTERS

We will now start looking at the Z80 CPU in detail, beginning with its programmable registers.

The Z80 has two sets of 8-bit programmable registers, and two Program Status Words. At any time one set of programmable registers and one Program Status Word will be active and accessible.

In addition, the Z80 has a 16-bit Program Counter, a 16-bit Stack Pointer, two 16-bit Index registers, an 8-bit Interrupt Vector and an 8-bit Memory Refresh register.

Figure 7-3 illustrates the Z80 registers. Within this figure, the 8080A registers' subset is shaded.



Shaded registers represent the 8080A subset.

Figure 7-3. Z80 Programmable Registers

The Z80 uses its Program Status Word, its A, B, C, D, E, H, and L registers, plus the Stack Pointer and the Program Counter exactly as the 8080A uses these locations; therefore no additional discussion of these registers is needed.

The Program Status Word, plus registers A, B, C, D, E, H and L are duplicated. Single Z80 instruction allow you to switch access from one register set to another, or to exchange the contents of selected registers. At any time, one or the other set of registers, but not both, are accessible.

There are two 16-bit Index registers, marked IX and IY. These are more accurately looked upon as base registers, as will become apparent when we examine Z80 addressing modes.

The Interrupt Vector register performs a function similar to the ICW2 byte of the 8259 PICU device. Z80 interrupt acknowledge logic gives you the option of initiating an interrupt service routine with a Call instruction, where the high order address byte for the call is provided by the Interrupt Vector register. The 8085 also provides this capability.

The Memory Refresh Counter register represents a feature of microcomputer systems which has been overlooked by everyone except Fairchild and Zilog. Dynamic memory devices will not hold their contents for very long, irrespective of whether power is off or on. A dynamic memory must therefore be accessed at millisecond intervals. Dynamic memory devices compensate for this short-coming by being very cheap — and dynamic refresh circuitry is very simple. Using a technique akin to direct memory access, dynamic refresh circuitry will periodically access dynamic memories, rewriting the contents of individual memory words on each access. About the only logic needed by dynamic refresh is a counter via which it keeps track of its progress through the dynamic memory; that is the purpose of the Z80 Memory Refresh Counter register. The Z80 also has a special DMA refresh control signal; therefore the Z80 provides all necessary dynamic refresh logic.

Z80 ADDRESSING MODES

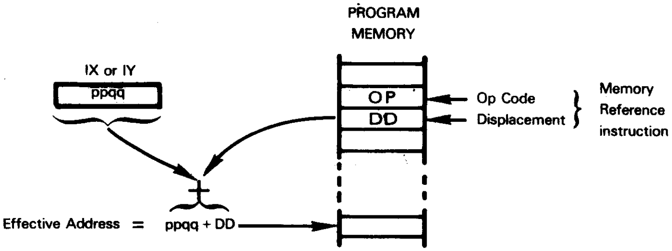
Z80 instructions use all of the 8080A addressing modes; the Z80 also has these two enhancements:

1) A number of memory reference instructions use the IX and IY registers for indexed, or base relative addressing.

2) There are some two-byte program relative Jump instructions.

Z80 INDEXED ADDRESSING

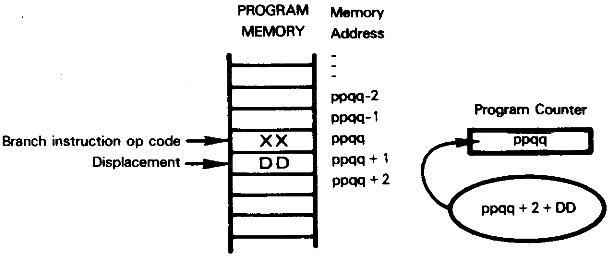
A memory reference instruction that uses the IX or IY register will include a single data displacement byte. The 8-bit value provided by the instruction object code is added to the 16-bit value provided by the identified Index register in order to compute the effective memory address:



p, q and D represent any hexadecimal digits;
DD represents an 8-bit, signed binary value.

This is standard microcomputer indexed addressing and is less powerful than having the memory reference instruction provide a 16-bit base address or displacement; for a discussion of these addressing modes see "Volume I — Basic Concepts", Chapter 6.

The program relative, two-byte Jump instructions provided by the Z80 provide standard two-byte, program relative addressing. A single, 8-bit displacement is provided by the Jump instruction's object code; this 8-bit displacement is added, as a signed binary value, to the contents of the Program Counter — after the Program Counter has been incremented to point to the sequential instruction:



Next instruction object code will be fetched from memory location ppqq + 2 + DD. p, q and D represent any hexadecimal digits. DD represents a signed binary, 8-bit value.

For a discussion of program relative addressing, see "Volume I — Basic Concepts".

The Z80 addressing enhancements are of significant value when comparing the Z80 to the 8080A.

The value of the Index register comes not so much from having an additional addressing option, but rather IX and IY allow an efficient programmer to husband his CPU register space more effectively. Look upon IX and IY as performing memory addressing tasks which the 8080A would have to perform using the BC and DE registers. By freeing up the BC and DE registers for data manipulation, you can significantly reduce the number of memory reference instructions executed by the Z80.

The two-byte program relative Jump instruction is useful because in most programs 80% of the Jump instructions branch to a memory location that is within 128 bytes of the Jump. That is the rationale for most microcomputers offering two-byte as well as three-byte Jump instructions.

Z80 STATUS

The Z80 and 8080A both use the Program Status Word in order to store status flags. These are the Z80 status flags:

Carry (C)
Zero (Z)
Sign (S)
Parity/Overflow (P/O)
Auxiliary Carry (A_C)
Subtract (N)

Statuses are recorded in the Program Status Word by the Z80, as compared to the 8080A, as follows:

7	6	5	4	3	2	1	0	Bit No.
S	Z	X	A_C	X	P/O	N	C	Z80 Program Status Word

7	6	5	4	3	2	1	0	Bit No.
S	Z	X	A_C	X	P	X	C	8080A Program Status Word

The Parity/Overflow and Subtract statuses differ from the 8080A. All other statuses are the same.

The 8080A has a Parity status but no Overflow status. The Z80 uses a single status flag for both operations, which makes a lot of sense. The Z80 Overflow status is absolutely standard, therefore only has meaning when signed binary arithmetic is being performed — at which time the Parity status has no meaning. Within the Z80, therefore, this single status is used by arithmetic operations to record overflow and by other operations to record parity. For a complete discussion of the Overflow status see "Volume I — Basic Concepts".

The Subtract status is used by the DAA instruction for BCD operations, to differentiate between decimal addition or subtraction. The Subtract and Auxiliary Carry statuses cannot be used as conditions for program branching (conditional Jump, Call or Return instructions).

Z80 CPU PINS AND SIGNALS

The Z80 CPU pins and signals are illustrated in Figure 7-4. Figure 7-2 provides the direct comparison between Z80 CPU signals and the standard 8080A, 8228, 8224 three-chip systems.

Let us first look at the Data and Address Busses.

The 16 address lines A0 - A15 output memory and I/O device addresses. The address lines are tristate; they may be floated by the Z80 CPU, giving external logic control of the Address Bus. **There is no difference between Z80 and 8080A Address Bus lines.**

The Data Bus lines D0 - D7 transmit bidirectional data into or out of the Z80 CPU. Like the Address Bus lines, the Data Bus lines are tristate. **The Z80 Data Bus lines do differ from the 8080A equivalent.** The 8080A Data Bus is multiplexed; status output on the Data Bus by the 8080A during the T_2 clock period of every machine cycle is strobed by the SYNC pulse. The Z80 does not multiplex the Data Bus in this way.

Z80 SYSTEM CONTROL SIGNALS

Control signals are described next; these may be divided into system control, CPU control and Bus control. First we will describe the System control signals.

$\overline{\text{M1}}$ identifies the instruction fetch machine cycle of an instruction's execution. Its function is similar, but not identical to the 8080A SYNC pulse.

$\overline{\text{MREQ}}$ identifies any memory access operation in progress; it is a tristate control signal.

$\overline{\text{IORQ}}$ identifies any I/O operation in progress. When $\overline{\text{IORQ}}$ is low, A0 - A7 contain a valid I/O port address. **$\overline{\text{IORQ}}$ is also used as an interrupt acknowledge; an interrupt is acknowledged by $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ being output low** — a unique combination, since $\overline{\text{M1}}$ is otherwise low only during an instruction fetch, which cannot address an I/O device.

$\overline{\text{RD}}$ is a tristate signal which indicates that the CPU wishes to read data from either memory or an I/O device, as identified by $\overline{\text{MREQ}}$ or IORQ.

$\overline{\text{WR}}$ is a tristate control signal which indicates that the CPU wishes to write data to memory or an I/O device as indicated by $\overline{\text{MREQ}}$ and $\overline{\text{IORQ}}$.

$\overline{\text{RFSH}}$ is a control signal usd to refresh dynamic memories. When $\overline{\text{RFSH}}$ is output low, the current $\overline{\text{MREQ}}$ signal should be used to refresh dynamic memory, as addressed by the lower seven bits of the Address Bus, A0 - A6.

Next we will describe CPU control signals.

Z80 CPU CONTROL SIGNALS

$\overline{\text{HALT}}$ is output low following execution of a Halt instruction. The CPU now enters a Halt state during which it continuously re-executes a NOP instruction in order to maintain memory refresh activity. A Halt can only be terminated with an interrupt.

$\overline{\text{WAIT}}$ is equivalent to the 8080A READY input. External logic which cannot respond to a CPU access request within the allowed time interval extends the time interval by pulling the $\overline{\text{WAIT}}$ input low. In response to WAIT low, the Z80 enters a Wait state during which the CPU inserts an integral number of clock periods; taken together, these clock periods constitute a Wait state.

$\overline{\text{INT}}$ and $\overline{\text{NMI}}$ are two interrupt request inputs. The difference between these two signals is that $\overline{\text{NMI}}$ has higher priority and cannot be disabled.

Z80 BUS CONTROL SIGNALS

There are two Bus control signals.

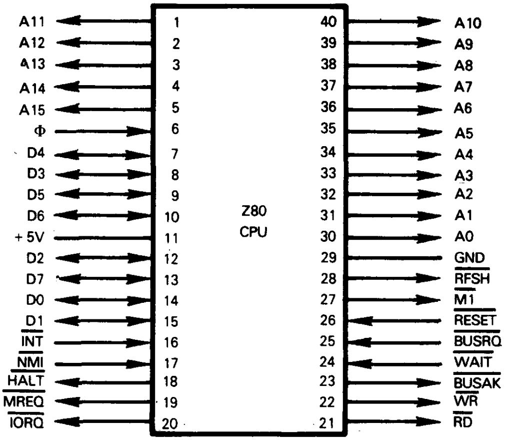
$\overline{\text{RESET}}$ is a standard reset control input. When the Z80 is reset, this is what happens:

The Program Counter, IV and R registers' contents are all set to zero.

Interrupt requests via $\overline{\text{INT}}$ are disabled.

All tristate bus signals are floated.

$\overline{\text{BUSRQ}}$ and $\overline{\text{BUSAK}}$ are bus request and acknowledge signals. In order to perform any kind of DMA operation, external logic must acquire control of the microcomputer System Bus. This is done by inputting $\overline{\text{BUSRQ}}$ low; at the conclusion of the current machine cycle, the Z80 CPU will float all tristate bus lines and will acknowledge the bus request by outputting $\overline{\text{BUSAK}}$ low.



PIN NAME	DESCRIPTION	TYPE
A0 - A15	Address Bus	Tristate, Output
D0 - D7	Data Bus	Tristate, Bidirectional
$\overline{\text{M1}}$	Identifies instruction fetch machine cycle	Output
$\overline{\text{MREQ}}$	Memory request — indicates that CPU is performing memory access	Tristate, Output
$\overline{\text{IORQ}}$	I/O request — indicates I/O operation in progress	Tristate, Output
$\overline{\text{RD}}$	CPU read from memory or I/O device	Tristate, Output
$\overline{\text{WR}}$	CPU write to memory or I/O device	Tristate, Output
$\overline{\text{RFSH}}$	Refresh dynamic memories	Output
$\overline{\text{HALT}}$	CPU Halt executed	Output
$\overline{\text{WAIT}}$	Wait state request	Input
$\overline{\text{INT}}$	Interrupt request	Input
$\overline{\text{NMI}}$	Nonmaskable interrupt request	Input
$\overline{\text{RESET}}$	Reset and initialize CPU	Input
$\overline{\text{BUSRQ}}$	Request for control of Address, Data and Control Busses	Input
$\overline{\text{BUSAK}}$	Bus acknowledge	Output
Φ	CPU clock	Input
+5V, GND	Power and Ground	

Figure 7-4. Z80 CPU Signals And Pin Assignments

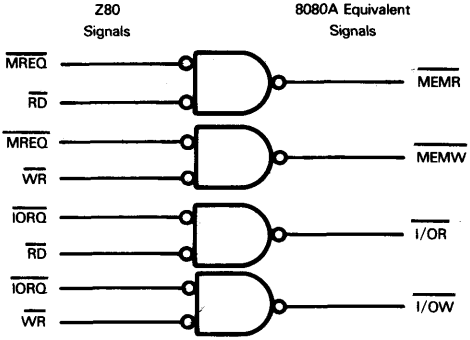
Z80 - 8080A SIGNAL COMPATIBILITY

If you are designing a new product around the Z80 CPU, then questions of Z80 - 8080A signal compatibility are irrelevant; you will design for the CPU on hand.

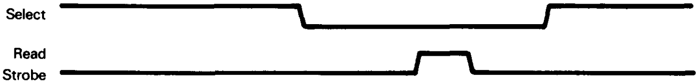
If you are replacing an 8080A with a Z80, then it would be helpful to have some type of lookup table which directly relates 8080A signals to Z80 signals. Unfortunately, such a lookup table cannot easily be created. The problem is that the Z80 is an implementation of three devices: the 8080A CPU, the 8224 Clock, and 8228 System Controller; but there are very many 8080A configurations that do not include an 8228 System Controller.

Possibly the most important conceptual difference between the Z80 and 8080A involves read and write control signals. **The 8228 System Controller develops four discrete control signals for memory read, memory write, I/O read and I/O write. The**

Z80 has a general read and a general write, coupled with an I/O select and a memory select. By adding logic, it would be easy enough to generate the four discrete 8080A signals from the two Z80 signal pairs; here is one elementary possibility:

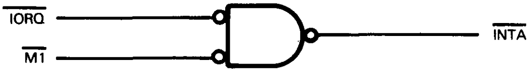


If your design allows it, however, it would be wiser to extend the Z80 philosophy to the various support devices surrounding the CPU. Recall from our discussion of 8080A support devices in Chapter 4 that every device requires separate device select and device access logic. For some arbitrary read operation, timing might be illustrated as follows:



With an 8080A scheme, select logic is decoded from Address Bus lines, while strobe logic depends on one of the four control lines $\overline{I/OR}$, $\overline{I/OW}$, $\overline{MEMR}$ or $\overline{MEMW}$. Using the Z80 philosophy, the memory select ($\overline{MREQ}$) or I/O select ($\overline{IORQ}$) control lines become part of the device select logic, while the read ($\overline{RD}$) or write ($\overline{WR}$) controls generate the strobe.

The Z80 has no interrupt acknowledge signal; rather it combines $\overline{IORQ}$ with $\overline{M1}$ as follows:



The 8080A HOLD and HLDA signals are functionally reproduced by the Z80 $\overline{BUSRQ}$ and $\overline{BUSAK}$ signals.

The 8080A SYNC pulse has no direct Z80 equivalent. $\overline{M1}$ is pulsed low during an instruction fetch, or an interrupt acknowledge, but it is not pulsed low during the initial time periods of an instruction's second or subsequent machine cycles. **Frequently the complement of $\overline{M1}$ can be used instead of SYNC** to drive those 8080A peripheral devices that require the SYNC pulse.

The Z80 has no signals equivalent to 8080A INTE, WAIT or Φ2. There is also no signal equivalent to the 8228 $\overline{BUSEN}$.

If for any reason external logic must know when interrupts have been disabled internally by the CPU, then the Z80 will be at a loss to provide any signal equivalent to the 8080A control signals. Remember INTE in an 8080A system tells external logic when the CPU has enabled or disabled all interrupts;since external logic can do nothing about interrupts being disabled, and requesting an interrupt at this time does neither good nor harm, knowing that the condition exists is generally irrelevant.

The single Z80 $\overline{\text{WAIT}}$ input serves the function of the 8080A READY input. Irrespective of when the WAIT is requested, a Wait clock period will only be inserted between T_2 and T_3; moreover, as we will see shortly, there are certain Z80 instructions which automatically insert a Wait state, without waiting for external demand. You would need relatively complex logic to decode instruction object codes, clock signal and the $\overline{\text{WAIT}}$ input if your Z80 system is to generate the equivalent of an 8080A $\overline{\text{WAIT}}$ output. In all probability, it would be simpler to find an alternative scheme that did not require a signal equivalent to the 8080A $\overline{\text{WAIT}}$ output.

The Z80 simply has no second clock equivalent to 8080A $\Phi 2$. Any device that needs clock signal $\Phi 2$ cannot be used in Z80 configurations.

The 8228 $\overline{\text{BUSEN}}$ input is used by external logic to float the System Bus. In a Z80 system, CPU logic floats the System Bus; therefore $\overline{\text{BUSEN}}$ becomes irrelevant.

The 8080A CPU has no signals equivalent to Z80 $\overline{\text{RFSH}}$, $\overline{\text{HALT}}$ and $\overline{\text{NMI}}$.

$\overline{\text{RFSH}}$ applies to dynamic memory refresh only; it is irrelevant within the context of a Z80 - 8080A signal comparison.

$\overline{\text{NMI}}$, being a nonmaskable interrupt request, also has no 8080A equivalent logic.

The Z80 $\overline{\text{HALT}}$ output needs some discussion. One of the more confusing aspects of the 8080A is the interaction of Wait, Halt and Hold states. Let us look at these three states, comparing the Z80 and 8080A configurations and in the process we will see the purpose of the Z80 $\overline{\text{HALT}}$ output.

The purpose of the Wait state is to elongate a memory reference machine cycle in deference to slow external memory or I/O devices. The Wait state consists of one or more Wait clock periods inserted between T_2 and T_3 of a machine cycle. The 8080A and the Z80 handle Wait states in exactly the same way, except for the fact that the Z80 has no Wait acknowledge output and under certain circumstances will automatically insert Wait clock periods.

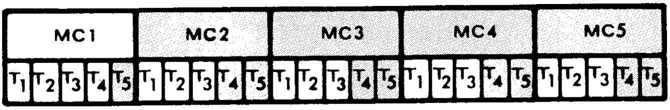
The purpose of the Hold condition is to allow external logic to acquire control of the System Bus and perform Direct Memory Access operations. Again both the Z80 and the 8080A have very similar Hold states. The only significant difference is that the Z80 initiates a Hold state at the conclusion of a machine cycle, whereas the 8080A initiates the Hold state during time period T_3 or T_4. The 8228 System Controller also needs a high $\overline{\text{BUSEN}}$ input in order to float its Data and Control Busses while the Z80 has no equivalent need.

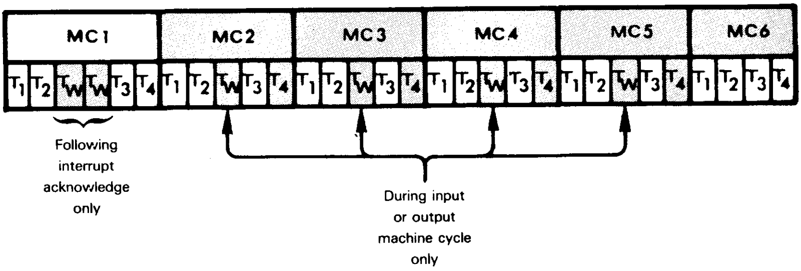
The big difference between the Z80 and the 8080A comes within the Halt state. When the 8080A executes a Halt instruction, it goes into a Halt state, which differs from a Hold state. There are some complex interactions between Hold, Halt, Wait and interrupts within 8080A systems. None of these complications exists in the Z80 system, since the Z80 has no Halt state. After executing a Halt instruction, the Z80 outputs $\overline{\text{HALT}}$ low, then proceeds to continuously execute a NOP instruction. This allows dynamic memory refresh logic to continue operating. **If you are replacing an 8080A with a Z80, you must give careful attention to the Halt state. This is one condition where unexpected incompatibilities can arise.**

Z80 TIMING AND INSTRUCTION EXECUTION

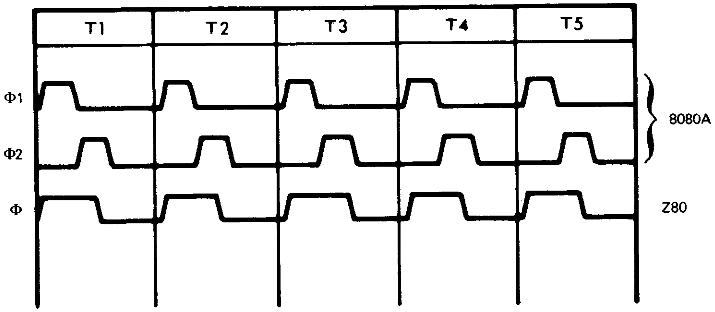
Z80 timing is conceptually similar to, but far simpler than 8080A timing. Like the 8080A, the Z80 divides its instructions into machine cycles and clock periods. However, all Z80 machine cycles consist of either three or four clock periods. Some instructions always insert Wait clock periods, in which case five or six clock periods may be present in a machine cycle. Recall that 8080A machine cycles may have three, four or five clock periods.

The 8080A may require from one to five machine cycles in order to execute an instruction; Z80 instructions execute in one to six machine cycles. If we shade optional machine cycles and time periods Z80 and 8080A instruction time subdivisions may be compared and illustrated as follows:





Z80 clock signals are also far simpler than the 8080A equivalent. Where the 8080A uses two clock signals the Z80 uses one. Clock logic may be compared as follows:



INSTRUCTION FETCH EXECUTION SEQUENCES

As compared to the 8080A, Z80 instruction timing is marvelously simple. Gone is the SYNC pulse and the decoding of Data Bus for status. Every instruction's timing degenerates into an instruction fetch, optionally followed by memory or I/O read or write. Add to this a few variations for Wait state, interrupt acknowledge and bus floating and you are done.

Let us begin by looking at an instruction fetch. Timing is illustrated in Figure 7-5. Look at Figure 4-5 to obtain an immediate comparison of the Z80 and the 8080A.

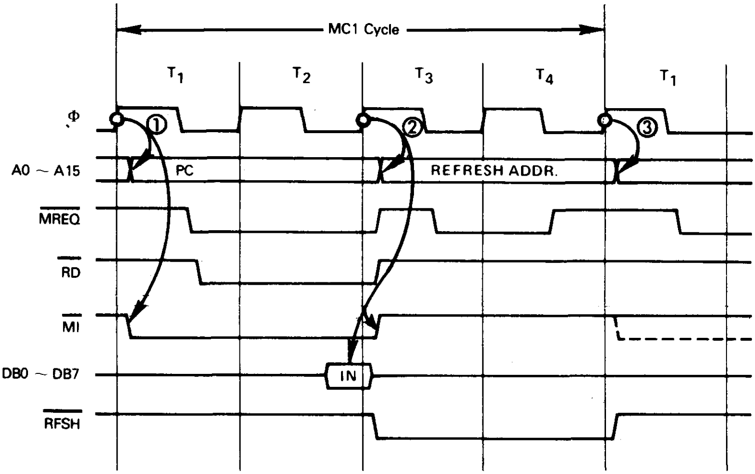


Figure 7-5. Z80 Instruction Fetch Sequence

Referring to Figure 7-5, note that the instruction fetch cycle is identified by $\overline{M1}$ output low during T$_1$ and T$_2$ (①,②). Since there is no status on the Data Bus to worry about, the Program Counter contents are output immediately on the Address Bus and stay stable for the duration of T$_1$ and T$_2$.

Since an instruction fetch is also a memory operation, $\overline{MREQ}$ and $\overline{RD}$ controls are both output low. This occurs half-way through T$_1$, at which time the Address Bus will stabilize. The falling edges of $\overline{MREQ}$ and $\overline{RD}$ can therefore be used to select a memory device and strobe data out. The CPU polls data on the Data Bus at the rising edge of the T$_3$ clock (②).

Clock perods T$_3$ and T$_4$ of the instruction fetch machine cycle are used by the Z80 CPU for internal operations. These clock periods are also used **to refresh dynamic memory.** As soon as the Program Counter contents are taken off the Address Bus (②), the refresh address from the Refresh register is output on lines A0 - A6 of the Address Bus. This address stays on the Address Bus until the conclusion of T$_4$ (③).

Since a memory refresh is a memory access operation, $\overline{MREQ}$ is again output low; however, it is accompanied by $\overline{RFSH}$ rather than $\overline{RD}$ low. Thus memory reference logic does not attempt to read data during a refresh cycle.

A MEMORY READ OPERATION

Memory interface logic responds to an instruction fetch and a memory read in exactly the same way. There are, however, a few differeces between memory read and instruction fetch timing. Memory read timing is illustrated in Figure 7-6. The principal difference to note is that during a memory read operation, the data is sampled on the trailing edge of the T$_3$ clock pulse, whereas during an instruction fetch it is sampled on the leading edge of this clock pulse. Also a normal memory read machine cycle will consist of three clock periods, while the normal instruction fetch consists of four clock periods. Remember also that the Z80 identifies an instruction fetch machine cycle by outputting $\overline{M1}$ low during the first two clock periods of the instruction fetch machine cycle.

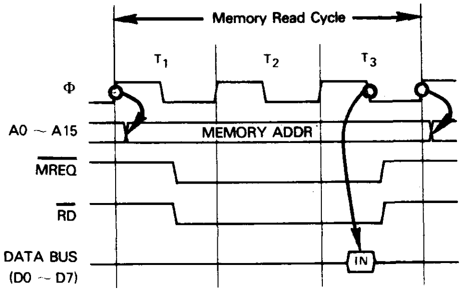


Figure 7-6. Z80 Memory Read Timing

MEMORY WRITE OPERATION

Figure 7-7 illustrates memory write timing for the Z80. The only differences between memory read and memory write timing are the obvious ones: $\overline{\text{WR}}$ is pulsed low for a write, and can be used as a strobe by memory interface logic to read data off the Data Bus.

THE WAIT STATE

Like the 8080A, **the Z80 allows a $\overline{\text{Wait}}$ state to occur between clock periods T_2 and T_3 of a machine cycle.** The Wait state frees external logic or memory from having to operate at CPU speed.

The Z80 CPU samples the $\overline{\text{WAIT}}$ input on the falling edge of Φ during T_2. Providing $\overline{\text{WAIT}}$ is low on the falling edge of Φ during T_2, Wait clock periods will be inserted. The number of Wait clock periods inserted depends strictly on how long the $\overline{\text{WAIT}}$ input is held low. As soon as the Z80 detects WAIT high on the falling edge of Φ, it will initiate T_3 on the next rising edge of Φ.

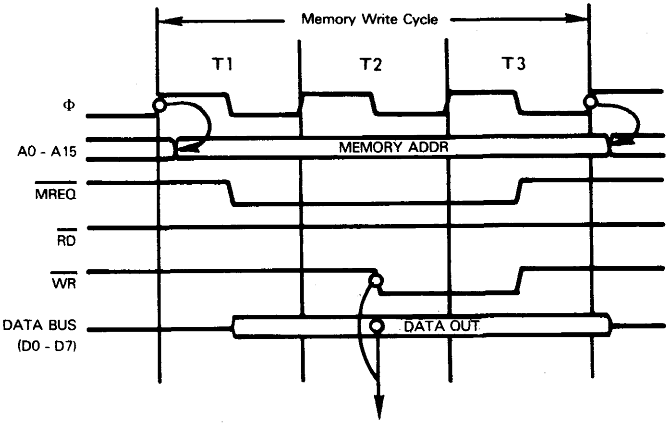


Figure 7-7. Z80 Memory Write Timing

Note that the single Z80 $\overline{\text{WAIT}}$ signal replaces the READY and WAIT 8080A signals. As this would imply, no signal is output telling external logic the Z80 has entered the Wait state. **In the event that external logic needs to know whether or not a Wait state has been entered, these are the rules:**

1) The Z80 will sample $\overline{\text{WAIT}}$ on the falling edge of Φ in T_2.
2) If $\overline{\text{WAIT}}$ is low, then the Z80 will continue to sample the Wait input for all subsequent Wait state clock periods.
3) The Z80 will not sample the $\overline{\text{WAIT}}$ input during any clock period other than T_2 or a Wait state.

Figure 7-8 illustrates Z80 Wait state timing.

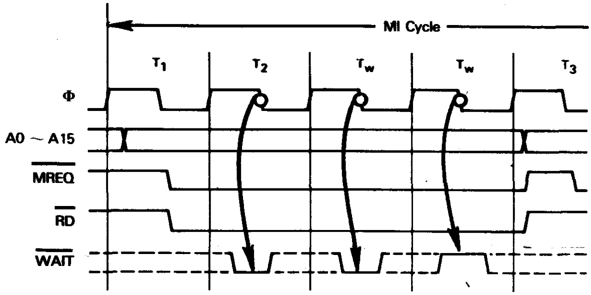


Figure 7-8. Z80 Wait State Timing

INPUT OR OUTPUT GENERATION

Timing for Z80 input and output generation is given in Figures 7-9 and 7-10, respectively.

The important point to note is that Zilog has acknowledged the infrequency with which typical I/O logic can operate at CPU speed. **One Wait clock period is therefore automatically inserted between T_2 and T_3 for all input or output machine cycles.** Otherwise timing differs from memory read and write operations only in that $\overline{\text{IORQ}}$ is output low rather than $\overline{\text{MREQ}}$.

Note that there is absolutely nothing to prevent you from selecting I/O devices within the memory space. This is something we did consistently in Chapter 4 when describing 8080A support devices. But if you adopt this design policy, remember that your I/O logic must execute at CPU speed, unless you insert Wait states.

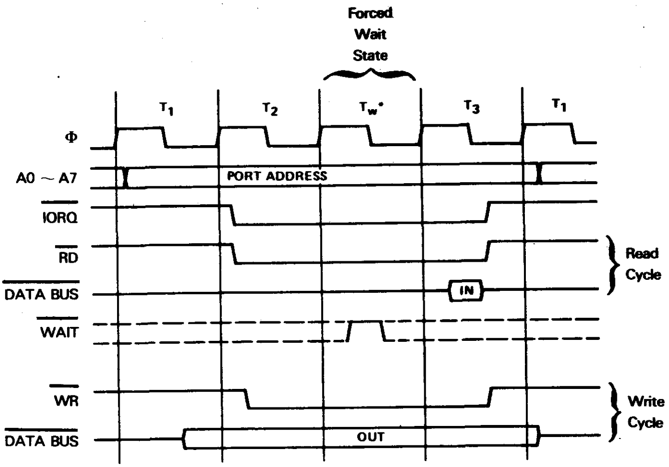


Figure 7-9. Z80 Input Or Output Cycles

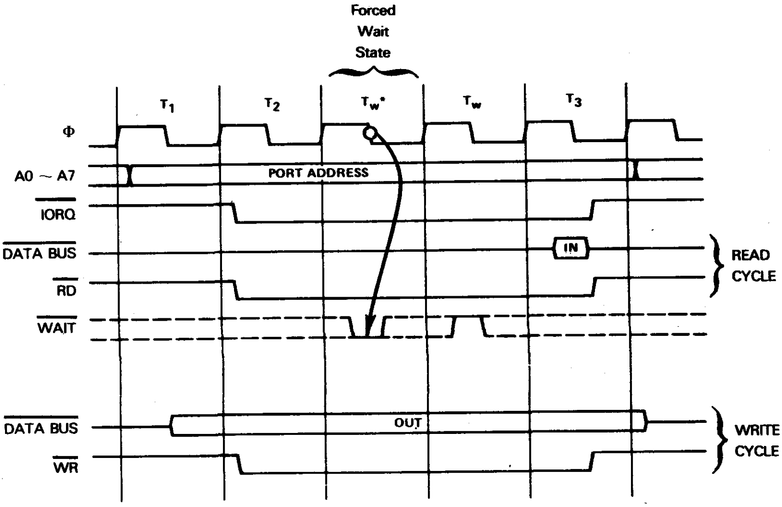


Figure 7-10. Z80 Input Or Output Cycles With Wait States

BUS REQUESTS

The Z80 does not have a Hold state as described for the 8080A, but Z80 bus request logic is equivalent. **The Z80 will float Address, Data and tristate Control Bus lines upon sensing a low BUSRQ signal.** BUSRQ is sampled by the Z80 CPU on the rising edge of the last clock pulse of any machine cycle. If BUSRQ is sampled low, then tristate lines are floated by the CPU, which also outputs BUSAK low. the Z80 CPU continues to sample BUSRQ on the rising edge of every clock pulse. As soon as BUSRQ is sensed high, floating will cease on the next clock pulse. This timing is illustrated in Figure 7-11.

One significant difference between the Z80 and 8080A results from differences between the Hold and bus floating states. As the logic we have described for the Z80 would imply, it will only float the System Bus in between machine cycles. The 8080A, on the other hand, will enter a Hold state variably during T_3 or T_4 of the machine cycle, depending on the type of operation in progress. It is therefore possible for the Z80 to float its bus three clock periods later than an 8080A in a similar configuration.

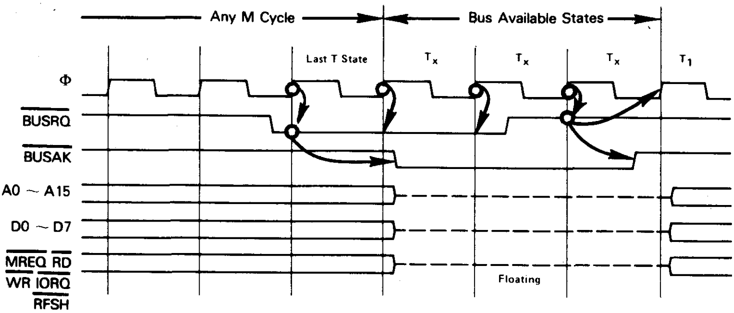

Any M Cycle
Bus Available States
Last T State
T_x
T_x
T_x
T_1
Φ
$\overline{\text{BUSRQ}}$
$\overline{\text{BUSAK}}$
A0 ~ A15
D0 ~ D7
$\overline{\text{MREQ}}$ $\overline{\text{RD}}$
$\overline{\text{WR}}$ $\overline{\text{IORQ}}$
$\overline{\text{RFSH}}$
Floating

Figure 7-11. Z80 Bus Timing

Note also that **if you are using the dynamic memory refresh logic of the Z80, then during long bus floats, external logic must refresh dynamic memory.** The simplest way around this problem in a Z80 system is to ensure that DMA operations acquire the System Bus for many short periods of time, rather than for a single long access.

EXTERNAL INTERRUPTS

The Z80 has two interrupt request input signals, one of which cannot be disabled.

Timing for the lower priority interrupt request acknowledge sequence differs significantly from the single 8080A interrupt request, and is illustrated in Figure 7-12.

The interrupt request signal $\overline{\text{INT}}$ is sampled by the Z80 CPU on the rising edge of the last clock pulse of any instruction's execution.

An interrupt request will be denied if interrupts have been disabled under program control, or if the $\overline{\text{BUSRQ}}$ signal is also low. Thus a DMA access will have priority over maskable interrupts.

The Z80 CPU acknowledges an interrupt request by outputting $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ low. This occurs in a special interrupt acknowledge machine cycle, as illustrated in Figure 7-12. Note that this machine cycle has two Wait states inserted so that external logic will have time for any type of daisy chained priority interrupt scheme to be implemented.

When $\overline{\text{IORQ}}$ is output low while $\overline{\text{M1}}$ is low, external logic must interpret this signal combination as requiring an interrupt vector to be placed on the Data Bus by the acknowledged external interrupt requesting source. This interrupt vector can take one of three forms; the form depends on which of the three modes you have selected for the Z80 under program control.

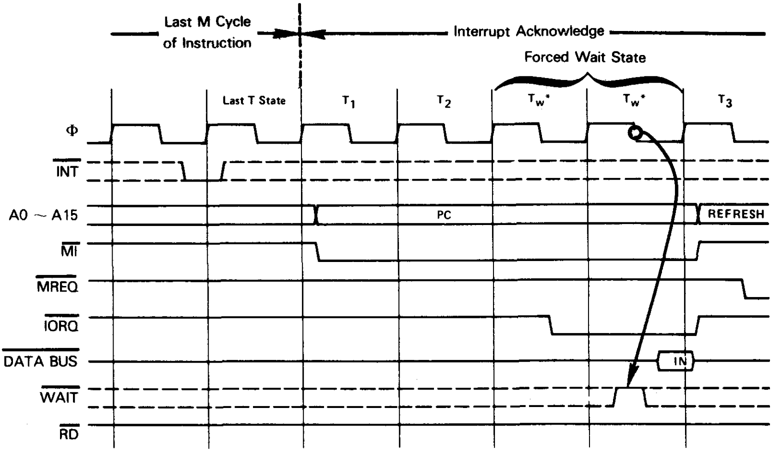
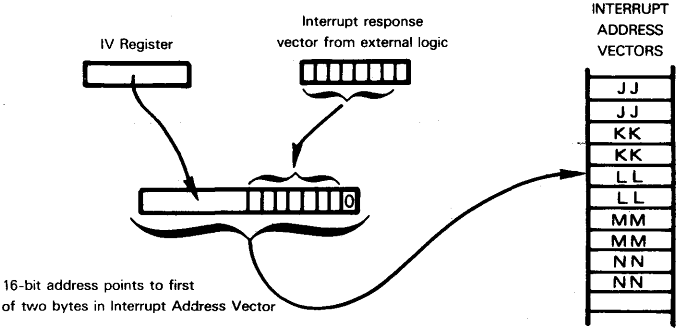


Figure 7-12. Z80 Response To A Maskable Interrupt Request

In **Mode 0,** the interrupt vector will be interpreted as a single-byte object code, representing the first instruction to be executed following the interrupt acknowledge. This **is equivalent to the standard RST instruction response used by the 8080A.** Whenever you are replacing an 8080A with a Z80, therefore, the Z80 must operate in interrupt response Mode 0.

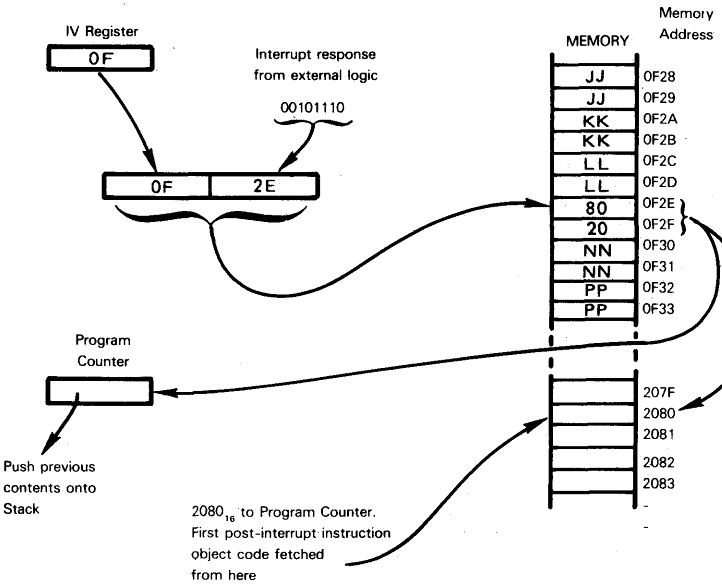
Z80 interrupt response logic in **Mode 1 automatically assumes that the first instruction executed following the interrupt response will be a Restart, branching to memory location 0056_{16}.** If the Z80 is in Mode 1, no interrupt vector is needed.

Z80 Mode 2 interrupt response has no 8080A equivalent. When you operate the Z80 **in Mode 2, you must create a table of 16-bit interrupt address vectors,** which can reside anywhere in addressable memory. These 16-bit addresses identify the first executable instruction of interrupt service routines. When an interrupt is acknowledged by the CPU in Mode 2, **the acknowledged external logic must place an interrupt response vector on the Data Bus. The Z80 CPU will combine the IV register contents with the interrupt acknowledge vector to form a 16-bit address, which accesses the interrupt address vector table.** Since 16-bit addresses must lie at even memory address boundaries, only seven of the eight bits provided by the acknowledged external logic will be used to create the table address; the low order bit will be set to 0. Thus the table of 16-bit interrupt address vectors will be accessed as follows:


IV Register
Interrupt response vector from external logic
INTERRUPT ADDRESS VECTORS
0
JJ
JJ
KK
KK
LL
LL
MM
MM
NN
NN
16-bit address points to first of two bytes in Interrupt Address Vector

The Z80 CPU will execute a Call to the memory location obtained from the interrupt address vector table.

Let us clarify this logic with a simple example. Suppose that you have 64 possible external interrupts; each interrupt has its own interrupt service routine, therefore 64 starting addresses will be stored in 128 bytes of memory. Let us arbitrarily assume that these 128 bytes are stored in a table with memory addresses $0F00_{16}$ through $0F7F_{16}$. Now in order to use Mode 2, you must initially load the value $0F_{16}$ into the Z80 IV register. Subsequently an external interrupt request is acknowledged and the acknowledged external logic returns on the Data Bus the vector $2E_{16}$; this is what will happen:



Z80 WAIT STATES DURING INTERRUPT ACKNOWLEDGE

If two Wait states are insufficient for external logic to arbitrate interrupt priorities and place the required vector on the Data Bus, **then additional Wait states can be inserted** in the usual way by inputting $\overline{\text{WAIT}}$ low. Timing is illustrated in Figure 7-13.

The nonmaskable interrupt differs from the maskable interrupt in two significant ways.

First of all the nonmaskable interrupt has priority over both the maskable interrupt and bus requests.

Next, the nonmaskable interrupt operates in Mode 1 only. Following the interrupt acknowledge, an RST instruction will always be executed, with a Call to memory location 0066_{16}. No other RST instruction can be executed and no interrupt vector should be placed on the Data Bus; if a vector is placed on the Data Bus, it will be ignored.

Nonmaskable interrupt timing is illustrated in Figure 7-14.

THE HALT INSTRUCTION

When a Halt instruction is executed by the Z80 CPU, a sequence of NOP instructions is executed until an interrupt request is received. Both maskable and nonmaskable interrupt request lines are sampled on the rising edge of Φ during T_4 of every NOP instruction's machine cycle.

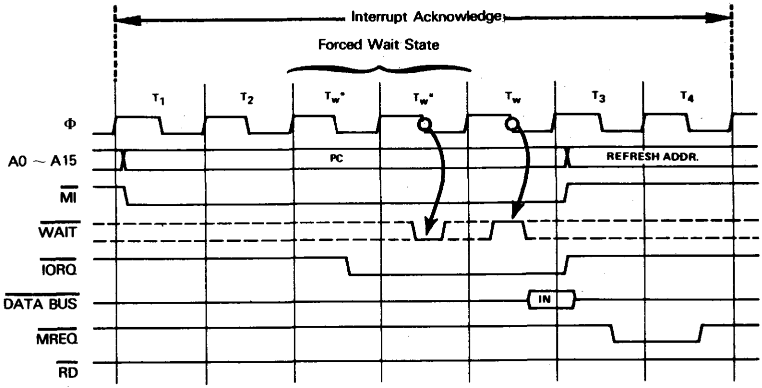


Figure 7-13. Wait States During Z80 Response To A Maskable Interrupt Request

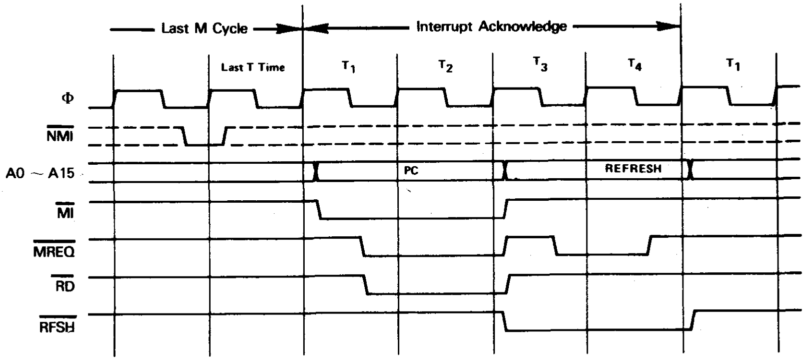
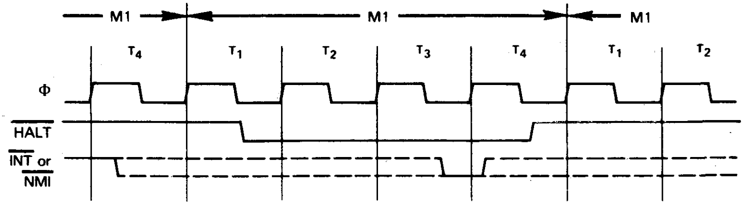


Figure 7-14. Z80 Response To A Nonmaskable Interrupt Request

The Halt state will terminate when any interrupt request is detected, at which time the appropriate interrupt acknowledge sequence will be initiated, as illustrated in Figures 7-13 and 7-14.

Note that the Z80 executes the sequence of NOP instructions during a Halt so that it can continue to generate dynamic memory refresh signals.

Halt instruction timing is illustrated in Figure 7-15.



HALT INSTRUCTION
IS RECEIVED
DURING THIS
MEMORY CYCLE

. Figure 7-15. Z80 Halt Instruction Timing

In Table 7-2, symbols are used as follows:

A,F,B,C,D,E,H,L	The 8-bit registers. A is the Accumulator and F is the Program Status Word.
AF',BE',CD',HL'	The alternative register pairs
ADDR	A 16-bit memory address
x(B)	Bit B of 8-bit register or memory location x
COND	Condition for program branching. Conditions are: NZ — not zero (Z = 0) Z — zero (Z = 1) NC — not carry (C = 0) C — carry (C = 1) PO — odd parity (P = 0) PE — even parity (P = 1) P — sign positive (S = 0) M — sign negative (S = 1)
DATA	An 8-bit binary data unit
DATA16	A 16-bit binary data unit
DISP	An 8-bit signed binary address displacement
xx(HI)	The high order 8 bits of a 16-bit quantity xx
IV	Interrupt vector register (8 bits)
IX, IY	The Index registers (16 bits each)
LABEL	A 16-bit instruction memory address
xx(LO)	The low order 8 bits of a 16-bit quantity xx
PC	Program Counter
PORT	An 8-bit I/O port address
PR	Any of the following register pairs: BC DE HL AF
R	The Refresh register (8 bits)

REG	Any of the following registers: A B C D E H L
RP	Any of the following register pairs: BC DE HL SP
SP	Stack Pointer (16 bits)
STATUSES	C — Carry status Z — Zero status S — Sign status P/O — Parity/Overflow status A_C — Auxiliary Carry status N — Subtract status The following symbols are used in the status columns: X — flag is affected by operation (blank) — flag is not affected by operation 1 — flag is set by operation 0 — flag is reset by operation ? — flag is unknown after operation P — flag shows parity status O — flag shows overflow status I — flag shows interrupt enabled/disabled status
[ ]	Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If an I/O port number is enclosed within the brackets, then the I/O port contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.
[[ ]]	Implied memory addressing; the contents of the memory location designated by the contents of a register.
∧	Logical AND
∨	Logical OR
$\veebar$	Logical Exclusive-OR
:	Compare operands. x:y sets status flags but does not change x or y.
←	Data is transferred in the direction of the arrow
←→	Data is exchanged between the two locations designated on either side of the arrow

*Address Bus: A0 - A7: [C]
A8 - A15: [B]

Table 7-2. A Summary Of The Z80 Instruction Set

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
I/O	IN	A,(PORT)	2							[A] ← [PORT] Input to Accumulator from directly addressed I/O port. Address bus: A0-A7: PORT A8-A15: [A]
	IN	REG,(C)	2		X	X	P	X	0	[REG] ← [[C]] Input to register from I/O port addressed by the contents of C.* If second byte is 70_{16} only the flags will be affected.
	INIR		2		1	?	?	?	1	Repeat until [B] = 0: [[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from low addresses to high. Contents of B serve as a count of bytes remaining to be transferred.*
	INDR		2		1	?	?	?	1	Repeat until [B] = 0: [[HL]] ← [[C]] [B] ←[B] - 1 [HL] ← [HL] - 1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from high addresses to low. Contents of B serve as a count of bytes remaining to be transferred.*
	INI		2		X	?	?	?	1	[[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a byte of data from I/O port addressed by contents of C to memory location addressed by contents of HL. Decrement byte count and increment destination address.*

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
I/O (Continued)	IND		2							[[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a byte of data from I/O port addressed by contents of C to memory location addressed by contents of HL. Decrement both byte count and destination address.*
	OUT	(PORT),A	2							[PORT] ← [A] Output from Accumulator to directly addressed I/O port. Address Bus: A0-A7: PORT A8-A15: [A]
	OUT	(C),REG	2							[[C]] ← [REG] Output from register to I/O port addressed by the contents of C.*
	OTIR		2		1	?	?	?	1	Repeat until [B] = 0: [[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a block of data from memory location addressed by contents of HL to I/O port addressed by contents of C, going from low memory to high. Contents of B serve as a count of bytes remaining to be transferred.*
	OTDR		2		1	?	?	?	1	Repeat until [B] = 0: [[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a block of data from memory location addressed by contents of HL to I/O port addressed by contents of C, going from high memory to low. Contents of B serve as a count of bytes remaining to be transferred.*

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
I/O (Continued)	OUTI		2		X	?	?	?	1	[[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a byte of data from memory location addressed by contents of HL to I/O port addressed by contents of C. Decrement byte count and increment source address.*
	OUTD		2		X	?	?	?	1	[[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a byte of data from memory location addressed by contents of HL to I/O port addressed by contents of C. Decrement both byte count and source address.*
PRIMARY MEMORY REFERENCE	LD	A,(ADDR)	3							[A] ← [ADDR] Load Accumulator from directly addressed memory location.
	LD	HL,(ADDR)	3							[H] ← [ADDR + 1], [L] ← [ADDR] Load HL from directly addressed memory.
	LD	RP,(ADDR) IX,(ADDR) IY,(ADDR)	4							[RP(HI)] ← [ADDR + 1], [RP(LO)] ← [ADDR] or [IX(HI)] ← [ADDR + 1].[IX(LO)] ← [ADDR] or [IY(HI)] ← [ADDR + 1], [IY(LO)] ← [ADDR] Load register pair of Index register from directly addressed memory.
	LD	(ADDR),A	3							[ADDR] ← [A] Store Accumulator contents in directly addressed memory location.
	LD	(ADDR),HL	3							[ADDR + 1] ← [H], [ADDR] ← [L] Store contents of HL to directly addressed memory location.
	LD	(ADDR),RP (ADDR),IX (ADDR),IY	4							[ADDR + 1] ← [RP(HI)], [ADDR] ← [RP(LO)] or [ADDR + 1] ← [IX(HI)], [ADDR] ← [IX(LO)] or [ADDR + 1] ← [IY(HI)], [ADDR] ← [IY(LO)] Store contents of register pair or Index register to directly addressed memory.
	LD	A,(BC) A,(DE)	1							[A] ← [[BC]] or [A] ← [[DE]] Load Accumulator from memory location addressed by the contents of the specified register pair.

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
PRIMARY MEMORY REFERENCE (Continued)	LD	REG,(HL)	1							[REG] ← [[HL]] Load register from memory location addressed by contents of HL.
	LD	(BC),A (DE),A	1							[[BC]] ← [A] or [[DE]] ← [A] Store Accumulator to memory location addressed by the contents of the specified register pair.
	LD	(HL),REG	1							[[HL]] ← [REG] Store register contents to memory location addressed by the contents of HL.
	LD	REG,(IX + DISP) REG,(IY + DISP)	3							[REG] ← [[IX] + DISP] or [REG] ← [[IY] + DISP] Load register from memory location using base relative addressing.
	LD	(IX + DISP),REG (IY + DISP),REG	3							[[IX] + DISP] ← [REG] or [[IY] + DISP] ← [REG] Store register to memory location addressed relative to contents of Index register.
BLOCK TRANSFER AND SEARCH	LDIR		2				0	0	0	Repeat until [BC] = 0: [[DE]] ← [[HL]] [DE] ← [DE] + 1 [HL] ← [HL] + 1 [BC] ← [BC] - 1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from low addresses to high. Contents of BC serve as a count of bytes to be transferred.
	LDDR		2				0	0	0	Repeat until [BC] = 0: [[DE]] ← [[HL]] [DE] ← [DE] - 1 [HL] ← [HL] - 1 [BC] ← [BC] - 1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from high addresses to low. Contents of BC serve as a count of bytes to be transferred.

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
BLOCK TRANSFER AND SEARCH (Continued)	LDI		2				X	0	0	[[DE]] ← [[HL]] [DE] ← [DE] + 1 [HL] ← [HL] + 1 [BC] ← [BC] - 1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Increment source and destination addresses and decrement byte count.
	LDD		2				X	0	0	[[DE]] ← [[HL]] [DE] ← [DE] - 1 [HL] ← [HL] - 1 [BC] ← [BC] - 1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Decrement source and destination addresses and byte count.
	CPIR		2		X	X	X	X	1	Repeat until [A] = [[HL]] or [BC] = 0: [A]:[[HL]] [HL] ← [HL] + 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from low addresses to high. Stop when a match is found or when the byte count becomes zero.
	CPDR		2		X	X	X	X	1	Repeat until [A] = [[HL]] or [BC] = 0: [A]:[[HL]] [HL] ← [HL] - 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from high addresses to low. Stop when a match is found or when the byte count becomes zero.

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
BLOCK TRANSFER AND SEARCH (Continued)	CPI		2		X	X	X	X	1	[A]:[[HL]] [HL] ← [HL] + 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Increment address and decrement byte count.
	CPD		2		X	X	X	X	1	[A]:[[HL]] [HL] ← [HL] - 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Decrement address and byte count.
SECONDARY MEMORY REFERENCE	ADD	(HL)	1	X	X	X	0	X	0	[A] ← [A] + [[HL]] Add to Accumulator using implied addressing.
	ADD	(IX + DISP) (IY + DISP)	3	X	X	X	0	X	0	[A] ← [A] + [[IX + DISP] or [A] ← [A] + [[IY] + DISP] Add to Accumulator using base relative addressing
	ADC	(HL)	1	X	X	X	0	X	0	[A] ← [A] + [[HL]] + C Add with Carry using implied addressing.
	ADC	(IX + DISP) (IY + DISP)	3	X	X	X	0	X	0	[A]←[A] + [[IX] + DISP] + C or [A]←[A] + [[IY] + DISP] + C Add with Carry using base relative addressing
	SUB	(HL)	1	X	X	X	0	X	1	[A]←[A]-[[HL]] Subtract from Accumulator using implied addressing
	SUB	(IX + DISP) (IY + DISP)	3	X	X	X	0	X	1	[A]←[A]-[[IX] + DISP] or [A]←[A]-[[IY] + DISP] Subtract using base relative addressing
	SBC	(HL)	1	X	X	X	0	X	1	[A]←[A]-[[HL]]-C Subtract with Carry using implied addressing

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
SECONDARY MEMORY REFERENCE (Continued)	SBC	(IX + DISP) (IY + DISP)	3	X	X	X	O	X	1	[A]←[A]-[[IX]+DISP]-C or [A]←[A]-[[IY]+DISP]-C Subtract with Carry using base relative addressing
	AND	(HL)	1	0	X	X	P	X	1	[A]←[A]∧[[HL]] AND with Accumulator using implied addressing
	AND	(IX + DISP) (IY + DISP)	3	0	X	X	P	X	1	[A]←[A]∧[[IX]+DISP] or [A]←[A]∧[[IY]+DISP] AND with Accumulator using base relative addressing
	OR	(HL)	1	0	X	X	P	X	0	[A]←[A]V[[HL]] OR with Accumulator using implied addressing
	OR	(IX + DISP) (IY + DISP)	3	0	X	X	P	X	0	[A]←[A]V[[IX]+DISP] or [A]←[A]V[[IY]+DISP] OR with Accumulator using base relative addressing
	XOR	(HL)	1	0	X	X	P	X	0	[A]←[A]⊻[[HL]] Exclusive-OR with Accumulator using implied addressing
	XOR	(IX + DISP) (IY + DISP)	3	0	X	X	P	X	0	[A]←[A]⊻[[IX]+Disp] or [A]←[A]⊻[[IY]+DISP] Exclusive-OR with Accumulator using base relative addressing
	CP	(HL)	1	0	X	X	O	X	1	[A]: [[HL]] Compare with Accumulator using implied addressing
	CP	(IX + DISP) (IY + DISP)	3	0	X	X	O	X	1	[A]: [[IX]+DISP] or [A]: [[IY]+DISP] Compare with Accumulator using base relative addressing
	INC	(HL)	1		X	X	O	X	0	[[HL]]←[[HL]]+1 Increment using implied addressing

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
SECONDARY MEMORY REFERENCE (Continued)	INC	(IX + DISP) (IY + DISP)	3		X	X	O	X	0	[[IX] + DISP]←[[IX] + DISP] + 1 or [[IY] + DISP]←[[IY] + DISP] + 1 Increment using base relative addressing
	DEC	(HL)	1		X	X	O	X	1	[[HL]]←[[HL]]-1 Decrement using implied addressing
	DEC	(IX + DISP) (IY + DISP)	3		X	X	O	X	1	[[IX] + DISP]←[[IX] + DISP]-1 or [[IY] + DISP]←[[IY] + DISP]-1 Decrement using base relative addressing
IMMEDIATE	LD	REG,DATA	2							[REG]←DATA Load immediate into register
	LD	RP,DATA16	3							[RP]←DATA16 Load 16 bits of immediate data into register pair
	LD	IX,DATA16 IY,DATA16	4							[IX]←DATA16 or [IY]←DATA16 Load 16 bits of immediate data into Index register
	LD	(HL),DATA	2							[[HL]]←DATA Load immediate into memory location using implied addressing
	LD	(IX + DISP),DATA (IY + DISP),DATA	4							[[IX] + DISP]←DATA or [[IY] + DISP]←DATA Load immediate into memory location using base relative addressing
JUMP	JP	LABEL	3							[PC]←LABEL Jump to instruction at address LABEL
	JR	DISP	2							[PC]←[PC] + 2 + DISP Jump relative to present contents of Program Counter
	JP	(HL)	1							[PC]←[HL] Jump to address contained in HL
	JP	(IX) (IY)	2							[PC]←[IX] or [PC]←[IY] Jump to address contained in Index register

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
SUBROUTINE CALL AND RETURN	CALL	LABEL	3							[[SP]-1]←[PC(HI)] [[SP]-2]←[PC(LO)] [SP]←[SP]-2 [PC]←LABEL Jump to subroutine starting at LABEL
	CALL	COND,LABEL	3							Jump to subroutine if condition is satisfied; otherwise, continue in sequence
	RET		1							[PC(LO)]←[[SP]] [PC(HI)]←[[SP]+1] [SP]←[SP]+2 Return from subroutine
	RET	COND	1							Return from subroutine if condition is satisfied; otherwise, continue in sequence
IMMEDIATE OPERATE	ADD	DATA	2	X	X	X	O	X	0	[A]←[A]+DATA Add immediate to Accumulator
	ADC	DATA	2	X	X	X	O	X	0	[A]←[A]+DATA+C Add immediate with Carry
	SUB	DATA	2	X	X	X	O	X	1	[A]←[A]-DATA Subtract immediate from Accumulator
	SBC	DATA	2	X	X	X	O	X	1	[A]←[A]-DATA-C Subtract immediate with Carry
	AND	DATA	2	0	X	X	P	X	1	[A]←[A]ΛDATA AND immediate with Accumulator
	OR	DATA	2	0	X	X	P	X	0	[A]←[A]V DATA OR immediate with Accumulator
	XOR	DATA	2	0	X	X	P	X	0	[A]←[A]⊻DATA Exclusive-OR immediate with Accumulator
	CP	DATA	2	X	X	X	O	X	1	[A]:DATA Compare immediate data with Accumulator contents

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
JUMP ON CONDITION	JP	COND,LABEL	3							If COND, then [PC]←LABEL Jump to instruction at address LABEL if the condition is satisfied
	JR	C,DISP	2							If C=1, then [PC]←[PC]+2+DISP Jump relative to contents of Program Counter if Carry flag is set
	JR	NC,DISP	2							If C=0, then [PC]←[PC]+2+DISP Jump relative to contents of Program Counter if Carry flag is reset
	JR	Z,DISP	2							If Z=1, then [PC]←[PC]+2+DISP Jump relative to contents of Program Counter if Zero flag is set
	JR	NZ,DISP	2							If Z=0, then [PC]←[PC]+2+DISP Jump relative to contents of Program Counter if Zero flag is reset
	DJNZ	DISP	2							[B]←[B]-1 If [B]≠0, then [PC]←[PC]+2+DISP Decrement contents of B and Jump relative to contents of Program Counter if result is not 0
REGISTER-REGISTER MOVE	LD	DST,SRC	1							[DST]←[SRC] Move contents of source register to destination register. SRC and DST may each be A, B, C, D, E, H or L.
	LD	A,IV	2		X	X	I	0	0	[A]←[IV] Move contents of interrupt vector to Accumulator
	LD	A,R	2		X	X	I	0	0	[A]←[R] Move contents of Refresh register to Accumulator
	LD	IV,A	2							[IV]←[A] Load interrupt vector from Accumulator
	LD	R,A	2							[R]←[A] Load Refresh register from Accumulator
	LD	SP,HL	1							[SP]←[HL] Move contents of HL to Stack Pointer
	LD	SP,IX SP,IY	2							[SP]←[IX] or [SP]←[IY] Move contents of Index register to Stack Pointer

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
REGISTER-REGISTER MOVE (Continued)	EX	DE,HL	1							[DE]⟷[HL] Exchange DE and HL contents
	EX	AF,AF′	1							[AF]⟷[AF′] Exchange program status and alternate program status
	EXX		1							$\begin{pmatrix} {[BC]} \\ {[DE]} \\ {[HL]} \end{pmatrix} \longleftrightarrow \begin{pmatrix} {[BC']} \\ {[DE']} \\ {[HL']} \end{pmatrix}$ Exchange register pairs and alternate register pairs.
REGISTER-REGISTER OPERATE	ADD	REG	1	X	X	X	O	X	0	[A]←[A]+[REG] Add contents of register to Accumulator
	ADC	REG	1	X	X	X	O	X	0	[A]←[A]+[REG]+C Add contents of register and Carry to Accumulator.
	SUB	REG	1	X	X	X	O	X	1	[A]←[A]-[REG] Subtract contents of register from Accumulator
	SBC	REG	1	X	X	X	O	X	1	[A]←[A]-[REG]-C Subtract contents of register and Carry from Accumulator
	AND	REG	1	0	X	X	P	X	1	[A]←[A]$\wedge$[REG] AND contents of register with contents of Accumulator
	OR	REG	1	0	X	X	P	X	0	[A]←[A]$\vee$[REG] OR contents of register with contents of Accumulator
	XOR	REG	1	0	X	X	P	X	0	[A]←[A]$\veebar$[REG] Exclusive-OR contents of register with contents of Accumulator
	CP	REG	1	X	X	X	O	X	1	[A]:[REG] Compare contents of register with contents of Accumulator
	ADD	HL,RP	1	X				?	0	[HL]←[HL]+[RP] 16-bit add register pair contents to contents of HL

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
REGISTER-REGISTER OPERATE (Continued)	ADC	HL,RP	2	X	X	X	O	?	0	[HL]←[HL]+[RP]+C 16-bit add with Carry register pair contents to contents of HL
	SBC	HL,RP	2	X	X	X	O	?	1	[HL]←[HL]-[RP]-C 16-bit subtract with Carry register pair contents from contents of HL
	ADD	IX,PP	2	X				?	0	[IX]←[IX]+[PP] 16-bit add register pair contents to contents of Index register (PP=BC, DE, IX, SP)
	ADD	IY,RR	2	X				?	0	[IY]←[IY]+[RR] 16-bit add register pair contents to contents of IY Index register (RR=BC, DE, IY, SP)
REGISTER OPERATE	DAA		1	X	X	X	P	X		Decimal adjust Accumulator, assuming that Accumulator contents are the sum or difference of BCD operands.
	CPL		1					1	1	[A]←[$\overline{A}$] Complement Accumulator (ones complement)
	NEG		2	X	X	X	O	X	1	[A]←[$\overline{A}$]+1 Negate Accumulator (twos complement)
	INC	REG	1		X	X	O	X	0	[REG]←[REG]+1 Increment register contents
	INC	RP	1							[RP]←[RP]+1 Increment contents of register pair
	INC	IX IY	2							[IX]←[IX]+1 or [IY]←[IY]+1 Increment contents of Index register
	DEC	REG	1		X	X	O	X	1	[REG]←[REG]-1 Decrement register contents
	DEC	RP	1							[RP]←[RP]-1 Decrement contents of register pair
	DEC	IX IY	2							[IX]←[IX]-1 or [IY]←[IY]-1 Decrement contents of Index register

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

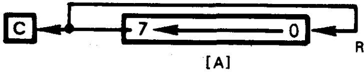
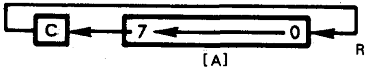
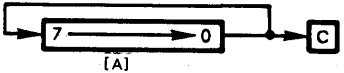
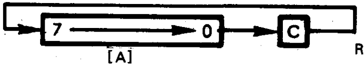
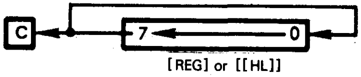
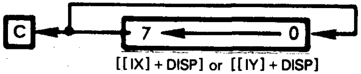
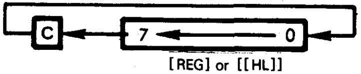
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
REGISTER OPERATE (Continued)	RLCA		1	X				0	0	 Rotate Accumulator left with branch Carry
	RLA		1	X				0	0	 Rotate Accumulator left through Carry
	RRCA		1	X				0	0	 Rotate Accumulator right with branch Carry
	RRA		1	X				0	0	 Rotate Accumulator right through Carry
	RLC	REG (HL)	2	X	X	X	P	0	0	 Rotate contents of register or memory location (implied addressing) left with branch Carry
	RLC	(IX + DISP) (IY + DISP)	4	X	X	X	P	0	0	 Rotate contents of memory location (base relative addressing) left with branch Carry
	RL	REG (HL)	2	X	X	X	P	0	0	 Rotate contents of register or memory location (implied addressing) left through Carry

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

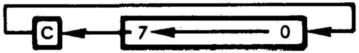
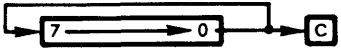
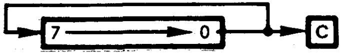
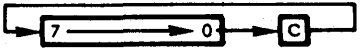
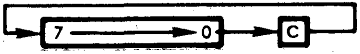
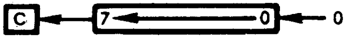
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
REGISTER OPERATE (Continued)	RL	(IX + DISP) (IY + DISP)	4	X	X	X	P	0	0	 [[IX] + DISP] or [[IY] + DISP] Rotate contents of memory location (base relative addressing) left through Carry
	RRC	REG (HL)	2	X	X	X	P	0	0	 [REG] or [[HL]] Rotate contents of register or memory location (implied addressing) right with branch Carry
	RRC	(IX + DISP) (IY + DISP)	4	X	X	X	P	0	0	 [[IX] + DISP] or [[IY] + DISP] Rotate contents of memory location (base relative addressing) right with branch Carry
	RR	REG (HL)	2	X	X	X	P	0	0	 [REG] or [[HL]] Rotate contents of register or memory location (implied addressing) right through Carry
	RR	(IX + DISP) (IY + DISP)	4	X	X	X	P	0	0	 [[IX] + DISP] or [[IY] + DISP] Rotate contents of memory location (base relative addressing) right through Carry.
	SLA	REG (HL)	2	X	X	X	P	0	0	 [REG] or [[HL]] Shift contents of register or memory location (implied addressing) left with branch Carry.

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

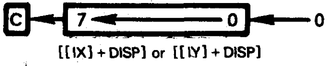
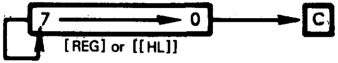
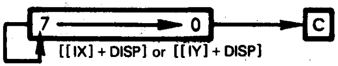
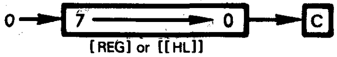
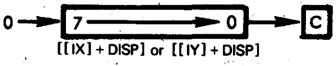
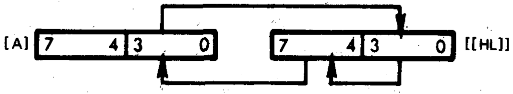
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
REGISTER OPERATE (Continued)	SLA	(IX + DISP) (IY + DISP)	4	X	X	X	P	0	0	 Shift contents of memory location (base relative addressing) left with branch Carry.
	SRA	REG (HL)	2	X	X	X	P	0	0	 Arithmetic shift right contents of register or memory location (implied addressing).
	SRA	[IX + DISP] [IY + DISP]	4	X	X	X	P	0	0	 Arithmetic shift right contents of memory location (base relative addressing).
	SRL	REG (HL)	2	X	X	X	P	0	0	 Shift contents of register or memory location (implied addressing) right with branch Carry.
	SRL	(IX + DISP) (IY + DISP)	4	X	X	X	P	0	0	 Shift contents of memory location (base relative addressing) right with branch Carry.
	RLD		2		X	X	P	0	0	 Rotate one BCD digit left between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

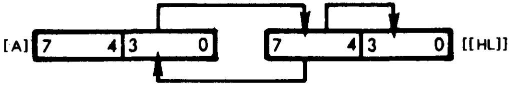
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
REGISTER OPERATE (Continued)	RRD		2		X	X	P	0	0	 Rotate one BCD digit right between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.
BIT MANIPULATION	BIT	B,REG	2		X	?	?	1	0	Z←REG(B) Zero flag contains complement of the selected register bit
	BIT	B,(HL)	2		X	?	?	1	0	Z←[[HL]](B) Zero flag contains complement of selected bit of the memory location (implied addressing).
	BIT	B,(IX + DISP) B,(IY + DISP)	4		X	?	?	1	0	Z←[[IX] + DISP](B) or [[IY] + DISP](B) Zero flag contains complement of the selected bit of the memory location (base relative addressing).
	SET	B,REG	2							REG(B)←1 Set indicated register bit
	SET	B,(HL)	2							[[HL]](B) ← 1 Set indicated bit of memory location (implied addressing).
	SET	B,(IX + DISP) B,(IY + DISP)	4							[[IX] + DISP](B)←1 or [[IY] + DISP](B)←1 Set indicated bit of memory location (base relative addressing).
	RES	B,REG	2							REG(B)←0 Reset indicated register bit
	RES	B,(HL)	2							[[HL]](B)←0 Reset indicated bit in memory location (implied addressing).
	RES	B,(IX + DISP) B,(IY + DISP)	4							[[IX] + DISP](B)←0 or [[IY] + DISP](B)←0 Reset indicated bit of memory location (base relative addressing).

Table 7-2. A Summary Of The Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES	OPERATION PERFORMED
				C Z S P/O A_C N	
STACK	PUSH	PR	1		[[SP]-1]←[PR(HI)] [[SP]-2]←[PR(LO)] [SP]←[SP]-2 Put contents of register pair on top of Stack and decrement Stack Pointer.
	PUSH	IX IY	2		[[SP]-1]←[IX(HI)] or [SP]-1]←[IY(HI)] [[SP]-2]←[IX(LO)] or [[SP]-2]←[IY(LO)] [SP]←[SP]-2 Put contents of Index register on top of Stack and decrement Stack Pointer.
	POP	PR	1		[PR(LO)]←[[SP]] [PR(HI)]←[[SP]+1] [SP]←[SP]+2 Put contents of top of Stack in register pair and increment Stack Pointer.
	POP	IX IY	2		[IX(LO)]←[[SP]] or [IY(LO)]←[[SP]] [IX(HI)]←[[SP]+1] or [IY(HI)]←[[SP]+1] [SP]←[SP]+2 Put contents of top of Stack in Index register and increment Stack Pointer.
	EX	(SP),HL	1		[H]←[[SP]+1] [L]←[[SP]] Exchange contents of HL and top of Stack.
	EX	(SP),IX (SP),IY	2		[IX(HI)]←[[SP]+1] or [IY(HI)]←[[SP]+1] [IX(LO)]←→[[SP]] or [IY(LO)]←→[[SP]] Exchange contents of Index register and top of Stack.

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
INTERRUPT	DI		1							Disable interrupts
	EI		1							Enable interrupts
	RST	N	1							PUSH PC, [PC]←(8 · N)$_{16}$
										Restart at designated location
	RETI		2							Return from interrupt
	RETN		2							Return from nonmaskable interrupt
	IM	0	2							Set interrupt mode 0, 1, or 2
		1								
		2								
STATUS	SCF		1	1				0	0	C←1
										Set Carry flag
	CCF		1	X				?	0	C←$\overline{C}$
										Complement Carry flag
	NOP		1							No operation — volatile memories are refreshed
	HALT		1							CPU halts, executes NOPs to refresh volatile memories.

Table 7-3. A Summary Of Instruction Object Codes And Execution Cycles With 8080A Mnemonics For Identical Instructions

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC		8080A CLOCK PERIODS
ADC	DATA	CE YY	2	7	ACI	DATA	7
ADC	(HL)	8E	1	7	ADC	M	7
ADC	HL,RP	ED 01xx1010	2	15			
ADC	(IX + DISP)	DD 8E YY	3	19			
ADC	(IY + DISP)	FD 8E YY	3	19			
ADC	REG	10001xxx	1	4	ADC	REG	4
ADD	DATA	C6 YY	2	7	ADI	DATA	7
ADD	(HL)	86	1	7	ADD	M	7
ADD	HL,RP	00xx1001	1	11	DAD	RB	10
ADD	(IX + DISP)	DD 86 YY	3	19			
ADD	IX,PP	DD 00xx1001	2	15			
ADD	(IY + DISP)	FD 86 YY	3	19			
ADD	IY,RR	FD 00xx1001	2	15			
ADD	REG	10000xxx	1	4	ADD	REG	4
AND	DATA	E6 YY	2	7	ANI	DATA	7
AND	(HL)	A6	1	7	ANA	M	7
AND	(IX + DISP)	DD A6 YY	3	19			
AND	(IY + DISP)	FD A6 YY	3	19			
AND	REG	10100xxx	1	4	ANA	REG	4
BIT	B,(HL)	CB 01bbb110	2	12			
BIT	B,(IX + DISP)	DD CB YY 01bbb110	4	20			
BIT	B,(IY + DISP)	FD CB YY 01bbb110	4	20			
BIT	B,REG	CB 01bbbxxx	2	8			
CALL	LABEL	CD ppqq	3	17	CALL	LABEL	17
CALL	C,LABEL	DC ppqq	3	10/17	CC	LABEL	11/17
CALL	M,LABEL	FC ppqq	3	10/17	CM	LABEL	11/17
CALL	NC,LABEL	D4 ppqq	3	10/17	CNC	LABEL	11/17
CALL	NZ,LABEL	C4 ppqq	3	10/17	CNZ	LABEL	11/17
CALL	P,LABEL	F4 ppqq	3	10/17	CP	LABEL	11/17
CALL	PE,LABEL	EC ppqq	3	10/17	CPE	LABEL	11/17
CALL	PO,LABEL	E4 ppqq	3	10/17	CPO	LABEL	11/17
CALL	Z,LABEL	CC ppqq	3	10/17	CZ	LABEL	11/17
CCF		3F	1	4	CMC		4
CP	DATA	FE YY	2	7	CPI	DATA	7
CP	(HL)	BE	1	7	CMP	M	7
CP	(IX + DISP)	DD BE YY	3	19			
CP	(IY + DISP)	FD BE YY	3	19	CMP	REG	19
CP	REG	10111xxx	1	4			
CPD		ED A9	2	16			
CPDR		ED B9	2	21/16			*
CPI		ED A1	2	16			
CPIR		ED B1	2	21/16			*
CPL		2F	1	4	CMA		4
DAA		27	1	4	DAA		4
DEC	(HL)	35	1	11	DCR	M	10
DEC	IX	DD 2B	2	10			
DEC	(IX + DISP)	DD 35 YY	3	23			
DEC	IY	FD 2B	2	10			
DEC	(IY + DISP)	FD 35 YY	3	23			
DEC	RP	00xx1011	1	6	DCX	RP	5

Table 7-3. A Summary Of Instruction Object Codes And Execution Cycles With 8080A Mnemonics For Identical Instructions (Continued)

INSTRUCTION	OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC	8080A CLOCK PERIODS
DEC REG	00xxx101	1	4	DCR REG	5
DI	F3	1	4	DI	4
DJNZ DISP	10 YY	2	8/13		
EI	FB	1	4	EI	4
EX AF,AF'	08	1	4		
EX DE,HL	EB	1	4	XCHG	4
EX (SP),HL	E3	1	19	XTHL	18
EX (SP),IX	DD E3	2	23		
EX (SP),IY	FD E3	2	23		
EXX	D9	1	4		
HALT	76	1	4	HLT	4
IM 0	ED 46	2	8		
IM 1	ED 56	2	8		
IM 2	ED 5E	2	8		
IN A,PORT	DB YY	2	10	IN PORT	10
IN REG,(C)	ED 01ddd000	2	11		
INC (HL)	34	1	11	IMR M	10
INC IX	DD 23	2	10		
INC (IX + DISP)	DD 34 YY	3	23		
INC IY	FD 23	2	10		
INC (IY + DISP)	FD 34 YY	3	23		
INC RP	00xx0011	1	6	INX RP	5
INC REG	00xxx100	1	4	INR REG	5
IND	ED AA	2	15		*
INDR	ED BA	2	20/15		
INI	ED A2	2	15		
INIR	ED B2	2	20/15		*
JP LABEL	C3 ppqq	3	10	JMP LABEL	10
JP C,LABEL	DA ppqq	3	10	JC LABEL	10
JP (HL)	E9	1	4	PCHL	5
JP (IX)	DD E9	2	8		
JP (IY)	FD E9	2	8		
JP M,LABEL	FA ppqq	3	10	JM LABEL	10
JP NC,LABEL	D2 ppqq	3	10	JNC LABEL	10
JP NZ,LABEL	C2 ppqq	3	10	JNZ LABEL	10
JP P,LABEL	F2 ppqq	3	10	JP LABEL	10
JP PE,LABEL	EA ppqq	3	10	JPE LABEL	10
JP PO,LABEL	E2 ppqq	3	10	JPO LABEL	10
JP Z,LABEL	CA ppqq	3	10	JZ LABEL	10
JR C,DISP	38 YY	2	7/12		
JR DISP	18 YY	2	12		
JR NC,DISP	30 YY	2	7/12		
JR NZ,DISP	20 YY	2	7/12		
JR Z,DISP	28 YY	2	7/12		
LD A,(ADDR)	3A ppqq	3	13	LDA ADDR	13
LD A,(BC)	0A	1	7	LDAX B	7
LD A,(DE)	1A	1	7	LDAX D	7
LD A,I	ED 57	2	9		
LD A,R	ED 5F	2	9		
LD (ADDR),A	32 ppqq	3	13	STA ADDR	13
LD (ADDR),BC	ED 43 ppqq	4	20		
LD (ADDR),DE	ED 53 ppqq	4	20		

Table 7-3. A Summary Of Instruction Object Codes And Execution Cycles With 8080A Mnemonics For Identical Instructions (Continued)

INSTRUCTION	OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC	8080A CLOCK PERIODS
LD (ADDR),HL	22 ppqq	3	16	SHLD ADDR	16
LD (ADDR),IX	DD 22 ppqq	4	20		
LD (ADDR),IY	LD FD 22 ppqq	4	20		
LD (ADDR),SP	ED 73 ppqq	4	20		
LD (BC),A	02	1	7	STAX B	7
LD (DE),A	12	1	7	STAX D	7
LD HL,(ADDR)	2A ppqq	3	16	LHLD ADDR	16
LD (HL),DATA	36 YY	2	10	MVI M,DATA	10
LD (HL),REG	01110sss	1	7	MOV M,REG	7
LD I,A	ED 47	2	9		
LD IX,(ADDR)	DD 2A ppqq	4	20		
LD IX,DATA16	DD 21 YYYY	4	14		
LD (IX + DISP),DATA	DD 36 YY YY	4	19		
LD (IX + DISP),REG	DD 01110sss YY	3	19		
LD IY,(ADDR)	FD 2A ppqq	4	20		
LD IY,DATA16	FD 21 YYYY	4	14		
LD (IY + DISP),DATA	FD 36 YYYY	4	19		
LD (IY + DISP),REG	FD 01110sss YY	3	19		
LD R,A	ED 4F	2	9		
LD REG,DATA	00ddd110 YY	2	7	MVI REG DATA	7
LD REG,(HL)	01ddd110	1	7	MOV REG,M	7
LD REG,(IX + DISP)	DD 01ddd110 YY	3	19		
LD REG,(IY + DISP)	FD 01ddd110 YY	3	19		
LD REG,REG	01dddsss	1	4	MOV REG,REG	5
LD RP,(ADDR)	ED 01xx1011 ppqq	4	20		
LD RP,DATA16	00xx0001 YYYY	3	10	LXI RP,DATA16	10
LD SP,HL	F9	1	6	PSHL	5
LD SP,IX	DD F9	2	10		
LD SP,IY	FD F9	2	10		
LDD	ED A8	2	16		
LDDR	ED B8	2	21/16		*
LDI	ED A0	2	16		
LDIR	ED B0	2	21/16		*
NEG	ED 44	2	8		
NOP	00	1	4	NOP	4
OR DATA	F6 YY	2	7	ORI DATA	7
OR (HL)	B6	1	7	ORA M	7
OR (IX + DISP)	DD B6 YY	3	19		
OR (IY + DISP)	FD B6 YY	3	19		
OR REG	10110xxx	1	4	ORA REG	5
OTDR	ED BB	2	20/15		*
OTIR	ED B3	2	20/15		*
OUT (C),REG,	01sss001				

Table·7-3. A Summary Of Instruction Object Codes And Execution Cycles With 8080A Mnemonics For Identical Instructions (Continued)

INSTRUCTION	OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC	8080A CLOCK PERIODS
OUT PORT,A	D3 YY	2	11	OUT PORT	10
OUTD	ED AB	2	15		
OUTI	ED A3	2	15		
POP IX	DD E1	2	14		
POP IY	FD E1	2	14		
POP PR	11xx0001	1	10	POP RP	10
PUSH IX	DD E5	2	15		
PUSH IY	FD E5	2	15		
PUSH PR	11xx0101	1	11	PUSH RP	11
RES B,(HL)	CB 10bbb110	2	15		
RES B,(IX + DISP)	DD CB YY 10bbb110	4	23		
RES B,(IY + DISP)	FD CB YY 10bbb110	4	23		
RES B,REG	CB 10bbbxxx	2	8		
RET	C9	1	10	RET	10
RET C	D8	1	5/11	RC	5/11
RET M	F8	1	5/11	RM	5/11
RET NC	D0	1	5/11	RNC	5/11
RET NZ	C0	1	5/11	RNZ	5/11
RET P	F0	1	5/11	RP	5/11
RET PE	E8	1	5/11	RPE	5/11
RET PO	E0	1	5/11	RPO	5/11
RET Z	C8	1	5/11	RZ	5/11
RETI	ED 4D	2	14		
RETN	ED 45	2	14		
RL (HL)	CB 16	2	15		
RL (IX + DISP)	DD CB YY 16	4	23		
RL (IY + DISP)	FD CB YY 16	4	23		
RL REG	CB 00010xxx	2	8		
RLA	17	1	4	RAL	4
RLC (HL)	CB 06	2	15		
RLC (IX + DISP)	DD CB YY 06	4	23		
RLC (IY + DISP)	FD CB YY 06	4	23		
RLC REG	CB 00000xxx	2	8		
RLCA	07	1	4	RLC	4
RLD	ED 6F	2	18		
RR (HL)	CB 1E	2	15		
RR (IX + DISP)	DD CB YY 1E	4	23		
RR (IY + DISP)	FD CB YY 1E	4	23		
RR REG	CB 00011xxx	2	8		
RRA	1F	1	4	RAR	4
RRC (HL)	CB 0E	2	15		
RRC (IX + DISP)	DD CB YY 0E	4	23		
RRC (IY + DISP)	FD CB YY 0E	4	23		

Table 7-3. A Summary Of Instruction Object Codes And Execution Cycles With 8080A Mnemonics For Identical Instructions (Continued)

INSTRUCTION	OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC	8080A CLOCK PERIODS
RRC REG	CB 00001xxx	2	8		
RRCA	0F	1	4	RRC	4
RRD	ED 67	2	18		
RST N	11xxx111	1	11	RST N	11
SBC DATA	DE YY	2	7	SBI DATA	7
SBC (HL)	9E	1	7	SBB M	7
SBC HL,RP	ED 01xx0010	2	15		
SBC (IX + DISP)	DD 9E YY	3	19		
SBC (IY + DISP)	FD 9E YY	3	19		
SBC REG	10011xxx	1	4	SBB REG	4
SCF	37	1	4	STC	4
SET B,(HL)	CB 11bbb110	2	15		
SET B,(IX + DISP)	DD CB YY 11bbb110	4	23		
SET B,(IY + DISP)	FD CB YY 11bbb110	4	23		
SET B,REG	CB 11bbbxxx	2	8		
SLA (HL)	CB 26	2	15		
SLA (IX + DISP)	DD CB YY 26	4	23		
SLA (IY + DISP)	FD CB YY 26	4	23		
SLA REG	CB 00100xxx	2	8		
SRA (HL)	CB 2E	2	15		
SRA (IX + DISP)	DD CB YY 2E	4	23		
SRA (IY + DISP)	FD CB YY 2E	4	23		
SRA REG	CB 00101xxx	2	8		
SRL (HL)	CB 3E	2	15		
SRL (IX + DISP)	DD CB YY 3E	4	23		
SRL (IY + DISP)	FD CB YY 3E	4	23		
SRL REG	CB 00111xxx	2	8		
SUB DATA	D6 YY	2	7	SUI DATA	7
SUB (HL)	96	1	7	SUB M	7
SUB (IX + DISP)	DD 96 YY	3	19		
SUB (IY + DISP)	FD 96 YY	3	19		
SUB REG	10010xxx	1	4	SUB REG	4
XOR DATA	EE YY	2	7	XRI DATA	7
XOR (HL)	AE	1	7	XRA M	7
XOR (IX + DISP)	DD AE YY	3	19		
XOR (IY + DISP)	FD AE YY	3	19		
XOR REG	10101xxx	1	4	XRA REG	4

x	represents an optional binary digit.
bbb	represents optional binary digits identifying a bit location in a register or memory byte.
ddd	represents optional binary digits identifying a destination register.
sss	represent optional binary digits identifying a source register.
ppqq	represents a four hexadecimal digit memory address.
YY	represents two hexadecimal data digits.
YYYY	represents four hexadecimal data digits.

When two possible execution times are shown (i.e., 5/11), it indicates that the number of clock periods depends on condition flags.

*Execution time shown is for one iteration.

THE Z80 INSTRUCTION SET

We are going to describe the Z80 instruction set as an 8080A enhancement. Table 7-2 summarizes the Z80 instruction set in the standard format used for all microcomputers in this book; unfortunately, the fact that the 8080A instruction set is a subset of Table 7-2 is not immediately obvious, since a number of significant conceptual differences exist between the Zilog and 8080A assembly language mnemonics. Table 7-3 therefore shows Z80 equivalents for every 8080A instruction. The few incompatibilities which exist are identified.

Also because of Z80 mnemonics, the Zilog instruction set is not easily forced into the standard instruction categories that we have selected for consistency. In particular, Z80 mnemonics group Memory Reference, Register-Register Move and Immediate instruction into a single "Load and Exchange" category. The same holds true for Z80 Arithmetic and Logical instructions; in Table 7-2 these become Secondary Memory Reference, Register-Register Operate and Immediate Operate instructions.

INPUT/OUTPUT INSTRUCTIONS

These are the types of input/output instructions provided by the Z80:

1) **The standard 8080A IN and OUT instructions,** whereby the second byte of instruction object code provides an I/O port address, which appears on Address Bus lines A0 - A7.

2) **Register indirect Input and Output instructions.** These instructions transfer data between Register A, B, C, D, E, H or L, and the I/O port identified by the contents of Register C. Thus the instruction:

```
        LD      C,PORTN     ;LOAD PORT NUMBER INTO REGISTER C
        -
        -
        -
        IN      D,(C)       ;INPUT DATA FROM PORTN TO REGISTER D
```

   is equivalent to:

```
        IN      A,(PORTN)
        LD      D,A
```

   The I/O port address, now the contents of Register C, is output on A0 - A7 in the usual way.

3) **Block Transfer I/O instructions.** These instructions move a block of data between the I/O port identified by Register C and a memory location addressed by the H and L register pair. Register B is used as a block byte counter. After each byte of data within the block is transferred, the contents of Register B are decremented; you can specify block transfer I/O instructions that will either increment or decrement the memory address in Registers H and L. Here is a programming example with the 8080A equivalent:

```
                Z80                        8080A
        LD      B,COUNT             MVI     B,COUNT
        LD      C,PORTN             LXI     H,START
        LD      HL,START    LOOP:   IN      PORTN
        OTIR                        MOV     M,A
                                    INX     H
                                    DCR     B
                                    JNZ     LOOP
```

These instruction sequences input COUNT bytes from I/O port PORTN, and store the data in a memory buffer whose beginning address is START. COUNT and PORTN are symbols representing 8-bit numbers. START is an address label. The block transfer I/O instruction will continue executing until the B register has decremented to 0.

4) **Single Step Block Transfer I/O instructions.** These are identical to the block transfer I/O instructions described in category 3 above, except that instruction execution ceases after one iterative step. Referring to the OTIR instruction example, if the OTIR instruction were replaced by an OUTI instruction, a single byte of data would be transferred from PORTN to the memory location addressed by START. The address START would be incremented, Register B contents would be decremented, then instruction execution would cease.

When a block transfer or single step, block transfer I/O instruction is executed, C register contents, which identify the I/O port, are output on the lower eight Address Bus lines in the usual way; however, B register contents are output on the higher eight address lines A15 - A8. Therefore external logic can, if it wishes, determine the extent of the transfer.

Let us now look at the advantages gained by having the new Z80 I/O instructions.

The value of the Register Indirect I/O instructions is that programs stored in ROM can access any I/O port. If I/O port assignments change, then all you need to do is modify that small portion of program which loads the I/O port address into the C register.

The Block Transfer I/O instructions must be approached with an element of caution. In response to the execution of a single instruction's object code, up to 256 bytes of data may be transferred between memory and an I/O port. This data transfer occurs at CPU speed — which means external logic must input or output data at the same speed. If external logic cannot operate fast enough, it can insert Wait states in order to slow the CPU, but that takes additional logic; and one might argue that the traditional methods of polling on status to effect block I/O transfers is cheaper than adding extra Wait state logic.

Note that all Z80 enhanced I/O instructions require two bytes of object code.

PRIMARY MEMORY REFERENCE INSTRUCTIONS

Instructions that we classify as Primary Memory Reference constitute a subset of the Load instructions, as classifed by Zilog. **Within the Primary Memory Reference instructions category, as we define it, Zilog offers a single enhancement: base relative addressing.** Instructions that move data between a register and memory may specify the memory address as the contents of an Index register; plus an 8-bit displacement provided by the instruction object code. Here is a programming example of Zilog base relative addressing and the 8080A equivalent:

```
               Z80                      8080A
        LD     IX,BASE            LSI       H,BASE
        LD     C,(IX + DISP)      LXI       D,DISP
                                  DAD       D
                                  MOV       C,M
```

Observe that the two Z80 instructions do not use any CPU registers — other than the IX Index register. The 8080A uses the DE and HL registers. Here is an example of the true value that results from having Index registers. The Z80 can use the DE and HL registers to store temporary data, which the 8080A cannot do; the 8080A would have to store such temporary data in external read/write memory.

The biggest single advantage that accrues to the Z80 from having indexed addressing is the fact that well written Z80 programs will contain far fewer memory reference instructions than equivalent 8080A programs; therefore Z80 programs will execute faster.

Other primary memory reference instructions provided by the Z80, and not present in the 8080A, include instructions which load data into the Index registers and store Index registers' contents in memory. Since the 8080A does not have Index registers, it cannot have memory reference instructions for them. The Z80 also has instructions which transfer 16-bit data between directly addressed memory and any register pair, except AF. Recall that in the 8080A, HL is the only register pair which stores to memory and loads from memory using direct addressing.

BLOCK TRANSFER AND SEARCH INSTRUCTIONS

We classify the Zilog Block Transfer and Search instructions in a separate category, since our hypothetical computer, as described in Volume I, had no equivalent instructions.

A Block Transfer instruction allows you to move up to 65,536 bytes of data between two memory buffers which may be anywhere in memory. The H and L registers address the source buffer, the D and E registers address the destination buffer, and the B and C registers hold the byte count.

After every byte of data is transferred, the B and C registers' contents are decremented; instruction execution ceases after the B and C registers decrement to zero. You have the option of incrementing or decrementing the source and destination addresses following the transfer of each data byte. Thus you can transfer data from low to high memory, or from high to low memory. Here is a programming example of the Z80 Block Move instruction, along with the 8080A equivalent:

```
                Z80                        8080A
        LD        BC,COUNT                 LXI       B,COUNT
        LD        DE,DEST                  LXI       D,DEST
        LD        HL,SRCE                  LXI       H,SRCE
        LDIR                      LOOP:    MOV       A,M
                                           INX       H
                                           XCHG
                                           MOV       M,A
                                           INX       H
                                           XCHG
                                           DCX       B
                                           MOV       A,B
                                           ORA       C
                                           JNZ       LOOP
```

The two instruction sequences illustrated above move a block of data, COUNT bytes long, from a buffer whose starting address is SRCE to another buffer whose starting address is DEST. SRCE and DEST are 16-bit address labels. COUNT is a symbol representing a 16-bit data value.

The Z80 - 8080A comparison above is one that makes the 8080A look particularly bad. This is because it emphasizes 8080A weaknesses; the 8080A requires memory addresses to be incremented as separate steps. Also, after decrementing the counter in Registers B and C, status is not set, therefore BC contents are tested by loading B into A and ORing with C.

You can use Block Move instructions in Z80 configurations that include dynamic memory. While the Block Move is being executed, dynamic memory is refreshed.

The Block Search instruction will search a block of data in memory, looking for a match with the Accumulator contents. The H and L registers address memory, while the B and C registers again act as a byte counter. When a match between Accumulator contents and a memory location is found, the Search instruction ceases executing. After every Compare, the B and C registers' contents are decremented; once again you have the option of either incrementing or decrementing H and L registers' contents. Thus you can search a block of memory from high address down, or from low address up.

The results of every step in a Block Search are reported in the Z and P/O statuses. If a match is found between Accumulator and memory contents, then Z is set to 1; otherwise Z will equal 0. When the B and C registers count out to zero, the P/O status will be reset to 0; otherwise the P/O status will equal 1.

Here is an example of a program using the Z80 Block Search instruction, along with 8080A program equivalent:

```
                    Z80                        8080A
          LD      A,REFC                       LXI     BC,COUNT
          LD      BC,COUNT                     LXI     HL,SRCE
          LD      HL,SRCE           LOOP:      MVI     A,REFC
          CPDR                                 CMP     M
          JR      Z,FOUND                      JEQ     FOUND
;NO MATCH FOUND                                DCX     H
          -                                    DCX     B
          -                                    MOV     A,B
;MATCH FOUND                                   ORA     C
FOUND:                                         JNZ     LOOP
          -                         ;NO MATCH FOUND
          -                                    -
          -                                    -
                                               -
                                    ;MATCH FOUND
                                    FOUND:     -
                                               -
                                               -
```

Each of the above instruction sequences tries to match a character represented by the symbol REFC with the contents of bytes in a memory buffer. The memory buffer is origined at SRCE and is COUNT bytes long.

In the example illustrated above, SRCE is the highest memory address for the buffer, which is searched towards the low memory address. FOUND is the label for the first instruction in the sequence which is executed if a match is found. If no match is found, that is, the BC registers count out to 0, program execution continues with the next sequential instruction.

The Z80 Block Search instruction is particularly useful when searching a large memory buffer for a byte that may frequently occur. Suppose you have an ASCII text in which Control codes have been imbedded. For the sake of argument, let us assume that all Control codes are two bytes long, where the first byte has the hexadecimal value 02 and the second byte identifies the Control code. You can use one set of registers in order to search the text buffer for Control codes, while using the second set of registers to process the text buffer after each Control code has been located.

All you need to do in the Block Search instruction sequence illustrated above is follow the CPDR instruction with an EXX instruction; after executing the instruction sequence following MATCH FOUND, again execute an EXX instruction before returning to search for the next Control code.

Each of the Block Move and Block Search instructions has a single step equivalent. The single step instruction moves one byte of data, or compares the Accumulator contents with the next byte in a data buffer; addresses and counters are incremented and decremented as for the Block Move and Search instructions, however execution ceases after a single step has been completed.

SECONDARY MEMORY REFERENCE (MEMORY OPERATE) INSTRUCTIONS

Instructions that we classify as Secondary Memory Reference, or Memory Operate, constitute a portion of the arithmetic and logical instructions, as defined by the Z80. **Within the Memory Operate group of instructions, the single enhancement offered by the Z80 is a duplicate set of instructions that uses base relative addressing.** We have already discussed this enhancement in connection with Primary Memory Reference instructions. Here is a programming example with the 8080A equivalent:

```
              Z80                   8080A
LD      IX,BASE        LXI      H,BASE
ADD     (IX + DISP)    LXI      D,DISP
                       DAD      D
                       ADD      M
```

The same comments we made regarding the use of indexed addressing in the Primary Memory Reference example apply to the instruction sequences above.

IMMEDIATE INSTRUCTIONS

Within the group of instructions that we classify as Immediate, the Z80 offers two enhancements:

1) Instructions are provided to load immediate data into the additional Z80 registers.

2) You can use base relative addressing to load a byte of data immediately into read/write memory.

JUMP INSTRUCTIONS

In addition to the standard Jump instruction offered by the 8080A, the Z80 has a two-byte, unconditional Branch instruction, and two instructions which allow you to jump to the memory location specified by an Index register.

The two indexed Jump instructions transfer the contents of the identified Index register to the Program Counter.

The two-byte Jump instruction interprets the second object code byte as an 8-bit signed binary number, which is added to the Program Counter, after the Program Counter has been incremented to point to the next instruction. This is a standard program relative branch, as described in Volume I.

Note that the Z80 uses many of the spare 8080A object codes to implement the two-byte Branch and Branch-on-Condition instructions. This makes sense; it would certainly not make much sense to have two bytes of object code followed by a single branch byte, since that would create a three-byte Branch instruction — offering no advantage over the three-byte Jump instructions which already exist.

SUBROUTINE CALL AND RETURN INSTRUCTIONS

The Z80 instructions in this group are identical to 8080A equivalents.

IMMEDIATE OPERATE INSTRUCTIONS

Z80 Immediate Operate instructions, as we define them, are identical to those in the 8080A instruction set.

JUMP-ON-CONDITION INSTRUCTION

The Z80 offers two significant Jump-on-Condition instruction enhancements over the 8080A:

1) **There are two-byte equivalents for four of the more commonly used Jump-on-Condition instructions.** The two-byte Jump-on-Condition instructions execute exactly as described for the two-byte Jump instruction.

2) **There is a decrement and Jump-on-Nonzero instruction** which is particularly useful in any kind of iterative loop. When this instruction is executed, the B register contents are decremented; if the B register contents, after being decremented, equal zero, the next sequential instruction is executed. If after being decremented the B register contents are not zero, then a Jump occurs. This is a two-byte instruction, where the Jump is specified by a single 8-bit signed binary value.

Here is an example of how the DJNZ instruction may be used along with the 8080A equivalent:

```
                Z80                           8080A
        AND     A                       ANA     A
        LD      IX,VALA                 LXI     H,VALA
        LD      IY,VALB                 LXI     D,VALB
        LD      B,CNT                   MVI     B,CNT
LOOP:   LD      A,(IX)          LOOP:   MOV     A,M
        ADC     A,(IY)                  XCHG
        LD      (IX),A                  ADC     M
        INC     IX                      INX     H
        INC     IY                      XCHG
        DJNZ    LOOP                    MOV     M,A
                                        INX     H
                                        DCR     B
                                        JNZ     LOOP
```

The two instruction sequences illustrated above perform simple multibyte binary addition. The contents of two buffers, origined at VALA and VALB, are summed; the results are stored in buffer VALA.

The first instruction in each sequence is executed in order to clear the Carry status. Like the 8080A, the Z80 does not have an instruction which sets the Carry status to 0, while performing no other operation.

REGISTER-REGISTER MOVE INSTRUCTIONS

Register-Register Move instructions, as we defined them in this book, constitute a subset of the Z80 Load instructions. All Z80 Exchange instructions, except those that exchange with the top of the Stack, are also classified as Register-Register Move instructions.

The Z80 enhancements within this instruction group apply strictly to the additional registers implemented within the Z80. That is to say, because the Z80 has registers which the 8080A does not have, the Z80 must also have instructions to move data in and out of these additional registers.

The instructions which exchange data between registers and their alternates need comment. Note that you can swap the entire set of duplicated registers, or you can swap selected register pairs. If you use these instructions following an interrupt acknowledge, you do not have to save the contents of the registers on the Stack. Of course, this will only work for a single interrupt level. There are also occasions when the alternate set of registers can be used effectively in normal programming logic, as we illustrated when describing the Block Search instruction.

REGISTER-REGISTER OPERATE INSTRUCTIONS

There are a few new Z80 Register-Register Operate instructions which do the following:

1) Add without Carry the contents of a register pair to an Index register.

2) Add with Carry to HL the contents of a register pair.

3) Subtract with Carry from HL the contents of a register pair.

REGISTER OPERATE INSTRUCTIONS

Within this category, the Z80 has three enhancements:

1) You can increment or decrement the contents of an Index register.

2) A rich variety of Shift and Rotate instructions have been added. These instructions are illustrated in Table 7-2. In particular, note the RLD and RRD instructions, which are very useful when performing multidigit BCD left and right shifts.

BIT MANIPULATION INSTRUCTIONS

The 8080A has no equivalent for this set of Z80 instructions. We give these instructions a separate category in Table 7-2 because of their extreme importance in microprocessor applications.

Bit manipulation instructions are particularly important for signal processing. A single signal is a binary entity; it is not part of an 8-bit unit. One of the great oversights among microprocessor designers has been to ignore bit manipulation instructions. **The Z80 has instructions that set to 1 (SET), reset to 0 (RES) or test (BIT) individual bits in memory or any general purpose register.** The result of a bit test is reported in the Zero status.

Here are some Z80 instructions with 8080A equivalents:

```
          Z80                      8080A
BIT       4,A               MOV    B,A
                            ANI    10H
```

The 8080A tests Accumulator bits destructively — all untested bits are cleared; Accumulator contents must therefore be saved before testing. We can also contrive an example to emphasize the strengths of the Z80 bit instructions:

```
          Z80                      8080A
LD        IY,BASE           LXI      H,BASE
SET       2,(IY + DISP)     LXI      D,DISP
                            DAD      D
                            MVI      A,4
                            ORA      M
```

Once again, note that the 8080A needs to use the D, E, H and L registers.

Note that all Z80 Bit instructions operate on memory or CPU registers. But in most microcomputer applications individual pins at I/O ports will most frequently be set, reset or tested. The Z80 has no I/O Bit instructions. If you wish, you can interface I/O devices so that they are addressed as memory locations; however, in that case, you cannot use Block I/O instructions.

The 8080A can do anything that a Z80 Bit Manipulation instruction can do but an additional Mask instruction is needed and the Accumulator, is involved. On the surface these seem to be small penalties; but it is the frequency with which Bit Manipulation instructions are needed that escalates small penalties into major aggravations.

STACK INSTRUCTIONS

Additional Stack instructions provided by the Z80 allow the Z80 Index registers to be pushed onto the Stack, popped from the Stack, or exchanged with the top of the Stack.

INTERRUPT INSTRUCTIONS

In addition to the 8080A Interrupt instructions, the Z80 has two Return-from-Interrupt instructions. **RETI and RETN are used to return from maskable and nonmaskable interrupt service routines, respectively.**

RETI and RETN are two-byte instructions. **Within the CPU these instructions enable interrupts, but otherwise execute exactly as a Return-from-Subroutine (RET) instruction. However, devices designed by Zilog to support the Z80 CPU use the RETI and RETN instructions in a unique way.** Any support device that has logic to request an interrupt also includes logic which tests the Data Bus contents during the low M1 pulse. Upon detecting the second byte of an RETI or RETN instruction's object code, a device which has had an interrupt request acknowledged determines that the interrupt has been serviced.

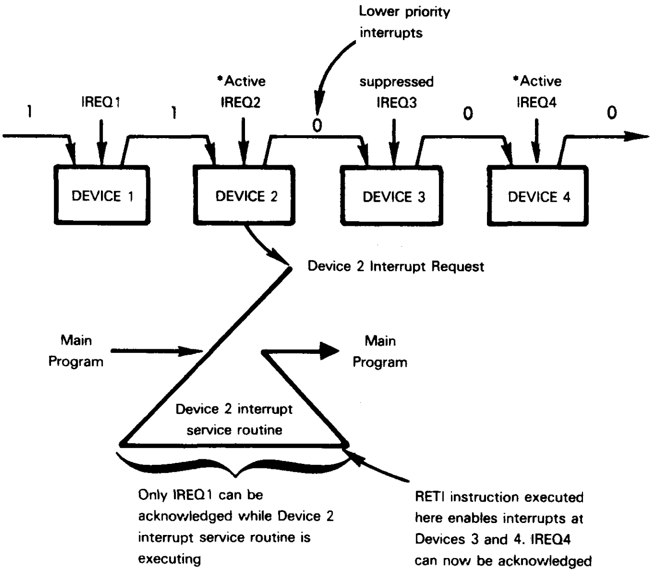
Why does a support device need to know that an interrupt service routine has completed execution? The reason is that Zilog extends interrupt priority arbitration logic beyond the interrupt acknowledge process to the entire interrupt service routine.

This is the scheme adopted by the 8259 PICU. After reading the next paragraph, if you are still unclear on concepts, refer back to the 8259 PICU discussion in Chapter 4.

Consider the typical daisy chain scheme used to set interrupt priorities in a multiple interrupt microcomputer system. Daisy chaining has been described in good detail in Volume I. When more than one device is requesting an interrupt, an acknowledge ripples down the daisy chain until trapped by the interrupt requesting device electrically closest to the CPU. As soon as the interrupt acknowledge process has ceased, an interrupt service routine is executed for the acknowledged interrupt; acknowledged external logic will now remove its interrupt request. Unless the CPU disables further interrupts, a lower priority device can immediately interrupt the service routine of a higher priority device. With the Zilog system, that is not the case. A device which has its interrupt request acknowledged continues to suppress interrupt requests from all lower priority devices in a daisy chain, until the second object code byte for an RETI or RETN instruction is detected on the Data Bus. The acknowledged device responds to an RETI or RETN instruction's object code by re-enabling interrupts for devices with lower priority in the daisy chain.

Providing a Zilog microcomputer system has been designed to make correct use of the RETI and RETN instructions, interrupt priority arbitration logic will allow an interrupt service routine to be interrupted only by a high priority interrupt request.

Here is an illustration of the Zilog interrupt priority arbitration scheme:



Only IREQ1 can be acknowledged while Device 2 interrupt service routine is executing

RETI instruction executed here enables interrupts at Devices 3 and 4. IREQ4 can now be acknowledged

The three IM instructions allow you to specify that the CPU will respond to maskable interrupts in Mode 0, 1 or 2. These three interrupt response modes have already been described.

STATUS AND MISCELLANEOUS INSTRUCTIONS

Z80 and 8080A instructions in these categories are identical.

THE BENCHMARK PROGRAM

Our benchmark program is coded for the Z80 as follows:

```
        LD      BC,LENGTH       ;LOAD PROGRAM LENGTH INTO BC
        LD      IX,TABLE        ;LOAD TABLE BASE ADDRESS
        LD      E,(IX)          ;LOAD ADDRESS OF FIRST FREE TABLE
        LD      D,(IX + 1)      ;BYTE OUT OF FIRST TWO TABLE BYTES
        LD      HL,IOBUF        ;LOAD SOURCE ADDRESS INTO HL
        LDIR                    ;EXECUTE BLOCK MOVE
```

The program above makes absolutely no assumptions. Both source and destination tables may have any length and may be located anywhere in memory.

Notice that there is no instruction execution loop, since the LDIR block move will not stop executing until the entire block of data has been moved.

SUPPORT DEVICES THAT MAY BE USED WITH THE Z80

The Z80 signal interface is very close to that of the 8080A. When looking at Z80 signals we saw how they may be combined to generate 8080A equivalents. Thus **8080A support devices described in Chapter 4 may be used with the Z80 CPU. Exceptions are the 8259 Priority Interrupt Control Unit and the TMS5501 multifunction device.**

The 8259 Priority Interrupt Control Unit should not be used with the Z80 CPU because the Z80 CPU provides essentially the same capabilities within the CPU chip itself. So far as signal interface is concerned, you could use an 8259 with a Z80, but it would make no sense.

The TMS5501 cannot be used with a Z80 because it assumes status on the Data Bus — as output by the 8080A without an 8228 System Controller.

The 8085 support devices — the 8155, the 8355 and the 8755 — **are difficult to use with the Z80;** you have to multiplex the low order eight Z80 address lines and the Z80 8-bit Data Bus to simulate the 8085 multiplexed bus lines. Logic needed to perform this bus multiplexing would likely be more expensive than discrete packages that implement individual functions provided by the 8155 and 8355 multifunction devices.

Using MC6800 support devices with the Z80 is not practical. MC6800 support devices all require a synchronizing clock signal whose characteristics cannot be generated simply from the Z80 clock signal.

The Z80 support devices (which we are about to describe) are not general purpose devices. The Z80 PIO and CTC devices decode the $\overline{M1}$, $\overline{IORQ}$ and $\overline{RD}$ control signals to identify a number of functions. Table 7-5 defines the manner in which the Z80 PIO decodes these three signals. Were you to use the Z80 PIO or CTC with any other microprocessor, you would have to multiplex the other microprocessor's control signals in order to create equivalents of $\overline{M1}$, $\overline{IORQ}$, and $\overline{RD}$; this may not be straightforward.

THE Z80 PARALLEL I/O INTERFACE (PIO)

The Z80 PIO is Zilog's parallel interface device; it may be looked upon as a replacement for the 8255 PPI, but it is equivalent to the PPI at a functional level only. No attempt has been made to make the Z80 PIO an upward compatible replacement for the 8255 PPI.

The Z80 PIO has 16 I/O pins, divided into two 8-bit I/O ports. Each I/O port has two associated control lines. This makes the Z80 PIO more like the Motorola MC6820 than the 8255 PPI.

The two Z80 PIO I/O ports may be separately specified as input, output or control ports. When specified as a control port, pins may be individually assigned to input or output. Port A may be used as a bidirectional I/O port.

The Z80 PIO also provides a significant interrupt handling capability. This includes:

- **The ability to define conditions which will initiate an interrupt.**
- **Interrupt priority arbitration**
- **Vectored response to an interrupt acknowledge**

Figure 7-16 illustrates that part of our general microcomputer system logic which has been implemented on the Z80 PIO.

The Z80 PIO is packaged as a 40-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL compatible. The device is fabricated using N-channel silicon gate depletion load technology.

Z80PIO PINS AND SIGNALS

Z80 PIO pins and signals are illustrated in Figure 7-17. Signals are very straightforward; therefore their functions will be summarized before we discuss device characteristics and operation.

Let us first consider the PIO CPU interface.

All data transfers between the PIO and the CPU occur via the Data Bus, which connects to pins D0 - D7.

For the PIO to be selected, a low input must be present at $\overline{\text{CE}}$. There are two additional address lines. **B/$\overline{\text{A}}$ SEL selects Port A if low and Port B if high. For the selected I/O port, C/$\overline{\text{D}}$ SEL selects a data buffer when low and a control buffer when high.** Device select logic is summarized in Table 7-4.

Table 7-4. Z80 PIO Select Logic

SIGNAL			SELECTED LOCATION
$\overline{\text{CE}}$	B/$\overline{\text{A}}$ SEL	C/$\overline{\text{D}}$ SEL	
0	0	0	Port A data buffer
0	0	1	Port A control buffer
0	1	0	Port B data buffer
0	1	1	Port B control buffer
1	X	X	Device not selected

Z80 PIO device control logic is not straightforward. Of the control signals output by the Z80 CPU, three are input to the PIO; $\overline{\text{M1}}$, $\overline{\text{IORQ}}$, and $\overline{\text{RD}}$. $\overline{\text{WR}}$ is not output to the PIO. **Table 7-5 illustrates the way in which Z80 PIO interprets $\overline{\text{M1}}$, $\overline{\text{IORQ}}$ and $\overline{\text{RD}}$.** Observe that $\overline{\text{RD}}$ is being treated as a signal with two active states: low $\overline{\text{RD}}$ specifies a read operation, whereas high $\overline{\text{RD}}$ specifies a write operation. This does not conform to the CPU, which treats $\overline{\text{RD}}$ and $\overline{\text{WR}}$ as signals with a low active state only.

Let us now look at the PIO external logic interface.

A0 - A7 represent the eight bidirectional I/O Port A lines; I/O Port A is supported by two control signals, A RDY and $\overline{\text{A STB}}$.

Table 7-5. Z80 PIO Interpretation Of Control Signals

SIGNALS*			FUNCTIONAL INTERPRETATION
$\overline{\text{M1}}$	$\overline{\text{IORQ}}$	$\overline{\text{RD}}$	
0	0	0	No function
0	0	1	Interrupt acknowledge
0	1	0	Check for end of interrupt service routine
0	1	1	Reset
1	0	0	Read from PIO to CPU
1	0	1	Write from CPU to PIO
1	1	0	No function
1	1	1	No function

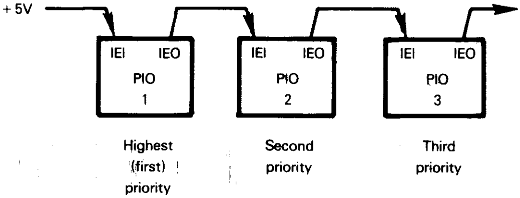
* These interpretations only apply if the device has been selected

Similarly, I/O Port B is implemented via the eight bidirectional lines B0 - B7 and the two associated control lines B RDY and $\overline{\text{B STB}}$.

The I/O Port A and B control lines provide handshaking logic which we will describe shortly.

Now consider interrupt control signals.

IEI and IEO are standard daisy chain interrupt priority signals. When more than one PIO is present in a system, the highest priority PIO (i.e., the one electrically closest to the CPU) will have IEI tied to +5V and will connect its IEO to the IEI for the next highest priority PIO in the daisy chain:



If you are unsure of daisy chain priority networks refer back to Volume I for clarification.

$\overline{\text{INT}}$ is a standard interrupt request signal which is output by the Z80 PIO and must be connected as an input to the Z80 CPU interrupt request. Observe that there is no interrupt acknowledge line since $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ simultaneously low constitute an interrupt acknowledge and will thus be decoded by the Z80 PIO.

Clock, power and ground signals are absolutely standard. The same clock signal is used by the PIO and the Z80 CPU.

Observe that there is no Reset signal to the PIO. $\overline{\text{M1}}$ low with both $\overline{\text{RD}}$ and $\overline{\text{IORQ}}$ high constitutes a reset. We will describe the effect of a Z80 PIO reset after discussing operating modes.

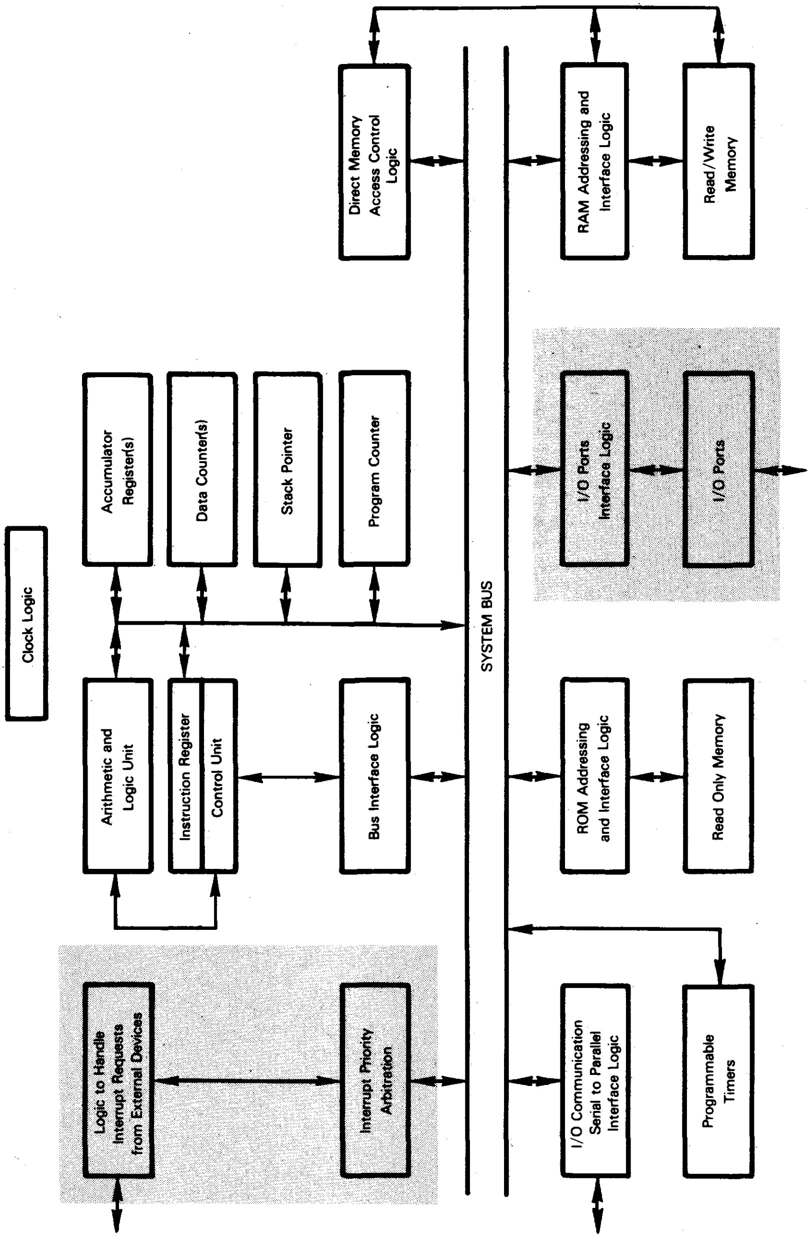
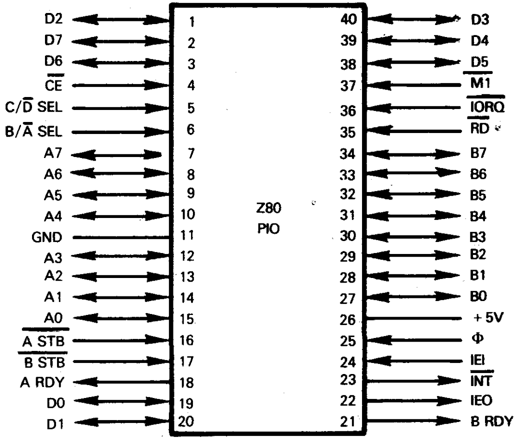

Clock Logic
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Direct Memory Access Control Logic
Arithmetic and Logic Unit
Instruction Register
Control Unit
Bus Interface Logic
SYSTEM BUS
Logic to Handle Interrupt Requests from External Devices
Interrupt Priority Arbitration
I/O Communication Serial to Parallel Interface Logic
Programmable Timers
ROM Addressing and Interface Logic
Read Only Memory
I/O Ports Interface Logic
I/O Ports
RAM Addressing and Interface Logic
Read/Write Memory

Figure 7-16. Logic Functions Of The Z80 PIO

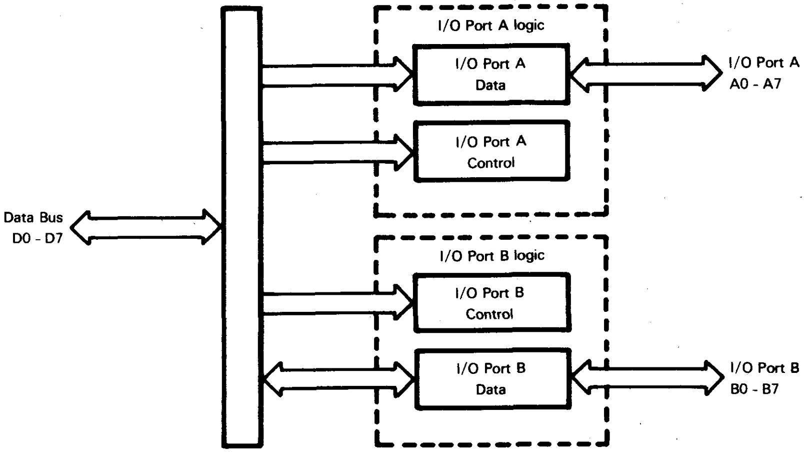


PIN NAME	DESCRIPTION	TYPE
D0 - D7	Data Bus	Tristate, Bidirectional
$\overline{\text{CE}}$	Device Enable	Input
B/$\overline{\text{A}}$ SEL	Select Port A or Port B	Input
C/$\overline{\text{D}}$ SEL	Select Control or Data	Input
$\overline{\text{M1}}$	Instruction fetch machine cycle signal from CPU	Input
$\overline{\text{IORQ}}$	Input/Output request from CPU	Input
$\overline{\text{RD}}$	Read cycle status from CPU	Input
A0 - A7	Port A Bus	Tristate, Bidirectional
A RDY	Register A Ready	Output
$\overline{\text{A STB}}$	Port A strobe pulse	Input
B0 - B7	Port B Bus	Tristate, Bidirectional
B RDY	Register B Ready	Output
$\overline{\text{B STB}}$	Port B strobe pulse	Input
IEI	Interrupt enable in	Input
IEO	Interrupt enable out	Output
$\overline{\text{INT}}$	Interrupt request	Output
Φ, + 5V,GND	Clock, Power and Ground	

Figure 7-17. Z80 PIO Signals And Pin Assignments

Z80 PIO OPERATING MODES

To the programmer, a Z80 PIO will be accessed as four addressable locations:



By loading appropriate information into the Control register you determine the mode in which the I/O port is to operate.

The Z80 PIO has operating modes which are equivalent to those of the 8255 PPI, plus an additional mode which the 8255 PPI does not have. However, 8255 PPI Mode 0 provides 24 I/O lines, as against a maximum of 16 I/O lines available with the Z80 PIO.

Zilog literature uses Mode 0, Mode 1 and Mode 2 to describe the ways in which the Z80 PIO can operate; in order to avoid confusion between mode designations as used by the Z80 PIO and the 8255 PPI, mode equivalences are given in Table 7-6.

Table 7-6. Z80 PIO And 8255 Mode Equivalences

Z80 PIO	8255 PPI	INTERPRETATION
Mode 3*	Mode 0	Simple input or output
Mode 0	Mode 1	Output with handshaking
Mode 1	Mode 1	Input with handshaking
Mode 2	Mode 2	Bidirectional I/O with handshaking
Mode 3	None	Port pins individually assigned as controls

*Special case of Mode 3

Let us now look at the Z80 PIO modes in more detail.

Output mode (Mode 0) allows Port A and/or Port B to be used as a conduit for transferring data to external logic. Figure 7-18 illustrates timing for Mode 0. An output cycle is initiated when the CPU executes any Output instruction accessing the I/O port. The Z80 PIO does not receive the $\overline{WR}$ pulse from the CPU, therefore it derives an equivalent signal by ANDing RD • $\overline{CE}$ • $\overline{C/D}$ • $\overline{IORQ}$.

This pseudo write pulse ($\overline{WR}$* in Figure 7-18) is used to strobe data off the Data Bus and into the addressed I/O port's Output register. After the pseudo write pulse goes high, on the next high-to-low transition of the clock pulse Φ, the RDY control signal is output high to external logic. RDY remains high until external logic returns a low pulse on the STB acknowledge. On the following high-to-low clock pulse Φ transition, RDY returns low. The low-to-high STB transition also generates an interrupt request.

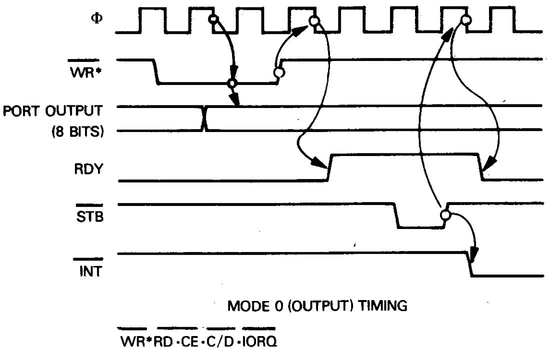
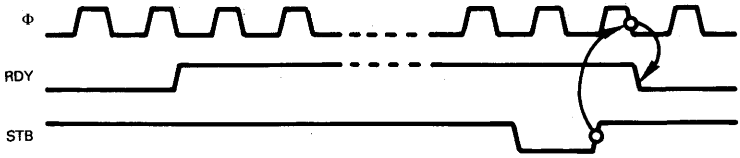
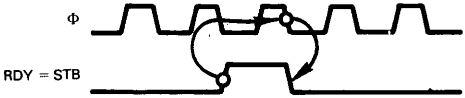


Figure 7-18. Mode 0 (Output) Timing

The RDY and STB signal transition logic has been designed to let RDY create STB. If you connect these two signals, the RDY low-to-high transition becomes the STB low-to-high transition and RDY is strobed high for one clock pulse only. This may be illustrated as follows:





Timing for input mode (Mode 1) is illustrated in Figure 7-19. External logic initiates an input cycle by pulsing STB low. This low pulse causes the Z80 PIO to load data from the I/O port pins into the port Input register. On the rising edge of the STB pulse an interrupt request will be triggered.

On the falling edge of the Φ clock pulse which follows STB input high, RDY will be output low informing external logic that its data has been received but has not yet been read. RDY will remain low until the CPU has read the data, at which time RDY will be returned high.

It is up to external logic to ensure that data is not input to the Z80 PIO while RDY is low. If external logic does input data to the Z80 PIO while RDY is low, then the previous data will be overwritten and lost — and no error status will be reported.

In bidirectional mode (Mode 2), the control lines supporting I/O Ports A and B are both applied to bidirectional data being transferred via Port A; Port B must be set to bit control (Mode 3).

Figure 7-20 illustrates timing for bidirectional data transfers. This figure is simply a combination of Figures 7-18 and 7-19 where the A control lines apply to data output while the B control lines apply to data input. The only unique feature of Figure 7-20 is that bidirectional data being output via Port A is stable only for the duration of the $\overline{\text{A STB}}$ low pulse. This is necessary in bidirectional mode since the Port A pins must be ready to receive input data as soon as the output operation has been completed.

Once again, it is up to external logic to make sure that it conforms with the timing requirements of bidirectional mode operation. External logic must read output data while $\overline{\text{A STB}}$ is low. If external logic does not read data at this time, the data will not be read and the Z80 PIO will not report an error status to the CPU; there is no signal that external logic sends back to the Z80 PIO following a successful read.

Also, it is up to external logic to make sure that it transmits data to Port A only while B RDY is high and A RDY is low. If external logic tries to input data while the Z80 PIO is outputting data, input data will not be accepted. If external logic tries to input data before previously input data has been read, the previously input data will be lost and no error status will be reported.

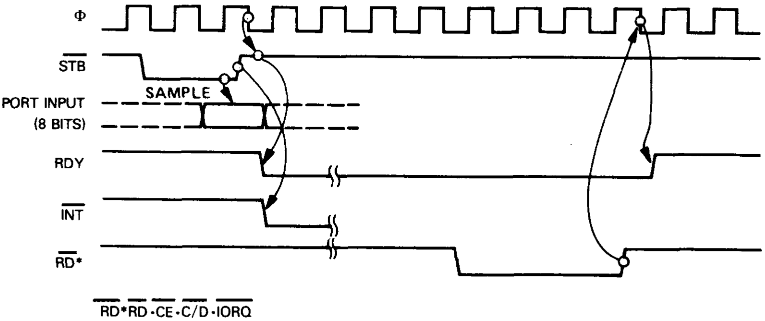


Figure 7-19. Mode 1 (Input) Timing

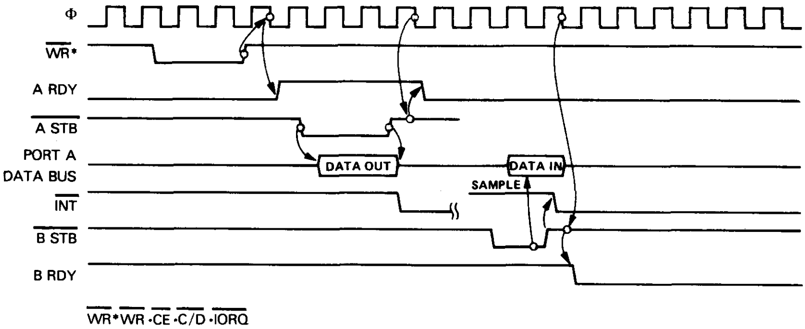


Figure 7-20. Port A, Mode 2 (Bidirectional) Timing

Control mode (Mode 3) does not use control signals. You must define every pin of an I/O port in Mode 3 as an input or an output pin. The section on programming the Z80 PIO explains how to do this. Timing associated with the actual transfer of data at a single pin is as illustrated in Figures 7-18 and 7-19, ignoring the RDY and $\overline{STB}$ signals. If all the pins of a single port are defined in the same direction, then that port can be used for simple parallel input or output (without handshaking).

Z80 PIO INTERRUPT SERVICING

The Z80 PIO has a single interrupt request line via which it transmts interrupt requests to the CPU.

An interrupt request can originate from I/O Port A logic, or from I/O Port B logic. In the case of simultaneous interrupt requests, I/O Port A logic has higher priority.

An interrupt request may be created in one of two ways. We have already seen in our discussion of Modes 0, 1 and 2 that appropriate control signal transitions will activate the interrupt request line; that is the first way in which an interrupt request may occur. In Mode 3 you can program either I/O port to generate an interrupt request based on the status of signals at individual I/O port pins; you can specify which I/O port pins will contribute to interrupt request logic and what the pin states must be for the interrupt request to occur. In a microcomputer system that has more than one Z80 PIO interrupt, priorities are arbitrated using daisy chain logic as we have already described. But there is a significant difference between priority arbitration within a Z80 system as compared to typical priority arbitration. Figure 7-21 illustrates interrupt acknowledge timing.

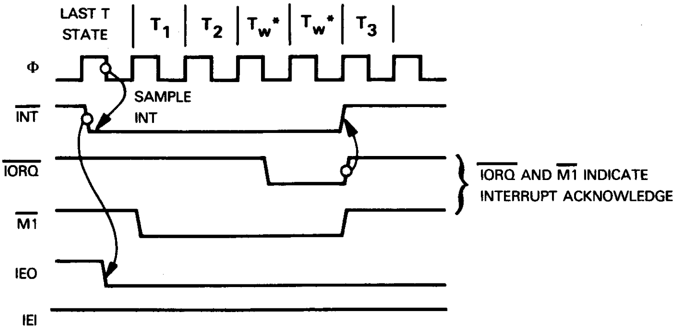


Figure 7-21. Interrupt Acknowledge Timing

The Z80 PIO requires the CPU to execute an RETI instruction upon concluding an interrupt service routine. Following an interrupt, an acknowledged Z80 PIO contінously scans the Data Bus whenever $\overline{M1}$ is pulsed low. Until an RETI instruction's object code is detected, the acknowledged Z80 PIO will continuously output IEO low, thus disabling all lower priority Z80 PIOs. As soon as an RETI instruction's object code is detected on the Data Bus, the Z80 PIO will output IEO high, thus enabling lower priority Z80 PIOs. What this means is that interrupt priorities extend to the interrupt service routine as well as the interrupt request arbitration logic. Once an interrupt has been acknowledged, all lower priority interrupt requests will be denied until the acknowledged interrupt service routine has completed execution and has executed an RETI instruction. However, higher priority interrupts can be acknowledged and in turn interrupt an executing service routine. This is identical to the priority arbitration logic which we described for the 8259 PICU.

You can, if you wish, enable lower priority interrupts by executing an RETI instruction before an interrupt service routine has completed execution. But this requires that you execute an RETI instruction in order to return from a subroutine within the interrupted service routine. This instruction sequence may be illustrated as follows:

```
;START OF INTERRUPT SERVICE ROUTINE
        -
        -
        -
        CALL    ENABLE     ;ENABLE ALL INTERRUPTS AT PIO DEVICES
        -
        -
        -
        RET                ;END OF INTERRUPT SERVICE ROUTINE
ENABLE  RETI
```

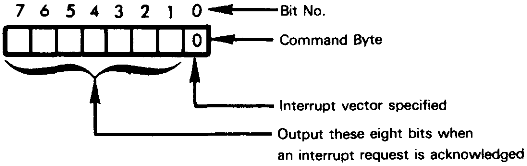
If you simply executed an RETI instruction shortly after entering an interrupt service routine, you would make a hasty exit from the routine — before completing the tasks that have to be performed in response to the acknowledged interrupt.

PROGRAMMING THE Z80 PIO

You program the Z80 PIO by inputting and outputting data in conjunction with a series of commands.

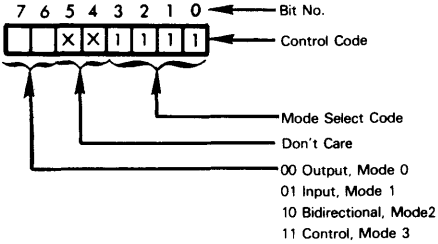
Let us start by identifying command format.

If the 0 bit of a command is low, then the receiving I/O port logic will interpret the command as an interrupt vector, with which it must respond to an interrupt acknowledge, assuming that the CPU is operating in interrupt Mode 2:

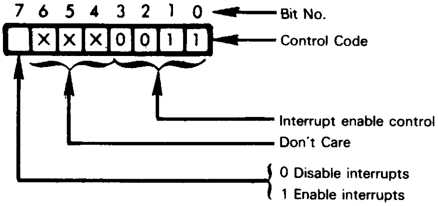

7 6 5 4 3 2 1 0
Bit No.
0
Command Byte
Interrupt vector specified
Output these eight bits when an interrupt request is acknowledged

Do not confuse CPU interrupt modes with I/O port modes; they have nothing in common.

In order to define an I/O port's mode you must output a Control code to the I/O port's Control buffer. This is the Control code format:



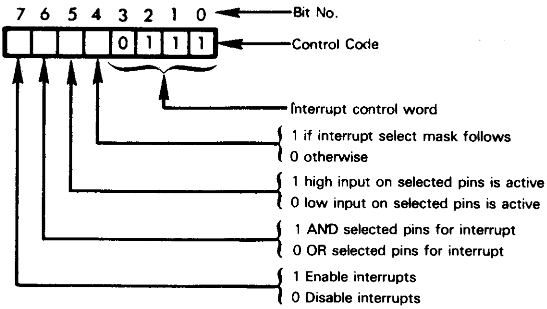
Observe that the same address, the I/O Port A or B Control buffer address, is used when outputting a Control code, an interrupt vector, or a mode select. The low order four bits of the Control code determine the way in which the Control code will be interpreted. **The following Control code will enable or disable interrupts:**



If a Mode Select Control code is output specifying that an I/O port will operate in Mode 3, then the next byte output is assumed to be a pin direction mask. 1 identifies an input pin, whereas 0 identifies an output pin. Here is a sample instruction sequence:

```
LD      C,(PORTAC)    ;LOAD PORT A CONTROL ADDRESS INTO
                      REGISTER C
LD      A,0CFH        ;LOAD MODE 3 SELECT INTO ACCUMULATOR
OUT     (C),A         ;OUTPUT TO PORT A CONTROL REGISTER
LD      A,3AH         ;DEFINE PINS 5, 4, 3 AND 1 AS INPUTS,
OUT     (C),A         ;PINS 7, 6, 2 AND 0 AS OUTPUTS
```

If you set an I/O port to Mode 3, then you can define the conditions which will cause an interrupt request; you do this by outputting the following interrupt Control code:



7 6 5 4 3 2 1 0
Bit No.
0 1 1 1
Control Code
Interrupt control word
1 if interrupt select mask follows
0 otherwise
1 high input on selected pins is active
0 low input on selected pins is active
1 AND selected pins for interrupt
0 OR selected pins for interrupt
1 Enable interrupts
0 Disable interrupts

When you output an interrupt Control code, as illustrated above, if bit 4 is 1, Z80 PIO logic will assume that the next Control code output is an interrupt mask. An interrupt mask selects the pins that will contribute to interrupt request logic. A 0 bit selects a pin, while a 1 bit deselects the pin.

Combining the various Control codes that have been described we can now illustrate a typical sequence of instructions for accessing a Z80 PIO. Assume that PIO I/O port addresses are:

Port A data	4
Port A command	5
Port B data	6
Port B command	7

We are going to set I/O Port B to Mode 3, with an interrupt request triggered by either pin 6, 3 or 2 high. Pins 6, 3, 2 and 1 will be input pins, while pins 7, 5, 4 and 0 are outputs. The Port B interrupt vector will be 04. Port A will be a bidirectional I/O port with an interrupt vector of 02. Here is the initialization instruction sequence:

```
LD      A,8FH       ;SET PORT A TO MODE 2
OUT     (5),A
LD      A,2         ;OUTPUT INTERRUPT VECTOR
OUT     (5),A
LD      C,7         ;SET PORT B ADDRESS IN C
LD      A,0CFH      ;SET PORT B TO MODE 3
OUT     (C),A
LD      A,4EH       ;OUTPUT PIN DIRECTION MASK
OUT     (C),A
LD      A,4         ;OUTPUT INTERRUPT VECTOR
OUT     (C),A
LD      A,0B7H      ;OUTPUT INTERRUPT CONTROL WORD
OUT     (C),A
LD      A,0B3H      ;OUTPUT INTERRUPT MASK
OUT     (C),A
```

THE Z80 CLOCK TIMER CIRCUIT (CTC)

The Z80 Clock Timer Circuit is a programmable device which contains four sets of timing logic. Each set of timing logic can be programmed independently as an interval timer or an external event counter.

The master Z80 system clock is used by interval timer logic. A time out may be identified by an interrupt request.

An external signal is used to decrement when logic is functioning as an event counter. An interrupt may be requested when the predetermined number of events count out.

If you compare the Z80 CTC with the 8253 Counter/Timer described in Chapter 4, **you will see that the Z80 CTC has four sets of counter/timer logic as compared to the three sts of the 8253;** however the 8253 has more programmable options. In addition to functioning as an event counter or an interval timer, the 8253 can be programmed to generate a variety of square waves and pulse output signals.

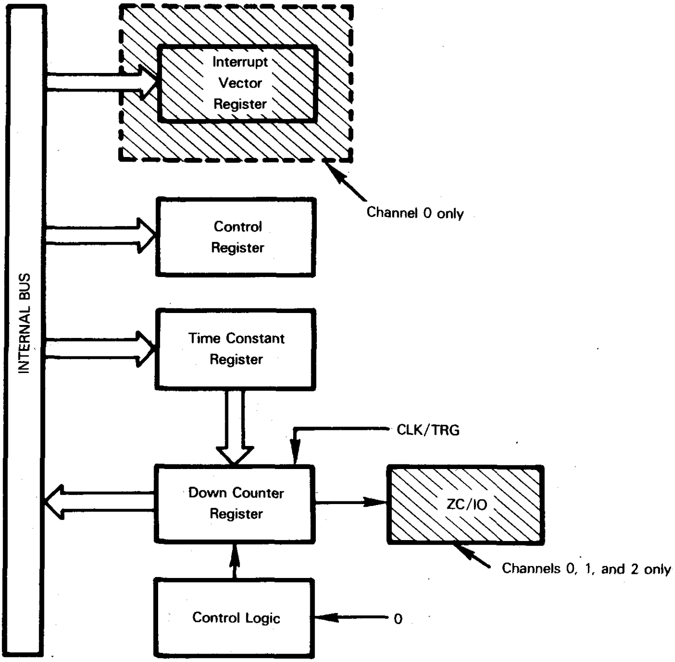
The Z80 CTC is fabricated using N-channel depletion load technology. It is packaged as a 28-pin DIP. All pins are TTL compatible.

Z80 CTC FUNCTIONAL ORGANIZATION

Before we examine pins, signals, and operating characterics of the Z80 CTC in detail, let us take an overall look at device logic.

There are four counter/timer logic elements in a Z80 CTC; each is referred to as a "channel".

Each of the four counter/timer channels may be visualized as consisting of three 8-bit registers and two control signals. This may be illustrated as follows:



An initial counter or timer constant is loaded into the Time Constant register. The value in the Time Constant register is maintained unaltered until you write a new value into this register.

The initial Timer Constant is loaded into the Down Counter register at the beginning of a counter or timer operation; the contents of the Down Counter register are decremented. You can at any time read the contents of the Down Counter register in order to determine how far a time interval or event counting sequence has progressed.

The Channel Control register contains a Control code which defines the channel's programmable options. There are four Control registers, one for each of the four channels. Thus one channel's operations in no way influence operations for any other channel.

There is an Interrupt Vector register which is addressed as though it were part of channel 0 logic. This register contains the address which is transmitted by the Z80 CTC upon receiving an interrupt acknowledge. The Z80 CTC assumes that the Z80 CPU is operating in Interrupt mode 2 — in which mode the device requesting an interrupt responds to an acknowledge by providing the second byte of a subroutine address which the CPU will Call. For details refer to our earlier discussion of the Z80 CPU.

Z80 CTC PINS AND SIGNALS

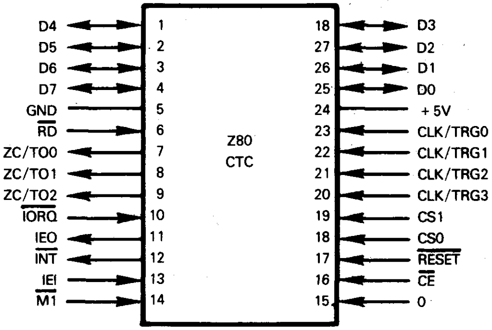
Z80 CTC pins and signals are illustrated in Figure 7-22.

D0 - D7 is the bidirectional Data Bus via which parallel data is transferred between the CPU and any register of the Z80 CTC.

$\overline{\text{CE}}$ is the master chip select signal for the Z80 CTC. This signal must be low for the device to be selected.

While $\overline{\text{CE}}$ is low, **CS0 and CS1 are used to select one of the four counter/timer logic channels** as follows:

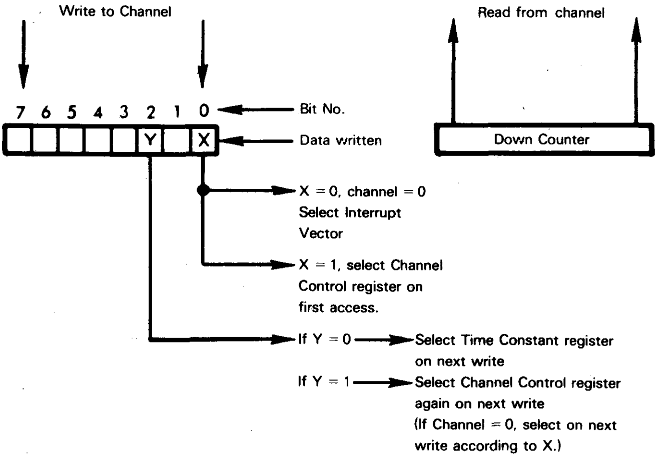
CS1	CS0	Channel
0	0	0
0	1	1
1	0	2
1	1	3



PIN NAME	DESCRIPTION	TYPE
D0-D7	Data Bus	Bidirectional, tristate
CLK/TRG0,	External Clock or timer trigger	Input
CLK/TRG1,		
CLK/TRG2,		
CLK/TRG3,		
ZC/TO0	Zero Count or timeout indicator	Output
ZC/TO1		
ZC/TO2		
$\overline{\text{M1}}$	Instruction fetch machine cycle signal from CPU	Input
$\overline{\text{IORQ}}$	Input/Output request from CPU	Input
$\overline{\text{RD}}$	Read cycle status from CPU	Input
$\overline{\text{RESET}}$	Device Reset	Input
IEI	Interrupt enable in	Input
IEO	Interrupt enable out	Output
$\overline{\text{INT}}$	Interrupt request	Output
$\overline{\text{CE}}$	Device enable	Input
CS0, CS1	Register select	Input
Φ, +5V, GND	Clock, power and ground	

Figure 7-22. Z80-CTC Signals and Pin Assignments

CS0 and CS1 select registers associated with counter/timer logic, to be accessed by read and write operations. The actual register which will be accessed is determined as follows:



Write to Channel
Read from channel
7 6 5 4 3 2 1 0
Bit No.
Y
X
Data written
Down Counter
X = 0, channel = 0 Select Interrupt Vector
X = 1, select Channel Control register on first access.
If Y = 0
Select Time Constant register on next write
If Y = 1
Select Channel Control register again on next write (If Channel = 0, select on next write according to X.)

As the illustration above would imply, the Down Counter register is the only location of any channel whose contents can be read. All other registers are write only locations.

When you write to a channel, bits 0 and 2 of the data byte being written determine the data destination as follows:

1) If bit 0 is 0 and you are selecting channel 0, then the data is written to the Interrupt Vector register.

2) If bit 0 is 0 and you select channel 1, 2 or 3, the data destination is undefined.

3) If bit 0 is 1, then on the first access of any channel the data will be written to the Channel Control register.

4) If within the data byte written to a Channel Control register bit 0 is 1 and bit 2 is 0, then the next data byte written to this channel will be loaded into the time Constant register, irrespective of whether bit 0 is 0 or 1. The data written will be interpreted as a time constant; select logic will immediately revert to selecting the Channel Control register or the Interrupt Vector register on the next write, depending on the condition of bit 0 of the next data byte.

$\overline{\text{M1}}$, $\overline{\text{IORQ}}$ and $\overline{\text{RD}}$ are three control signals input to the Z80 CTC. Combinations of these three control signals **control logic within the Z80 CTC, as described for the Z80 PIO. An exception is the device Reset.** The Z80 CTC has its own Reset input. The PIO decodes a Reset when $\overline{\text{M1}}$ is low while $\overline{\text{IORQ}}$ and $\overline{\text{RD}}$ are high. With the exception of the RESET function, Table 7-5 defines the manner in which the Z80 CTC interprets $\overline{\text{M1}}$, $\overline{\text{IORQ}}$, and $\overline{\text{RD}}$ signals.

Interrupt logic has three associated signals: IEI, IEO and $\overline{\text{INT}}$. These signals operate exactly as described for the Z80 PIO.

The Z80 CTC requests an interrupt with a low $\overline{\text{INT}}$ output.

IEI and IEO are used to implement daisy chain priority interrupt logic as described for the PIO.

Each of the four counter/timer channels has a CLK/TRG input control. This signal can be used to trigger timer logic; it is also used as a decrement control by counter logic.

Counter/timer logic systems 0, 1 and 2 have a ZC/TO output. This signal is pulsed high on a time out or a count out.

When a low input is applied to the $\overline{\text{RESET}}$ pin, the Z80 CTC is reset. AT this time all counter/timer logic is stopped, $\overline{\text{INT}}$ is output high, IEO is output at the IEI level and the Data Bus is floated. Register contents are not cleared during a reset.

Z80 CTC OPERATING MODES

The Z80 CTC is accessed by the CPU as four I/O ports or four memory locations. Timing for any CTC access conforms to descriptions given earlier in this chapter for the CPU.

Let us begin by looking at a counter/timer operating as a timer.

Using an appropriate Control code (described later) you select Timer mode for the channel and specify that an initial time constant is to follow.

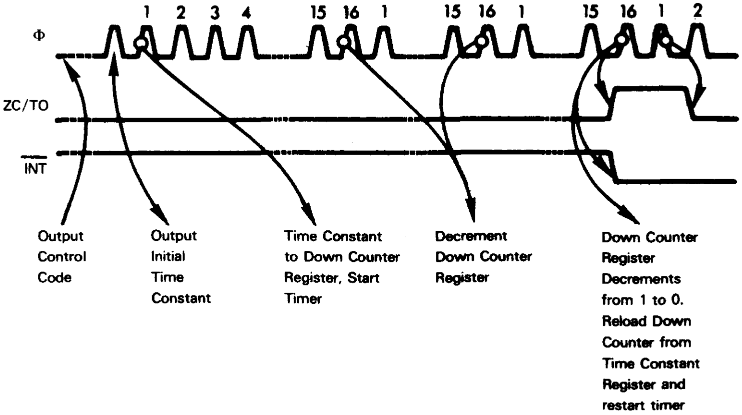
You load an initial constant into the Time Constant register, after which timer operations begin.

You have the option of using the CLK/TRG input to start the timer, in which case timer logic is initiated by external logic. The alternative is to initiate the timer under program control, in which case the timer starts on the clock pulse following the Time Constant register being loaded.

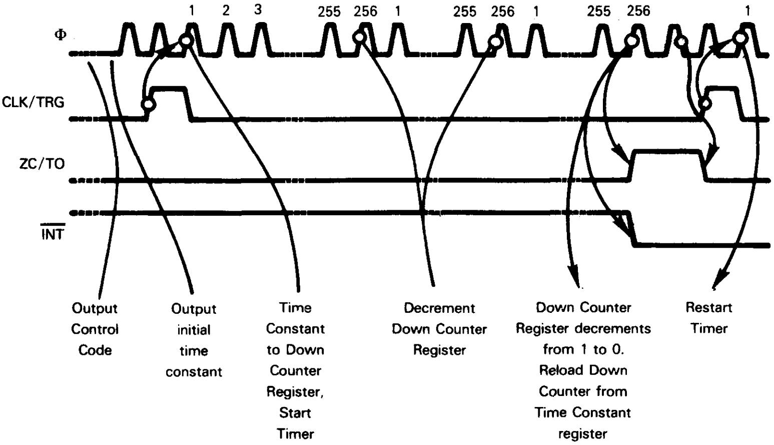
When timer operations begin, the Time Constant register contents are transmitted to the Down Counter register. The Down Counter register contents are decremented on every 16th system clock pulse, or on every 256th system clock pulse. You make the selection via the Control code. Assuming a 500 nanosecond clock, therefore, the timer will decrement the Down Counter register contents every 8 microseconds, or every 128 microseconds.

When timer logic decrements the Down Counter register contents from 1 to 0 a time out occurs. At this time ZC/TO is pulsed high, the Time Constant register contents are reloaded into the Down Counter register and timer logic starts again. Thus timer logic is free running; once started, the timer will run continuously until stopped by an appropriate Control code.

Here is a timing example for a timer started under program control and decrementing the Down Counter register on every 16th clock pulse:

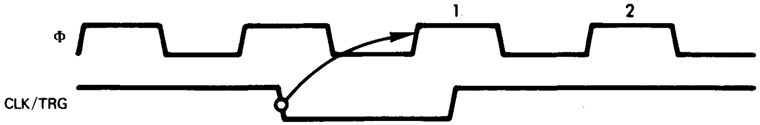

1 2 3 4 15 16 1 15 16 1 15 16 1 2
Φ
ZC/TO
$\overline{\text{INT}}$
Output Control Code
Output Initial Time Constant
Time Constant to Down Counter Register, Start Timer
Decrement Down Counter Register
Down Counter Register Decrements from 1 to 0. Reload Down Counter from Time Constant Register and restart timer

Here is a timing example for a timer whose operations are initiated by CLK/TRG, where the Down Counter register contents are decremented on every 256th clock pulse:


1 2 3 255 256 1 255 256 1 255 256 1
Φ
CLK/TRG
ZC/TO
$\overline{\text{INT}}$
Output Control Code
Output initial time constant
Time Constant to Down Counter Register, Start Timer
Decrement Down Counter Register
Down Counter Register decrements from 1 to 0. Reload Down Counter from Time Constant register
Restart Timer

Observe that every time out is marked by a ZC/TO high pulse. $\overline{\text{INT}}$ is also output low providing interrupt logic is enabled at the channel.

In the illustration above CLK/TRG is shown as a high true signal. You can specify CLK/TRG as a low true signal via the Channel Control code; the timer will be initiated as follows:



For exact timing requirements see the data sheets at the end of this chapter.

You can at any time write new data into the Time Constant register. If you do this while the timer is running, nothing happens until the next time out; at that time the new Time Constant register contents will be transferred to the Down Counter register and subsequent time intervals will be computed based on the new time Constant Register contents.

If you are unforntunate enough to output data to the Time Constant register while a time out is in progress and the Time Constant register contents are being transferred to the Down Counter register, then an undefined value will be loaded into the Down Counter register; however, following the next time out the new value in the Time Constant register will apply; that is to say, there will only be one undefined time interval.

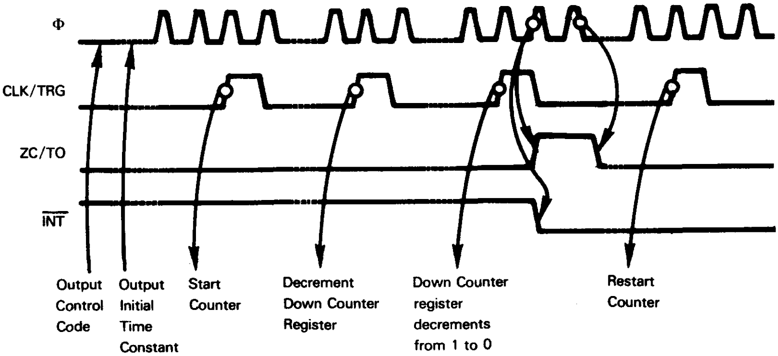
Let us now look at a counter/timer operating as a counter.

Using an appropriate Control code (described later) you select Counter mode for the channel and specify that an initial time constant is to follow.

You load an initial constant into the Time Constant register, after which counter operations begin.

When counter operations begin, the Time Constant register contents are transmitted to the Down Counter register. The Down Counter register contents are decremented every time the CLK/TRG input makes an active transition. Counter logic begins on the first active transition of CLK/TRG following data being loaded into the Time Constant register. The active transition of CLK/TRG may be selected under program control as low-to-high or high-to-low.

When counter logic decrements the Down Counter register contents from 1 to 0, a count out occurs. At this time the ZC/TO signal is pulsed high; an interrupt request occurs, providing the channel's interrupt logic has been enabled. The Time Constant register contents are reloaded into the Down Counter register and counter operations begin again. That is to say, counter logic is free running and will continue to re-execute until specifically stopped by an appropriate Control code. Counter logic timing may be illustrated as follows:


Φ
CLK/TRG
ZC/TO
$\overline{\text{INT}}$
Output Control Code
Output Initial Time Constant
Start Counter
Decrement Down Counter Register
Down Counter register decrements from 1 to 0
Restart Counter

Z80 CTC INTERRUPT LOGIC

Every Z80 CTC channel has its own interrupt logic. A channel's interrupt logic generates an interrupt request when the channel counts out or times out. All interrupt requests are transmitted to the CPU via the $\overline{\text{INT}}$ output. This is true if one, or more than one channel is requesting an interrupt. If more than one channel is requesting an interrupt, then priorities are arbitrated as follows:

Highest Priority	Channel 0
	Channel 1
	Channel 2
Lowest Priority	Channel 3

Every channel's interrupt logic can be individually enabled or disabled under program control.

The Z80 CTC device's overall interrupt logic is identical to that which we have already described for the Z80 PIO.

The interrupt request is transmitted to the CPU via a low $\overline{\text{INT}}$ signal.

The CPU acknowledges the interrupt by outputting $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ low as illustrated in the data sheets at the end of this chapter.

The device requesting an interrupt which is electrically closest to the CPU in the daisy chain acknowledges the interrupt. Presuming this is a Z80 CTC, the CTC places its interrupt vector on the Data Bus; it is assumed that the CPU is operating in Interrupt mode 2. The Z80 CTC immediately outputs IEO low, disabling all devices below it in the daisy chain.

When an RETI instruction is executed, Z80 CTC logic sets IEO high again.

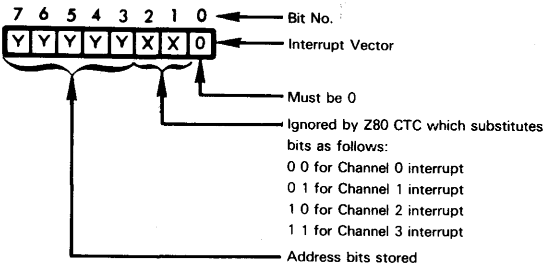
For more information on Z80 interrupt logic refer back to discussions of this subject given earlier in the chapter for the Z80 CPU and the PIO.

PROGRAMMING THE Z80 CTC

These are the steps required to program a Z80 CTC:

1) **Output an interrupt vector once, when initializing the Z80 CTC.**
2) **For each active counter/timer channel, output one or more Control codes. Control codes are used initially to set counter/timer operating conditions and to load the Time Constant register. Subsequently Control codes are used to start and stop the counter/timer, or to change the initial time constant.**

The interrupt vector is written to a counter/timer by outputting a byte of data to counter/timer channel 0 with a 0 in the low order bit. The interrupt vector may be illustrated as follows:


7 6 5 4 3 2 1 0
Bit No.
Y Y Y Y Y X X 0
Interrupt Vector
Must be 0
Ignored by Z80 CTC which substitutes bits as follows:
0 0 for Channel 0 interrupt
0 1 for Channel 1 interrupt
1 0 for Channel 2 interrupt
1 1 for Channel 3 interrupt
Address bits stored

The Control code which must be output to each active channel will be interpreted as illustrated in Figure 7-23.

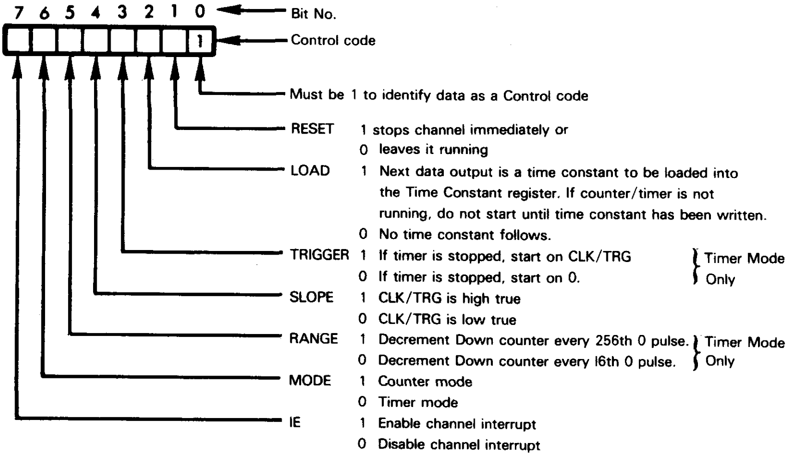

7 6 5 4 3 2 1 0
Bit No.
1
Control code
Must be 1 to identify data as a Control code
RESET 1 stops channel immediately or
0 leaves it running
LOAD 1 Next data output is a time constant to be loaded into the Time Constant register. If counter/timer is not running, do not start until time constant has been written.
0 No time constant follows.
TRIGGER 1 If timer is stopped, start on CLK/TRG
0 If timer is stopped, start on 0.
Timer Mode Only
SLOPE 1 CLK/TRG is high true
0 CLK/TRG is low true
RANGE 1 Decrement Down counter every 256th 0 pulse.
0 Decrement Down counter every l6th 0 pulse.
Timer Mode Only
MODE 1 Counter mode
0 Timer mode
IE 1 Enable channel interrupt
0 Disable channel interrupt

Figure 7-23. Z80 CTC Control Code Interpretation

Bit 0 must be 1 to identify the data as a Control code. If bit 0 is 0, then the data is interpreted as an interrupt vector — providing Channel 0 is addressed; the data is undefined otherwise.

Bit 1 is used to stop the channel when it is running. If bit 1 is 0, then every time the channel times out the Down Counter register is immediately reloaded from the Time Constant register contents and channel operations restart according to current options. If bit 1 is 1, the channel stops immediately; the ZC/TO output is inactive and channel interrupt logic is disabled. The channel must be restarted by outputting a new Control code.

Bit 2 is used to output time constants. If bit 2 is 1, then the next data output to the channel will be interpreted as a time constant. If bit 2 is 0, then the next data output to the channel will be interpreted as another Control code, or an interrupt vector, depending on the bit 0 value.

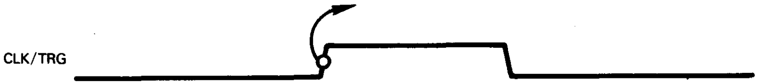
Bit 3 applies to Timer mode only; assuming that the timer is not running, it determines whether timer operations will be initiated by the system clock signal Φ, or by CLK/TRG.

If bit 3 is 0 then timer operations are initiated by system clock signal Φ; the timer will start on the next leading edge of Φ, unless the current Control code specifies (via bit 2) that a new time constant is to be output, in which case the timer will start on the rising edge of Φ which immediately follows output of the time constant. Timing for these two cases has been illustrated earlier.

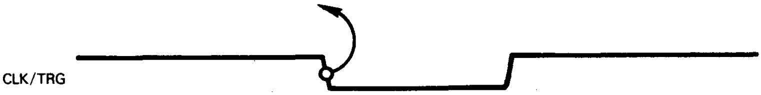
If bit 3 is 1, then the active transition of the CLK/TRG signal initiates the timer. Once again, if bit 2 of the current Control code specifies that a new time constant is to be output then timer logic cannot be started until this new time constant has been output. Timing has been illustrated earlier.

Bit 4 determines whether the low-to-high or the high-to-low transition of CLK/TRG is active. Assuming that bit 6 has specified Timer mode and bit 3 has specified the timer will be triggered externally by CLK/TRG, the active transition of CLK/TRG starts the timer. If bit 6 is not 0 or bit 3 is not 1, then the active transition of CLK/TRG decrements the counter.

If bit 4 specifies that a low-to-high transition of CLK/TRG will be active then CLK/TRG is a high true signal. This may be illustrated as follows:



If bit 4 specifies that the high-to-low transition of CLK/TRG will be active then CLK/TRG is low true. This may be illustrated as follows:



Bit 5 applies to Timer mode only. If bit 5 is 0, Down Counter register contents will be decremented every 16th system clock pulse (Φ). If bit 5 is 1, the Down Counter register contents will be decremented every 256th system clock pulse (Φ).

Bit 6 determines whether the channel will be operated as a counter or a timer. If bit 6 is 0, Timer mode is selected; Counter mode is selected if bit 6 is 1.

Bit 7 is an interrupt enable/disable flag. If 0, the channel's interrupt logic is disabled; if 1, the channel's interrupt logic is enabled.

Let us now look at the programming example. Here are the assumed operating conditions for the Z80 CTC:

1) Channel 0 is operating as a counter with an initial time constant of 80_{16} and interrupt logic enabled.
2) Channel 1 is operating as a timer. It decrements on every 16th system clock pulse and has an initial time constant of 40_{16}; its interrupts are disabled and CLK/TRG starts the timer on its low-to-high transition.
3) Channel 2 is operating as a timer. It decrements every 256th system clock pulse and has an initial time constant of $C8_{16}$; its interrupts are enabled and the system clock starts the timer.
4) Channel 3 is inactive.

The CPU is operating with interrupt logic in Mode 2. CTC interrupt service routine starting addresses are stored at memory locations $2C40_{16}$, $2C42_{16}$ and $2C44_{16}$. The CTC is accessed as I/O ports $B8_{16}$, $B9_{16}$ and BB_{16}.

Here is the appropriate CTC initiation instruction sequence:

```
            LD      A,2CH       ;LOAD INTERRUPT VECTOR REGISTER OF CPU
            LD      I,A
            IM      2           ;SELECT CPU INTERRUPT MODE 2
            LD      A,40H       ;OUTPUT INTERRUPT VECTOR TO
            OUT     (0B8H),A    ;CHANNEL 0
;START CHANNEL 0
            LD      A,0C5H      ;OUTPUT THE CONTROL CODE TO CHANNEL 0
            OUT     (0B8H),A
            LD      A,80H       ;OUTPUT THE INITIAL COUNT TO CHANNEL 0
            OUT     (0B8H),A    ;CHANNEL 0 BEGINS OPERATING.
;START CHANNEL1
            LD      A,1DH       ;OUTPUT THE CONTROL CODE TO CHANNEL 1
            OUT     (0B9H),A
            LD      A,40H       ;OUTPUT THE INITIAL TIMER CONSTANT TO CHAN-
                                NEL1
            OUT     (0B9H),A    ;CHANNEL 1 BEGINS OPERATING. (IF TRANSITION
                                OCCURS)
;START CHANNEL 2
            LD      A,0A5H      ;OUTPUT THE CONTROL CODE TO CHANNEL 2
            OUT     (0BAH),A
            LD      A,0C8H      ;OUTPUT THE INITIAL TIMER CONSTANT TO CHAN-
                                NEL 2
            OUT     (0BAH),A    ;CHANNEL 2 BEGINS OPERATING
```

THE Z80 DMA DIRECT MEMORY ACCESS CONTROLLER

This is one of the most remarkable support devices described in this book. Although designed to work with the Z80 CPU, it can — and should — be considered in any microcomputer system that transfers data blocks. Of the Z80 support devices described in this Chapter, only the Z80 DMA uses separate Read and Write Control strobes — which is what makes it universally usable with microprocessors in general.

While the Z80 DMA device is described as a direct memory access controller, it is in reality more than that. Almost any conceivable block transfer operation can be handled by the Z80 DMA device. Specifically, three types of DMA operations are available, each one executable in one of four modes. These are the three types of DMA operations:

Type 1 — Transfer a block of data.

Type 2 — Transfer a block of data, identifying a match byte each time it is encountered.

Type 3 — Seek a match byte in a block of data.

You identify what a "match byte" is by specifying the "match byte" bit pattern. But you do not have to specify all eight bits; a "match byte" may be keyed to the pattern of one or more bits within the byte.

The four modes of operation available with each type of DMA transfer are:

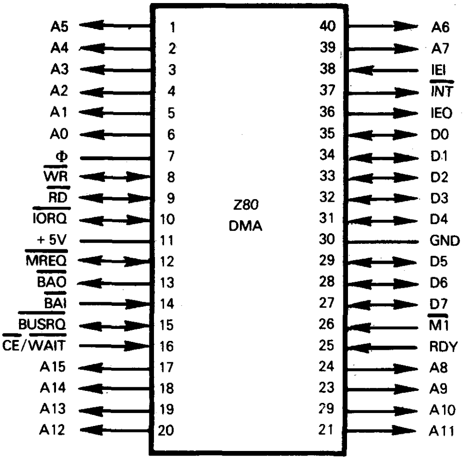
1) Single byte mode, where each direct memory access operates on a single byte of data.
2) Burst mode, where the Z80 DMA device keeps control of the bus for as long as data is continuously ready.
3) Continuous mode, where the Z80 DMA device retains bus access for the entire DMA operation.
4) Transparent mode, where the DMA transfer occurs during memory refresh time; therefore it does not slow down program execution.

Any DMA operation may be continuous, or it may stop when the end of a block and/or a match byte is detected.

Even within the multitude of DMA options described above, data operations are controlled by a bewildering variety of programmable options covering both the data blocks operated on and the control signals accompanying the data operations.

But the Z80 DMA device pays a price for these options; each Z80 DMA device supports a single DMA channel. **Unlike the 8257 DMA Controller and direct memory access devices in general, the Z80 DMA devices take charge of the Data Bus while performing DMA operations.** Thus, each DMA transfer becomes two events: a Read machine cycle followed by a Write machine cycle. The Z80 DMA device receives the data via the Data Bus on the Read machine cycle and sustains it on the Data Bus during the subsequent Write machine cycle, if any. In contrast, the 8257 DMA Controller is disconnected from the Data Bus during any actual DMA transfer, creating only control signals which must be appropriately interpreted as Read and Write strobes by logic at the two ends of the DMA transfer. **The Z80 DMA philosophy is more versatile and makes the devices easier to design with.**

We will not use our standard logic illustration to represent logic of the Z80 DMA devices since any such illustration would be highly misleading. While the box marked "Direct Memory Access Control Logic" would be the only area of the figure shaded, in reality additional parallel data transfer capability is provided.



PIN NAME	DESCRIPTION	TYPE
A0-A15	DMA Address Bus	Tristate, Output
D0-D7	Data Bus	Tristate, Bidirectional
$\overline{M1}$	Identifies instruction fetch machine cycle	Input
$\overline{MREQ}$	Memory request - identifies a memory access	Tristate, Bidirectional
$\overline{IORQ}$	I/O request - identifies an I/O access	Tristate, Bidirectional
$\overline{RD}$	Read from memory or I/O device	Tristate, Bidirectional
$\overline{WR}$	Write to memory or I/O device	Tristate, Bidirectional
$\overline{CE}/\overline{WAIT}$	Multiplexed chip select and machine cycle extend	Input
RDY	Memory or I/O port ready	Input
$\overline{INT}$	Interrupt request	Output
IEI	Interrupt enable in	Input
IEO	Interrupt enable out	Output
$\overline{BUSRQ}$	Bus Request	Bidirectional
$\overline{BAI}$	Bus Acknowledge in	Input
$\overline{BAO}$	Bus Acknowledge out	Output
Φ, +5V, GND	Timing, power and ground	

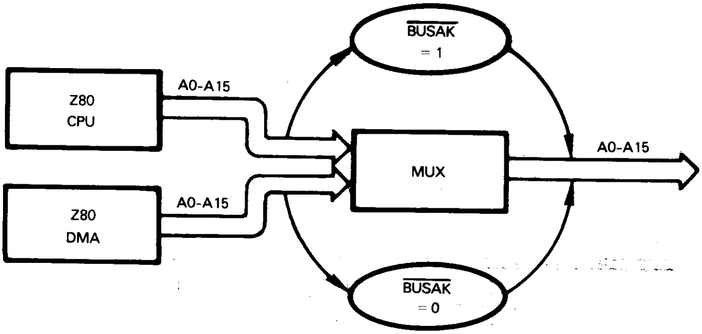
Figure 7-24. Z80 DMA Signals And Pin Assignments

The Z80 DMA device is fabricated using N-channel depletion load MOS technology. It is packaged as a 40-pin DIP. All signals are TTL compatible.

Z80 DMA DEVICE PINS AND SIGNALS

Figure 7-24 summarizes Z80 DMA device pins and signals.

The 16 address lines A0-A15 output memory and I/O device addresses. Memory and I/O device addresses output by the Z80 DMA device are the addresses used during the DMA operation. **The Z80 DMA address lines must be multiplexed with the Z80 CPU Address Bus** since only one or the other can be active at any time. This may be illustrated as follows:



The Data Bus lines D0 - D7 are used by the Z80 CPU to write data to Z80 DMA internal registers, **or to read** data from Z80 DMA internal registers. Unlike other DMA devices, however, **the Z80 DMA device also uses its Data Bus pins to receive and transmit** the data byte which is being transferred **via direct memory access.** This requires the Data Bus to be multiplexed, as illustrated above for the Address Bus.

A standard DMA controller such as the 8257, described in Chapter 4, does not require the Data Bus to be multiplexed, as illustrated above, since the DMA controller itself plays no part in the actual transfer of data under DMA control. The only time the 8257 DMA controller uses its Data Bus is when it responds to CPU accesses — at which time it is just another I/O support device.

CE/WAIT is a multiplexed input control which is used to input a select signal, and may be used additionally to input a machine cycle extend signal. The chip select logic applies only while the CPU is accessing one of the Z80 DMA internal registers. The CPU accesses the Z80 DMA as a single I/O port. The CPU cannot access the Z80 DMA device as a memory location. Thus the low true CE input must be appropriately decoded off the Address Bus, conditioned by control signals RD, WR and/or IORQ, when the CPU is performing a Read or Write operation. MREQ is not active while the CPU is accessing Z80 DMA internal registers. When the Z80 DMA device is performing a DMA operation, it is bus master; therefore, chip select logic and the CE signal no longer apply. At that time the pin may optionally be used to receive a WAIT input, which, as described for the Z80 CPU, causes Wait clock periods to be inserted within Read or Write machine cycles created by the Z80 DMA device.

MREQ, IORQ, RD and WR are memory request, I/O request, Read and Write control signals, respectively. These are the master control signals identifying data on the Data Bus. We have described these signals and their use earlier in this Chapter while discussing the Z80 CPU. These four control signals are bidirectional at the Z80 DMA device. They are control inputs when the CPU is accessing internal registers of the Z80 DMA device; they are control outputs when the Z80 DMA device is controlling data transfer via Direct Memory Access.

M1 is the control signal output by the Z80 CPU to identify an instruction fetch machine cycle. This signal is input to the Z80 DMA device so that the device can detect an instruction fetch machine cycle — during the second half of which dynamic memory refresh occurs. M1 is not output by the Z80 DMA device since an instruction fetch cannot occur during a DMA data transfer operation. M1 is also used to identify an interrupt acknowledge and to decode an RETI instruction.

RDY is a control input which **is transmitted to the Z80 DMA device by the external source or destination logic** when this source or destination logic is ready to transmit or receive data using direct memory access.

The three interrupt logic signals $\overline{\textbf{INT}}$, IEI, and IEO are standard Z80 interrupt signals. The Z80 DMA requests interrupts by outputting a low signal via $\overline{\text{INT}}$. During the interrupt enable process, IEI is the daisy chain priority input and IEO is the daisy chain priority output. These signals have been described in detail in the Z80 PIO discussion.

The Z80 DMA device also uses the $\overline{\textbf{INT}}$ signal to output a control pulse to external logic. This control pulse is directed to external logic while the Z80 DMA is bus master — and the Z80 CPU cannot therefore receive interrupt requests.

The Z80 DMA device requests bus access by outputting a low $\overline{\textbf{BUSRQ}}$ signal. This is transmitted directly to the Z80 CPU $\overline{\text{BUSRQ}}$ input. When the CPU floats its System Bus it responds by outputting $\overline{\text{BUSAK}}$ low; the low **$\overline{\textbf{BUSAK}}$** signal **must be connected to $\overline{\textbf{BAI}}$, which is a daisy chain bus acknowledge priority input. $\overline{\textbf{BAO}}$ is the daisy chain bus acknowledge priority output.** $\overline{\text{BAI}}$ and $\overline{\text{BAO}}$ provide daisy chain logic exactly as described for IEI and IEO, except that $\overline{\text{BAI}}$ and $\overline{\text{BAO}}$ are negative true signals. Thus Z80 DMA devices may be daisy chained separately for interrupt logic and bus access logic.

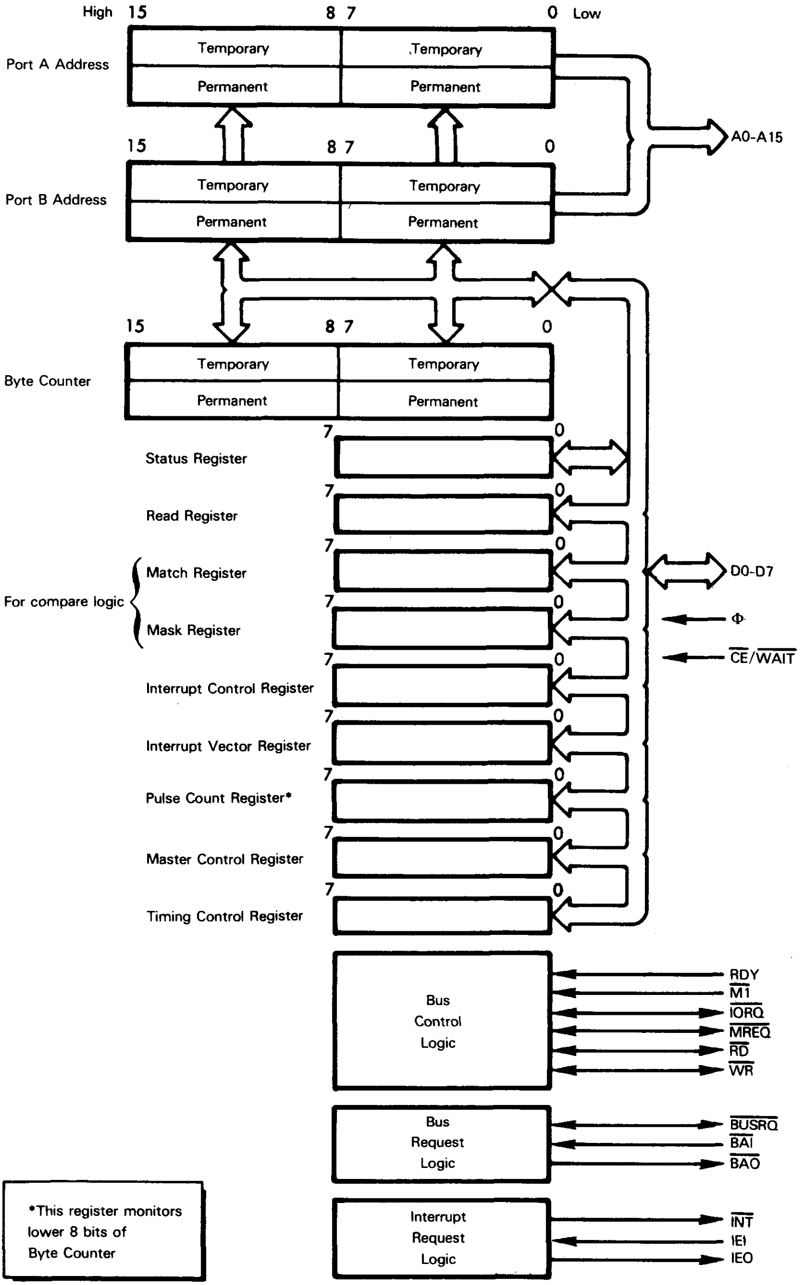


Figure 7-25. Functional Logic Of The Z80 DMA Device

Z80 DMA REGISTERS, TYPES OF TRANSFER AND MODES OF OPERATION

Z80 DMA device logic is illustrated functionally in Figure 7-25.

Z80 DMA PORT A ADDRESS REGISTERS

Z80 DMA PORT B ADDRESS REGISTERS

Z80 DMA BYTE COUNT REGISTERS

During any DMA operation the Z80 DMA device uses the Port A and Port B Address registers to identify the data source and the data destination. The Byte Counter registers define the initial data block length. The Port A Address register, the Port B Address register and the Byte Counter register are all duplicated. Data loaded into any one of these registers has a permanent storage location, plus a temporary storage location. Data is preserved in the permanent storage location while it is being incremented or decremented in the temporary storage location. This allows you subsequently to restore addresses and byte counts to their original values so that a DMA operation may be restarted or executed continuously.

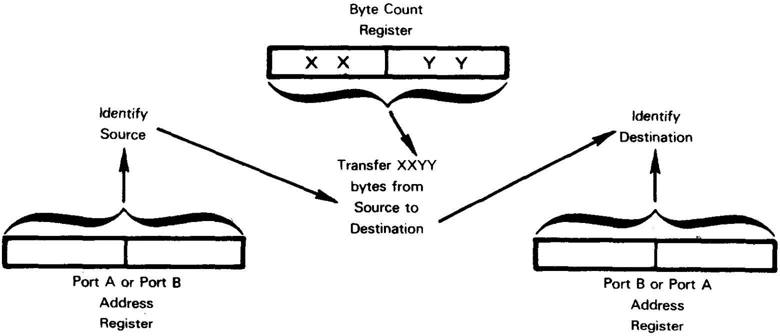
Initially the Byte Counter temporary registers are set to 0; they are then incremented until they match the block length, as defined by the Byte Counter permanent registers. Meanwhile, whenever the low order Byte Counter temporary register contents equal the Pulse Counter register contents, a pulse is output, if so programmed. Then the original DMA operation restarts. You can also issue a discrete Restart command after a DMA operation has stopped; when the Restart command is executed, permanent register contents are moved to the temporary registers and the DMA operation begins again according to prevailing programmed options.

Three types of DMA operations may be performed, as follows:

Z80 DMA TYPES OF TRANSFER

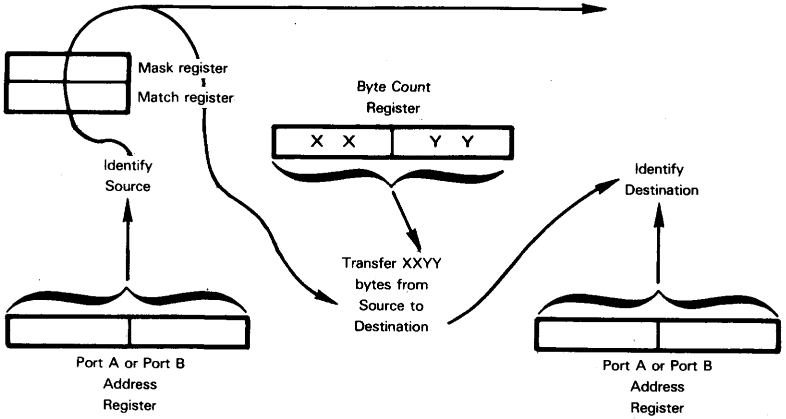
TYPE 1 — Transfer a Block of Data.

A block of data whose length is defined initially by the Byte Count register, is transferred from a source to a destination:



TYPE 2 — Transfer and Detect Match Bytes

A block of data is transferred from a source to a destination, as described for operation Type 1 above, but match bytes are detected during the data transfer. A status flag, an interrupt request or an end of transmission may occur when the match byte is found:



Z80 DMA MATCH AND MASK REGISTERS

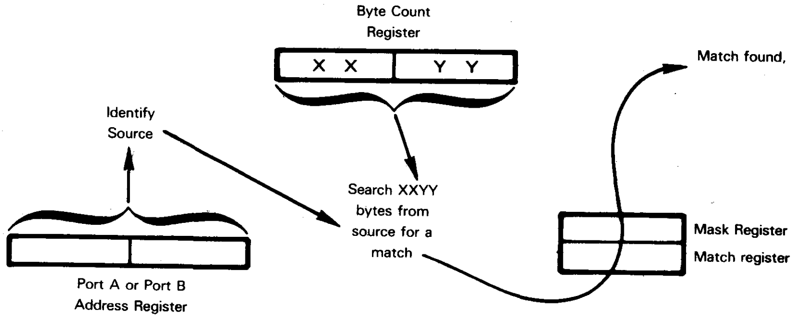
The match byte has a bit pattern which you define, under program control, using the Match and Mask registers.

The Match register contents will be compared, bit by bit, with each data byte arriving from the source. The results of the Match operation are passed via the Mask register: only these bit positions identified by a 0 in the Mask register contribute to a match. Providing all contributing bits are equivalent, a match will be found. This may be illustrated as follows:

11110111 ←	Data Byte	→ 01010111
11010110 ←	Match Byte	→ 11010110
00100001 ←	Data XOR Match	→ 10000001
10000001 ←	Mask Byte	→ 10000001
X010000X ←	Result	→ X000000X
No Match		Match

TYPE 3 — Search for Match Byte

Data transmitted from the source is searched for a match byte:



No data is transmitted to the destination.

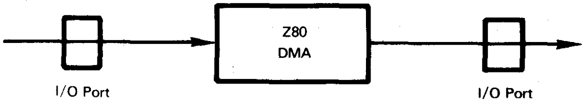
The Byte Counter register is 16 bits wide; therefore up to 65,536 bytes may be handled within a single DMA operation. **The Byte Counter register must have the initial DMA block length loaded into it.** If data is transferred up to a match byte, or a match byte is being sought, (operation Types 2 or 3 above) then when the DMA operation ceases the Byte Counter register will identify the number of bytes up to the match byte in the DMA block.

The source and the destination for a DMA data transfer may each either be an I/O port or a memory location. **The Port A and Port B Address registers within the Z80 DMA device identify the source and the destination.** Under program control you specify whether Port A identifies the source and Port B identifies the destination, or vice versa.

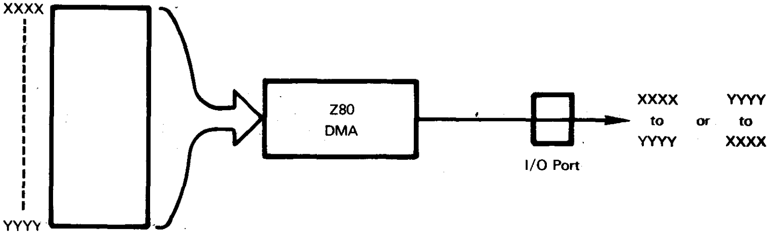
Port A and Port B Address registers are each 16 bits wide: thus, up to 65,536 bytes of memory may be addressed.

An I/O port is addressed by the low order byte of Port A and/or Port B. The Z80 DMA device discriminates betweeen I/O port and memory addresses by outputting appropriate bus control signals — $\overline{\text{MREQ}}$ with a memory address and $\overline{\text{IORQ}}$ with an I/O port address.

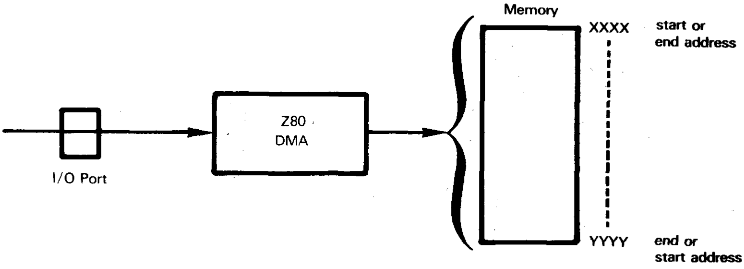
The initial addresses loaded into the Port A Address registers and the Port B Address registers may be incremented, decremented, or left unaltered. Separate specifications may be made for the Port A Address registers as against the Port B Address registers. If increment or decrement logic is specified, then the address is incremented or decremented after each byte access. Thus, you can transfer data between two fixed locations such as I/O ports:



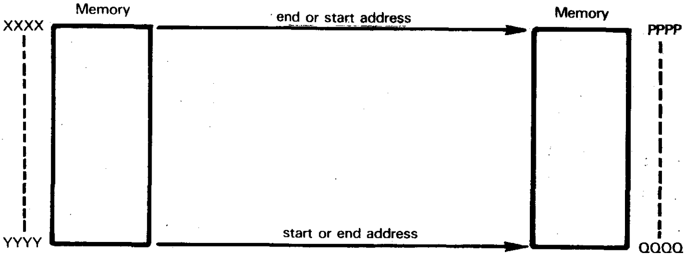
Data may be transferred from memory to an I/O port:

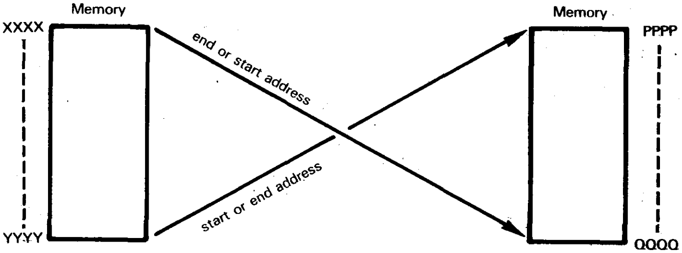


Or you may transfer data from an I/O port to memory:



When memory is being accessed, the memory area may be addressed beginning at the high address or at the low address, as illustrated above. Here are some memory to memory options:





There is nothing to stop you addressing an I/O port via the Port A Address registers, or the Port B Address registers, and incrementing or decrementing the I/O port address. This will result in numerically sequential I/O port addresses being accessed as the DMA operation proceeds. Because of the small I/O port address space typically available, (in the case of the Z80, 256 I/O port addresses are allowed) incrementing or decrementing I/O port addresses makes little sense and will rarely be used. Thus, in the standard case, memory addresses will be either incremented or decremented, while I/O port addresses are left constant.

The various types of DMA transfer we have described and the address manipulations allowed define the data source, and the data destination, plus the manner in which data is handled during a DMA access. There are, in addition, four operating modes available, where modes define the way in which the Z80 DMA and CPU devices compete for bus access. These are the four allowed modes:

1) **Single byte mode.** The Z80 DMA device requests bus access, and upon receiving it processes a single byte of data, then returns bus control to the CPU.

2) **Burst Mode.** Once the Z80 DMA device gains control of the bus, it keeps control for as long as data is continuously ready to be processed. The Z80 DMA device returns bus control to the CPU when no data is ready to be processed.

3) **Continuous mode.** Once the Z80 DMA device gains control of the bus, it keeps control until the entire DMA operation has been completed, as per the type of DMA operation specified, but proceeds with the operation only as directed by RDY.

4) **Transparent mode.** All DMA transfers occur during the second half of instruction fetch machine cycles, within the time allotted to memory refresh logic. Some complicated timing is associated with the use of Transparent mode.

Irrespective of the DMA mode you employ, you must remember that Dynamic Memory Refresh logic of the Z80 CPU is inactive while the Z80 DMA device has

control of the System Bus. This is no problem in single byte mode, since the bus is floated for very short time intervals. In Burst and Continuous mode you must make sure that data blocks being transferred are short enough for bus control to return to the CPU within the time interval allowed by Dynamic Memory Refresh logic.

It is theoretically possible to operate the Z80 DMA device in Transparent mode while simultaneously using Z80 CPU logic to refresh dynamic memory; however, timing problems associated with this dual use of the instruction fetch machine cycle's refresh time may become complicated.

You can perform all three types of DMA operations — "Transfer a block of data", "Transfer and look for match found" or "Search for match byte" using any one of the four DMA modes.

Z80 DMA INTERRUPT LOGIC

Z80 DMA INTERRUPTS

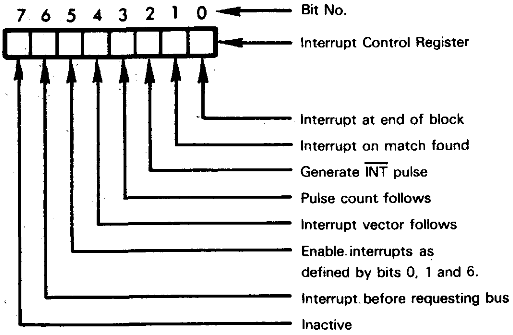
You have the option of controlling DMA operations with interrupt requests. Also you have the option of transmitting control pulses to external logic via the interrupt request line while the bus is being floated — and the CPU will therefore not detect interrupt requests.

Z80 DMA INTERRUPT CONTROL REGISTER

Z80 DMA INTERRUPT VECTOR REGISTER

The Interrupt Control register must be loaded with a Control code that defines the conditions under which an interrupt request will occur. The Interrupt Vector register holds the address byte which will be transmitted to the CPU in response to an interrupt acknowledge. It is assumed that the CPU is operating its interrupt logic in Mode 2.

The Interrupt Control code is interpreted as follows:


7 6 5 4 3 2 1 0
Bit No.
Interrupt Control Register
Interrupt at end of block
Interrupt on match found
Generate $\overline{\text{INT}}$ pulse
Pulse count follows
Interrupt vector follows
Enable interrupts as defined by bits 0, 1 and 6.
Interrupt before requesting bus
Inactive

In the Control code illustrated above, a 1 in any bit position selects the option.

Two forms of activity are being specified by the Interrupt Control code illustrated above: interrupt requests to the CPU and control pulses to external logic.

Let us first look at specifications for interrupt requests to the CPU.

Providing overall device interrupt logic has been enabled, then interrupt control bits 0, 1 and 6 determine the particular conditions which can generate an interrupt request.

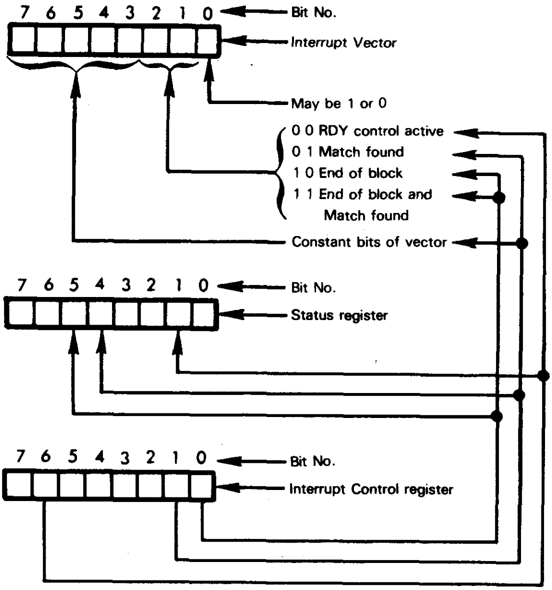
That is to say, providing interrupts have been enabled, an interrupt request will occur at one or more of the following times:

1) Prior to the Z80 DMA device requesting bus access, after RDY has gone active.
2) When the end of a DMA block has been reached, that is to say, the Byte Counter registers match the Block Length register.
3) When a match is found during a "transfer and identify match byte" operation, or during a "search and identify match byte" operation.

When a Z80 DMA device interrupt request is acknowledged, an address vector byte is placed on the Data Bus — assuming that the CPU is operating its interrupt logic in CPU interrupt Mode 2. The interrupt vector byte comes from the Interrupt Vector register. You must load this interrupt vector byte into the Interrupt Vector register. You do this by outputting an Interrupt Control code with a 1 in bit 4. This causes the next data byte output to be written into the Interrupt Vector register, if bit 3 of the Control code is 0. If bit 3 is 1, then the next data byte is a pulse count, and the interrupt vector is the second byte written to the device after the Interrupt Control code.

We will for the moment quickly pass over Z80 DMA register addressing logic since this is a subject covered in detail later on.

Now the interrupt vector which is returned to the CPU following an interrupt acknowledge may optionally have vector bits 1 and 2 modified in order to identify the reason for the interrupt request. If bit 5 of the Interrupt Control code is 1, then bits 1 and 2 of the interrupt vector are modified as follows:



Note carefully that the interrupt vector will have bits 1 and 2 modified only if you have made this selection under program control, and then only conditions which you have programmed to request an interrupt can modify the interrupt vector. For example, suppose you have not selected interrupt requests when a match byte is detected; then you cannot have the "match byte detect" condition modify the interrupt vector.

The fact that we have not yet examined the Status register in detail should not bother you, since you will now be quite familiar with Status registers in general — individual bits of the Status register are set or reset to identify conditions internal to the Z80 DMA device; and that is how the Status register is being used in the illustration above.

Let us look at the various interactions of the Status register, Interrupt Vector register, and Interrupt Control register.

When the RDY control input from external logic is true, Status register bit 1 will be set. If the RDY line goes true while the CPU is bus master, then the Z80 DMA device will request bus access by outputting $\overline{\text{BUSRQ}}$ low. If the Interrupt Control register bit 6 is 1, then as soon as RDY goes high while the CPU is bus master, the Z80 DMA device will request an interrupt by outputting a low signal via $\overline{\text{INT}}$; at this time it will not issue a $\overline{\text{BUSRQ}}$. When the interrupt is acknowledged, the Z80 DMA device will respond by outputting to the Data Bus the contents of the Interrupt Vector register.

Now there is an obvious problem to preceding a bus request with an interrupt request: the bus request, coming immediately after the interrupt request, will prevent the CPU from acknowledging the interrupt request. Z80 DMA device logic solves this problem by allowing you to specify under program control that Z80 DMA logic will remain disabled until the Z80 DMA device detects an RETI instruction object code, or it receives a re-enabling Control code from the CPU.

The "End of Block" and the "Match Found" conditions are much more straightforward. When an End of Block is reached, Status register bit 5 is set. When a match is found during a "transfer and search for match byte" or a "search for match byte" operation, Status register bit 4 is set. Status register bit settings occur whether or not associated interrupt logic is active. "Match found" interrupt logic is enabled by a 1 in Interrupt Control register bit 1. "End of Block" interrupt requests are enabled by a 1 in bit 0 of the Interrupt Control register. When the interrupt is acknowledged, the Z80 DMA device responds by outputting on the Data Bus the contents of the Interrupt Vector register.

Let us now look at Interrupt Control code specifications covering pulses output to external logic via the $\overline{\text{INT}}$ line. These pulses are intended for external logic associated with the data source or destination; they occur while the Z80 DMA device is bus master, therefore, the CPU cannot respond by acknowledging an interrupt.

Z80 DMA PULSE COUNT REGISTER

You specify that $\overline{\text{INT}}$ pulses will be created by outputting an Interrupt Control code with 1 in bit 2. The $\overline{\text{INT}}$ line will now pulse low **after a fixed number of bytes have been processed via direct memory accesses. This "pulse count" is determined by the contents of the Pulse Count register.** The Pulse Count register is an 8-bit register, which means that anywhere between 1 and 256 DMA accesses may separate low pulses on the $\overline{\text{INT}}$ line. The $\overline{\text{INT}}$ low pulse is one machine cycle long.

You write a pulse count to the Pulse Count register by first outputting an Interrupt Control code with 1 in bit 3. This specifies that the next byte written to the Z80 DMA device is a pulse count, to be stored in the Pulse Count register.

If an Interrupt Control code is output with bits 3 and 4 both 1, then the next two bytes written to the Z80 DMA device will be interpreted as a pulse count, and then an interrupt vector, in that order. This may be illustrated as follows:

```
LD      A,3FH      ;OUTPUT AN INTERRUPT CONTROL CODE TO THE
OUT     (DMA),A    ;Z80 DMA DEVICE
LD      A,PCNT     ;OUTPUT A PULSE COUNT
OUT     (DMA),A
LD      A,IVEC     ;OUTPUT AN INTERRUPT VECTOR
OUT     (DMA),A
```

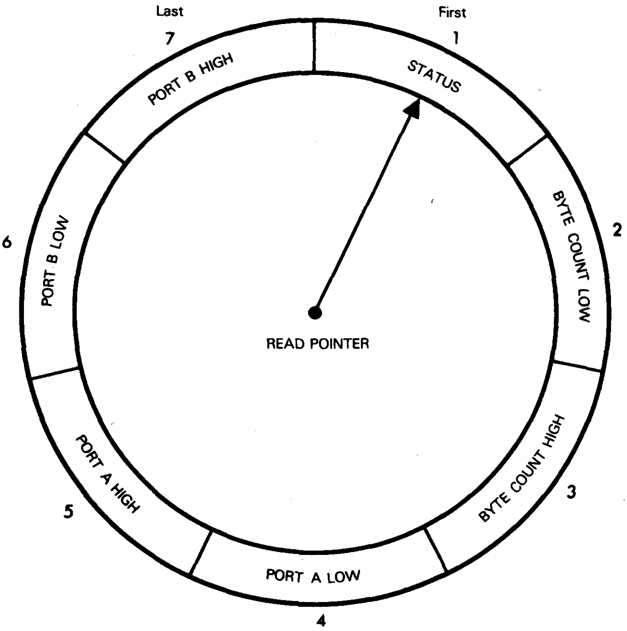
Z80 DMA REGISTER ADDRESSING

In the instruction sequence illustrated above, the same I/O port address, represented by the symbol DMA, is used to access the Interrupt Control register, the Pulse Count register, and the Interrupt Vector register. In fact, you will use the same I/O port address to read the contents of any readable Z80 DMA register or write to any Z80 DMA register.

Let us examine the way in which Z80 DMA logic allows you to access such a large number of registers using a single I/O port or memory address.

You can write into any of the registers illustrated in Figure 7-25. You use the same I/O port or memory address to access all of the registers; however, you use appropriate Control codes to select the registers into which you will write. Control codes are described later in this chapter. Following an explanation of Control codes, **Figure 7-26 summarizes the manner in which Control codes define register accesses.**

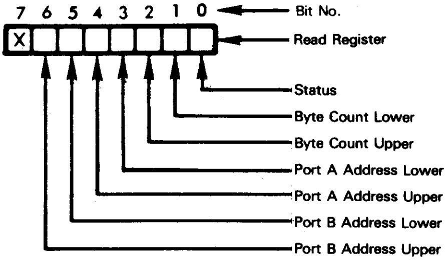
You can only read the contents of seven Z80 DMA registers. There are: Port A Address High and Low, Port B Address High and Low, Byte Counter High and Low and the Status register. The single Z80 DMA I/O port or memory address applies to all seven readable registers. Read or input instructions access the seven registers sequentially as follows:



There are a number of conditions which will reset the read pointer to the first readable register. Following such a reset, the first read or input instruction accessing the Z80 DMA device will access the first readable register. The next read or input instruction accessing the Z80 DMA device will read the contents of the Byte Counter Low register — and so the sequence will continue down to the Port B High register.

If you do not wish to read the contents of all seven readable registers, then you can specify those registers whose contents you wish to read by writing an appropriate Control code to the Read register. Read register bits are assigned to the seven readable locations as follows:

Z80 DMA READ REGISTER



A 1 in any bit position allows the associated register contents to be read; a 0 in any bit position will cause the associated register to be skipped during a read sequence.

Suppose, for example, the Read register contents are:

00011001

Sequential reads would access Z80 DMA registers in the following order:

Status
Port A low
Port A high

Z80 DMA TIMING

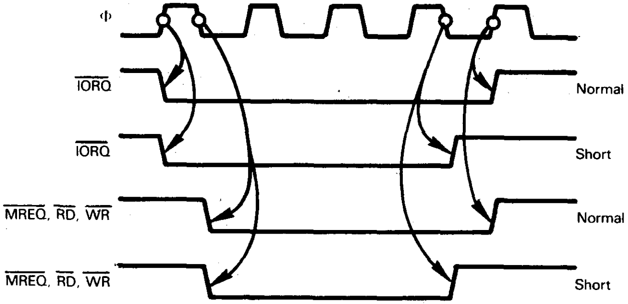
Let us now look at timing associated with DMA operations. When the CPU accesses the Z80 DMA device, timing conforms to standard CPU requirements and needs little further discussion.

Exact Z80 DMA timing requirements are given in the data sheets at the end of this chapter.

When a byte of data is actually transferred via Direct Memory Access, the Z80 DMA device executes a Read or input machine cycle, followed by a Write or output machine cycle. Timing for these machine cycles is given in Figures 7-6, 7-7, 7-9 and 7-10.

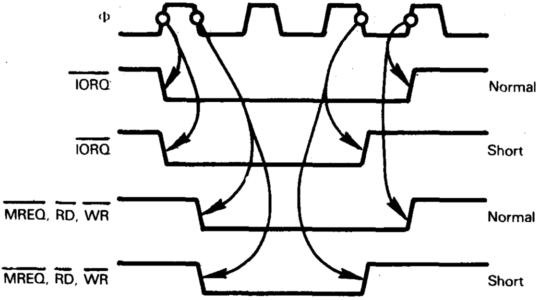
But the Z80 DMA device does not limit itself to operating within a Z80 system strictly in accordance with Z80 CPU timing. Under program control you can select anywhere from one to four clock periods as the machine cycle width for Z80 DMA Read, Write input or output machine cycles. You also have the option of terminating the control signal half a clock period earlier than specified by the Z80 CPU. These options may be illustrated as follows:

a) Four clock period machine cycles.



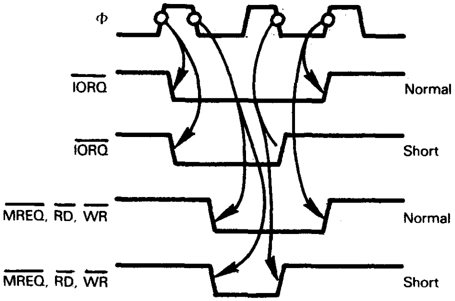
Φ
$\overline{\text{IORQ}}$
Normal
$\overline{\text{IORQ}}$
Short
$\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$
Normal
$\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$
Short

b) Three clock period machine cycles.



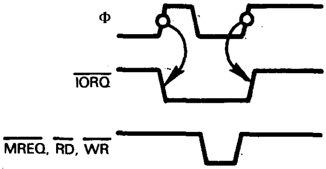
Φ
$\overline{\text{IORQ}}$
Normal
$\overline{\text{IORQ}}$
Short
$\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$
Normal
$\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$
Short

c) Two clock period machine cycles.



Φ
$\overline{\text{IORQ}}$
Normal
$\overline{\text{IORQ}}$
Short
$\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$
Normal
$\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$
Short

d) One clock period machine cycle.



Φ
$\overline{\text{IORQ}}$
$\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$

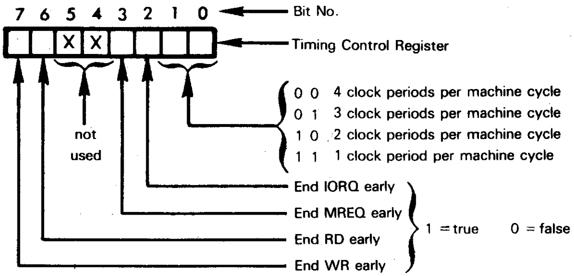
The two clock period machine cycle can be used with any of the control signals. The two clock period machine cycle is generally used in Transparent mode. Transparent mode performs DMA operations during the second half of an Instruction Fetch machine cycle, which is otherwise set aside for dynamic memory refresh logic. The second half of an Instruction Fetch machine cycle is two clock periods long, therefore the two clock period machine cycle option should be used when the Z80 DMA device is operating in Transparent mode.

The main concern here is to ensure that the two cycles of refresh are utilized fully; i.e., with a 1 cycle read and a 1 cycle write a full byte could be transferred in 1 refresh or 2 bytes read. With a 2 cycle read/write the write will take place on a subsequent refresh. However, a 3 cycle or 4 cycle operation may not be used nor may a 1 cycle operation be used in combination with a two cycle operation.

If Z80 DMA machine cycles are three clock periods long, or four clock periods long, then you can insert Wait states after clock period 2 when accessing memory, or 3 when accessing an I/O device. Z80 DMA and CPU Wait state logic and timing are identical. WAIT for a memory access machine cycle is sampled on the falling edge of T2; thus WAIT may be used for memory accesses of 2, 3, or 4 cycles. For an I/O access machine cycle, WAIT is sampled on the negative edge of T3; thus, it may be used only with I/O accesses of 3 or 4 cycles. In any case, WAIT is sampled only if so programmed. The Z80 DMA device uses its Chip Enable pin to receive a "Wait" request while DMA logic is active, if programmed. At that time chip enable logic is inactive and therefore meaningless.

Z80 DMA TIMING CONTROL REGISTER

The timing options you select are determined by a Control code written into the Timing Control register whose contents are interpreted as follows:

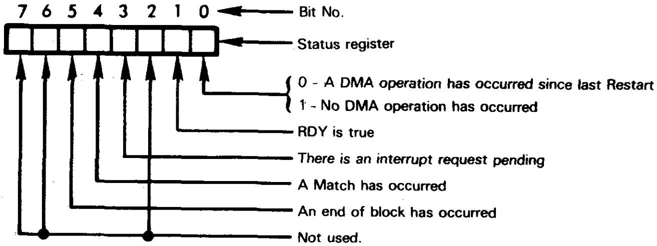

7 6 5 4 3 2 1 0
Bit No.
X X
Timing Control Register
not used
0 0 4 clock periods per machine cycle
0 1 3 clock periods per machine cycle
1 0 2 clock periods per machine cycle
1 1 1 clock period per machine cycle
End IORQ early
End MREQ early
End RD early
End WR early
1 = true 0 = false

Now in addition to the control signal timing variations described above, you can also use the WAIT input to insert Wait clock periods between T2 and T3. Thus, you can expand the duration of a machine cycle, and therefore the control signal active pulse, beyond four clock periods. For Wait state timing, see Figure 7-8 and associated text.

If you output a timing Control code to the Timing Control register, you will modify the Read or input and the Write or output machine cycles executed by the Z80 DMA device during any DMA operation. There are, however, specific Control codes which allow you to restore timing associated with one of the two I/O ports to standard timing. These are type 2D Control codes, described later in this Chapter. Using these Control codes in conjunction with the Timing Control register illustrated above, you have the options of operating the input or output port of the Z80 DMA device with non-standard timing while the other port operates with standard timing.

Z80 DMA STATUS REGISTER

The Z80 DMA device has a Status register whose contents are interpreted as follows:


7 6 5 4 3 2 1 0
Bit No.
Status register
0 - A DMA operation has occurred since last Restart
1 - No DMA operation has occurred
RDY is true
There is an interrupt request pending
A Match has occurred
An end of block has occurred
Not used.

A 0 in any Status register bit position represents a "true" condition. A 1 represents a "false" condition.

You cannot write into the Status register; this is a read only location and, as we discussed earlier, it is the register selected by read logic whenever the Z80 CPU device is reset.

Only bits 4 and 5 of the Status register are reset to 0 following execution of a Reset Status command (type 2D).

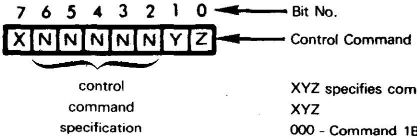
In our previous discussion of the Interrupt Vector register and the Interrupt Control register, we saw how bits 1, 4, and 5 interact with interrupt logic.

Z80 DMA CONTROL COMMANDS

The main programmable options of the Z80 DMA device are selected by writing a sequence of Control Commands into the Master Control register. Whenever you write data to a Z80 DMA device, the data will be interpreted as a Control Command, to be loaded into the Master Control register, unless a prior Control Command has specified alternative data to follow. In this case, the next data byte, or bytes written to the Z80 DMA device will be stored in the selected register location, or locations, following which data will again be written into the Master Control register.

Z80 DMA MASTER CONTROL REGISTER

Now the Command codes written to the Master Control register have to specify a bewildering profusion of operating and addressing options. This is accomplished by setting aside some Control Command bits to identify how others will be interpreted. Thus **six different Control Command interpretations are possible. This may be illustrated as follows:**

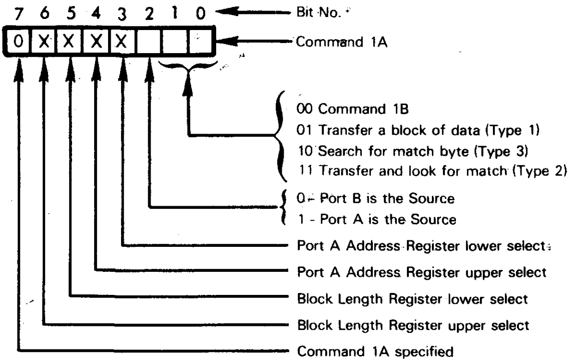


XYZ specifies command bytes as follows:

XYZ
000 - Command 1B
001 - Command 1A
010 - Command 1A
011 - Command 1A
100 - Command 2A
101 - Command 2B
110 - Command 2C
111 - Command 2D

Let us first look at Command 1A. It may be illustrated as follows:

Z80 DMA COMMAND 1A


7 6 5 4 3 2 1 0
Bit No.
0 X X X X
Command 1A
00 Command 1B
01 Transfer a block of data (Type 1)
10 Search for match byte (Type 3)
11 Transfer and look for match (Type 2)
0 - Port B is the Source
1 - Port A is the Source
Port A Address Register lower select
Port A Address Register upper select
Block Length Register lower select
Block Length Register upper select
Command 1A specified

Command 1A is one of the overall operation definition commands.

Bits 0 and 1 identify which of the three types of DMA operation is to occur. The three types of DMA operation have been described earlier in the chapter.

Bit 2 determines whether Port A is the input port and Port B the output port, or vice versa.

Bits 3 through 6 specify data bytes that may follow. If X is 1 in any bit position, then the subsequent data byte or bytes will be interpreted as data going to the selected location. Since there are four register select bits, you can have up to four data bytes following a type 1A Command code. When more than one data byte is specified, the sequence is:

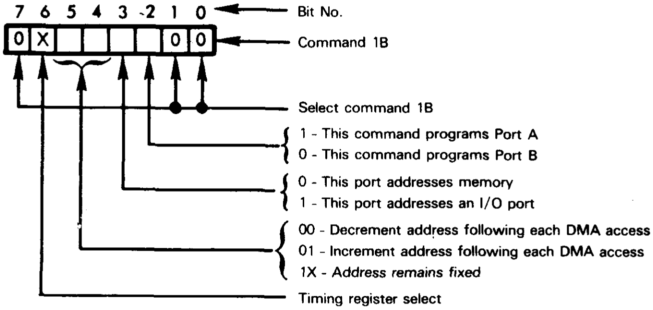
Port A lower
Port A upper
Block Length lower
Block Length upper

Here is an example of data following Command code 1A:

```
        LD      A,7DH           ;OUTPUT CONTROL CODE 1A TO Z80 DMA.
        OUT     (DMA),A         ;FOUR DATA VALUES FOLLOW.
        LD      A,ADLO          ;OUTPUT PORT A ADDRESS, LOW ORDER BYTE
        OUT     (DMA),A
        LD      A,ADHI          ;OUTPUT PORT A ADDRESS, HIGH ORDER BYTE
        OUT     (DMA),A
        LD      A,BLKLO         ;OUTPUT BLOCK LENGTH, LOW ORDER BYTE
        OUT     (DMA),A
        LD      A,BLKHI         ;OUTPUT BLOCK LENGTH, HIGH ORDER BYTE
        OUT     (DMA),A
```

Command 1B is the next command providing basic set up *information* for the Z80 DMA device. It is interpreted as follows:

Z80 DMA COMMAND 1B

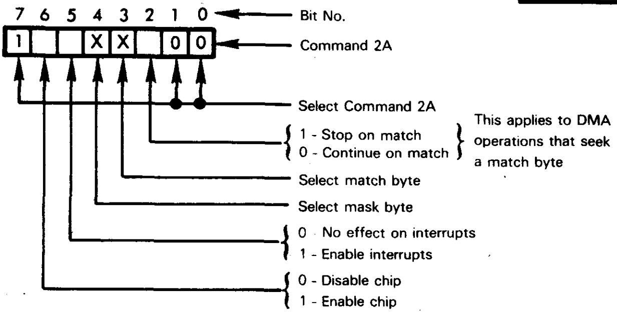

7 6 5 4 3 2 1 0
Bit No.
0 X 0 0
Command 1B
Select command 1B
1 - This command programs Port A
0 - This command programs Port B
0 - This port addresses memory
1 - This port addresses an I/O port
00 - Decrement address following each DMA access
01 - Increment address following each DMA access
1X - Address remains fixed
Timing register select

Command 1B will be output twice: for Port A (with bit 2 = 1) and for Port B (with bit 2 = 0). Via bits 3,4 and 5 you indicate whether an I/O port or block of memory is being addressed by the selected port. If a block memory is being addressed, you further specify whether the address is to be incremented, decremented or left alone following each DMA access.

If you are using non-standard DMA timing, then at least one of the two type 1B commands will have a 1 in bit 6, specifying that the Timing Control code will be the next data output to the Z80 DMA device. The Timing Control code has already been described.

Command 2A is interpreted as follows:

Z80 DMA COMMAND 2A



Command 2A or one of the variations of Command 2D must be the last command output to the Z80 DMA device before it starts executing the required DMA operations. Command 2A enables the Z80 DMA chip via a 1 in bit 6. Until the Z80 DMA device has been enabled its DMA logic is inactive, and you are limited to executing memory or I/O access instructions that read from Z80 DMA registers, or write to Z80 DMA registers.

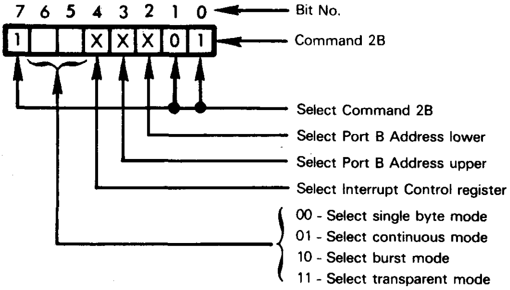
Bits 2, 3 and 4 of Command 2A apply to match logic.

Bits 3 and 4 are used to output data to the Match register, and/or to the Mask register. A 1 in bit 3 specifies that the next byte output will be written to the Mask register. A 1 in bit 4 specifies that the next byte written must be stored in the Match register. If there are 1's in both bit positions 3 and 4, the next two bytes written out will be loaded into the Mask register and then the Match register.

During a "Transfer-and-search" or a "Search-for-match-byte" operation, bit 2 of Command 2A determines whether the DMA operation will stop when the match occurs, or whether the DMA operation will continue, simply identifying the match, using whatever identification technique you have selected.

Z80 DMA COMMAND 2B

Command 2B is interpreted as follows:



7 6 5 4 3 2 1 0
Bit No.
1 X X X 0 1
Command 2B
Select Command 2B
Select Port B Address lower
Select Port B Address upper
Select Interrupt Control register
00 - Select single byte mode
01 - Select continuous mode
10 - Select burst mode
11 - Select transparent mode

Command 2B is used to specify the mode in which the Z80 DMA device will operate. Bits 5 and 6 select one of the four modes which we have already described.

Command code 2B, bits 2, 3 and 4 are used to load data into Port B Address Lower, Port B Address Upper and the Interrupt Control register. As with the previous Control codes, Control 2B may be followed by 0, 1, 2 or 3 data bytes, depending upon the contents of bits 2, 3 and 4. If more than one data byte follows, then the sequence is:

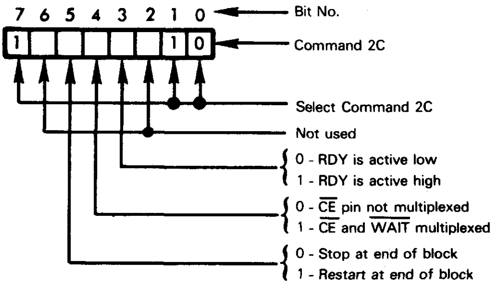
Port A Address Lower

Port B Address Upper

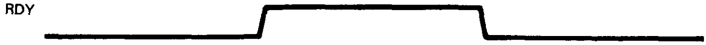
Interrupt Control

Z80 DMA COMMAND 2C

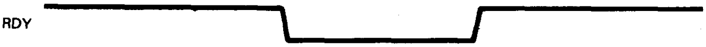
Command 2C specifies three operating options of the Z80 DMA device; it is interpreted as follows:



Via bit 3 you specify whether the RDY control input from external logic will be active high:



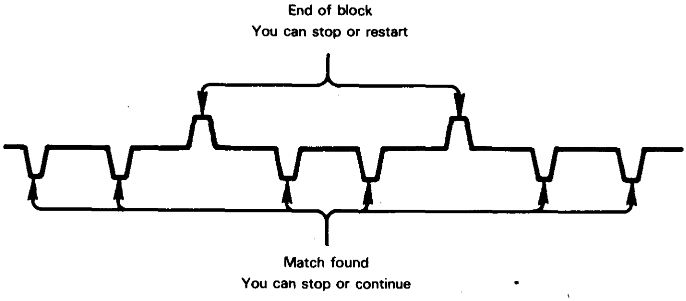
Or whether RDY will be active low:



If bit 4 is 1, then while DMA operations are in progress, pin 16 is used to receive a $\overline{\text{WAIT}}$ input. This allows external logic to extend the Read and Write machine cycles created by the Z80 DMA device, as described for the Z80 CPU earlier in this Chapter.

Bit 5 determines whether or not the DMA operation will be continuous. If bit 5 is 1, then as soon as the Block Length register matches the Byte Counter, initial values for the DMA operation are restored and the DMA operation begins again.

If bit 5 is 0, however, the DMA operation ceases when the Byte Counter register goes to zero. Observe that there is an interaction between the automatic restart provided by Command 2C above and the Stop On Compare condition specified by Command 2A. The Stop On Compare condition applies to a match while the automatic restart applies to an end of block. This may be illustrated as follows:

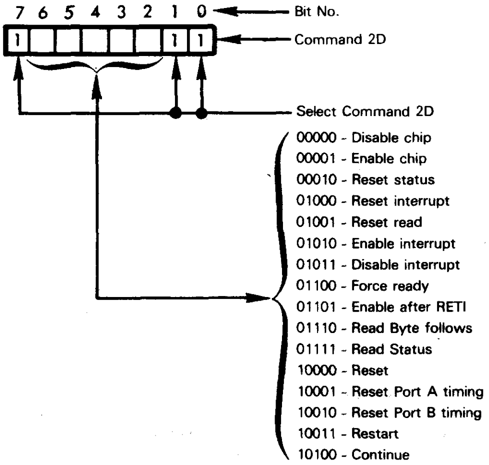


In the illustration above, notice that following an "end of block" we say that you can stop or "restart", whereas when a "match" is found we say that you can stop or "continue". **What is the significance of "continue" versus "restart"?** The answer is that following an "end of block", when you "restart", you will reload the temporary Address and Block Counter registers from their permanent equivalents. When you "continue", you simply carry on the DMA operation with the prior temporary Address and Byte Counter register contents.

The "restart" and "continue" options illustrated above are automatic, which means that there is no discernible pause between a block or match found terminating one DMA sequence and the next DMA sequence starting. If you select the "stop" option, however, you can issue specific Restart or Continue commands which reload temporary Address and/or Byte Counter registers from their permanent equivalents before restarting the DMA operation. These are type 2D commands described below.

Z80 DMA COMMAND 2D

Command 2D is used to control events occurring at the Z80 DMA device while it is enabled, or while DMA logic is being executed. Command 2D allows you to specify a number of single controls as follows:



00000 - Disable chip
00001 - Enable chip
00010 - Reset status
01000 - Reset interrupt
01001 - Reset read
01010 - Enable interrupt
01011 - Disable interrupt
01100 - Force ready
01101 - Enable after RETI
01110 - Read Byte follows
01111 - Read Status
10000 - Reset
10001 - Reset Port A timing
10010 - Reset Port B timing
10011 - Restart
10100 - Continue

The Disable Chip command disables the Z80 DMA device's DMA logic, but it does not reset any register or logic condition. Following a disable chip instruction, an enable chip instruction must be issued in order to continue operations.

The Enable Chip command duplicates a 1 in bit 6 of Command code 2A. This instruction allows the Z80 DMA device to execute DMA operations as specified by current selected options.

The Reset Status command resets bits 4 and 5 of the Status register to 0:

The Reset Interrupt command has the same effect on Z80 DMA logic as an RETI instruction being executed. For example, if interrupts precede each bus request, then following the interrupt request, the bus request can occur either after an RETI instruction has been executed, or after a Reset Interrupt command has been output as a Control code.

The Reset Read command will cause the next CPU Read of the Z80 DMA to access the first readable register, as defined by the Read Byte register. That is, the Reset Read command resets the read pointer, illustrated earlier.

The Enable Interrupt and Disable Interrupt commands are equivalent to Command 2A bit 5. These commands either enable or disable all interrupt logic at the Z80 DMA device.

The Force Ready command is equivalent to the RDY signal being input true by external logic. It causes all events to occur just as they would had external logic input RDY true.

The Enable After RETI command forces the Z80 DMA device to postpone all subsequent bus requests until an RETI instruction has been executed. Typically you will output an "Enable After RETI" command upon entering a service routine which must not be slowed down by DMA operations. Observe that a "Reset Interrupt" command being output will also allow bus requests to occur again.

The Read Byte Follows command specifies that the next data byte written to the Z80 DMA device must be loaded into the Read register.

The Read Status command forces read logic to select the Status register. The "Read Status" command does not cause the Z80 DMA device to output the Status register contents — but the next time the CPU reads data from the Z80 DMA device it will get the Status register contents. Then the read pointer returns to its location before the Read Status command — rather than the next readable register after the Status register.

The Reset command resets the entire Z80 DMA device. The chip and its interrupt logic are all disabled.

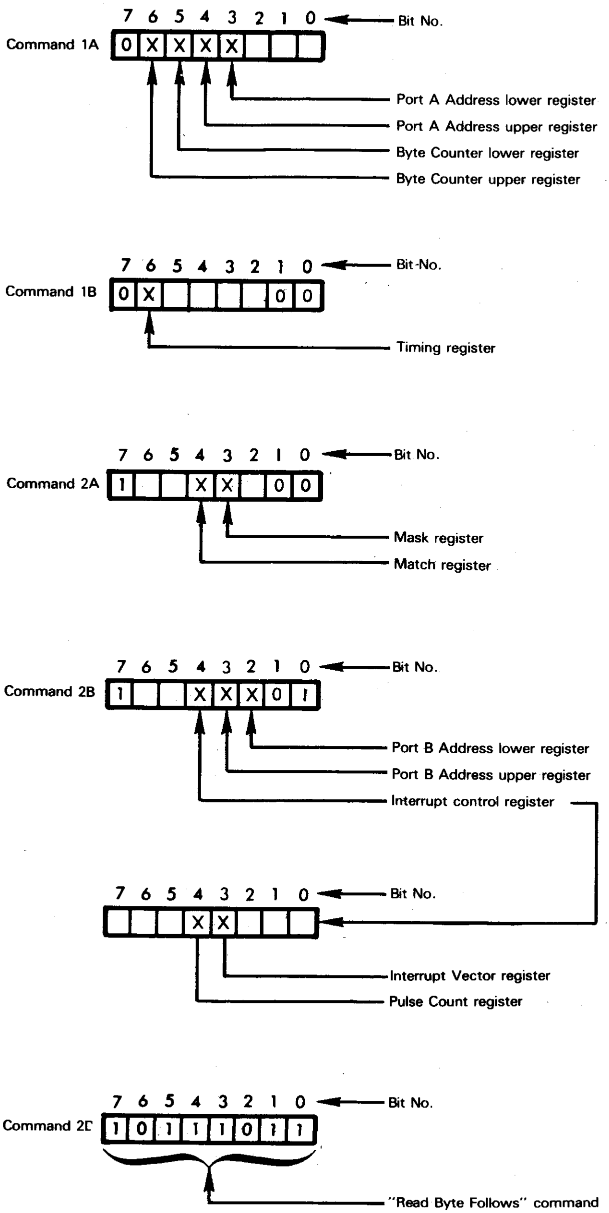
The Reset Port A timing and Reset Port B timing commands return timing to standard CPU specifications for the selected Z80 DMA I/O port. In other words, this negates specifications made by data output to the Timing Control register, but for one port only.

The Restart command reloads the Port A Address, Port B Address and Block Length registers with the values most recently output to these registers. The most recently specified DMA operation then is re-executed.

The Continue command reloads the Block Length registers with the most recently output values, then restarts the most recently specified DMA operation, leaving the Port A and Port B Address registers with their current values. In other words, you restart using addresses at the end of the previous DMA operation.

Z80 DMA REGISTER SELECT SUMMARY

Figure 7-26 summarizes the way in which Command code are used to write data into Z80 DMA registers.



Unless otherwise specified, the Master Control register is selected by a write operation addressing the Z80 DMA device.

Figure 7-26. A Summary Of Z80 DMA Writable Register Locations

ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +150°C
Voltage On Any Pin with Respect to Ground	-0.3V to +7V
Power Dissipation	1.1W

*Comment

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

- **D. C. CHARACTERISTICS**

T_A = 0°C to 70°C, V_{cc} = 5V ± 5% unless otherwise specified

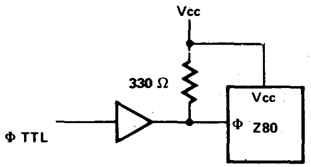
Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{cc} [1]		V_{cc}	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{cc}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL}=1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH}= -100μA
I_{CC}	Power Supply Current			200	mA	t_c=400nsec
I_{LI}	Input Leakage Current			10	μA	V_{IN}=0 to V_{cc}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT}=2.4 to V_{cc}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	V_{OUT}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \leq V_{IN} \leq V_{cc}$

- **CAPACITANCE**

T_A = 25°C, f = 1 MHz

Symbol	Parameter	Typ.	Max.	Unit	Test Condition
C_Φ	Clock Capacitance		20	pF	Unmeasured Pins Returned to Ground
C_{IN}	Input Capacitance		5	pF	
C_{OUT}	Output Capacitance		10	pF	

[1] Clock Driver



An external clock pull-up resistor of (330Ω) will meet both the A.C. and D.C. clock requirements.

A.C. Characteristics Z80-CPU

T_A = 0°C to 70°C, Vcc = +5V ± 5%, Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
Φ	t_c	Clock Period	.4	[12]	μsec	
	$t_w(\Phi H)$	Clock Pulse Width, Clock High	180	∞	nsec	
	$t_w(\Phi L)$	Clock Pulse Width, Clock Low	180	2000	nsec	
	$t_{r,f}$	Clock Rise and Fall Time		30	nsec	
A_{0-15}	$t_{D(AD)}$	Address Output Delay		160	nsec	C_L = 100pF
	$t_{F(AD)}$	Delay to Float		110	nsec	
	t_{acm}	Address Stable Prior to $\overline{MREQ}$ (Memory Cycle)	[1]		nsec	
	t_{aci}	Address Stable Prior to $\overline{IORQ}$, $\overline{RD}$ or $\overline{WR}$ (I/O Cycle)	[2]		nsec	
	t_{ca}	Address Stable from $\overline{RD}$ or $\overline{WR}$	[3]		nsec	
	t_{caf}	Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	[4]		nsec	
D_{0-7}	$t_{D(D)}$	Data Output Delay		260	nsec	C_L = 200pF
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		nsec	
	$t_{S\overline{\Phi}(D)}$	Data Setup Time to Falling Edge of Clock During M2 to M5	60		nsec	
	t_{dcm}	Data Stable Prior to $\overline{WR}$ (Memory Cycle)	[5]		nsec	
	t_{dci}	Data Stable Prior to $\overline{WR}$ (I/O Cycle)	[6]		nsec	
	t_{cdf}	Data Stable From $\overline{WR}$	[7]			
	t_H	Any Hold Time for Setup Time	0		nsec	
$\overline{MREQ}$	$t_{DL\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low		100	nsec	C_L = 50pF
	$t_{DH\Phi(MR)}$	$\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High		100	nsec	
	$t_{DH\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High		100	nsec	
	$t_{w(\overline{MRL})}$	Pulse Width, $\overline{MREQ}$ Low	[8]		nsec	
	$t_{w(\overline{MRH})}$	Pulse Width, $\overline{MREQ}$ High	[9]		nsec	
$\overline{IORQ}$	$t_{DL\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ Low		90	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ Low		110	nsec	
	$t_{DH\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ High		100	nsec	
	$t_{DH\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ High		110	nsec	
$\overline{RD}$	$t_{DL\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low		100	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low		130	nsec	
	$t_{DH\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High		100	nsec	
	$t_{DH\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		110	nsec	
$\overline{WR}$	$t_{DL\Phi(WR)}$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low		80	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ Low		90	nsec	
	$t_{DH\Phi(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High		100	nsec	
	$t_{w(\overline{WRL})}$	Pulse Width, $\overline{WR}$ Low	[10]		nsec	
$\overline{M1}$	$t_{DL(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ Low		130	nsec	C_L = 30pF
	$t_{DH(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ High		130	nsec	
$\overline{RFSH}$	$t_{DL(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ Low		180	nsec	C_L = 30pF
	$t_{DH(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ High		150	nsec	
$\overline{WAIT}$	$t_{s(WT)}$	$\overline{WAIT}$ Setup Time to Falling Edge of Clock	70		nsec	
$\overline{HALT}$	$t_{D(HT)}$	$\overline{HALT}$ Delay Time From Falling Edge of Clock		300	nsec	C_L = 50pF
$\overline{INT}$	$t_{s(IT)}$	$\overline{INT}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{NMI}$	$t_{w(\overline{NML})}$	Pulse Width, $\overline{NMI}$ Low	80		nsec	
$\overline{BUSRQ}$	$t_{s(BQ)}$	$\overline{BUSRQ}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{BUSAK}$	$t_{DL(BA)}$	$\overline{BUSAK}$ Delay From Rising Edge of Clock, $\overline{BUSAK}$ Low		120	nsec	C_L = 50pF
	$t_{DH(BA)}$	$\overline{BUSAK}$ Delay From Falling Edge of Clock, $\overline{BUSAK}$ High		110	nsec	
$\overline{RESET}$	$t_{s(RS)}$	$\overline{RESET}$ Setup Time to Rising Edge of Clock	90		nsec	
	$t_{F(C)}$	Delay to Float ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ and $\overline{WR}$)		100	nsec	
	t_{mr}	$\overline{M1}$ Stable Prior to $\overline{IORQ}$ (Interrupt Ack.)	[11]		nsec	

[12] $t_c = t_{w(\Phi H)} + t_{w(\Phi L)} + t_r + t_f$

[1] $t_{acm} = t_{w(\Phi H)} + t_f - 75$

[2] $t_{aci} = t_c - 80$

[3] $t_{ca} = t_{w(\Phi L)} + t_r - 40$

[4] $t_{caf} = t_{w(\Phi L)} + t_r - 60$

[5] $t_{dcm} = t_c - 180$

[6] $t_{dci} = t_{w(\Phi L)} + t_r - 180$

[7] $t_{cdf} = t_{w(\Phi L)} + t_r - 50$

[8] $t_{w(\overline{MRL})} = t_c - 40$

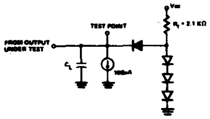
[9] $t_{w(\overline{MRH})} = t_{w(\Phi H)} + t_f - 30$

[10] $t_{w(WR)} = t_c - 40$

[11] $t_{mr} = 2t_c + t_{w(\Phi H)} + t_f - 80$

NOTES:

1. Data should be enabled onto the CPU data bus when $\overline{RD}$ is active. During interrupt acknowledge data should be enabled when $\overline{M1}$ and $\overline{IORQ}$ are both active.
2. All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.
3. The $\overline{RESET}$ signal must be active for a minimum of 3 clock cycles.
4. Output Delay vs. Loaded Capacitance
   TA = 70°C Vcc = +5V ±5%
   (1) ΔC_L = +100pF ($A_\emptyset - A_{15}$ and Control Signals), add 30 ns to timing shown.
   (2) ΔC_L = -50pF ($A_\emptyset - A_{15}$ and Control Signals), subtract 15 ns from timing shown.

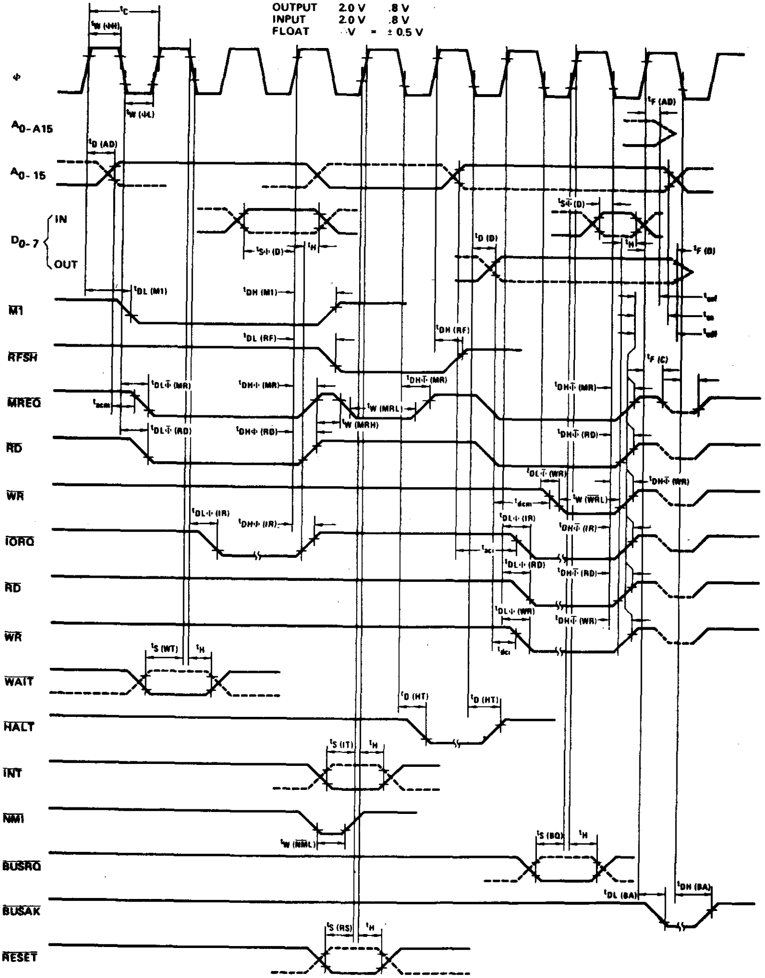


Load circuit for Output

TIMING WAVEFORMS

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	4.2 V	.8 V
OUTPUT	2.0 V	.8 V
INPUT	2.0 V	.8 V
FLOAT	ΔV	= ± 0.5 V



ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias	Specified operating range.
Storage Temperature	-65° C to +150° C
Voltage On Any Pin With Respect To Ground	-0.3 V to +7 V
Power Dissipation	.6 W

Note: All AC and DC characteristics remain the same for the military grade parts except I_{cc}.

I_{cc} = 130 mA. MAX

***Comment**

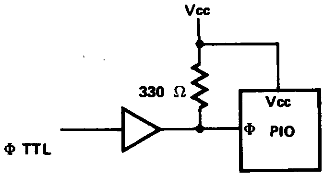
Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

TA = 0° C to 70° C, Vcc = 5 V ± 5% unless otherwise specified

Symbol	Parameter	Min.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3	0.45	V	
V_{IHC}	Clock Input High Voltage	Vcc-.2	Vcc	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{IH}	Input High Voltage	2.0	Vcc	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 1.8 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} -250 μA
I_{CC}	Power Supply Current		70	mA	T_C = 400 n sec
I_{LI}	Input Leakage Current		10	μA	V_{IN} = 0 to Vcc
I_{LOH}	Tri-State Output Leakage Current in Float		10	μA	V_{OUT} = 2.4 to Vcc
I_{LOL}	Tri-State Output Leakage Current in Float		-10	μA	V_{OUT} = 0.4 V
I_{LD}	Data Bus Leakage Current in Input Mode		±10	μA	$0 \leq V_{IN} \leq$ Vcc
I_{OHD}	Darlington Drive Current	1.5		mA	V_{OH} = 1.5 V R_{EXT} = 390 Ω Port B Only

CLOCK DRIVER



An external pull-up resistor of 330 Ω will meet all A.C. and D.C. clock requirements.

A.C. CHARACTERISTICS

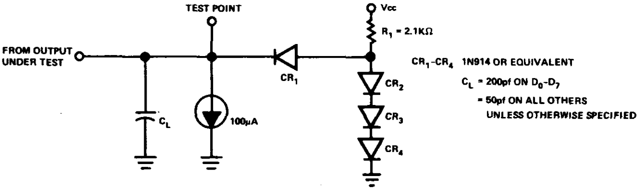
TA = 0° C to 70° C, Vcc = +5 V ± 5%, unless otherwise noted

Signal	Symbol	Parameter	Min.	Max.	Unit	Comments
Φ	t_c	Clock Period	.4	[1]	μsec	[1] $t_c = t_{W(\Phi H)} + t_{W(\Phi L)} + t_r + t_f$
	$t_{W(\Phi H)}$	Clock Pulse Width, Clock High	180	2000	nsec	
	$t_{W(\Phi L)}$	Clock Pulse Width, Clock Low	180	2000	nsec	
	t_r, t_f	Clock Rise and Fall Times		30	nsec	
D_0-D_7	$t_{HW(D)}$	Data Hold Time During Write Cycle	0		nsec	
	$t_{HR(D)}$	Data Hold Time From Rising Edge of $\overline{RD}$ During M1 Cycle	0		nsec	
	$t_{DR(D)}$	Data Output Delay During Read Cycle		430	nsec	
	$t_{DI(D)}$	Data Output Delay During INTA		[2]		
	$t_{F(D)}$	Delay to Floating Bus During Read Cycle		160	nsec	
	$t_{S(D)}$	Data Setup Time to Rising Edge of $\overline{IORQ}$ During Write Cycle	200		nsec	
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		nsec	
A_0-A_7, B_0-B_7	$t_{H(PD)}$	Port Data Hold Time From Rising Edge of $\overline{STROBE}$	0		nsec	Mode 1
	$t_{S(PD)}$	Port Data Setup Time to Rising Edge of $\overline{STROBE}$	260		nsec	Mode 1
	$t_{DS(PD)}$	Port Data Output Delay From Falling Edge of $\overline{STROBE}$		230	nsec	Mode 2
	$t_{F(PD)}$	Delay to Floating Port Data Bus From Rising Edge of $\overline{STROBE}$	200		nsec	Mode 2
	$t_{DI(PD)}$	Port Data Stable From Rising Edge of $\overline{IORQ}$ During Write Cycle		200	nsec	Mode 0
B/A, C/D, $\overline{CE}$	$t_{H(CS)}$	Control Signal Hold Time From Rising Edge of $\overline{IORQ}$	0		nsec	
	$t_{S(CS)}$	Control Signal Setup Time to Falling Edge of $\overline{IORQ}$	30		nsec	
$\overline{A\ STB}$, $\overline{B\ STB}$	$t_{W(ST)}$	Pulse Width, $\overline{STROBE}$	150		nsec	Mode 0 or 1
			[3]		nsec	Mode 2 Output [3] $t_{W(ST)} > t_{S(PD)}$
$\overline{INT}$	$t_{D(IT)}$	$\overline{INT}$ Delay Time From Rising Edge of $\overline{STROBE}$		490	nsec	Mode 0, 1 or 2
	$t_{D(IT3)}$	$\overline{INT}$ Delay Time From Data Match During Mode 3 Operation		420	nsec	Mode 3
IEI	$t_{S(IEI)}$	Setup Time Of IEI Prior To $\overline{IORQ}$ During Interrupt Acknowledge	0		nsec	
IEO	$t_{DL(IO)}$	IEO Delay Time From Falling Edge of IEI		210	nsec	
	$t_{DH(IO)}$	IEO Delay Time From Rising Edge of $\overline{RD}$ During RETI		$1.5t_c$ + 500	nsec	
	$t_{DM(IEO)}$	Delay Time From Falling Edge of $\overline{M1}$ To Falling Edge of IEO		310	nsec	See [4] Below
A RDY or B RDY	$t_{DH(RY)}$	READY Response Time From Rising Edge of $\overline{IORQ}$		t_c + 460	nsec	
	$t_{DL(RY)}$	READY Response Time From Rising Edge of $\overline{STROBE}$		t_c + 400	nsec	

[2] 380 ns for C_L = 100pF; 410 ns for C_L = 200pF

[4] 2.5 $t_c > (N-2)t_{DL(IO)} + t_{DM(IEO)} + t_{s(IEI)}$ + TTL buffer delay, if any
where N = number of PIO's in daisy chain

Output load circuit.



CAPACITANCE

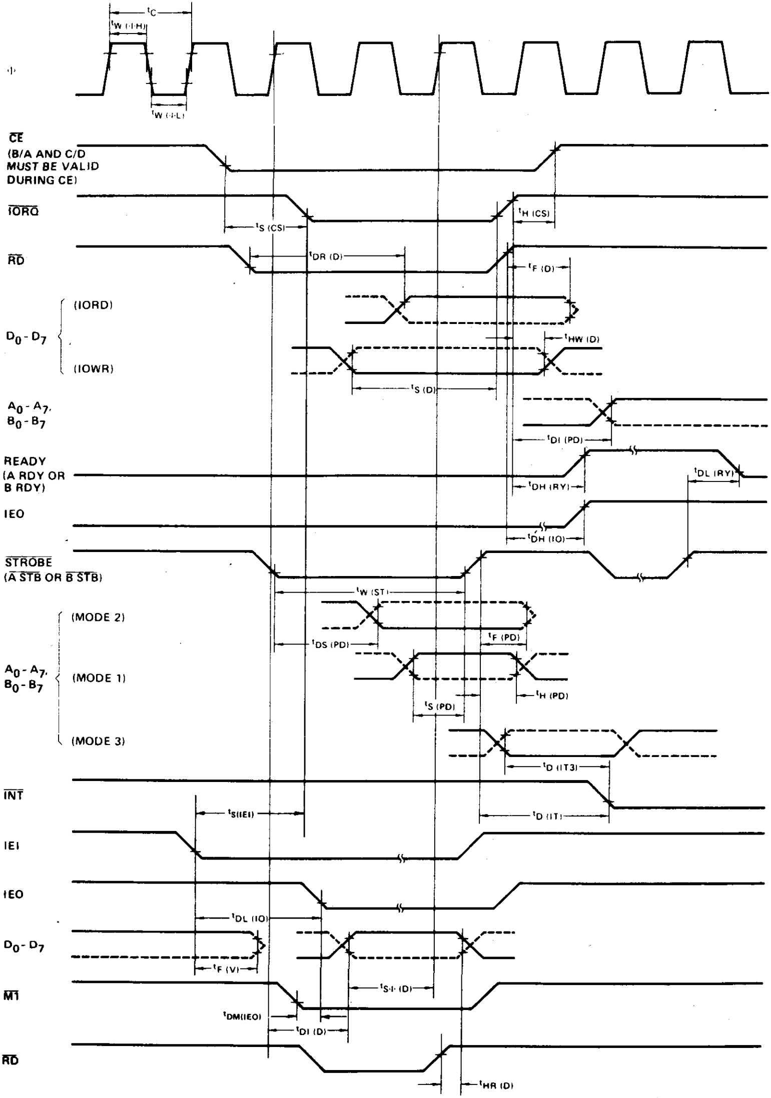
TA = 25° C, f = 1 MHz

Symbol	Parameter	Max.	Unit	Test Condition
C_Φ	Clock Capacitance	10	pF	Unmeasured Pins Returned to Ground
C_{IN}	Input Capacitance	5	pF	
C_{OUT}	Output Capacitance	10	pF	

TIMING CHART

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	4.2 V	.8 V
OUTPUT	2.0 V	.8 V
INPUT	2.0 V	.8 V
FLOAT	V	+ 0.5 V



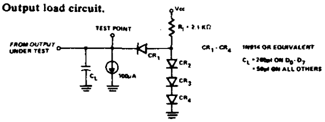
A.C. Characteristics

TA = 0° C to 70° C, Vcc = +5 V ± 5%, unless otherwise noted

Signal	Symbol	Parameter	Min.	Max.	Unit	Comments
Φ	$t_C(\Phi)$	Clock Period	0.4	2	μsec	
	$t_W(\Phi H)$	Clock Pulse Width, Clock High	180	∞	nsec	
	$t_W(\Phi L)$	Clock Pulse Width, Clock Low	180	2000	nsec	
	t_r, t_f	Clock Rise and Fall Times		30	nsec	
CS_0, CS_1	$t_{S\Phi}(CS)$	Channel Select Setup Time to Rising Edge of Φ during Read or Write Cycle	140		nsec	
	$t_{H\Phi}(CS)$	Channel Select Hold Time from Rising Edge of Φ during Write Cycle	$1.5t_c$ (Φ)		nsec	
$\overline{CE}$	$t_{S\Phi}(CE)$	Chip Enable Setup Time to Rising Edge of Φ during Read or Write Cycle	140		nsec	
	$t_{H\Phi}(CE)$	Chip Enable Hold Time from Rising Edge of Φ during Write Cycle	0		nsec	
$D_0–D_7$	$t_{DR}(D)$	Data Output Delay from Rising Edge of Φ during Read Cycle		305	nsec	
	$t_{S\Phi}(D)$	Data Setup Time to Rising Edge of Φ during Write Cycle or M1 Cycle	80		nsec	
	$t_{H\Phi}(D)$	Data Hold Time from Rising Edge of Φ during Write Cycle or M1 Cycle	0		nsec	
	$t_{DI}(D)$	Data Output Delay from Falling Edge of $\overline{IORQ}$ during INTA Cycle		205	nsec	
	$t_{FIM}(D)$	Delay to Floating Bus from Rising Edge of $\overline{IORQ}$ or $\overline{M1}$ during INTA Cycle		210	nsec	
	$t_{FR}(D)$	Delay to Floating Bus from Rising Edge of $\overline{CE}$, $\overline{IORQ}$, or $\overline{RD}$ during Read Cycle		210	nsec	
	$t_{FI}(D)$	Delay to Floating Bus from Falling Edge of IEI during INTA Cycle		230	nsec	
$\overline{IORQ}$	$t_{S\Phi}(IR)$	$\overline{IORQ}$ Setup Time to Rising Edge of Φ during Read or Write Cycle	140		nsec	
	$t_{H\Phi}(IR)$	$\overline{IORQ}$ Hold Time from Rising Edge of Φ during Write Cycle	$1.5t_c$ (Φ)		nsec	
IEO	$t_{DL}(IO)$	IEO Delay Time from Falling Edge of IEI		180	nsec	
	$t_{DH}(IO)$	IEO Delay Time from Rising Edge of IEI		180	nsec	
	$t_{D\bar{\Phi}}(IO)$	IEO Delay Time from Falling Edge of Φ during RETI		185	nsec	
$\overline{INT}$	$t_{DCK}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of CLK/TRG		$2t_C(\Phi)$+155	nsec	Counter Mode
	$t_{D\Phi}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of Φ		$t_C(\Phi)$+155	nsec	Timer Mode
$\overline{M1}$	$t_{SW\Phi}(M1)$	$\overline{M1}$ Setup Time to Rising Edge of Φ during Read or Write Cycle	115		nsec	
	$t_{SR\Phi}(M1)$	$\overline{M1}$ Setup Time to Rising Edge of Φ during INTA or M1 Cycle	115		nsec	
	$t_{H\Phi}(M1)$	$\overline{M1}$ Hold Time from Rising Edge of Φ	0		nsec	
$\overline{RD}$	$t_{SW\Phi}(RD)$	$\overline{RD}$ Setup Time to Rising Edge of Φ during Write or INTA Cycle	115		nsec	
	$t_{H\Phi}(RD)$	$\overline{RD}$ Hold Time from Rising Edge of Φ during INTA Cycle	0		nsec	
	$t_{SR\Phi}(RD)$	$\overline{RD}$ Setup Time to Rising Edge of Φ during Read or M1 Cycle	140		nsec	
	$t_{HW\Phi}(RD)$	$\overline{RD}$ Hold Time from Rising Edge of Φ during Write Cycle	0		nsec	
	$t_{HM\Phi}(RD)$	$\overline{RD}$ Hold Time from Rising Edge of Φ during M1 Cycle	0	$t_c(\Phi)/2$	nsec	
$CLK/TRG_{\emptyset-3}$	$t_C(CK)$	Clock Period	$2t_C(\Phi)$		nsec	Counter Mode
	$t_S(CK)$	Clock Setup Time to Rising Edge of Φ for immediate count	100		nsec	Counter Mode
	$t_S(TR)$	Trigger Setup Time to Rising Edge of Φ for enabling of prescaler on second succeeding Φ	100		nsec	Timer Mode
	t_r, t_f	Clock and Trigger Rise and Fall Times		50	nsec	Counter and Timer Modes
	$t_W(CTH)$	Clock and Trigger High Pulse Width	100		nsec	Counter and Timer Modes
	$t_W(CTL)$	Clock and Trigger Low Pulse Width	100		nsec	Counter and Timer Modes
$ZC/TO_{\emptyset-2}$	$t_{DH}(ZC)$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO High		185	nsec	Counter and Timer Modes
	$t_{DL}(ZC)$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO Low		185	nsec	Counter and Timer Modes

Notes: 1. The $\overline{\text{RESET}}$ signal must be active for a minimum of 2 clock cycles.

Output load circuit.



Capacitance

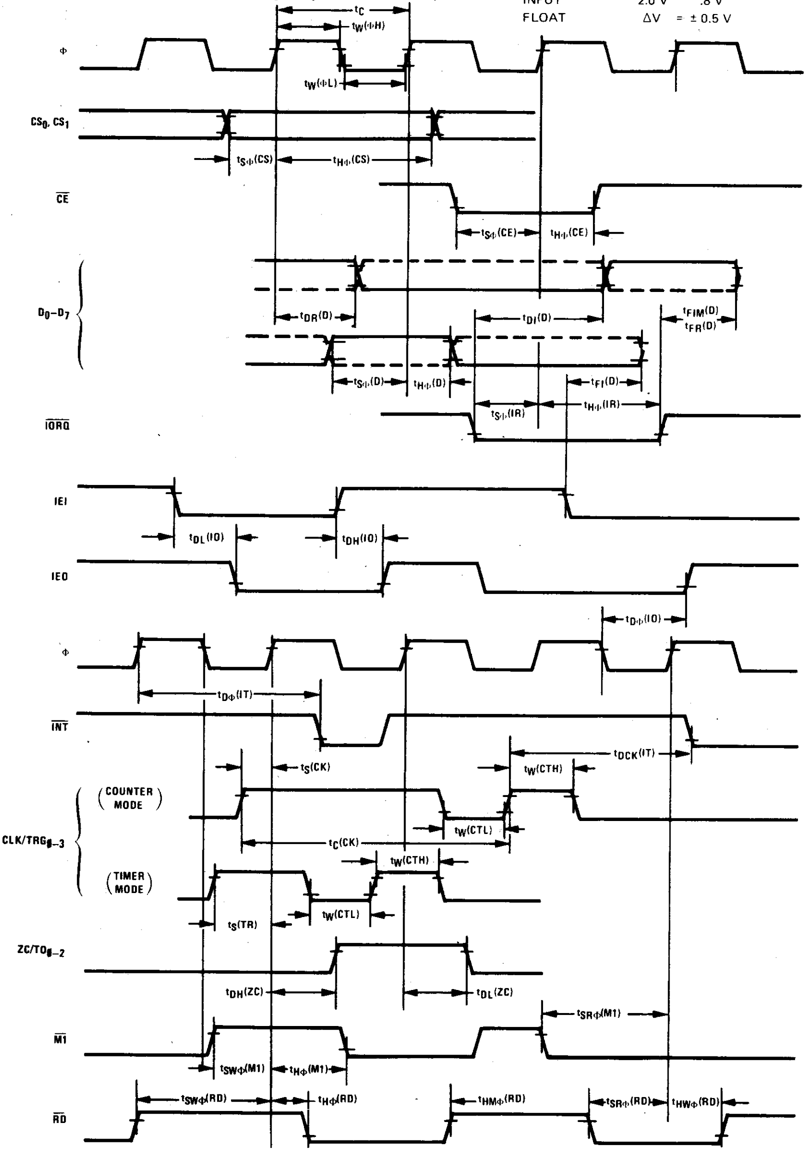
TA = 25° C, f = 1 MHz

Symbol	Parameter	Max.	Unit	Test Condition
C_Φ	Clock Capacitance	20	pF	Unmeasured Pins Returned to Ground
C_{IN}	Input Capacitance	5	pF	
C_{OUT}	Output Capacitance	10	pF	

A.C. Timing Diagram

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	4.2 V	.8 V
OUTPUT	2.0 V	.8 V
INPUT	2.0 V	.8 V
FLOAT	ΔV	= ± 0.5 V



Absolute Maximum Ratings

Temperature Under Bias	0° C to 70° C
Storage Temperature	-65° C to +150° C
Voltage On Any Pin With Respect To Ground	-0.3 V to +7 V
Power Dissipation	0.8W

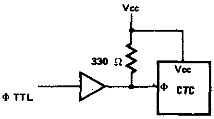
***Comment**
Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. Characteristics

TA = 0° C to 70° C, Vcc = 5 V ± 5% unless otherwise specified

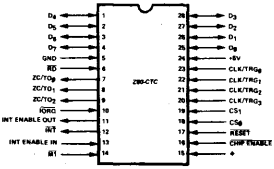
Symbol	Parameter	Min.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3	0.6	V	
V_{IHC}	Clock Input High Voltage [1]	Vcc-.2	Vcc	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{IH}	Input High Voltage	2.0	Vcc	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 1.8 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} -100 µA
I_{CC}	Power Supply Current		100	mA	T_C = 400 n sec
I_{LI}	Input Leakage Current		10	µA	V_{IN} = 0 to Vcc
I_{LOH}	Tri-State Output Leakage Current in Float		10	µA	V_{OUT} = 2.4 to Vcc
I_{LOL}	Tri-State Output Leakage Current in Float		-10	µA	V_{OUT} = 0.4 V
I_{OHD}	Darlington Drive Current	1.5		mA	V_{OH} = 1.5 V R_{EXT} = 390 Ω

[1] Clock Driver

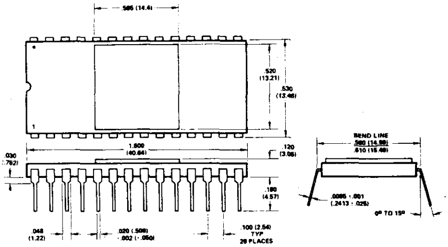


An external pull-up resistor of 330Ω will meet all clock AC and DC requirements.

Package Configuration



Package Outline



* DIMENSIONS FOR METRIC SYSTEM IN PARENTHESES (mm)

Zilog

10460 Bubb Road, Cupertino, California 95014
Telephone: (408) 446-4666   TWX 910-338-7621

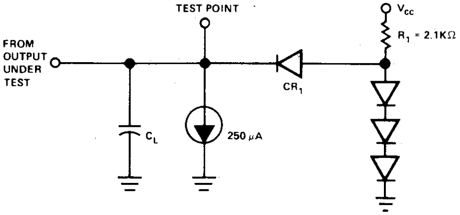
Printed in U.S.A.
12.76.2 10K

Capacitance

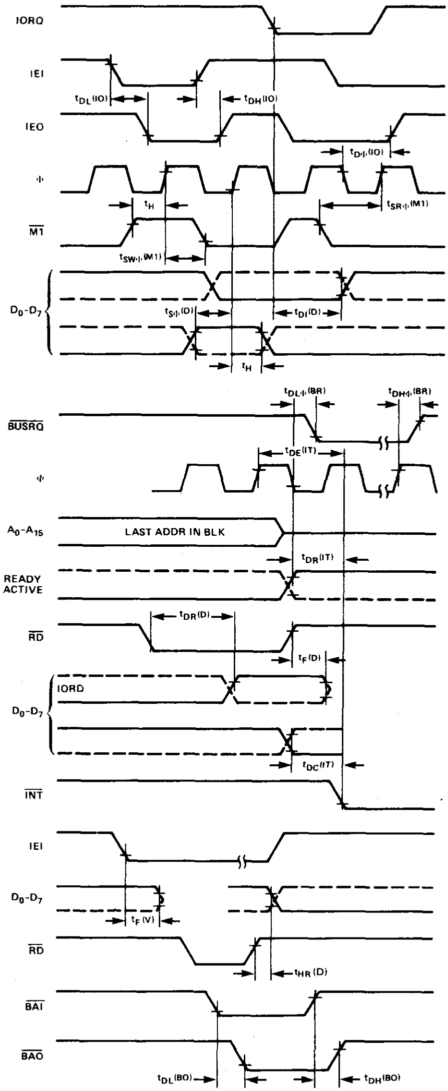
$T_A = 25°C, f = 1$ MHz

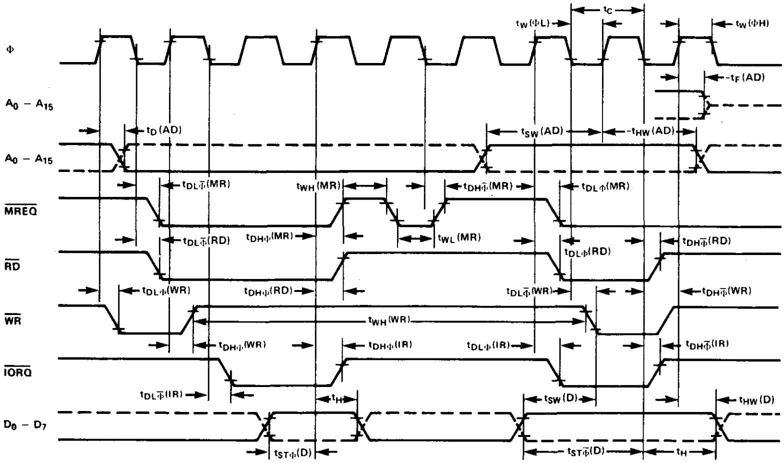
Symbol	Parameter	Max.	Unit	Test Condition
C_{Φ}	Clock Capacitance	20	pF	Unmeasured Pins Returned to Ground
C_{IN}	Input Capacitance	5	pF	
C_{OUT}	Output Capacitance	10	pF	

Load Circuit for Output



A.C. Timing Diagrams





Φ
$A_0 - A_{15}$
$A_0 - A_{15}$
$\overline{MREQ}$
$\overline{RD}$
$\overline{WR}$
$\overline{IORQ}$
$D_0 - D_7$
t_C
$t_W(\Phi L)$
$t_W(\Phi H)$
$t_F(AD)$
$t_D(AD)$
$t_{SW}(AD)$
$t_{HW}(AD)$
$t_{DL\bar{\Phi}}(MR)$
$t_{WH}(MR)$
$t_{DH\bar{\Phi}}(MR)$
$t_{DL\Phi}(MR)$
$t_{DL\bar{\Phi}}(RD)$
$t_{DH\Phi}(MR)$
$t_{WL}(MR)$
$t_{DL\Phi}(RD)$
$t_{DH\bar{\Phi}}(RD)$
$t_{DL\Phi}(WR)$
$t_{DH\Phi}(RD)$
$t_{DL\bar{\Phi}}(WR)$
$t_{DH\bar{\Phi}}(WR)$
$t_{WH}(WR)$
$t_{DH\Phi}(WR)$
$t_{DH\Phi}(IR)$
$t_{DL\Phi}(IR)$
$t_{DH\bar{\Phi}}(IR)$
$t_{DL\bar{\Phi}}(IR)$
t_H
$t_{SW}(D)$
$t_{HW}(D)$
$t_{ST\Phi}(D)$
$t_{ST\bar{\Phi}}(D)$
t_H

A.C. Characteristics

	Signal	Symbol	Parameter	Min.	Max.	Unit	Comments
1	Φ	t_C	Clock period	0.4		μsec	
2		$t_W(\Phi H)$	Clock pulse width, high	180	∞	nsec	
3		$t_W(\Phi L)$	Clock pulse width, low	180	2000		
4		t_r, t_f	Clock rise and fall times		30		
5	D_0-D_7	$t_{HW}(D)$	Data hold after rising edge of Φ during command cycle	0		nsec	
6		$t_{HR}(D)$	Data hold from rising edge of $\overline{RD}$ during M1 cycle	0			
7		$t_{DR}(D)$	Data output delay during DMA response cycle ($\overline{RD} \cdot \overline{CE} \cdot \overline{IORQ}$)		350		
8		$t_{DI}(D)$	Data output delay from falling edge of $\overline{IORQ}$ during INTA cycle		350		
9		$t_F(D)$	Delay to floating bus from rising edge of $\overline{RD}$ during response cycle		210		
10		$t_{SW\Phi}(D)$	Data set-up time to rising edge of Φ during command cycle	200			
11		$t_{S\Phi}(D)$	Data set-up time to rising edge of clock during M1 cycle	100			During INTA cycle
12		$t_F(V)$	Delay to Floating Bus from falling edge of IEI		200		
13		$t_{ST\overline{\Phi}}(D)$	Data set-up prior to Φ when Φ will end $\overline{RD}$		100		During a transfer or compare
14		$t_{ST\overline{\Phi}}(D)$	*Data set-up prior to* $\overline{\Phi}$ *when* $\overline{\Phi}$ *will end* $\overline{RD}$		100		
15	A_0-A_{15}	$t_D(AD)$	Address output delay		200	nsec	
16		$t_F(AD)$	Delay to float		120		
17		$t_{SW}(AD)$	Addr set-up to rising edge of Φ during command cycle		200		
18		$t_{HW}(AD)$	Addr hold after rising edge of Φ during command cycle		0		
19	$\overline{MREQ}$	$t_{DL\Phi}(MR)$	$\overline{MREQ}$ delay from rising edge of Φ, $\overline{MREQ}$ low		120	nsec	
20		$t_{DL\overline{\Phi}}(MR)$	$\overline{MREQ}$ delay from falling edge of Φ, $\overline{MREQ}$, low		130		
21		$t_{DH\Phi}(MR)$	$\overline{MREQ}$ delay from rising edge of Φ, $\overline{MREQ}$, high		130		
22		$t_{DH\Phi}(MR)$	$\overline{MREQ}$ delay from falling edge of Φ, $\overline{MREQ}$, high		150		
23		$t_{WL}(MR)$	*Pulse width,* $\overline{MREQ}$ *low*	200	NOTE [1]		
24		$t_{WH}(MR)$	Pulse width, $\overline{MREQ}$ high	200	NOTE [1]		
25	$\overline{INT}$	$t_{DC}(IT)$	$\overline{INT}$ delay time from compare		400	nsec	
26		$t_{DE}(IT)$	$\overline{INT}$ delay time from end of block if **selected**		400		
27		$t_{DR}(IT)$	$\overline{INT}$ delay time from start of **READY if selected**		400		
28	IEO	$t_{DL}(IO)$	IEO delay time from falling **edge of** IEI, IEO low		180	nsec	
29		$t_{DH}(IO)$	IEO delay time **from rising edge** of IEI, IEO high		180		
30		$t_{D\overline{\Phi}H}(IO)$	IEO delay time from falling edge of Φ during RETI, IEO low		185		
31	$\overline{IORQ}$	$t_{DL\Phi}(IR)$	$\overline{IORQ}$ **delay from** rising edge of Φ, $\overline{IORQ}$ low		110	nsec	
32		$t_{DL\overline{\Phi}}(IR)$	$\overline{IORQ}$ **delay** from falling edge of Φ, $\overline{IORQ}$ low		130		
33		$t_{DH\Phi}(IR)$	$\overline{IORQ}$ delay from rising edge of Φ, $\overline{IORQ}$ high		130		
34		$t_{DH\overline{\Phi}}(IR)$	$\overline{IORQ}$ delay in falling edge of Φ, $\overline{IORQ}$ high		150		
35		$t_s(C)$	*Control signal (*$\overline{IORQ}$, $\overline{CE}$*) setup before falling edge of* Φ, *Control low*		200		
36	$\overline{WR}$	$t_{WL}(WR)$	Pulse width $\overline{WR}$ low	200		nsec	
37		$t_{WH}(WR)$	Pulse width $\overline{WR}$ high	[2]			
38		$t_{DL\Phi}(WR)$	$\overline{WR}$ delay from rising edge of Φ, $\overline{WR}$ low		110		
39		$t_{DL\overline{\Phi}}(WR)$	$\overline{WR}$ delay from falling edge of Φ, $\overline{WR}$ low		130		
40		$t_{DH\Phi}(WR)$	$\overline{WR}$ delay from rising edge of Φ, $\overline{WR}$ high		130		
41		$t_{DH\overline{\Phi}}(WR)$	$\overline{WR}$ delay from falling edge of Φ, $\overline{WR}$ high		150		
42	$\overline{RD}$	$t_{DL\Phi}(RD)$	$\overline{RD}$ delay from rising edge of Φ, $\overline{RD}$ low		130	nsec	
43		$t_{DL\overline{\Phi}}(RD)$	$\overline{RD}$ delay from falling edge of Φ, $\overline{RD}$ low		150		
44		$t_{DH\Phi}(RD)$	$\overline{RD}$ delay from *rising* edge of Φ, $\overline{RD}$ *high*		130		
45		$t_{DH\overline{\Phi}}(RD)$	$\overline{RD}$ delay from falling edge of Φ, $\overline{RD}$ high		150		
46	$\overline{BUSRQ}$	$t_{DL\Phi}(BR)$	$\overline{BUSRQ}$ delay from rising edge of Φ, $\overline{BUSRQ}$ low		250	nsec	
47		$t_{DH\Phi}(BR)$	$\overline{BUSRQ}$ delay from rising edge of Φ		110		
48	$\overline{BAO}$	$t_{DL}(BO)$	$\overline{BAO}$ delay time from falling edge of $\overline{BAI}$, $\overline{BAO}$ low		180	nsec	
49		$t_{DH}(BO)$	$\overline{BAO}$ delay time from rising edge of $\overline{BAI}$, $\overline{BAO}$ high		180		
50	$\overline{M1}$	$t_{SW\Phi}(M1)$	$\overline{M1}$ set-up time to rising edge of Φ during command or response $\overline{M1}$ high		120	nsec	
51		$t_{SR\Phi}(M1)$	$\overline{M1}$ set-up time to rising edge of Φ during INTA or M1 cycle $\overline{M1}$ low		120		

NOTES: 1. Variable cycle length 1 clock period, MREQ start delay - ½ clock period MREQ stop = end of cycle
2. A rising edge and a complete t_c must occur between command writes.

Absolute Maximum Ratings

Temperature Under Bias	Specified operating range.
Storage Temperature	-65°C to +150°C
Voltage On Any Pin with Respect to Ground	-0.3V to +7V
Power Dissipation	1.5W

Note: All AC and DC characteristics remain the same for the military grade parts except I_{CC}.

I_{CC} = 200 mA.

*Comment

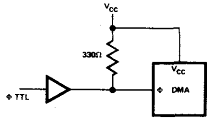
Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. Characteristics

T_A = 0°C to 70°C, V_{CC} = 5V ±5% unless otherwise specified

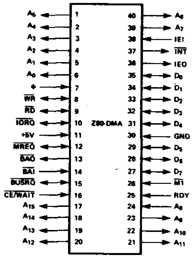
Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC}-.2[1]		V_{CC}	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 1.8 mA
V_{OH}	Output High Voltage	2.4		.	V	I_{OH} = -250 μA
V_{CC}	Power Supply Current			150	mA	t_C = 400 nsec
I_{LI}	Input Leakage Current			10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	V_{OUT} = 0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	0 < V_{IN} < V_{CC}

[1] Clock Driver

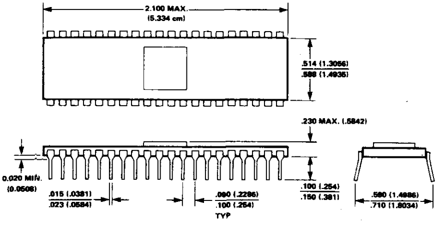


An external clock pull-up resistor of (330Ω) will meet both the AC and DC clock requirements.

Package Configuration



Package Outline



*Dimensions for metric system are in parentheses

Ordering Information

C – Ceramic
P – Plastic
S – Standard 5V ±5%, 0° to 70°C
E – Extended 5V ± 5% -40° to 85°C
M – Military 5V ±10% -55° to 125°C

Example:

Z80-DMA CS (Ceramic–Standard range)

10460 Bubb Road, Cupertino, California 95014
03-0013-00

Zilog

Telephone: (408) 446-4666 TWX 910-338-7621
Printed in U.S.A.

Chapter 8
THE MOTOROLA MC6800

The MC6800 was developed by Motorola as an enhancement of the Intel 8008, at the same time that Intel was developing the 8080A, also as an enhancement of the 8008.

When comparing the MC6800 to the 8080A, the most important feature of the MC6800 is its relative simplicity. Here are a few superficial, but illustrative comparisons between the two products:

1) As compared to the 8080A, MC6800 timing is very simple. MC6800 instructions execute in two or more machine cycles, all of which are identical in length. In contrast to the 8080A, which we described in Chapter 4, note that an MC6800 machine cycle and clock period are one and the same thing — each MC6800 machine cycle has a single clock period.
2) Whereas the 8080A has separate I/O instructions, the MC6800 includes memory and I/O within a single address space. Thus all I/O devices are accessed as memory locations.
3) The MC6800 has a simpler set of control signals, therefore it does not multiplex the Data Bus — and does not need any device equivalent to the 8228 System Controller.
4) Whereas the 8080A requires three levels of power supply, the MC6800 uses just one — +5V.
5) The instruction set of the MC6800 is much easier to comprehend than that of the 8080A. The MC6800 has fewer basic instruction types, with more memory addressing options; the 8080A, by way of contrast, has a large number of special, one-of-a-kind instructions.

It is very informative to extend the five comparisons above with the enhancements that Intel has made to the 8080A in order to come up with the 8085. Let us take the five points one at a time.

1) 8085 instruction execution timing is far simpler than the 8080A. But MC6800 timing is still far simpler than the 8085.
2) The 8085 retains the separate memory and I/O spaces of the 8080A.
3) The 8085 has separate control signals which do not need to be demultiplexed off the Data Bus, as required by the 8080A. The price paid by the 8085 is a multiplexed Data and Address Bus. Neither the MC6800 nor the 8085 need any device equivalent to the 8228 System Controller; however, the 8085 will need a bus demultiplexer in configurations that do not use the standard 8085 support devices.
4) The 8085, like the MC6800, has gone to a single +5V power supply.
5) The 8085 instruction set is almost identical to that of the 8080A.

An additional point worth noting is that the 8085 includes clock logic on the CPU chip. The MC6800 requires a separate clock logic chip.

Looking at the 8085, there are grounds for arguing that Intel has acknowledged that the MC6800 has some desirable characteristics not present in the 8080A. In

order to compete with the 8085, therefore, Motorola will not be required to make MC6800 enhancements of the same magnitude as Intel made going from the 8080A to the 8085. Specifically, these are the MC6800 characteristics which remain to be addressed by any MC6800 enhancement:

1) Clock logic must be moved on to the CPU chip.
2) Multifunction CPU and support devices must be developed so that Motorola can offer low chip count microcomputers.

Additional weaknesses of the MC6800 that have manifested themselves include:

1) An instruction set that makes excessive use of memory as a result of too few Index registers and a lack of data mobility between registers of the CPU. This is a weakness that was identified in the first version of this book.
2) The synchronizing E signal, required by support devices of the MC6800, render these support devices useless in any microcomputer system other than the MC6800. In contrast, 8080A support devices can be used widely in microcomputer systems not based on the 8080A CPU.

Future Motorola plans address many of the points raised above. The MC6802, described in this chapter, is the first step towards reducing chip counts in MC6800-based microcomputer systems. An MC6801, planned for release in late 1977 or early 1978, will provide a one chip MC6800-based microcomputer. The MC6809 will be the new enhanced MC6800, to compete with the 8085. The MC6809 will provide additional Index registers, plus instructions that move data between Accumulators and Index registers. The MC6809 will have clock logic on the CPU chip.

These are the devices described in this chapter:

- **The MC6800 CPU**
- **The MC6802 CPU with RAM**
- **The MC6870 series Clocks**
- **The MC6820 Peripheral Interface Adapter (PIA)**
- **The MC6850 Asynchronous Communications Interface Adapter (ACIA)**
- **The XC6852 Synchronous Serial Data Adapter (SSDA)**
- **The MC6828 Priority Interrupt Controller (PIC)**

Two new series of MC6800 parts offer higher speeds. Standard MC6800 parts use a 1 MHz clock signal. "A" parts use a 1.5 MHz clock signal, while "B" parts use a 2 MHz clock signal. There is, in addition, an MC6821 PIA which is identical to the MC6820 in operating characteristics, but has different physical characteristics.

The principal MC6800 manufacturer is:

MOTOROLA INCORPORATED
Semiconductor Products Division
3501 Ed Bluestein Boulevard
Austin, Texas 78721

The second sources are:

AMERICAN MICROSYSTEMS
3800 Homestead Road
Santa Clara, California 95051

FAIRCHILD SEMICONDUCTOR
464 Ellis Street
Mountain View, California 94040

HITACHI — Semiconductors And Integrated
Circuits Division of Hitachi LTD
1450 Josuihan-Cho-Kodaira-Shi
Tokyo, Japan

SESCOSEM — Thompson CSF
173 Haussmann Blvd.
Paris, France 75008

The MC6800 devices use a single +5V power supply. Using a one microsecond clock, instruction execution times range from 2 to 12 microseconds. A one microsecond clock is the standard for MC6800 microcomputer systems. 750 nanosecond clocks are standard for the 6800A series while 500 nanosecond clocks are standard for the 6800B series.

All MC6800 devices have TTL compatible signals.

N-channel silicon gate, depletion load MOS technology is used for the MC6800.

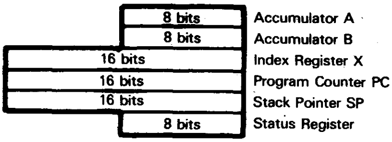
THE MC6800 CPU

Functions implemented on the MC6800 CPU are illustrated in Figure 8-1; they represent typical CPU logic. As compared to other microprocessors described in this book, the MC6800 might be considered deficient in requiring external clock logic; however, its principal competitor, the 8080A, requires external clock logic and Data Bus demultiplexing logic.

The need for external clock logic simply reflects the fact that the MC6800 is one of the earlier microprocessors.

THE MC6800 PROGRAMMABLE REGISTERS

The MC6800 has two Accumulators, a Status register, an Index register, a Stack Pointer and a Program Counter. These may be illustrated as follows:



The two Accumulators, A and B, are both primary Accumulators. The only instructions which apply to one Accumulator, but not the other, are the instructions which move statuses between Accumulator A and the Status register.

The Index register is a typical microcomputer Index register, as described in Volume I.

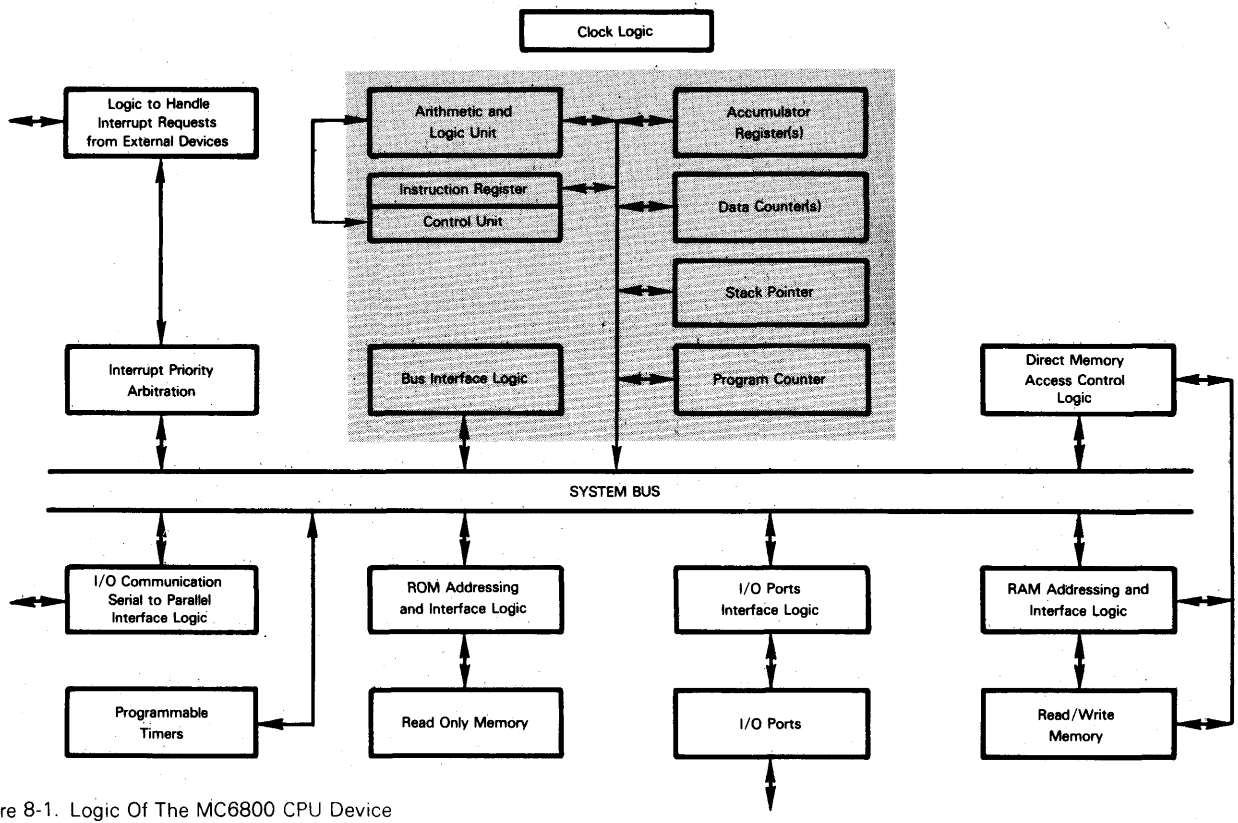

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 8-1. Logic Of The MC6800 CPU Device

The MC6800 has a Stack implemented in memory and indexed by the Stack Pointer, as described in Volume I. Because of the nature of the MC6800 instruction set, it is more realistic to look upon the MC6800 Stack Pointer as a cross between a Stack Pointer and a Data Counter. Memory reference instructions make it very easy to store the contents of either the Stack Pointer or the Index register in read/write memory; by maintaining a number of base page memory locations as storage for these two Address registers, each can be put to multiple use.

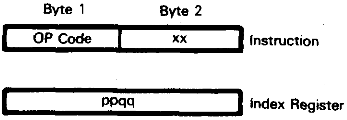
The Program Counter is a typical Program Counter, as described in Volume I.

MC6800 MEMORY ADDRESSING MODES

MC6800 memory reference instructions use direct addressing and indexed addressing.

The MC6800 has an unusually large variety of memory referencing instructions, all of which offer three-byte, direct memory addressing; a 16-bit direct address is provided by the second and third bytes of the instruction; therefore, 65,536 bytes of memory can be directly addressed. The commonly used memory reference instructions also have a base page, direct addressing option; this is a two-byte instruction, with a one-byte address which can directly address any one of the first 256 bytes of memory.

All memory reference instructions are available with indexed addressing. Indexed addressing on the MC6800 differs from indexed addressing as described in Volume I, in that the one-byte displacement provided by the memory reference instruction is added to the Index register as an unsigned 8-bit value:

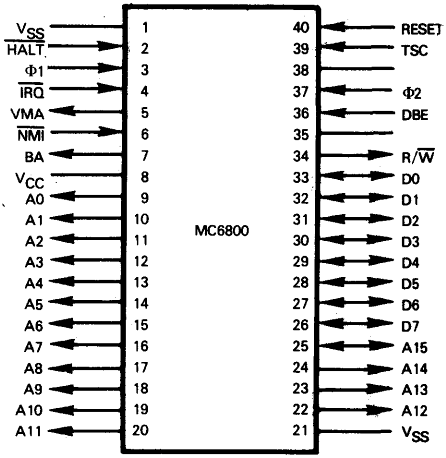


Effective Address = ppqq + 00xx
p, q and x represent any hexadecimal digits.

MC6800 programs can use the Stack Pointer as an Address register, but two bytes of read/write memory must be reserved for the current top of Stack address and interrupts must be disabled while the Stack Pointer is being used to address data memory. A single instruction allows an address to be loaded into the Stack Pointer; another single instruction allows the Stack Pointer contents to be stored in read/write memory. In most programs, the Stack is unused for much of the time; therefore, given the low MC6800 overhead involved with swapping addresses between the Stack Pointer and read/write memory, making dual use of the Stack Pointer is advisable.

Branch and Branch-on-Condition instructions use program relative, direct addressing; a single byte displacement is treated as a signed binary number which is added to the Program Counter, after Program Counter contents have been incremented to address the next sequential instruction. This allows displacements in the range +129 to -126 bytes.

One note of caution: Motorola's MC6800 literature uses the term "implied addressing" to describe instructions that identify one of the programmable registers. **The closest thing the MC6800 has to implied addressing, as the term is used in this book, is indexed addressing with a zero displacement.**



PIN NAME	DESCRIPTION	TYPE
*A0 - A15	Address Lines	Tristate, Output
*D0 - D7	Data Bus Lines	Tristate, Bidirectional
*$\overline{\text{HALT}}$	Halt	Input
*TSC	Three State Control	Input
*R/$\overline{\text{W}}$	Read/Write	Tristate, Output
*VMA	Valid Memory Address	Output
*DBE	Data Bus Enable	Input
*BA	Bus Available	Output
*$\overline{\text{IRQ}}$	Interrupt Request	Input
$\overline{\text{RESET}}$	Reset	Input
$\overline{\text{NMI}}$	Non-Maskable Interrupt	Input
Φ1, Φ2	Clock Signals	Input
V_{SS}, V_{CC}	Power	Input

*These signals connect to the System Bus.

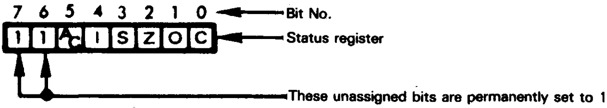
Figure 8-2. MC6800 CPU Signals And Pin Assignments

MC6800 STATUS FLAGS

The MC6800 has a Status register which maintains five status flags and an interrupt control bit. These are the five status flags:

Carry (C)
Overflow (O)
Sign (S)
Zero (Z)
Auxiliary Carry (A_C)

Statuses are assigned bit positions within the Status register as follows:



I is the external interrupt enable/disable flag. When it is 1, interrupts via $\overline{\text{IRQ}}$ are disabled; when it is 0, interrupts via $\overline{\text{IRQ}}$ are enabled.

MC6800 literature refers to the Sign bit as a negative bit, given the symbol N; the Overflow bit is given the symbol V. The Intermediate Carry bit represents the standard Carry out of bit 3 and is referred to as the Half Carry bit, given the symbol H. Statuses are nevertheless set and reset as described for our hypothetical microcomputer in Volume I.

MC6800 CPU PINS AND SIGNALS

The MC6800 CPU pins and signals are illustrated in Figure 8-2. A description of these signals is useful as a guide to the way in which the MC6800 microcomputer system works.

The Address Bus is a tristate bus; it is 16 bits wide and is used to address all types of memory and external devices.

The Data Bus is also a tristate bus; it is an 8-bit bidirectional bus via which data is transmitted between memory and all MC6800 microcomputer system devices.

Control signals on the MC6800 Control Bus may be divided into bus state controls, bus data identification, and interrupt processing.

MC6800 BUS STATE CONTROLS

These are the bus state control signals:

Three State Control (TSC). This input is used to float the Address Bus and the read/write control output.

Data Bus Enable (DBE). This signal is input low in order to float the Data Bus. When the Data Bus, the Address Bus and the read/write control output have all been floated, Direct Memory Access operations may be performed by external logic. DBE is frequently tied to the Φ2 clock input, in which case Φ2 and DBE are identical signals.

$\overline{\text{HALT}}$. When this signal is input low, the CPU ceases execution and floats the entire System Bus.

Bus Available (BA). This line is output high when the Data and Address Busses have been floated following a $\overline{\text{HALT}}$ input only. **When BA is low, the CPU is controlling the Data and Address Busses; information on these busses is identified by the following two control signals:**

Read/Write (R/$\overline{\text{W}}$). When high, this signal indicates that the CPU wishes to read data off the Data Bus; when low, this signal indicates that the CPU is outputting data on the Data Bus. The normal standby state for this signal is "read" (high).

Valid Memory Address (VMA). This signal is output high whenever a valid address has been output on the Address Bus.

There are three interrupt processing signals as follows:

$\overline{\text{IRQ}}$. *This signal is used to request an interrupt.* If interrupts have been enabled and the CPU is not in the Halt state, then it will acknowledge the interrupt at the end of the currently executing instruction.

Non-Maskable Interrupt ($\overline{\text{NMI}}$). This signal differs from $\overline{\text{IRQ}}$ in that it cannot be inhibited. Typically, this input is used for catastrophic interrupts such as power fail.

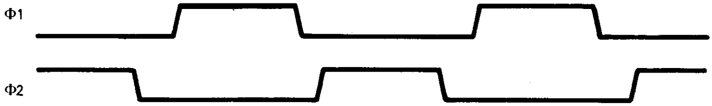
$\overline{\text{RESET}}$. This is a typical reset signal.

Note that a number of control signals output by the MC6800 are only capable of driving one standard TTL load. Some form of signal buffering and amplification will therefore be required in most systems.

MC6800 TIMING AND INSTRUCTION EXECUTION

MC6800 CLOCK SIGNALS

The MC6800 uses a relatively simple combination of two clock signals to time events within the microprocessor CPU and the microcomputer system in general. **These two clock signals may be illustrated as follows:**

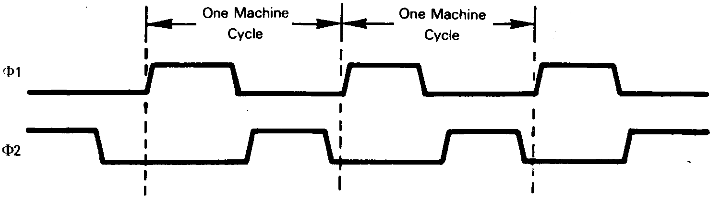


Observe that clock signals Φ1 and Φ2 both have high pulses which occur within the width of the other clock signal's low pulse.

A further timing signal, given the symbol E, is used by support devices within an MC6800 microcomputer system. Φ1, Φ2 and E timing signals are generated by the clock logic devices described later in this chapter.

MC6800 MACHINE CYCLE

Each repeating pattern of Φ1 and Φ2 signals constitutes a single machine cycle:



MC6800 instructions require between two and eight machine cycles to execute. Interrupt instructions are an exception, requiring longer instruction execution times.

MC6800 MACHINE CYCLE TYPES

So far as external logic is concerned, there are only three types of machine cycles which can occur during an instruction's execution:

1) **A read operation** during which a byte of data must be input to the CPU.
2) **A write operation** during which a byte of data is output by the CPU.
3) **An internal operation** during which no activity occurs on the System Bus.

All MC6800 instructions have timing which is a simple concatenation of the three basic machine cycle types.

Let us therefore begin by looking at these three basic machine cycles.

MC6800 READ MACHINE CYCLE

Figure 8-3 illustrates timing for a standard read machine cycle. Observe that in the normal course of events, neither the Address nor the Data Busses are available for DMA operations. The address output is stable for most of the machine cycle. Data needs to be stable for a short interval of time late in the machine cycle. Exact timing is given in MC6800 data sheets at the end of this chapter.

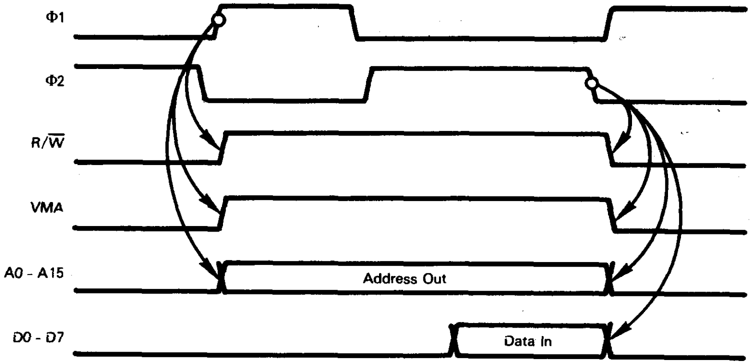


Figure 8-3. A Standard MC6800 Read Machine Cycle

MC6800 WRITE MACHINE CYCLE

Figure 8-4 illustrates a standard MC6800 write machine cycle. This machine cycle is not as straightforward as the read. The address to which data is being written is stable on the Address Bus for the duration of the machine cycles; however, the data being written is stable for a period within the high DBE pulse. While DBE is low, the Data Bus is floated.

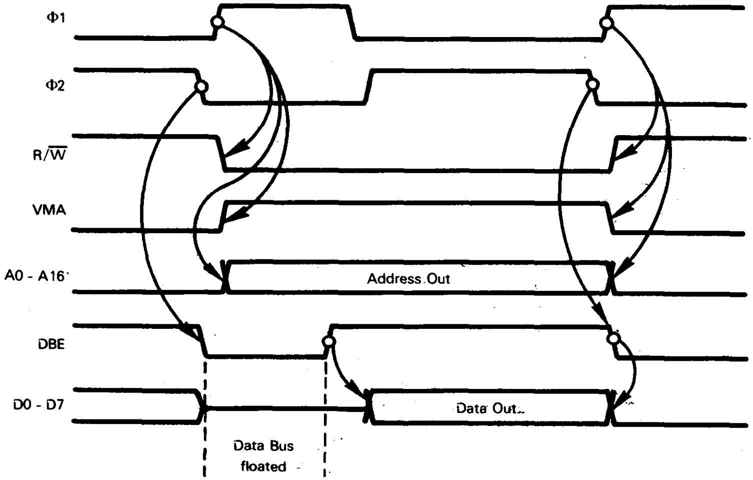
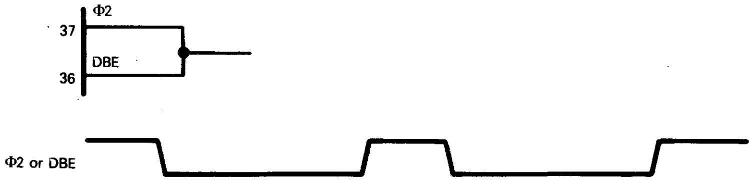


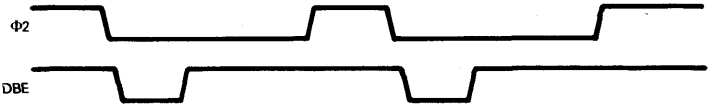
Figure 8-4. A Standard MC6800 Write Machine Cycle

Under normal circumstances, DBE is identical to Φ2:



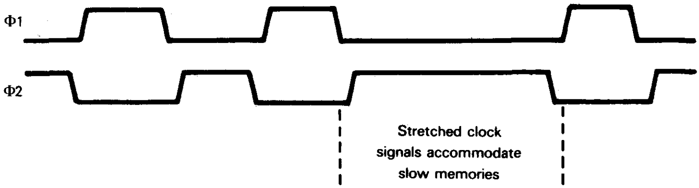
MC6800 WAIT STATE WITH SLOW MEMORY

If the high Φ2 pulse is too short for external logic to respond to the write, the slow external logic can be accommodated in two ways. You can input a DBE signal to the CPU that has a shorter low pulse and a longer high pulse. DBE and Φ2 are no longer identical signals:



There is some minimum time during which DBE must be low, since the CPU itself requires time to perform internal operations. This minimum time is given in the MC6800 data sheets at the end of this chapter.

You can also accommodate slow memories by stretching the system clocks; this may be illustrated as follows:



The standard clock devices, described later in this chapter, provide clock stretching logic. During a clock stretch, Φ1 and Φ2 cannot be held constant for more than 5 μsec; the MC6800 is a dynamic device, and longer static clock periods can result in loss of internal data.

MC6800 INTERNAL OPERATIONS MACHINE CYCLE

During an internal operation's machine cycle, there is no activity on the System Bus. R/$\overline{W}$ is in its normal high state and VMA is low.

Table 8-2 defines the way in which individual MC6800 instructions concatenate machine cycles and use the System Bus during the course of instruction execution.

The VMA and DBE signals require special mention, because their significance can easily be missed. External logic uses VMA as a signal identifying the address on the Address Bus as having been placed there by the CPU. DBE similarly identifies that portion of a machine cycle when the CPU is active at one end of the Data Bus, either transmitting or receiving data. And this is why these signals are so important: MC6800 microcomputer systems rely heavily on clock signal manipulation as a means of accommodating slow memories, implementing Direct Memory Access, or refreshing dynamic memory. On the next few pages we are going to see examples of how this is done. So long as you understand that the VMA and DBE signals identify the unmanipulated portions of a standard machine cycle, you will have no trouble locating the time slices within which special operations such as Direct Memory Access or dynamic memory refresh are occurring.

THE HOLD STATE, THE HALT STATE AND DIRECT MEMORY ACCESS

The Hold state typically describes a CPU condition during which System Busses are floated, so that external logic can perform Direct Memory Access operations.

Though the MC6800 literature does not talk about a Hold state, this microprocessor does indeed have two equivalent conditions.

You can float the Address and Data Busses separately, using the TSC and DBE signals.

You can enter an MC6800 Halt state, which is equivalent to our definition of a Hold state.

Let us begin by looking at the use of TSC and DBE signals.

The Three State Control signal (TSC), if input high, will float the Address Bus and R/$\overline{W}$ line. VMA and BA are forced low. The unusual feature of the Three State Control input is that when this signal is input high, you must simultaneously stop the clock by holding Φ1 high and Φ2 low. Timing is illustrated in Figure 8-5. Now the MC6800, being a dynamic device, will lose its data contents if the clock is stopped for more than 5 μsec. You must therefore float the Address Bus just long enough to perform a single Direct Memory Access.

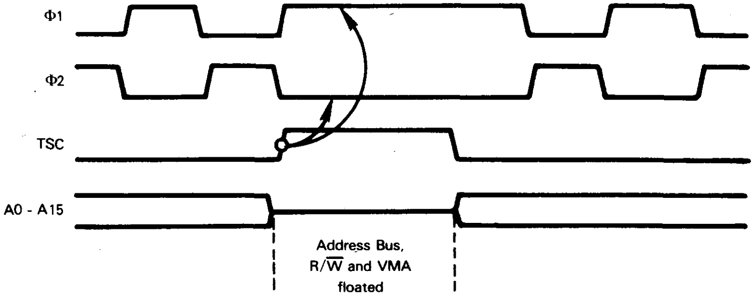


Figure 8-5. TSC Floating The Address Bus

Just as the Three State Control input floats the Address Bus, so the Data Bus Enable input (DBE) floats the Data Bus. When DBE is input low, the Data Bus is floated.

The clock devices, which are described later in this chapter, provide all necessary clock stretching logic.

There are two very important points to note regarding the use of Three State Control (TSC) and Data Bus Enable (DBE) signals.

First of all, note carefully that the Bus Available (BA) signal is held low when the busses are floated by the Three State Control (TSC) and Data Bus Enable (DBE) signals. The purpose of the Bus Available signal is to indicate that the System Bus is available during a Halt or Wait state, both of which we have yet to describe.

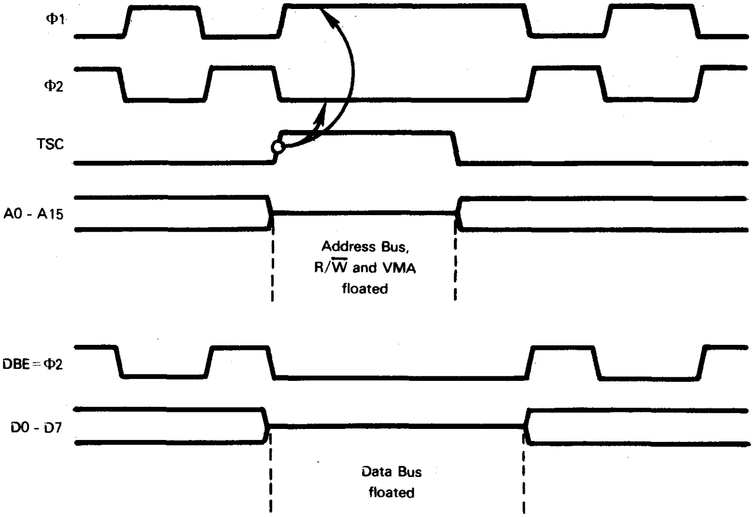


Figure 8-6. TSC Floating The Address And Data Busses When DBE Is Tied To Φ2

The second important feature of the Three State Control (TSC) and Data Bus Enable (DBE) signals is that they do indeed float the System Bus in two halves. Now in many MC6800 systems Φ2 and DBE are the same signal; in such a configuration you will automatically float the Data Bus whenever you float the Address Bus, as illustrated in Figure 8-6.

Now consider the MC6800 Halt state.

The Halt state of the MC6800 is equivalent to the Hold state of the 8080A. When a low $\overline{\text{HALT}}$ is input to the MC6800, then upon conclusion of the current instruction's execution, the System Bus is floated. Timing is illustrated in Figure 8-7. Observe that the Bus Available signal, BA, is output high; VMA is output low. The Address and Data Busses, and the R/$\overline{\text{W}}$ control are floated.

In summary, the MC6800 provides two means of performing Direct Memory Access operations. You can use the TSC and DBE inputs to gain control of the System Bus for as long as it takes to perform a single DMA access, or you can use the $\overline{\text{HALT}}$ input, following which external logic can gain control of the System Bus for as long as you wish.

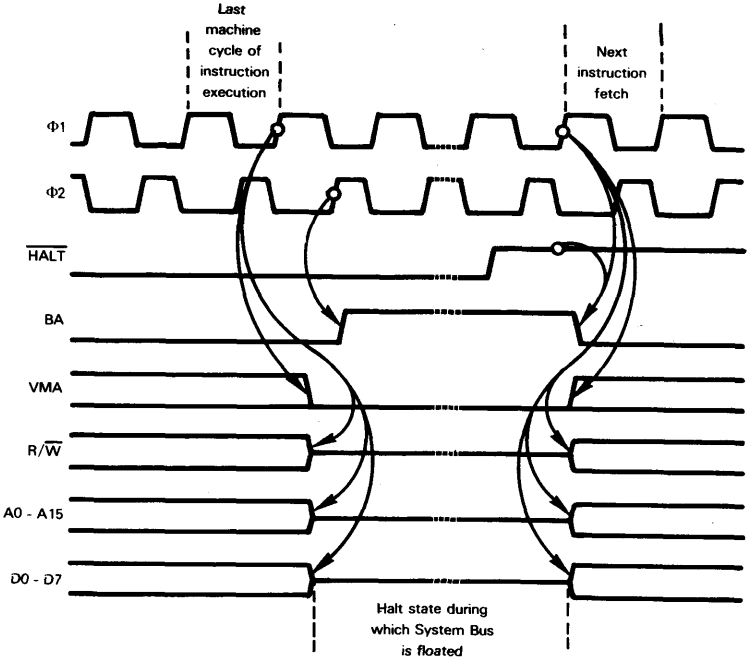
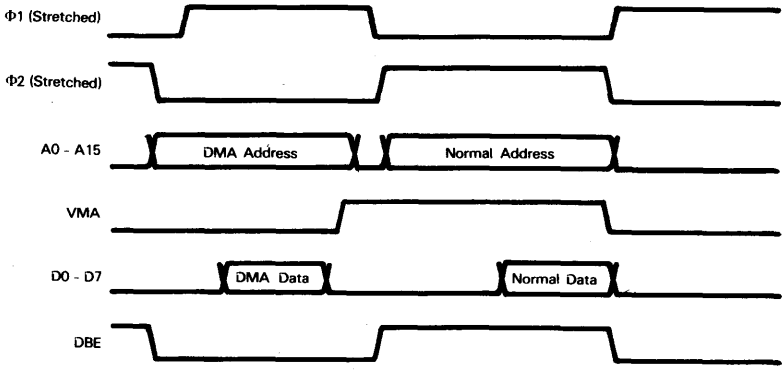

Last machine cycle of instruction execution
Next instruction fetch
Φ1
Φ2
HALT
BA
VMA
R/W
A0 - A15
D0 - D7
Halt state during which System Bus is floated

Figure 8-7. System Bus Floating During The Halt State

Conceptually, the MC6800 scheme for implementing Direct Memory Access or dynamic memory refresh, is very elegant. **If you stretch the Φ1 and Φ2 clock signals, then you can transfer the normal CPU generated address, and an extraneous address within one machine cycle. VMA identifies the CPU generated address. Within the one machine cycle you can perform two Data Bus transfers; the first is in response to the external address, while the second is in response to the CPU address. Now DBE identifies the CPU response.** This scheme may be illustrated as follows:



From this conceptually elegant beginning, some very complex design considerations can arise. These design considerations are not discussed in this book. If you are going to make use of clock stretching as illustrated above, you should first study vendor literature on the subject.

INTERRUPT PROCESSING, RESET AND THE WAIT STATE

MC6800 microcomputer system interrupt logic is based on polling rather than vectoring.

All normal interrupt requests, when acknowledged, result in an indirect addressing Call to a single high memory address. If more than one device can request an interrupt, then the basic assumption made is that the interrupt service routine will initially read the Status register contents of every device that might be requesting an interrupt; and by testing appropriate status bits, the interrupt service routine will determine which interrupt requests are active. If more than one interrupt request is active, interrupt service routine logic must decide the order in which interrupt requests will be acknowledged.

But be warned: this type of polling quickly becomes untenable as a means of controlling microcomputer systems with multiple random interrupts. If you have more than two or three competing external interrupts, the time taken to read Status register contents and arbitrate priority will become excessive. If your application demands numerous external interrupts, then you must resort to external hardware which implements interrupt vectoring. We will describe ways in which this can be done.

If you casually look at a description of MC6800 interrupt logic, you may at first believe that some level of interrupt vectoring is provided. In reality, that is not the case.

The MC6800 sets aside the eight highest addressable memory locations for interrupt processing purposes. Four 16-bit addresses are stored in these eight memory locations, identifying the interrupt service routine's starting address for the four possible sources of interrupt. This is how the eight memory locations are used:

These two memory locations:	Store the starting address for an interrupt service routine to be executed following this interrupt acknowledge:
FFF8 and FFF9	Normal external interrupt
FFFA and FFFB	Software interrupt
FFFC and FFFD	Non-maskable interrupt
FFFE and FFFF	Reset (or restart)

The lower address (FFF8, FFFA, FFFC, FFFE) holds the high order byte of the starting address.

MC6800 INTERRUPT PRIORITIES

In the event of simultaneous interrupt requests, **this is the priority sequence** during the acknowledge process:

Highest	(1)	Restart
	(2)	Non-maskable interrupt
	(3)	Software interrupt
Lowest	(4)	Normal external interrupt

Only the lowest priority interrupt is normally used by the typical support device that is capable of requesting interrupt service. The three higher priority interrupt levels represent special conditions and cannot be accessed by the standard external interrupt request.

We will begin our discussion of MC6800 interrupt processing by describing the four interrupts.

MC6800 NORMAL EXTERNAL INTERRUPTS

The normal external interrupt request is the standard interrupt present on all microprocessors that support interrupts; it is equivalent to the 8080A INT input. In very simple systems, the addresses $FFF8_{16}$ and $FFF9_{16}$ may indeed access real memory locations; in the multiple interrupt MC6800 microcomputer systems, $FFF9_{16}$ is more likely to select an 8-bit buffer within which an address vector is stored identifying the interrupting source. This is essentially how the MC6828 PIC works and is equivalent to the 8259 PICU operation.

MC6800 SOFTWARE INTERRUPT

MC6800 SWI INSTRUCTION

A software interrupt is initiated by the execution of the SWI instruction. What the SWI instruction does is cause the MC6800 to go through the complete logic of an interrupt request and acknowledge, even though the interrupting source is within the CPU. Software interrupts are typically used as a response to fatal errors occurring within program logic. Whenever your program logic encounters a situation that must not, or should not exist, the error condition is trapped by executing an SWI instruction; this causes a call to some general purpose, error recovery program.

MC6800 NON-MASKABLE INTERRUPT

The non-maskable interrupt cannot be disabled. Otherwise it is identical to the normal external interrupt request. Note that the 8080A has no non-maskable interrupt; however, the Zilog Z80 and the 8085 have incorporated this feature.

MC6800 RESET

A Reset is treated as the highest priority interrupt in an MC6800. How does the Reset differ from the non-maskable interrupt? Conceptually, the non-maskable interrupt is going to be triggered by a termination condition such as power failure, while the **Reset is going to be triggered by an initiating condition such as power being turned on.**

There are some differences between the MC6800's response to a Reset as compared to any other interrupt request.

Figure 8-8 illustrates MC6800 response to a normal external interrupt, a software interrupt, or a non-maskable interrupt. In each case, the interrupt request will be acknowledged upon completion of an instruction's execution. A normal external interrupt will only be acknowledged providing interrupts have been enabled.

If more than one interrupt request exists, then the highest priority interrupt will be acknowledged.

Following the interrupt acknowledge, interrupts are disabled by the CPU, which then pushes onto the Stack the contents of all internal registers. This process is illustrated in Figure 8-8. The Program Counter is then loaded with the appropriate interrupt service routine starting address, which will be fetched from memory locations $FFF8_{16}$ and $FFF9_{16}$, $FFFA_{16}$ and $FFFB_{16}$ or $FFFC_{16}$ and $FFFD_{16}$.

Referring to Figure 8-8, note that an interrupt is acknowledged following the last machine cycle for the instruction during which the interrupt request occurred. During the first two machine cycles following the interrupt acknowledge, an instruction fetch is executed, as it would have been had the interrupt not occurred. This instruction fetch is aborted and will reoccur after the interrupt service routine has completed execution Two machine cycles are expended performing this aborted instruction fetch.

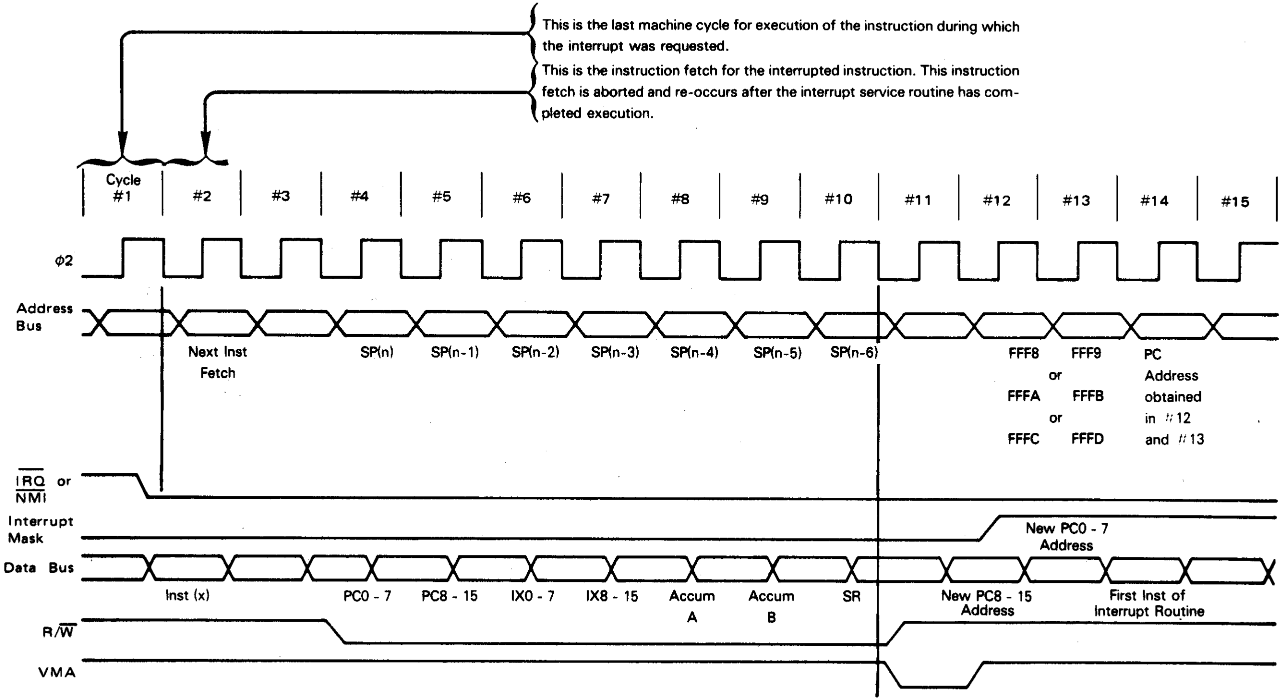

This is the last machine cycle for execution of the instruction during which the interrupt was requested.
This is the instruction fetch for the interrupted instruction. This instruction fetch is aborted and re-occurs after the interrupt service routine has completed execution.
Cycle #1
#2
#3
#4
#5
#6
#7
#8
#9
#10
#11
#12
#13
#14
#15
φ2
Address Bus
Next Inst Fetch
SP(n)
SP(n-1)
SP(n-2)
SP(n-3)
SP(n-4)
SP(n-5)
SP(n-6)
FFF8 or FFFA or FFFC
FFF9 or FFFB or FFFD
PC Address obtained in #12 and #13
$\overline{IRQ}$ or $\overline{NMI}$
Interrupt Mask
New PC0 - 7 Address
Data Bus
Inst (x)
PC0 - 7
PC8 - 15
IX0 - 7
IX8 - 15
Accum A
Accum B
SR
New PC8 - 15 Address
First Inst of Interrupt Routine
R/$\overline{W}$
VMA

Figure 8-8. MC6800 Interrupt Acknowledge Sequence

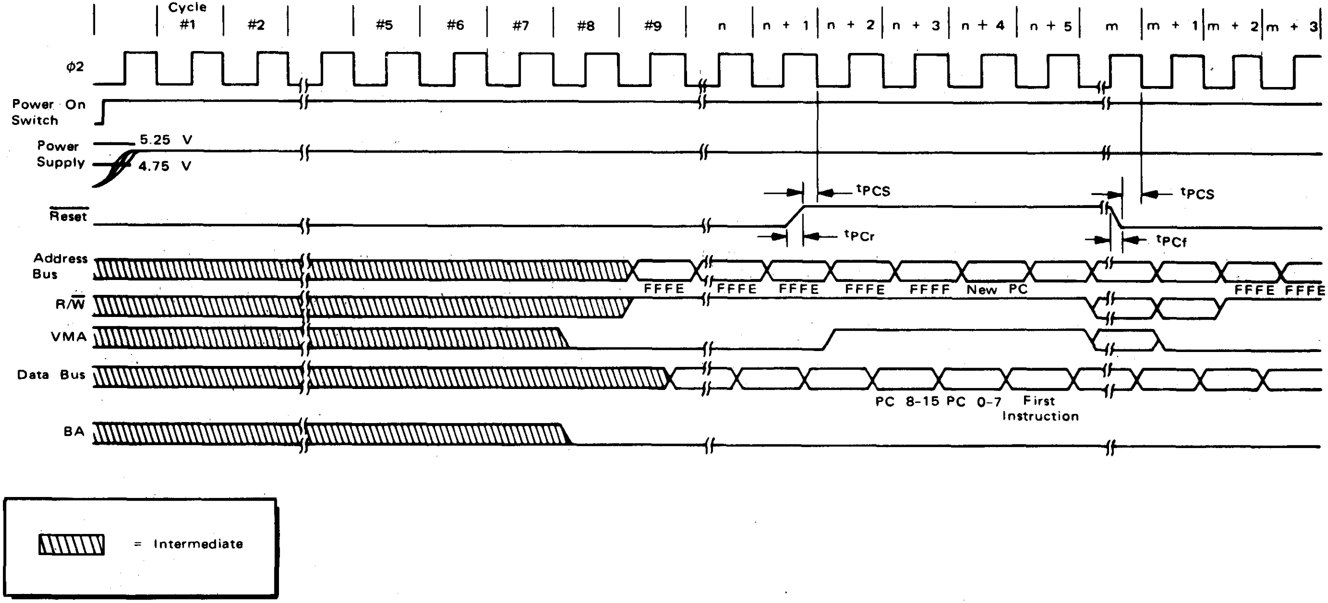


Figure 8-9. The Reset Sequence

Following the aborted instruction fetch, CPU registers' contents are pushed onto the Stack in the following order:

- Lower half of Program Counter
- Upper half of Program Counter
- Lower half of Index register
- Upper half of Index register
- Accumulator A
- Accumulator B
- Status register

When the 8080A acknowledges an interrupt, if CPU registers' contents are going to be saved on the Stack, you must execute individual instructions to perform the operations which the MC6800 performs automatically. The advantage of the MC6800's scheme is that it saves instruction execution time. The disadvantage of this scheme is that there are occasions when you do not need to bother saving registers' contents.

After all CPU registers' contents have been saved on the Stack, the next two machine cycles are used to fetch an address from the appropriate two high memory bytes. This address is loaded into the Program Counter, causing a branch to the appropriate interrupt service routine.

The MC6800 RESET OPERATION

We will now examine the MC6800 Reset operation.

Figure 8-9 illustrates Reset timing. First of all, note that $\overline{\text{RESET}}$ must be held low for at least eight machine cycles to give the CPU sufficient response time. On the high-to-low transition of $\overline{\text{RESET}}$ the CPU outputs VMA and BA low and R/$\overline{\text{W}}$ is high. On the subsequent low-to-high transition of $\overline{\text{RESET}}$, maskable interrupts are disabled, then the contents of memory locations FFFE$_{16}$ and FFFF$_{16}$ are fetched and loaded into the Program Counter. If $\overline{\text{RESET}}$ is not held low for a minimum of eight machine cycles, then when $\overline{\text{RESET}}$ is input high again, indeterminate program execution may follow.

It is absolutely vital that the $\overline{\text{RESET}}$ rise time is less than 100 nanoseconds on the low-to-high transition of $\overline{\text{RESET}}$.

We stated that the difference between a Reset and a non-maskable interrupt is that the Reset represents initiation conditions. This is illustrated in Figure 8-9, which includes the power supply level. The MC6800 will automatically trigger a Reset when power increases above +4.75 volts; this is in response to the normal powering up sequence.The fact that Reset represents initiation conditions also explains why no CPU registers' contents are saved, as occurs with any other interrupt. Clearly, if we are initiating operations, there can be no prior registers' contents to be saved. Therefore pushing registers' contents on the Stack would be pointless and impossible: it would be pointless because there is nothing to save; it would be impossible because when powering up, we have no idea what the Stack Pointer contains.

MC6800 RESET DURING POWER UP

Powering up an MC6800 microcomputer system represents a special Reset case. Those MC6800 microcomputer system devices that have an external Reset input control, expect this control to be held low while power is being turned on for the first eight clock cycles following power-up. When designing Reset logic be sure to keep this in mind.

MC6800 configurations using 8080A support devices are easy to design and commonly seen. Necessary system bus logic is described later in this chapter. But if you have such a mixed configuration, be sure to satisfy the separate and distinct Reset requirements of the MC6800 CPU as against the 8080A support devices.

MC6800 WAIT STATE

MC6800 WAI INSTRUCTION

We complete our discussion of the MC6800 interrupt logic with a discussion of the WAI instruction, which puts the MC6800 into a "Wait-for-interrupt" state.

A WAI instruction is executed when the CPU has nothing to do except wait for an interrupt. Rather than pushing registers' contents onto the Stack following the interrupt acknowledge, as illustrated in Figure 8-8, the WAI instruction pushes registers' contents onto the Stack while waiting for the interrupt, as illustrated in Figure 8-10. Thus some execution time is saved.

Once all registers' contents have been pushed onto the Stack, the MC6800 floats the System Bus in the Wait state.

This gives rise to another frequent use of the WAI instruction: block data transfers under DMA control.

MC6800 USE OF WAIT FOR DMA

Consider again the sequence of events which follows the WAI instruction execution:

1) All registers' contents are pushed onto the Stack.
2) The System Bus is floated.

This is very convenient if you are going to transfer a large block of data via DMA, because you will announce the end of the DMA transfer with an interrupt request. This method of handling block DMA transfers has been discussed in Volume I. Now when using an MC6800 microcomputer system; all you need to do is initiate the actual DMA transfer by executing a WAI instruction, knowing that once the DMA transfer has been completed, an interrupt will be requested and program execution can continue.

THE MC6800 INSTRUCTION SET

Table 8-1 summarizes the MC6800 instruction set; this instruction set is characterized by a heavy use of read/write memory and a rich variety of instructions that are able to manipulate the contents of memory locations as though they were programmable registers. Whereas the primary memory reference instructions offer base page direct addressing, extended direct addressing or indexed addressing, secondary memory reference instructions offer extended direct addressing and indexed addressing only. This simply means that secondary memory reference instructions use three-byte direct addressing even when a base page byte must be accessed.

Of the microcomputers described in this chapter, the MC6800 has one of the largest varieties of Branch-on-Condition instructions. Note that these and the unconditional Branch instructions are the only MC6800 instructions which use program relative direct addressing.

When comparing the MC6800 and 8080A instruction sets, the conclusion we must draw is that the MC6800 is going to have to rely on a large number of memory reference instructions. You are going to have to set up programs with this in mind. As a result, relatively simple programs will make the MC6800 look better than the 8080A, because the MC6800 has such a diverse variety of memory reference instructions. The moment a program starts to become complicated, the large number of 8080A registers is quickly going to become an advantage, since the MC6800 will be forced to execute memory reference instructions where the 8080A can use register-register instructions.

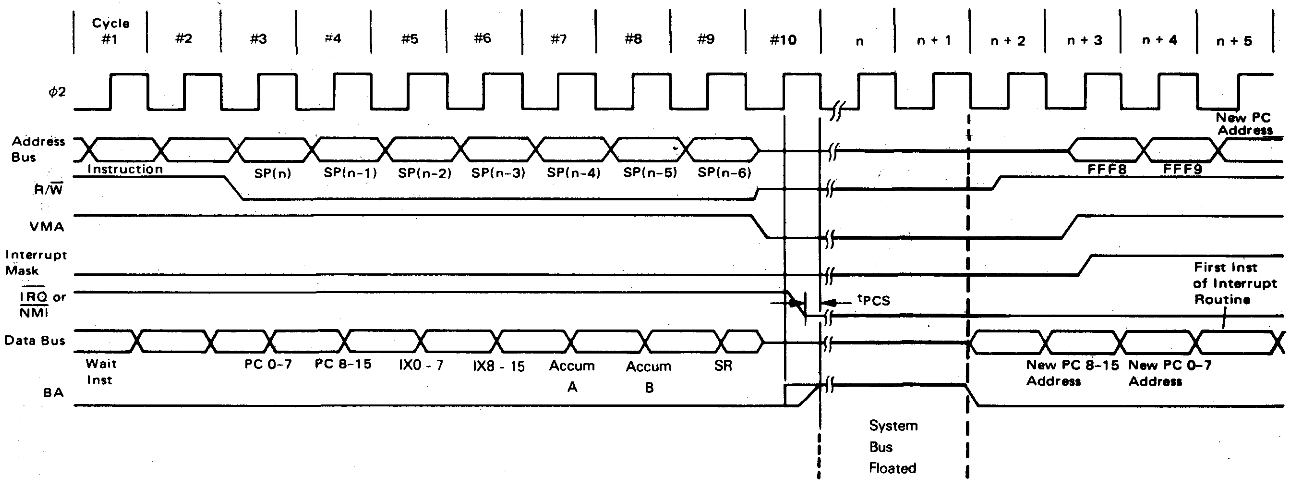


Figure 8-10. MC6800 Wait Instruction Execution Sequence

The SWI and WAI instructions within the interrupt instruction group are relatively unusual within microcomputer systems.

The SWI instruction initiates a normal interrupt sequence, taking the interrupt service routine's starting address from memory locations FFFA$_{16}$ and FFFB$_{16}$.

The WAI instruction prepares for an interrupt by saving the contents of all registers and status on the Stack; the System Bus is then floated while the CPU waits for an interrupt request to occur.

We have described both the SWI and WAI instructions in some detail earlier in this chapter.

The one set of instructions which are missing, and which would greatly enhance the MC6800 instruction set, are instructions that move data between the Accumulator and the Index register, or allow Accumulator contents to be added to the Index register.

THE BENCHMARK PROGRAM

The benchmark program is coded for the MC6800 as follows:

```
        STS     SSP       SAVE STACK POINTER CONTENTS IN MEMORY
        LDX     #TABLE    LOAD TABLE BASE ADDRESS INTO INDEX REGISTER
        LDX     0,X       LOAD ADDRESS OF FIRST FREE TABLE BYTE
        LDS     #IOBUF    LOAD I/O BUFFER STARTING ADDRESS
LOOP    PULL    A         LOAD NEXT BYTE INTO A
        STAA    0,X       STORE IN NEXT FREE TABLE BYTE
        INX               INCREMENT INDEX REGISTER
        DEC     IOCNT     DECREMENT I/O BYTE COUNT IN MEMORY
        BNE     LOOP      RETURN FOR MORE BYTES
        STX     TABLE     STORE NEW ADDRESS FOR FIRST FREE TABLE BYTE
        LDS     SSP       RELOAD STACK POINTER
```

The memory initialization for the MC6800 interpretation of the benchmark program is identical to the memory initialization for the 8080A benchmark program. The MC6800 assumes that there is some memory location in which the current real Stack address can be stored, so that the Stack Pointer may be used as a Data Counter.

In Table 8-1, symbols are used as follows:

ACX	Either Accumulator A or Accumulator B

The registers:

A,B	Accumulator
X	Index register
PC	Program Counter
SP	Stack Pointer
SR	Status register

Statuses shown:

C	Carry status
Z	Zero status
S	Sign status
O	Overflow status
I	Interrupt status
A_C	Auxiliary Carry status

Symbols in the STATUSES column:

(blank)	operation does not affect status
X	operation affects status
0	flag is cleared by the operation
1	flag is set by the operation

ADR8 An 8-bit (1-byte) quantity which may be used to directly address the first 256 locations in memory, or may be an 8-bit unsigned displacement to be added to the Index register.

ADR16 A 16-bit memory address

B2 Instruction Byte 2

B3 Instruction Byte 3

DATA An 8-bit binary data unit

DATA16 A 16-bit binary data unit

DISP An 8-bit signed binary address displacement

xx(HI) The high order 8 bits of the 16-bit quantity xx; for example, SP(HI) means bits 15 - 8 of the Stack Pointer.

xx(LO) The low order 8 bits of the 16-bit quantity xx; for example, PC(LO) means bits 7 - 0 of the Program Counter.

[ ] Contents of location enclosed within brackets.

[[ ]] Implied memory addressing; the contents of the memory location designated by the contents of a register.

[MEM] Symbol for memory location indicated by base page direct, extended direct, or indexed addressing.

That is:

[MEM] = [ADR8]

or

[ADR16]

or

[[X]+ADR8]

[M] Symbol for memory location indicated by extended direct or indexed addressing. That is:

[M]=[ADR16]

or

[[X]+ADR8]

Λ Logical AND

V Logical OR

⊻ Logical Exclusive-OR

← Data is transferred in the direction of the arrow.

Table 8-1. A Summary Of The MC6800 Instruction Set

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	O	A_C	I	
PRIMARY MEMORY REFERENCE AND I/O	LDA	ACX,ADR8	2		X	X	0			[ACX]←[MEM]
		ACX,ADR16	3							Load A or B using base page direct, extended direct, or indexed addressing.
	STA	ACX,ADR8	2		X	X	0			[MEM]←[ACX]
		ACX,ADR16	3							Store A or B using direct, extended, or indexed addressing.
	LDX	ADR8	2		X	X	0			[X(HI)]←[MEM], [X(LO)]←[MEM + 1]
		ADR16	3							Load Index register using direct, extended, or indexed addressing. Sign status reflects Index register bit 15.
	STX	ADR8	2		X	X	0			[MEM]←[X(HI)], [MEM + 1]←[X(LO)]
		ADR16	3							Store contents of Index register using direct, extended, or indexed addressing. Sign status reflects Index register bit 15.
	LDS	ADR8	2		X	X	0			[SP(HI)]←[MEM], [SP(LO)]←[MEM + 1]
		ADR16	3							Load Stack Pointer using direct, extended, or indexed addressing. Sign status reflects Stack Pointer bit 15.
	STS	ADR8	2		X	X	0			[MEM]←[SP(HI)], [MEM + 1]←[SP(LO)]
		ADR16	3							Store contents of Stack Pointer using direct, extended, or indexed addressing. Sign status reflects Stack Pointer bit 15.
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	ADD	ACX,ADR8	2	X	X	X	X	X		[ACX]←[ACX] + [MEM]
		ACX,ADR16	3							Add to Accumulator A or B using base page direct, extended direct, or indexed addressing.
	ADC	ACX,ADR8	2	X	X	X	X	X		[ACX]←[ACX] + [MEM] + C
		ACX,ADR16	3							Add with carry to Accumulator A or B using direct, extended, or indexed addressing.
	AND	ACX,ADR8	2		X	X	0			[ACX]←[ACX] ∧ [MEM]
		ACX,ADR16	3							AND with Accumulator A or B using direct, extended, or indexed addressing.
	BIT	ACX,ADR8	2		X	X	0			[ACX] ∧ [MEM]
		ACX,ADR16	3							AND with Accumulator A or B, but only Status register is affected.
	CMP	ACX,ADR8	2	X	X	X	X			[ACX] - [MEM]
		ACX,ADR16	3							Compare with Accumulator A or B (only Status register is affected).
	EOR	ACX,ADR8	2		X	X	0			[ACX]←[ACX] ⊻ [MEM]
		ACX,ADR16	3							Exclusive-OR with Accumulator A or B using direct, extended, or indexed addressing.
	ORA	ACX,ADR8	2		X	X	0			[ACX]←[ACX] V [MEM]
		ACX,ADR16	3							OR with Accumulator A or B using direct, extended, or indexed addressing.

Table 8-1. A Summary Of The MC6800 Instruction Set (Continued)

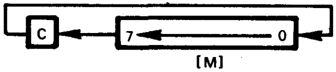
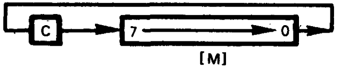
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	O	A_C	I	
SECONDARY MEMORY REFERENCE (MEMORY OPERATE) CONTINUED	SUB	ACX,ADR8	2	X	X	X	X			[ACX]←[ACX] - [MEM]
		ACX,ADR16	3							Subtract from Accumulator A or B using direct, extended, or indexed addressing.
	SBC	ACX,ADR8	2	X	X	X	X			[ACX]←[ACX] - [MEM] - C
		ACX,ADR16	3							Subtract with carry from Accumulator A or B using direct, extended, or indexed addressing.
	CPX	ADR8	2		X	X	X			[X(HI)] - [MEM], [X(LO)] - [MEM + 1]
		ADR16	3							Compare with contents of Index register (only Status register is affected). Sign and Overflow statuses reflect result on most significant byte.
	CLR	ADR8	2	0	1	0	0			[M] ← 00_{16}
		ADR16	3							Clear memory location using extended or indexed addressing.
	COM	ADR8	2	1	X	X	0			[M]←$[\overline{M}]$
		ADR16	3							Complement contents of memory location (ones complement).
	NEG	ADR8	2	X	X	X	X			[M] ← 00_{16} - [M]
		ADR16	3							Negate contents of memory location (twos complement). Carry status is set if result is 00_{16} and reset otherwise. Overflow status is set if result is 80_{16} and reset otherwise.
	DEC	ADR8	2		X	X	X			[M]←[M] - 1
		ADR16	3							Decrement contents of memory location, using extended or indexed addressing. Overflow status is set if operand was 80_{16} before execution, and cleared otherwise.
	INC	ADR8	2		X	X	X			[M]←[M] + 1
		ADR16	3							Increment contents of memory location, using extended or indexed addressing. Overflow status is set if operand was $7F_{16}$ before execution, and cleared otherwise.
	ROL	ADR8	2	X	X	X	X			, O ← S⊻C
		ADR16	3							Rotate contents of memory location left through carry.
	ROR	ADR8	2	X	X	X	X			, O ← S⊻C
		ADR16	3							Rotate contents of memory location right through carry.

Table 8-1. A Summary Of The MC6800 Instruction Set (Continued)

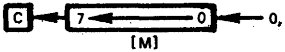
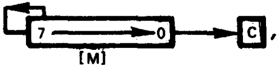
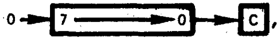
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C	Z	S	O	A_C	I	OPERATION PERFORMED
SECONDARY MEMORY REFERENCE (MEMORY OPERATE) CONTINUED	ASL	ADR8 ADR16	2 3	X	X	X	X			 O — S⊻C Arithmetic shift left. Bit 0 is set to 0.
	ASR	ADR8 ADR16	2 3	X	X	X	X			 O — S⊻C Arithmetic shift right. Bit 7 stays the same.
	LSR	ADR8 ADR16	2 3	X	X	0	X			 O — S⊻C Logical shift right. Bit 7 is set to 0.
	TST	ADR8 ADR16	2 3	0	X	X	0			$[M] - 00_{16}$ Test contents of memory location for zero or negative value.
IMMEDIATE	LDA	ACX,DATA	2		X	X	0			[ACX]— DATA Load A or B immediate.
	LDX	DATA16	3		X	X	0			[X(HI)]—[B2], [X(LO)]—[B3] Load Index register immediate. Sign status reflects Index register bit 15.
	LDS	DATA16	3		X	X	0			[SP(HI)]—[B2], [X(LO)]—[B3] Load Stack Pointer immediate. Sign status reflects Stack Pointer bit 15.
IMMEDIATE OPERATE	ADD	ACX,DATA	2	X	X	X	X	X		[ACX]←[ACX] + DATA Add immediate to Accumulator A or B.
	ADC	ACX,DATA	2	X	X	X	X	X		[ACX]—[ACX] + DATA + C Add immediate with carry to Accumulator A or B.
	AND	ACX,DATA	2		X	X	0			[ACX]—[ACX] Λ DATA AND immediate with Accumulator A or B.
	BIT	ACX,DATA	2		X	X	0			[ACX] Λ DATA AND immediate with Accumulator A or B, but only the Status register is affected.

Table 8-1. A Summary Of The MC6800 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	O	A_C	I	
IMMEDIATE OPERATE (CONTINUED)	CMP	ACX,DATA	2	X	X	X	X			[ACX] - DATA Compare immediate with Accumulator A or B (only the Status register is affected).
	EOR	ACX,DATA	2		X	X	0			[ACX]←[ACX]⊻DATA Exclusive-OR immediate with Accumulator A or B.
	ORA	ACX,DATA	2		X	X	0			[ACX]←[ACX] V DATA OR immediate with Accumulator A or B.
	SUB	ACX,DATA	2	X	X	X	X			[ACX]←[ACX] - DATA Subtract immediate from Accumulator A or B.
	SBC	ACX,DATA	2	X	X	X	X			[ACX]←[ACX] - DATA - C Subtract immediate with carry from Accumulator A or B.
	CPX	DATA16	3		X	X	X			[X(HI)] - [B2], [X(LO)] - [B3] Compare immediate with contents of Index register (only the Status register is affected). Sign and Overflow status reflect result on most significant byte.
JUMP	JMP	ADR8 ADR16	2 3							[PC]←[X] + ADR8 or [PC(HI)]←[B2], [PC(LO)]←[B3] Jump to indexed or extended address.
	JSR	ADR8 ADR16	2 3							[[SP]]←[PC(LO)], [[SP]-1]←[PC(HI)], [SP]←[SP]-2 [PC]←[X] + ADR8 or [PC(HI)]←[B2], [PC(LO)]←[B3] Jump to subroutine (indexed or extended addressing).
	BRA	DISP	2							[PC]←[PC] + DISP + 2 Unconditional branch relative to present Program Counter contents.
	BSR	DISP	2							[[SP]]←[PC(LO)], [[SP]-1]←[PC(HI)], [SP]←[SP]-2, [PC]←[PC] + DISP + 2 Unconditional branch to subroutine located relative to present Program Counter contents.

Table 8-1. A Summary Of The MC6800 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	O	A_C	I	
BRANCH ON CONDITION										[PC]←[PC] + DISP + 2 if the given condition is true:
	BCC	DISP	2							C = 0 (Branch if carry clear)
	BCS	DISP	2							C = 1 (Branch if carry set)
	BEQ	DISP	2							Z = 1 (Branch if equal to zero)
	BGE	DISP	2							S⊻O = 0 (Branch if greater than or equal to zero)
	BGT	DISP	2							Z V (S⊻O) = 0 (Branch if greater than zero)
	BHI	DISP	2							C V Z = 0 (Branch if Accumulator contents higher than comparand)
	BLE	DISP	2							Z V (S⊻O) = 1 (Branch if less than or equal to zero)
	BLS	DISP	2							C V Z = 1 (Branch if Accumulator contents less than or same as comparand)
	BLT	DISP	2							S⊻O = 1 (Branch if less than zero)
	BMI	DISP	2							S = 1 (Branch if minus)
	BNE	DISP	2							Z = 0 (Branch if not equal to zero)
	BVC	DISP	2							O = 0 (Branch if overflow clear)
	BVS	DISP	2							O = 1 (Branch if overflow set)
	BPL	DISP	2							S = 0 (Branch if plus)
REGISTER-REGISTER MOVE	TAB		1		X	X	0			[B]←[A] Move Accumulator A contents to Accumulator B.
	TBA		1		X	X	0			[A]←[B] Move Accumulator B contents to Accumulator A.
	TXS		1							[SP]←[X]-1 Move Index register contents to Stack Pointer and decrement.
	TSX		1							[X]←[SP] + 1 Move Stack Pointer contents to Index register and increment.
REGISTER REGISTER OPERATE	ABA		1	X	X	X	X	X		[A]←[A] + [B] Add contents of Accumulators A and B.
	CBA		1	X	X	X	X			[A] - [B] Compare contents of Accumulators A and B. Only the Status register is affected.
	SBA		1	X	X	X	X			[A]←[A] - [B] Subtract contents of Accumulator B from those of Accumulator A.

Table 8-1 A Summary Of The MC6800 Instruction Set (Continued)

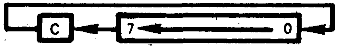
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C	Z	S	O	A_C	I	OPERATION PERFORMED
REGISTER OPERATE	CLR	ACX	1	0	1	0	0			[ACX] ← 00_{16} Clear Accumulator A or B.
	COM	ACX	1	1	X	X	0			[ACX] ← $\overline{[ACX]}$ Complement contents of Accumulator A or B (ones complement).
	NEG	ACX	1	X	X	X	X			[ACX] ← 00_{16} - [ACX] Negate contents of Accumulator A or B (twos complement). Carry status is set if result is 00_{16} and reset otherwise. Overflow status is set if result is 80_{16} and reset otherwise.
	DAA		1	X	X	X	X			Decimal adjust A. Convert contents of A (the binary sum of BCD operands) to BCD format. Carry status is set if value of upper four bits is greater than 9, but not cleared if previously set.
	DEC	ACX	1		X	X	X			[ACX] ← [ACX] - 1 Decrement contents of Accumulator A or B. Overflow status is set if operand was 80_{16} before execution, and cleared otherwise.
	DEX		1		X					[X] ← [X] - 1 Decrement contents of Index register.
	DES		1							[SP] ← [SP] - 1 Decrement contents of Stack Pointer.
	INC	ACX	1		X	X	X			[ACX] ← [ACX] + 1 Increment contents of Accumulator A or B. Overflow status is set if operand was $7F_{16}$ before execution, and cleared otherwise.
	INX		1		X					[X] ← [X] + 1 Increment contents of Index register.
	INS		1							[SP] ← [SP] + 1 Increment contents of Stack Pointer.
	ROL	ACX	1	X	X	X	X			 O ← S∀C [ACX] Rotate Accumulator A or B left through carry.

Table 8-1. A Summary Of The MC6800 Instruction Set (Continued)

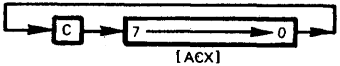
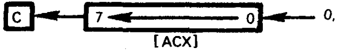
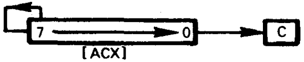
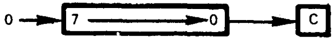
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	O	A_C	I	
REGISTER OPERATE (CONTINUED)	ROR	ACX	1	X	X	X	X			 O ← S⊻C Rotate Accumulator A or B right through carry.
	ASL	ACX	1	X	X	X	X			 O ← S⊻C Arithmetic shift left. Bit 0 is set to 0.
	ASR	ACX	1	X	X	X	X			 O ← S⊻C Arithmetic shift right. Bit 7 stays the same.
	LSR	ACX	1	X	X	0	X			 O ← S⊻C Logical shift right. Bit 7 is set to 0.
	TST	ACX	1	0	X	X	0			$[ACX] - 00_{16}$ Test contents of Accumulator A or B for zero or negative value.
STACK	PSH	ACX	1							[[SP]] ← [ACX] [SP] ← [SP] - 1 Push contents of Accumulator A or B onto top of Stack and decrement Stack Pointer.
	PUL	ACX	1							[SP] ← [SP] + 1 [ACX] ← [[SP]] *Increment Stack Pointer and pull Accumulator A or B from top of Stack.*
	RTS		1							[PC(HI)] ← [[SP] + 1], [PC(LO)] ← [[SP] + 2], [SP] ← [SP] + 2 Return from subroutine. Pull PC from top of Stack and increment Stack Pointer.

Table 8-1. A Summary Of The MC6800 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	O	A_C	I	
INTERRUPT	CLI		1						0	I ←0 Clear interrupt mask to enable interrupts.
	SEI		1						1	I ←1 Set interrupt mask to disable interrupts.
	RTI		1	X	X	X	X	X	X	[SR]—[[SP] + 1], [B]—[[SP] + 2], [A]—[[SP] + 3], [X(HI)]—[[SP] + 4], [X(LO)]—[[SP] + 5], [PC(HI)]—[[SP] + 6], [PC(LO)]—[[SP] + 7], [SP]—[SP] + 7 Return from interrupt. Pull registers from Stack and increment Stack Pointer.
	SWI		1						1	[[SP]]—[PC(LO)], [[SP]-1]—[PC(HI)], [[SP]-2]—[X(LO)], [[SP]-3]—[X(HI)], [[SP]-4]—[A], [[SP]-5]—[B], [[SP]-6]—[SR], [SP]—[SP]-7, [PC(HI)]—[$FFFA_{16}$] [PC(LO)]—[$FFFB_{16}$] Software Interrupt: push registers onto Stack, decrement Stack Pointer, and jump to interrupt subroutine.

Table 8-1. A Summary Of The MC6800 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	Z	S	O	A_C	I	
INTERRUPT (CONTINUED)	WAI		1						1	[[SP]]←[PC(LO)], [[SP]-1]←[PC(HI)], [[SP]-2]←[X(LO)], [[SP]-3]←[X(HI)], [[SP]-4]←[A], [[SP]-5]←[B], [[SP]-6]←[SR], [SP]←[SP]-7 Push registers onto Stack, decrement Stack Pointer, and wiat for interrupt. If [I]=1 when WAI is executed, a non-maskable interrupt is required to exit the Wait state. Otherwise, [I]← 1 when the interrupt occurs.
STATUS	CLC		1	0						C←0 Clear carry
	SEC		1	1						C←1 Set carry
	CLV		1				0			O←0 Clear overflow status bit
	SEV		1				1			O←1 Set overflow status bit
	TAP		1	X	X	X	X	X		[SR]←[A] Transfer contents of Accumulator A to Status register.
	TPA		1							[A]←[SR] Transfer contents of Status register to Accumulator A.
	NOP		1							No Operation

MC6800 SUMMARY OF CYCLE BY CYCLE OPERATION

This table provides a detailed description of the information present on the Address Bus, Data Bus, Valid Memory Address line (VMA), and the Read/Write line (R/W) during each cycle for each instruction.

This information is useful in comparing actual with expected results during debug of both software and hardware as the control program is executed. The information is categorized in groups according to Addressing Mode and Number of Cycles per instruction. (In general, instructions with the same Addressing Mode and Number of Cycles execute in the same manner; exceptions are indicated in the table.)

Table 8-2. Operation Summary

ADDRESS MODE AND INSTRUCTIONS	CYCLES	CYCLE NO.	VMA LINE	ADDRESS BUS	R/W LINE	DATA BUS
IMMEDIATE						
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	2	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Operand Data
CPX LDS LDX	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Operand Data (High Order Byte)
		3	1	Op Code Address + 2	1	Operand Data (Low Order Byte)
DIRECT						
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	1	Address of Operand	1	Operand Data
CPX LDS LDX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	1	Address of Operand	1	Operand Data (High Order Byte)
		4	1	Operand Address + 1	1	Operand Data (Low Order Byte)
STA	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Destination Address
		3	0	Destination Address	1	Irrelevant Data (Note 1)
		4	1	Destination Address	0	Data from Accumulator
STS STX	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	0	Address of Operand	1	Irrelevant Data (Note 1)
		4	1	Address of Operand	0	Register Data (High Order Byte)
		5	1	Address of Operand + 1	0	Register Data (Low Order Byte)
INDEXED						
JMP	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	1	Index Register Plus Offset	1	Operand Data
CPX LDS LDX	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	1	Index Register Plus Offset	1	Operand Data (High Order Byte)
		6	1	Index Register Plus Offset + 1	1	Operand Data (Low Order Byte)

Table 8-2. Operation Summary (Continued)

ADDRESS MODE AND INSTRUCTIONS	CYCLES	CYCLE NO.	VMA LINE	ADDRESS BUS	R/W LINE	DATA BUS
INDEXED (CONTINUED)						
STA	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	0	Index Register Plus Offset	1	Irrelevant Data (Note 1)
		6	1	Index Register Plus Offset	0	Operand Data
ASL LSR ASR NEG CLR ROL COM ROR DEC TST INC	7	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	1	Index Register Plus Offset	1	Current Operand Data
		6	0	Index Register Plus Offset	1	Irrelevant Data (Note 1)
		7	1/0 (Note 3)	Index Register Plus Offset	0	New Operand Data (Note 3)
STS STX	7	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	0	Index Register Plus Offset	1	Irrelevant Data (Note 1)
		6	1	Index Register Plus Offset	0	Operand Data (High Order Byte)
		7	1	Index Register Plus Offset + 1	0	Operand Data (Low Order Byte)
JSR	8	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer	0	Return Address (Low Order Byte)
		5	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		6	0	Stack Pointer – 2	1	Irrelevant Data (Note 1)
		7	0	Index Register	1	Irrelevant Data (Note 1)
		8	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
EXTENDED						
JMP	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Jump Address (High Order Byte)
		3	1	Op Code Address + 2	1	Jump Address (Low Order Byte)
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Operand Data
CPX LDS LDX	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Operand Data (High Order Byte)
		5	1	Address of Operand + 1	1	Operand Data (Low Order Byte)
STA A STA B	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Destination Address (High Order Byte)
		3	1	Op Code Address + 2	1	Destination Address (Low Order Byte)
		4	0	Operand Destination Address	1	Irrelevant Data (Note 1)
		5	1	Operand Destination Address	0	Data from Accumulator
ASL LSR ASR NEG CLR ROL COM ROR DEC TST INC	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Current Operand Data
		5	0	Address of Operand	1	Irrelevant Data (Note 1)
		6	1/0 (Note 3)	Address of Operand	0	New Operand Data (Note 3)

Table 8-2. Operation Summary (Continued)

ADDRESS MODE AND INSTRUCTIONS	CYCLES	CYCLE NO.	VMA LINE	ADDRESS BUS	R/W LINE	DATA BUS
EXTENDED (CONTINUED)						
STS STX	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	0	Address of Operand	1	Irrelevant Data (Note 1)
		5	1	Address of Operand	0	Operand Data (High Order Byte)
		6	1	Address of Operand + 1	0	Operand Data (Low Order Byte)
JSR	9	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Subroutine (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Subroutine (Low Order Byte)
		4	1	Subroutine Starting Address	1	Op Code of Next Instruction
		5	1	Stack Pointer	0	Return Address (Low Order Byte)
		6	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		7	0	Stack Pointer – 2	1	Irrelevant Data (Note 1)
		8	0	Op Code Address + 2	1	Irrelevant Data (Note 1)
		9	1	Op Code Address + 2	1	Address of Subroutine (Low Order Byte)
REGISTER-REGISTER						
ABA DAA SEC ASL DEC SEI ASR INC SEV CBA LSR TAB CLC NEG TAP CLI NOP TBA CLR ROL TPA CLV ROR TST COM SBA	2	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
DES DEX INS INX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Previous Register Contents	1	Irrelevant Data (Note 1)
		4	0	New Register Contents	1	Irrelevant Data (Note 1)
PSH	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	1	Stack Pointer	0	Accumulator Data
		4	0	Stack Pointer – 1	1	Accumulator Data
PUL	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Stack Pointer	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer + 1	1	Operand Data from Stack
TSX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Stack Pointer	1	Irrelevant Data (Note 1)
		4	0	New Index Register	1	Irrelevant Data (Note 1)
TXS	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Index Register	1	Irrelevant Data
		4	0	New Stack Pointer	1	Irrelevant Data
RTS	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (Note 2)
		3	0	Stack Pointer	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer + 1	1	Address of Next Instruction (High Order Byte)
		5	1	Stack Pointer + 2	1	Address of Next Instruction (Low Order Byte)

Table 8-2. Operation Summary (Continued)

ADDRESS MODE AND INSTRUCTIONS	CYCLES	CYCLE NO.	VMA LINE	ADDRESS BUS	R/W LINE	DATA BUS
REGISTER-REGISTER (CONTINUED)						
WAI	9	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	1	Stack Pointer	0	Return Address (Low Order Byte)
		4	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		5	1	Stack Pointer – 2	0	Index Register (Low Order Byte)
		6	1	Stack Pointer – 3	0	Index Register (High Order Byte)
		7	1	Stack Pointer – 4	0	Contents of Accumulator A
		8	1	Stack Pointer – 5	0	Contents of Accumulator B
		9	1	Stack Pointer – 6 (Note 4)	1	Contents of Cond. Code Register
RTI	10	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (Note 2)
		3	0	Stack Pointer	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer + 1	1	Contents of Cond. Code Register from Stack
		5	1	Stack Pointer + 2	1	Contents of Accumulator B from Stack
		6	1	Stack Pointer + 3	1	Contents of Accumulator A from Stack
		7	1	Stack Pointer + 4	1	Index Register from Stack (High Order Byte)
		8	1	Stack Pointer + 5	1	Index Register from Stack (Low Order Byte)
		9	1	Stack Pointer + 6	1	Next Instruction Address from Stack (High Order Byte)
		10	1	Stack Pointer + 7	1	Next Instruction Address from Stack (Low Order Byte)
SWI	12	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (Note 1)
		3	1	Stack Pointer	0	Return Address (Low Order Byte)
		4	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		5	1	Stack Pointer – 2	0	Index Register (Low Order Byte)
		6	1	Stack Pointer – 3	0	Index Register (High Order Byte)
		7	1	Stack Pointer – 4	0	Contents of Accumulator A
		8	1	Stack Pointer – 5	0	Contents of Accumulator B
		9	1	Stack Pointer – 6	0	Contents of Cond. Code Register
		10	0	Stack Pointer – 7	1	Irrelevant Data (Note 1)
		11	1	Vector Address FFFA (Hex)	1	Address of Subroutine (High Order Byte)
		12	1	Vector Address FFFB (Hex)	1	Address of Subroutine (Low Order Byte)
RELATIVE						
BCC BHI BNE BCS BLE BPL BEQ BLS BRA BGE BLT BVC BGT BMI BVS	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Branch Offset
		3	0	Op Code Address + 2	1	Irrelevant Data (Note 1)
		4	0	Branch Address	1	Irrelevant Data (Note 1)
BSR	8	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Branch Offset
		3	0	Return Address of Main Program	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer	0	Return Address (Low Order Byte)
		5	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		6	0	Stack Pointer – 2	1	Irrelevant Data (Note 1)
		7	0	Return Address of Main Program	1	Irrelevant Data (Note 1)
		8	0	Subroutine Address	1	Irrelevant Data (Note 1)

Note 1. If device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.

Note 2. Data is ignored by the MPU.

Note 3. For TST, VMA = 0 and Operand data does not change.

Note 4. While the MPU is waiting for the interrupt, Bus Available will go high indicating the following states of the control lines: VMA is low; Address Bus, R/W, and Data Bus are all in the high impedance state.

The following codes are used in Table 8-3:

aa two bits choosing the address mode:
00 immediate data
01 base page direct addressing
10 indexed addressing
11 extended direct addressing

pp the second byte of a two- or three-byte instruction.

qq the third byte of a three-byte instruction.

x one bit choosing the Accumulator:
0 Accumulator A
1 Accumulator B

yy two bits choosing the address mode:
00 (inherent addressing) Accumulator A
01 (inherent addressing) Accumulator B
10 indexed addressing
11 extended direct addressing

y one bit choosing the address mode:
0 indexed addressing
1 extended direct addressing

Two numbers in the "Machine Cycles" column (for example, 2 - 5) indicate that execution time depends on the addressing mode.

Table 8-3. MC6800 Instruction Set Object Codes

MNEMONIC	OPERAND(S)	OBJECT CODE	BYTE	MACHINE CYCLES
ABA		1B	1	2
ADC	ACX,	1xaa1001		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4
ADD	ACX,	1xaa1011		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4
AND	ACX,	1xaa0100		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4
ASL	ACX	01yy1000	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
ASR	ACX	01yy0111	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
BCC	DISP	24 pp	2	4
BCS	DISP	25 pp	2	4
BEQ	DISP	27 pp	2	4
BGE	DISP	2C pp	2	4
BGT	DISP	2E pp	2	4
BHI	DISP	22 pp	2	4
BIT	ACX,	1xaa0101		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4

Table 8-3. MC6800 Instruction Set Object Codes (Continued)

MNEMONIC	OPERAND(S)	OBJECT CODE	BYTE	MACHINE CYCLES
BLE	DISP	2F pp	2	4
BLS	DISP	23 pp	2	4
BLT	DISP	2D pp	2	4
BMI	DISP	2B pp	2	4
BNE	DISP	26 pp	2	4
BPL	DISP	2A pp	2	4
BRA	DISP	20 pp	2	4
BSR	DISP	8D pp	2	8
BVC	DISP	28 pp	2	4
BVS	DISP	29 pp	2	4
CBA		11	1	2
CLC		0C	1	2
CLI		0E	1	2
CLR	ACX	01yy1111	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
CLV		0A	1	2
CMP	ACX,	1xaa0001		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4
COM	ACX	01yy0011	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
CPX		10aa1100		
	ADR8	pp	2	4-6
	ADR16 or DATA16	qq	3	3-5
DAA		19	1	2
DEC	ACX	01yy1010	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
DES		34	1	4
DEX		09	1	4
EOR	ACX,	1xaa1000		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4
INC	ACX	01yy1100	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
INS		31	1	4
INX		08	1	4
JMP		011y1110		
	ADR8	pp	2	4
	ADR16	qq	3	3
JSR		101y1101		
	ADR8	pp	2	8
	ADR16	qq	3	9
LDA	ACX,	1xaa0110		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4
LDS		10aa1110		
	ADR8	pp	2	3-5
	ADR16 or DATA16	qq	3	4-6

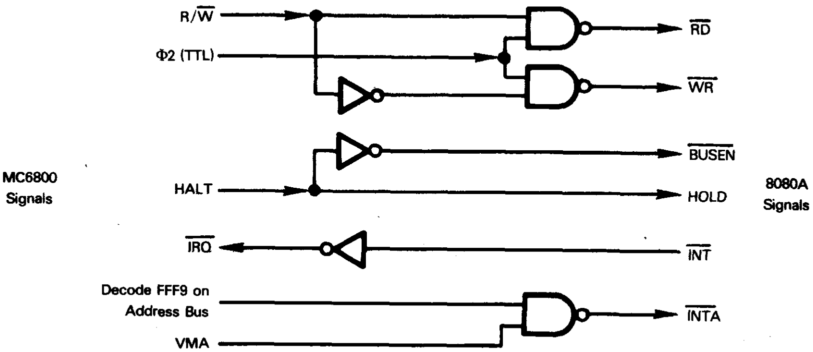
Table 8-3. MC6800 Instruction Set Object Codes (Continued)

MNEMONIC	OPERAND(S)	OBJECT CODE	BYTE	MACHINE CYCLES
LDX		11aa1110		
	ADR8	pp	2	3-5
	ADR16 or DATA16	qq	3	4-6
LSR	ACX	01yy0100	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
NEG	ACX	01yy0000	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
NOP		01	1	2
ORA	ACX,	1xaa1010		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4
PSH	ACX	0011011x	1	4
PUL	ACX	0011001x	1	4
ROL	ACX	01yy1001	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
ROR	ACX	01yy0110	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
RTI		3B	1	10
RTS		39	1	5
SBA		10	1	2
SBC	ACX,	1xaa0010		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4
SEC		0D	1	2
SEI		0F	1	2
SEV		0B	1	2
STA	ACX,	1xaa0111	*	
	ADR8	pp	2	4-6
	ADR16	qq	3	5
STS		10aa1111	*	
	ADR8	pp	2	5-7
	ADR16	qq	3	6
STX		11aa1111	*	
	ADR8	pp	2	5-7
	ADR16	qq	3	6
SUB	ACX,	1xaa0000		
	ADR8 or DATA	pp	2	2-5
	ADR16	qq	3	4
SWI		3F	1	12
TAB		16	1	2
TAP		06	1	2
TBA		17	1	2
TPA		07	1	2
TST	ACX	01yy1101	1	2
	ADR8	pp	2	7
	ADR16	qq	3	6
TSX		30	1	4
TXS		35	1	4
WAI		3E	1	9

*aa = 00 is not permitted.

SUPPORT DEVICES THAT MAY BE USED WITH THE MC6800

Using 8080A support devices with the MC6800 is very straightforward in terms of control signals generated. You must break out the single MC6800 R/$\overline{W}$ control signal into separate $\overline{RD}$ and $\overline{WR}$ control signals. Other signal interconnections are self-evident. Here is appropriate logic:



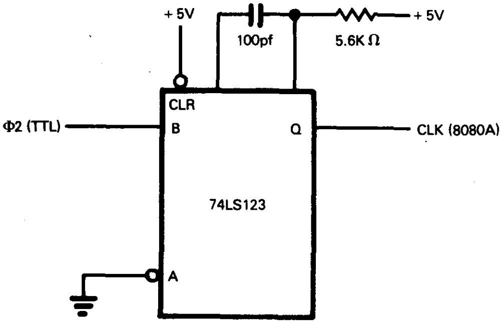
Signals illustrated above apply to communications between the MC6800 CPU and 8080A support devices. External memory will communicate with the MC6800 CPU using standard MC6800 timing.

There are some limitations imposed on communications between the MC6800 CPU and 8080A support devices.

As illustrated above, you must create an interrupt acknowledge control signal by decoding the second interrupt acknowledge address, $FFF9_{16}$, appearing on the Address Bus. Similarly, if you wish to create specific I/O read and write control signals, then you must decode off the Address Bus those memory addresses which you have assigned to I/O devices.

If you wish to extend instruction execution cycles for slow 8080A support devices, then you must use the MC6800 clock stretching logic for this purpose. Clearly the 8080A support devices cannot use Wait state logic since the MC6800 has no such logic.

You can generate an 8080A compatible system clock from the Φ2 (TTL) 6870 series clock as follows:



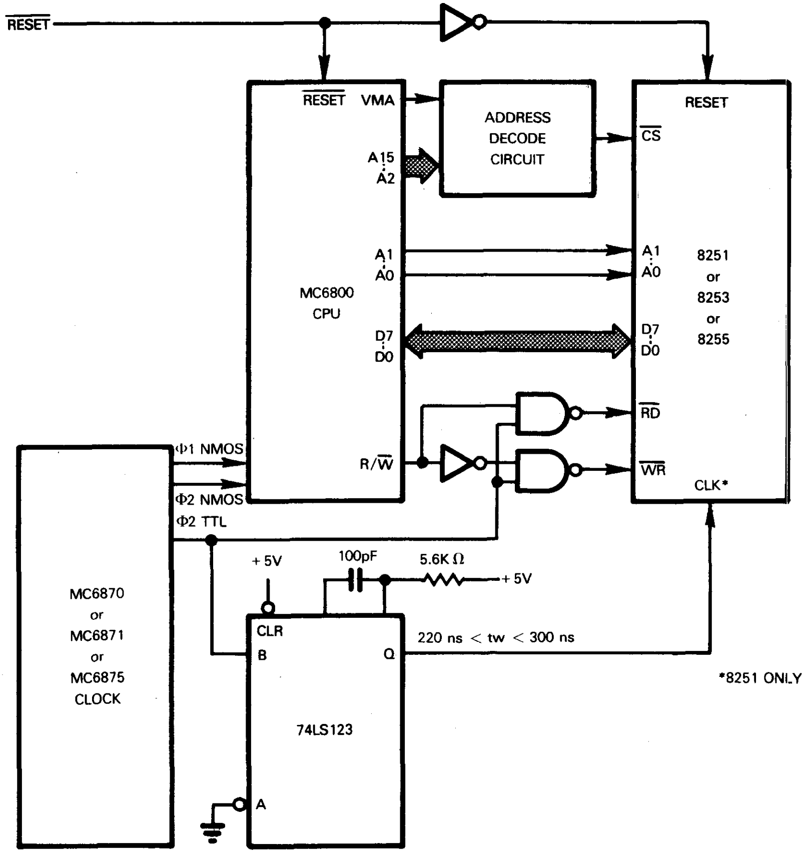


Figure 8-11. Use Of 8080A Support Devices With MC6800 CPU

Figure 8-11 illustrates the interface for an 8251, an 8253 or an 8255 device connected to an MC6800 CPU. Figure 8-12 provides the timing for 8080A support devices used with an MC6800 CPU.

The 8257 DMA device and the 8259 PICU should not be used in an MC6800 since MC6800 DMA and interrupt logic are not compatible with these devices.

8085 support devices could be used with an MC6800 but **would require that you multiplex the Data Bus and low order eight Address Bus lines,** as required by the 8155, 8355, and 8755. Extra logic needed to perform this bus multiplexing would probably destroy the cost effectiveness of the 8085 support devices in an MC6800 system.

The only Z80 support device that is practical in an MC6800 system is the Z80 DMA device. This is because the other Z80 support devices decode a Write state from a combination of the $\overline{\text{M1}}$, $\overline{\text{INT}}$, and $\overline{\text{RD}}$ control signals. The Z80 DMA device uses separate read and write control inputs; therefore it may be used with an MC6800 CPU. The logic needed to create Z80 DMA control inputs from MC6800 control signals is identical to the 8080A control signal logic illustrated above.

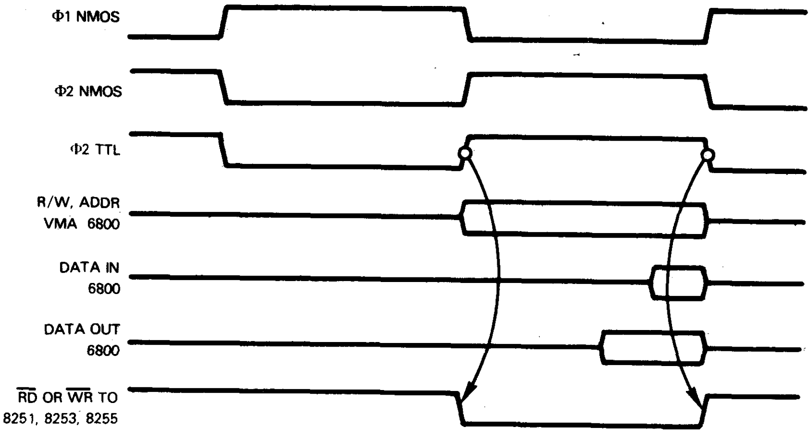


Figure 8-12. Timing For 8080A Support Devices Used With An MC6800 CPU

When using non-MC6800 support devices with the MC6800 CPU, remember that there is a particularly pernicious problem associated with MC6800 Reset logic on power-up. As discussed earlier in this chapter, the MC6800 does not internally disable interrupt requests until the trailing low-to-high transition of the RESET signal. Thus external devices capable of requesting an interrupt may randomly do so during the power on Reset sequence; and this may result in an interrupt being acknowledged following the initial system Reset, rather than the expected system initialization program getting executed. You must make certain that all support devices capable of requesting an interrupt are disabled by the leading high-to-low transition of RESET during the power-up sequence.

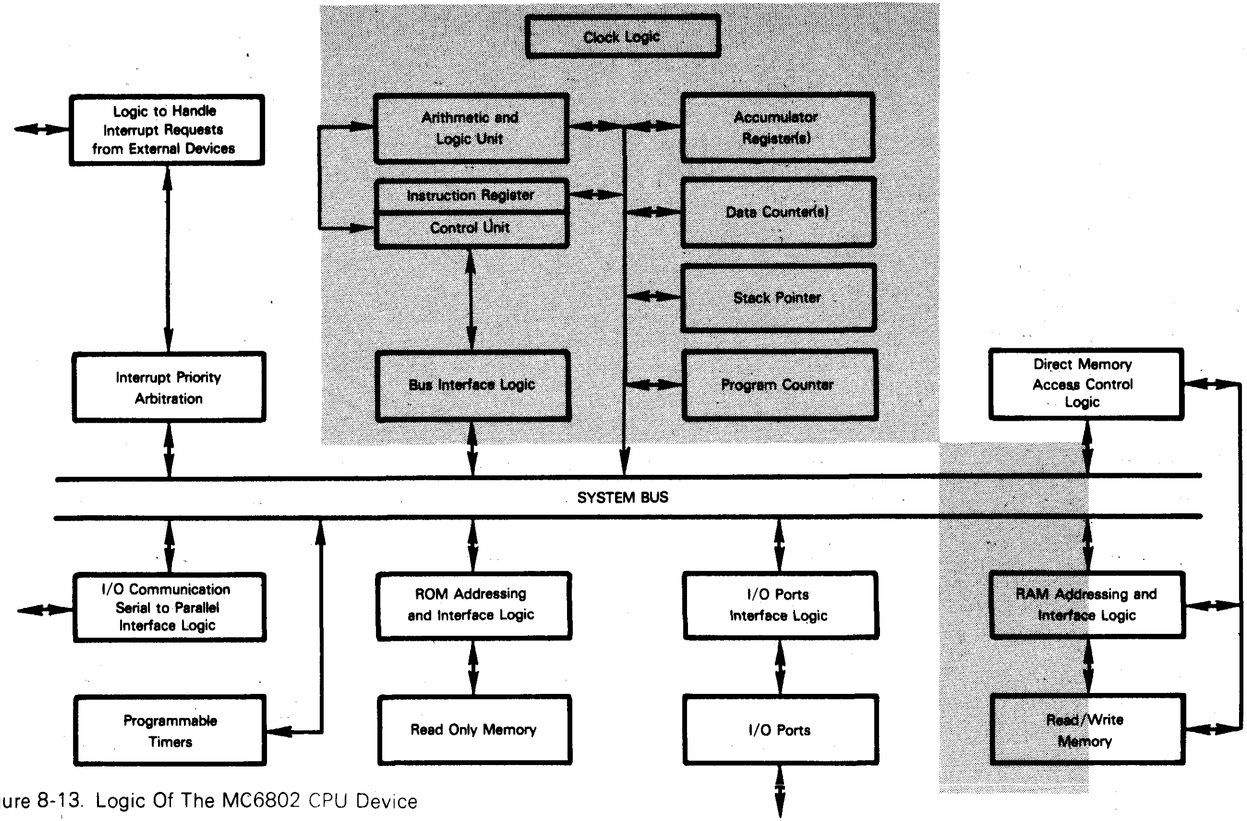
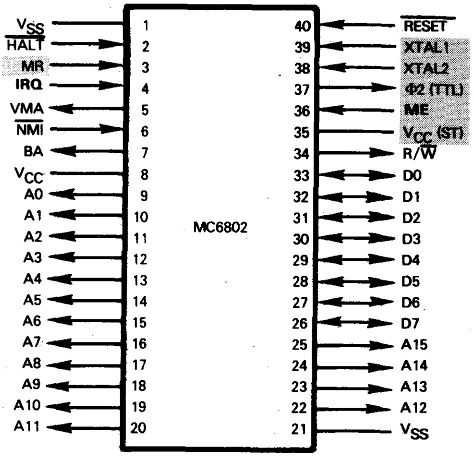

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 8-13. Logic Of The MC6802 CPU Device



PIN NAME	DESCRIPTION	TYPE
*A0 - A15	Address Lines	Tristate, Output
*D0 - D7	Data Bus Lines	Tristate, Bidirectional
*$\overline{\text{HALT}}$	Halt	Input
*MR	Memory Ready	Input
*ME	RAM Enable	Input
*R/$\overline{\text{W}}$	Read/Write	Tristate, Output
*VMA	Valid Memory Address	Output
*BA	Bus Available	Output
*IRQ	Interrupt Request	Input
$\overline{\text{RESET}}$	Reset	Input
$\overline{\text{NMI}}$	Non-Maskable Interrupt	Input
XTAL1,XTAL2	Crystal/Clock Connections	Input
Φ2 (TTL)	Clock Signal	Output
V_{SS},V_{CC}	Power	
V_{CC} (ST)	Standby Power	

*These signals connect to the System Bus.

Figure 8-14. MC6802 CPU Signals And Pin Assignments

THE MC6802 CPU WITH READ/WRITE MEMORY

The MC6802 is a combination of the MC6800 CPU, clock logic, and 128 bytes of read/write memory. Figure 8-13 illustrates logic of the MC6802 CPU device.

The actual CPU architecture and the instruction set of the MC6802 are identical to the MC6800 which we have already described.

The 128 bytes of read/write memory which are present on the MC6802 chip are accessed by memory addresses 0000_{16} through $007F_{16}$. The first 32 bytes of this read/write memory may be protected during power down by a special low power standby input.

MC6802 CPU pins and signals are illustrated in Figure 8-14. Pins and signals which differ from the MC6800 illustrated in Figure 8-2 are shaded. We will examine these new signals only.

Since clock logic is on the MC6802 chip, three pins are needed for this specific purpose. Normally **a crystal will be connected across XTAL1 and XTAL2.** A 4 MHz crystal should be used since the MC6802 has internal divide-by-four logic to create a 1 MHz system clock signal. **A TTL level system clock signal is output via Φ2 (TTL).**

You can, if you wish, drive the MC6802 using **an external clock signal;** this signal **is input via XTAL2;** it must not be faster than 1 MHz.

In order to provide the clock stretching logic that is a standard part of MC6800 microcomputer system, **a Memory Ready (MR) signal is present.** MR is normally low. When MR is high, clock logic stops within the MC6800 and the level of the Φ2 output remains unaltered.

Two signals have been added to support the on-chip read/write memroy. **ME is an enable signal for the on-chip memory.** ME must be input high for the on-chip memory to be accessed. If ME is low, on-chip memory cannot be written into or read. While on-chip memory is disabled its address space is also disabled, and addresses in the range 0000_{16} through $007F_{16}$ are deflected to external memory. Thus **the address space 0000_{16} through $007F_{16}$ is duplicated;** it accesses on-chip RAM when ME is high, but it accesses external RAM when ME is low.

The ME signal is also used during the power down sequence. ME must be low at least three clock signals before V_{CC} goes below +4.75V during power down.

V_{CC}(ST) is a new power input which is used to retain the contents of the 32 low order on-chip read/write memory bytes.

MC6800 signals which have been removed, going to the MC6802, **include the clock inputs Φ1 and Φ2,** plus the bus control signals **TSC and DBE.**

Obviously, the clock inputs must be removed since clock logic is now on the CPU chip.

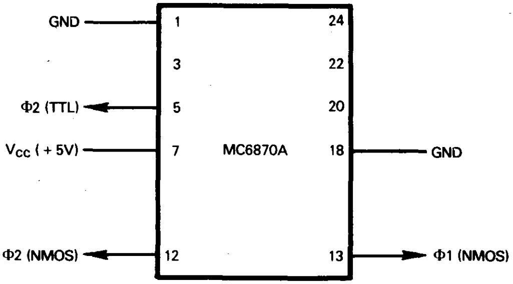
Removal of the System Bus control signals TSC and DBE reflects the fact that if you are going to need direct memory access, you are not going to use the MC6802. Only larger microcomputer systems need direct memory access; for such systems the MC6800 is available. The MC6802 is intended as half of a two-chip 6800 configuration, within which direct memory access would be meaningless.

Ultimately **a multifunction support device, tentatively identified as the MC6846, will be available to support the MC6802.** The MC6846 will provide read only memory, I/O ports and a timer. The MC6846 has not yet been characterized; therefore, it is not described in this book.

THE MC6870 TWO PHASE CLOCKS

Four clock logic devices supporting the MC6800 CPU are described.

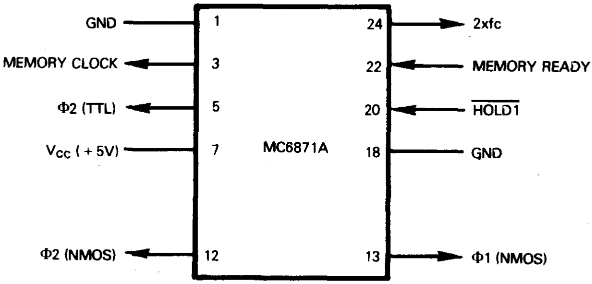
The MC6870A is a very elementary device providing minimum clock signals needed with an MC6800 microcomputer system. Its pin assignments are illustrated in Figure 8-15.



Pin Name	Description	Type
Φ1 (NMOS)	Φ1 Clock to MC6800	Output
Φ2 (NMOS)	Φ2 Clock to MC6800	Output
Φ2 (TTL)	Φ2 Clock to microcomputer system	Output
V_{CC}, GND	Power and Ground	

Figure 8-15. MC6870A Clock Device Pins And Signals

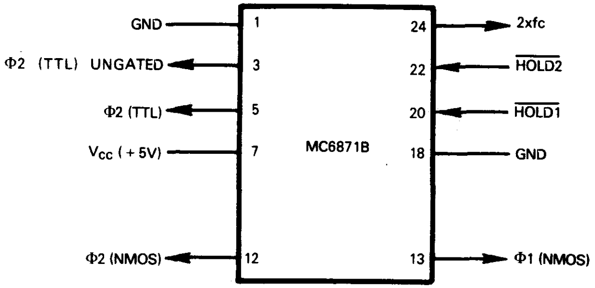
The first enhancement is provided by the MC6871A, illustrated in Figure 8-16, which adds clock signal stretching capabilities and a twice frequency clock output.



Pin Name	Description	Type
Φ1 (NMOS)	Φ1 Clock to MC6800	Output
Φ2 (NMOS)	Φ2 Clock to MC6800	Output
Φ2 (TTL)	Φ2 Clock to microcomputer system	Output
MEMORY CLOCK	Select to memory devices	Output
2xfc	Twice frequency clock	Output
$\overline{\text{HOLD1}}$	Stretch Φ1 high control	Input
MEMORY READY	Stretch Φ1 low control	Input
V_{CC},GND	Power and Ground	

Figure 8-16. MC6871A Clock Device Pins And Signals

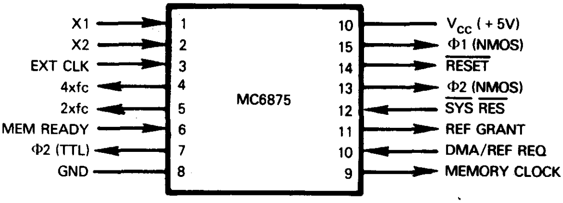
The MC6871B, illustrated in Figure 8-17, is a variation of the MC6871A.



Pin Name	Description	Type
Φ1 (NMOS)	Φ1 Clock to MC6800	Output
Φ2 (NMOS)	Φ2 Clock to MC6800	Output
Φ2 (TTL)	Φ2 Clock to microcomputer system	Output
Φ2 (TTL) UNGATED	Free-running Φ2 (TTL)	Output
2xfc	Twice frequency clock	Output
$\overline{\text{HOLD1}}$	Stretch Φ1 high control	Input
$\overline{\text{HOLD2}}$	Stretch Φ1 low control	Input
V_{CC}, GND	Power and Ground	

Figure 8-17. MC6871B Clock Device Pins And Signals

The MC6875 is the most versatile of the clock devices provided for the MC6800. It is illustrated in Figure 8-18.



Pin Name	Description	Type
Φ1 (NMOS)	Φ1 Clock to MC6800	Output
Φ2 (NMOS)	Φ2 Clock to MC6800	Output
Φ2 (TTL)	Φ2 Clock to microcomputer system	Output
MEMORY CLOCK	Free-running Φ2 (TTL)	Output
2xfc	Twice frequency clock	Output
4xfc	Four Times frequency clock	Output
$\overline{\text{DMA/REF REQ}}$	Stretch Φ1 high control	Input
REF GRANT	Stretch Φ1 high acknowledge	Output
MEM READY	Stretch Φ1 low control	Input
$\overline{\text{SYS RES}}$	Asynchronous system reset control	Input
$\overline{\text{RESET}}$	Synchronous reset control	Output
EXT CLK	External synchronization control	Input
X1, X2	External crystal connections	
V_{CC}, GND	Power and Ground	

Figure 8-18. MC6875 Clock Device Pins And Signals

Since these various clock logic devices represent essentially the same capabilities, but with increasing enhancements, we will describe logic and capabilities in the order of the device illustrations.

Much of the clock device logic we are going to describe stretches the Φ1 (NMOS) and Φ2 (NMOS) clock signals. But recall that stretching Φ1 (NMOS) and Φ2 (NMOS), *in itself, is only half of the logic needed to stretch the entire System Bus.* Additionally, the MC6800 needs a high TSC input to float the Address and R/$\overline{\text{W}}$ Bus lines while Φ1 (NMOS) is high. DBE must be input low in order to float the Data Bus lines while the clock is being stretched with Φ1 (NMOS) low.

THE MC6870A CLOCK DEVICE

This is a minimum clock device; it outputs Φ1 (NMOS) and Φ2 (NMOS), the two clock signals required by an MC6800 CPU.

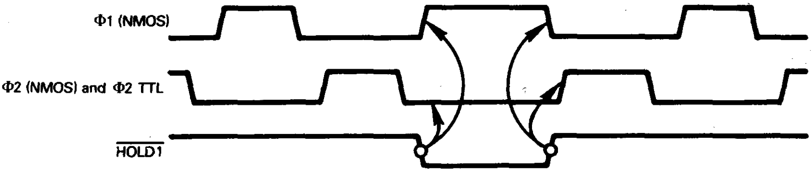
Φ2 (TTL) is alsp generated. Φ2 (TTL) is used to synchronize support devices; it has sufficient load capacity to drive five devices without signal buffering.

The MC6870A contains an internal crystal and oscillator; in its standard form clock signals with a 1 MHz frequency are generated. A variety of other clock frequencies can also be ordered.

THE MC6871A CLOCK DEVICE

In addition to the standard signals output by the MC6870A, the MC6871A provides two additional TTL output clock signals and externally controlled pulse stretching capabilities.

$\overline{\text{HOLD1}}$ is used to stretch the standard clock signals: Φ1 (NMOS), Φ2 (NMOS) and Φ2 (TTL), which we described for the MC6870A. Timing may be illustrated as follows:



It is very important that $\overline{\text{HOLD1}}$ makes its active high-to-low transition during a Φ1 (NMOS) high state. Subsequently, Φ1 (NMOS), Φ2 TTL clocks will be stretched until $\overline{\text{HOLD1}}$ makes a low-to-high transition.

$\overline{\text{HOLD1}}$ must make its low-to-high transition during the time when Φ1 would ordinarily be high — that is, if it had not been stretched. External logic can use the signals MEMORY CLOCK and 2xfc to synchronize this transition. As a result, it is possible for a maximum delay of one-half clock cycle to elapse between $\overline{\text{HOLD1}}$ going high and clock signals resuming their normal waveform.

As illustrated above, $\overline{\text{HOLD1}}$ stretches clocks with Φ1 (NMOS) high. If you refer back to our discussion of the MC6800, you will see that these clock levels identify the portion of a machine cycle when an address is being output. Typically, the clock will be stretched so that two addresses can be output: the first for a Direct Memory Access or dynamic memory refresh operation; the second for the normal address output which is required when any instruction is executed. Device select logic must discriminate between the two addresses being output; DMA or dynamic memory refresh logic must receive the first address only, while memory or I/O devices receive the second address only.

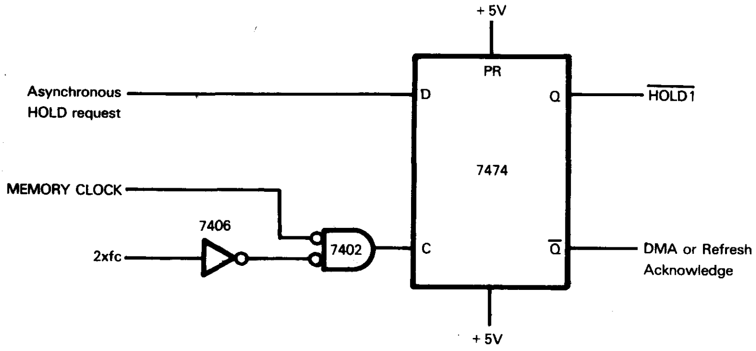
MC6800 STRETCHING ADDRESS TIMING

Two additional clock signals are output by the MC6871A: 2xfc and MEMORY CLOCK; they are not part of normal memory addressing logic, therefore these two clock signals are not stretched by $\overline{\text{HOLD1}}$.

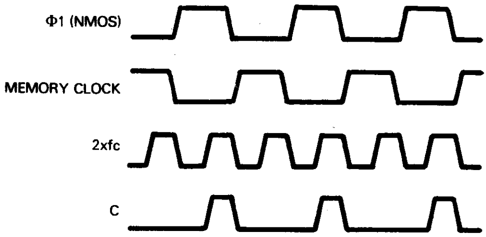
2xfc is a twice frequency clock signal which can be used for various synchronization logic around an MC6800 microcomputer system.

MEMORY CLOCK is identical in waveform to Φ2 TTL except MEMORY CLOCK is not stretched by $\overline{\text{HOLD1}}$.

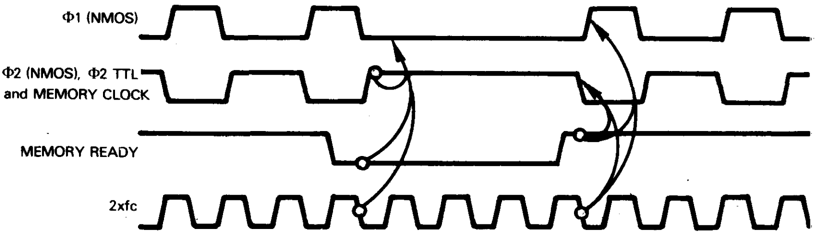
Notice that $\overline{\text{HOLD1}}$ must make its high-to-low transition while Φ1 (NMOS) is high. An asynchronous HOLD request must therefore be synchronized with Φ1 (NMOS) in order to generate a $\overline{\text{HOLD1}}$ MC6871A clock device input. This is a simple logic operation; here is one possibility:


+5V
PR
Asynchronous HOLD request
D
Q
$\overline{\text{HOLD1}}$
7474
MEMORY CLOCK
7406
2xfc
7402
C
$\overline{\text{Q}}$
DMA or Refresh Acknowledge
+5V

This circuit also ensures that the low-to-high transition of $\overline{\text{HOLD1}}$ will occur at a time when Φ1 (NMOS) would be high even if it were not stretched. The low-to-high clock transition occurs only during Φ1 (NMOS) high time:



MEMORY READY also stretches clock signals. Timing may be illustrated as follows:



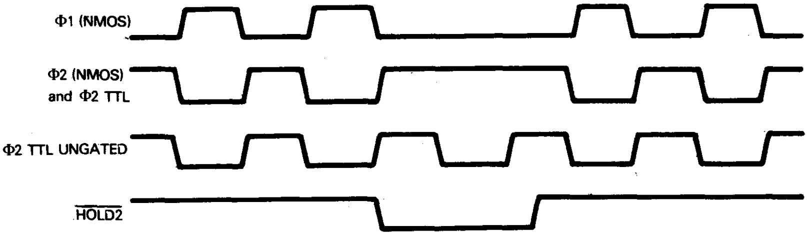
Clock signal stretching begins with Φ2 (NMOS) high following the MEMORY READY high-to-low transition. Clock stretching ends with the falling edge of 2xfc following the MEMORY READY low-to-high transition. Observe that MEMORY READY stretches MEMORY CLOCK, which HOLD1 does not do. 2xfc, however, is not stretched, either by HOLD1 or by MEMORY READY Also note that MEMORY READY does not require input synchronization, as does HOLD1.

If you refer back to the timing diagrams which illustrate MC6800 instructions' execution, you will see that MEMORY READY stretches clock signals during the data access portion of a machine cycle. This is the part of the machine cycle during which external memory has to respond to a CPU access; therefore, this is the portion of the machine cycle which must be stretched for slow memories — which is why MEMORY READY can be visualized as the signal which slow memories must input low in order to gain the access time they require.

The MC6871A contains an internal crystal oscillator. In its standard form, clock signals with a 1 MHz frequency are generated. A variety of other clock frequencies can also be ordered.

THE MC6871B CLOCK DEVICE

This device differs from the MC6871A in two ways. MEMORY READY is replaced by HOLD2 and MEMORY CLOCK is replaced by Φ2 (TTL) UNGATED. HOLD2 stretches clock signals with Φ1 (NMOS) low, just as MEMORY READY did; however, like HOLD1, HOLD2 must have its active transitions synchronized with the clock output — in this case with Φ2 high. Φ2 (TTL) UNGATED, however, is not stretched. Timing may be illustrated as follows:



THE MC6875 CLOCK DEVICE

This is the most sophisticated of the clock devices offered with the MC6800 microcomputer system. Its principal features are that it performs control input synchronization which must be handled externally by other clock devices; also, the MC6875 allows external timing.

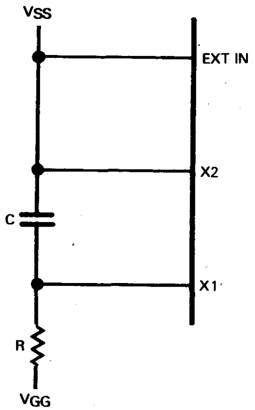
As we have already stated, clock signals are stretched with Φ1 and Φ2 low in order to allow a Direct Memory Access or dynamic memory refresh address to be output. **The MC6875 DMA/REF REQ input performs this clock stretching operation, just as HOLD1 does, except that DMA/REF REQ can be an asynchronous input.** MC6875 internal logic performs the synchronization operations which have to be handled externally for the MC6871A and MC6871B clocks. In addition, the MC6875 outputs REF GRANT high while the clocks are being stretched with Φ1 (NMOS) high. External DMA or dynamic memory refresh logic can use REF GRANT as an enable strobe.

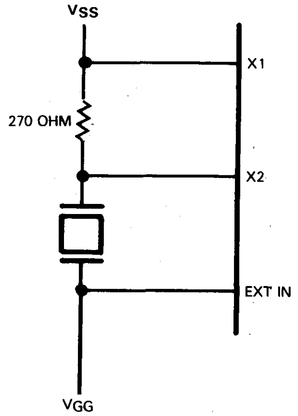
MEMORY READY and MEMORY CLOCK are as described for the MC6871A. MEMORY READY stretches clocks with Φ1 (NMOS) low. MEMORY CLOCK follows Φ2 (NMOS) and is stretched by MEMORY READY but not by DMA/REF REQ.

The MC6875 clock signal outputs Φ1 (NMOS) and Φ2 (NMOS) have sufficient capacity to drive two MC6800 CPUs. 4xfc is an additional oscillator running at four times the Φ1 and Φ2 clock rates.

X1, X2 and EXT IN are three signals which allow MC6875 clock rates to be controlled externally.

You can optionally attach a crystal oscillator or an RC network to X1, X2 as follows:





You can also input an external clock signal to EXT IN, in which case the MC6875 will adopt the frequency of the external signal. The external clock frequency must be four times the Φ1 and Φ2 clock frequency.

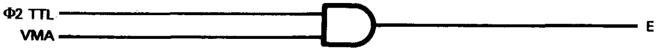
The MC6875 is able to take a ragged SYSTEM RESET input and convert it into a crisp RESET, which may be used throughout an MC6800 microcomputer system. SYSTEM RESET can be any input signal which is processed through a Schmitt trigger to create a RESET output, as described for the 8224 clock device in Chapter 4.

SOME STANDARD CLOCK SIGNAL INTERFACE LOGIC

There are a number of very common ways in which MC6870 series clock signals are used within MC6800 microcomputer systems.

You will find that all of the support devices described in the rest of this chapter require an enable synchronizing signal, given the symbol "E". This signal is usually generated as the AND of the MC6800 VMA output and the Φ2 TTL clock output:

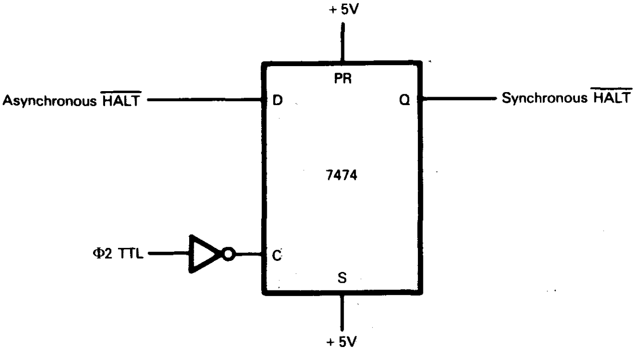
MC6800 ENABLE SIGNAL GENERATION



The purpose of ANDing Φ2 with VMA is to make sure that devices receiving signal E are inhibited while VMA is low — at which time the CPU cannot be accessing the support device.

The $\overline{\text{HALT}}$ signal, which is used in MC6800 microcomputer systems to float the System Bus for extended periods, must be a synchronous input. **You can create a synchronous $\overline{\text{HALT}}$ from an asynchronous $\overline{\text{HALT}}$ using Φ2 TTL as follows:**

MC6800 SYNCHRONOUS HALT GENERATION


+5V
PR
Asynchronous $\overline{\text{HALT}}$
D
Q
Synchronous $\overline{\text{HALT}}$
7474
Φ2 TTL
C
S
+5V

THE MC6820 AND MCS6520 PERIPHERAL INTERFACE ADAPTER (PIA)

This part is manufactured as the MC6820 by the companies listed at the beginning of this chapter. MOS Technology and its second source companies (whose products are described in Chapter 9) manufacture the same part, but call it the MCS6520.

The MC6820 PIA is a general purpose I/O device, designed for use within MC6800 microcomputer systems.

The MC6820 PIA provides 16 I/O pins, configured as two 8-bit I/O ports. We will refer to these as Port A and Port B. Individual pins of each I/O port may be used separately as inputs or outputs. Each I/O port has two associated control signals, one of which is input only, while the other is bidirectional. The only differences between I/O Ports A and B are in their electrical characteristics, and in their handshaking control capabilities. But these are very significant differences, as we will explain shortly.

Figure 8-19 illustrates that part of our general microcomputer system logic which has been implemented on the MC6820 PIA.

The MC6820 PIA is packaged as a 40-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL compatible.

The device is implemented using N-channel silicon gate MOS technology.

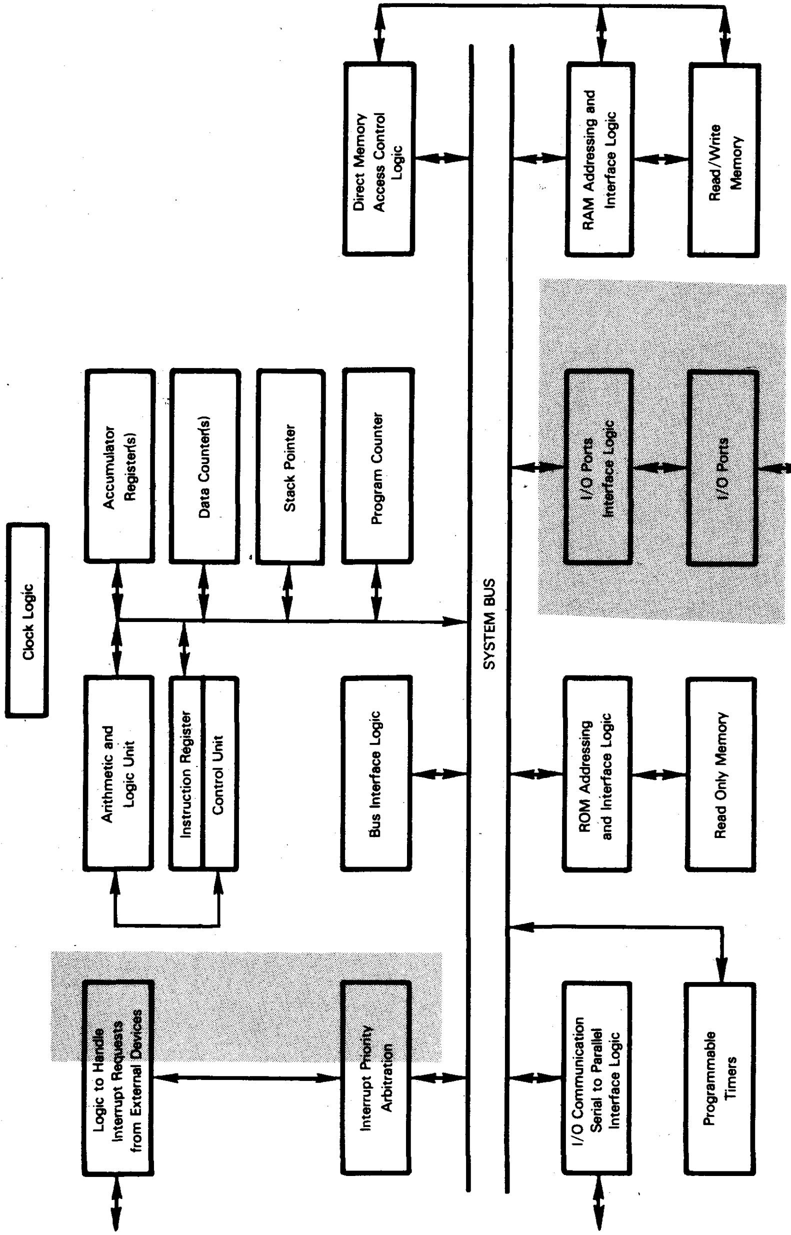
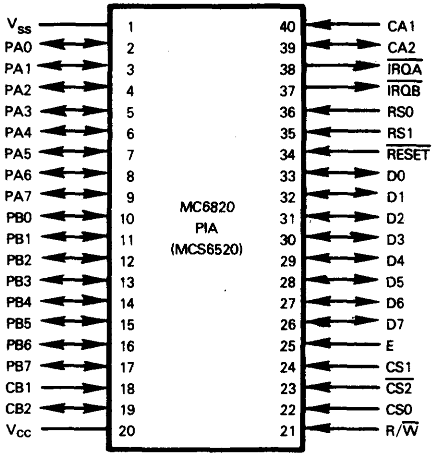

Clock Logic
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Direct Memory Access Control Logic
Arithmetic and Logic Unit
Instruction Register
Control Unit
Bus Interface Logic
Logic to Handle Interrupt Requests from External Devices
Interrupt Priority Arbitration
SYSTEM BUS
RAM Addressing and Interface Logic
Read/Write Memory
I/O Ports Interface Logic
I/O Ports
ROM Addressing and Interface Logic
Read Only Memory
I/O Communication Serial to Parallel Interface Logic
Programmable Timers

Figure 8-19. Logic Of The MC6820 PIA



Pin Name	Description	Type
D0 - D7	Data Bus to CPU	Tristate, bidirectional
PA0 - PA7	Port A peripheral Data Bus	Input or Output
PB0 - PB7	Port B peripheral Data Bus	Tristate, Input or Output
CS0, CS1, $\overline{\text{CS2}}$	Chip Select	Input
RS0, RS1	Register Select	Input
CA1	Interrupt input to Port A	Input
CA2	Port A peripheral control	Input or Output
CB1	Interrupt input to Port B	Input
CB2	Port B peripheral control	Input or Output
E	Device synchronization	Input
R/$\overline{\text{W}}$	Read/Write control	Input
$\overline{\text{IRQA}}$, $\overline{\text{IRQB}}$	Interrupt request	Output
$\overline{\text{RESET}}$	Reset	Input
V_{CC}, V_{SS}	Power and Ground	

Figure 8-20. MC6820 PIA Signals And Pin Assignments

THE MC6820 PIA PINS AND SIGNALS

The MC6820 pins and signals are illustrated in Figure 8-20. We will summarize signal functions before describing PIA operations.

Consider first the various Data Busses.

D0 - D7 represents the bidirectional Data Bus via which all communications between the CPU and the MC6820 occur.

PA0 - PA7 and PB0 - PB7 represent Data Busses connecting the two 8-bit I/O Ports A and B with external logic. The 16 I/O port pins may be looked upon as 16 individual signal lines, or two 8-bit I/O busses. Each I/O port pin can be individually assigned to input or output, but an individual pin cannot support bidirectional data transfers.

These are the differences between I/O Port A and B pins:

1) An I/O Port B pin which has been assigned to output will enter a tristate condition during an input operation; this is not the case for an I/O Port A pin.
2) Bits of I/O Port A may be set or reset at any time by voltage levels applied to associated pins. Irrespective of data that may be in a bit position following a Read or Write operation, an I/O Port A bit will be reset to zero any time a voltage of +0.8V or less is applied to a Port A pin. A 1 will be written into a Port A bit any time a voltage of +2V or more is applied to the Port A pin. I/O Port B bit contents are not affected by voltage levels at I/O Port B pins. For example, suppose that a 1 has been output to bit 2 of I/O Ports A and B. Subsequently suppose that pin 2 of I/O Ports A and B are drained excessively, so that voltage levels transiently drop to +0.5V. I/O Port A bit 2 will become 0, but I/O Port B bit 2 will retain a level of 1.
3) As outputs, I/O Port B pins may be used as a source of up to 1 mA at +1.5V, to directly drive the base of a transistor switch. This is not feasible using I/O Port A pins.

There are five device select pins.

CS0, CS1 and $\overline{\text{CS2}}$ are three typical chip select signals. For an MC6820 device to be selected, CS0 and CS1 must receive high inputs while $\overline{\text{CS2}}$ simultaneously receives a low input.

Providing CS0, CS1 and $\overline{\text{CS2}}$ have selected an MC6820 device, **RS0 and RS1 address one of four memory locations.** Thus an MC6820 device will appear to a programmer as four memory locations.

Any of the standard schemes described in Volume I can be used to address an MC6820 PIA. There is nothing unusual about the select logic with which you will assign four unique memory addresses to an MC6820.

There are four timing and control signals which interface an MC6820 with external logic.

CA1 and CA2 are control signals associated with I/O Port A. CA1 is an input only signal and is usually used by external logic to request an interrupt. CA2 is a bidirectional control signal which is used to implement various types of handshaking logic.

CB1 and CB2 are the control signals which support I/O Port B. These two signals are analogous to CA1 and CA2, although there are some differences in the handshaking logic associated with CB2 as compared to CA2.

There are two control signals associated with the MC6820 CPU interface.

E is the standard synchronization signal generated by the various MC6870 series clock devices. The trailing edge of E pulses synchronizes all logic and timing within the MC6820. Manufacturer literature refers to E as a device enable signal, but it is more accurately viewed as a device synchronization signal.

R/$\overline{\text{W}}$ is the standard Read/Write control signal output by the MC6800 CPU. When R/$\overline{\text{W}}$ is high, a Read operation is specified; that is, data transfer from the MC6820 PIA to the MC6800 CPU occurs. When R/$\overline{\text{W}}$ is low, a Write operation is specified; that is, data transfer from the CPU to the PIA occurs.

There are two interrupt request signals, $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$. Under program control you can specify the conditions under which an interrupt request can originate at logic associated with I/O Port A or I/O Port B. The actual interrupt request is transmitted to the MC6800 CPU via signal $\overline{\text{IRQA}}$ for I/O Port A logic, and via $\overline{\text{IRQB}}$ for I/O Port B logic. Interrupt requests originating at either signal will connect to the MC6800 $\overline{\text{IRQ}}$ input.

$\overline{\text{RESET}}$ is a standard Reset input. When it is input low, the contents of all MC6820 registers will be set to zero.

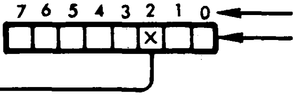
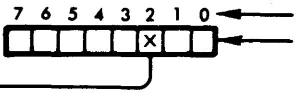
MC6820 OPERATIONS

As compared to the 8255 PPI, the MC6820 PIA has less formalized operating modes. The MC6820-to-external logic interface consists of two I/O ports, each of which has two dedicated control lines. **You have the option of assigning individual I/O port lines to input or output; as a completely separate operation you can use the two control lines to perform a limited amount of handshaking and interrupt processing — or you can ignore the control lines, in which case the I/O port is supporting simple input and/or output. Bidirectional I/O, equivalent to 8255 Mode 2, is not available. Figure 8-21 generally represents MC6820 functional organization and Table 8-4 summarizes the available operating modes.**

Table 8-4. MC6820 Operating Modes

OPERATING MODE	MC6800 AVAILABILITY
Simple input without handshaking	I/O Port A or B
Simple output without handshaking	I/O Port A or B
Bidirectional I/O without handshaking	Not available, but individual pins of either I/O port may be separately assigned to input or output
Input with handshaking	I/O Port A only
Output with handshaking	I/O Port B only
Bidirectional I/O with handshaking	Not Available

Table 8-5. Addressing MC6820 Internal Registers

SELECT LINES			ADDRESSED LOCATION
RS1	RS0	X	
0	1		 Bit No. I/O Port A Control register
0	0	0	I/O Port A Data Direction register
0	0	1	I/O Port A Data buffer
1	1		 Bit No. I/O Port B Control register
1	0	0	I/O Port B Data Direction register
1	0	1	I/O Port B Data buffer

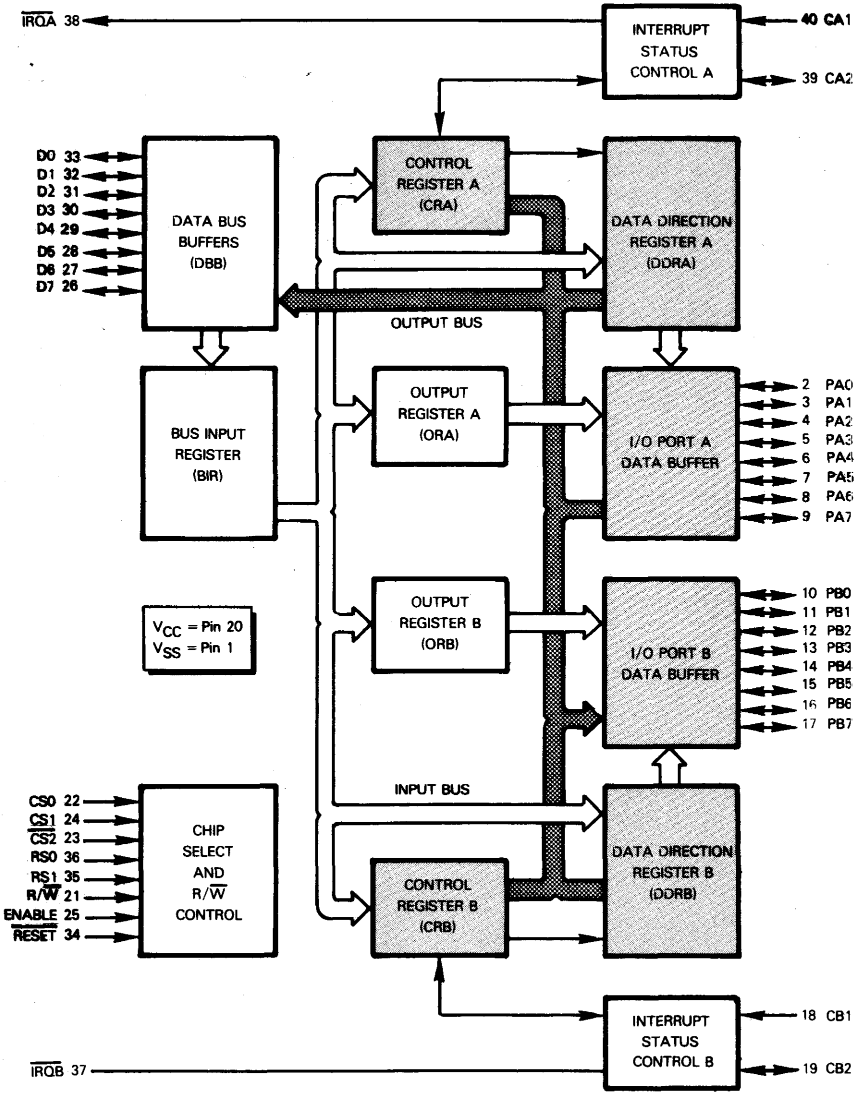


Figure 8-21. Functional Block Diagram For The MC6820 PIA

MC6820 REGISTERS ADDRESSING

There are six addressable locations within an MC6820 PIA; they are shaded in Figure 8-21. Since there are only two register select lines, RS0 and RS1, four unique addressable locations can be identified within the MC6820. Table 8-5 summarizes the manner in which the MC6820 uses four addresses to access six locations. Logic defined in Table 8-5 requires that you first output a Control code to each I/O port Control register; next you access either the I/O port Data Direction register, or the I/O port Data Buffer. You use the same memory address to access an I/O port Data Direction register and I/O port Data Buffer. Which location you access is determined by bit 2 of the I/O port's Control register.

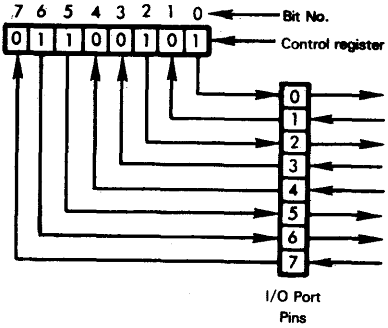
You must precede any I/O port Data Direction register, or Data Buffer access with a Control code, written to the I/O port's Control register. Once you have written a Control code to an I/O port Control register, you do not have to write another Control code for addressing purposes until you wish to switch from accessing the I/O port Data Direction register to the Data Buffer, or from accessing the Data Buffer to the Data Direction register.

To illustrate MC6820 addressing, suppose the four addresses C000$_{16}$, C001$_{16}$, C002$_{16}$ and C003$_{16}$ select an MC6820. This is how addressable locations within the MC6820 would actually be selected if address line A0 were connected to RS0 and A1 to RS1:

Address	Selected
C000$_{16}$	I/O Port A Data Direction register, if C001$_{16}$, bit 2 = 0
	I/O Port A Data buffer, if C001$_{16}$, bit 2 = 1
C001$_{16}$	I/O Port A Control register
C002$_{16}$	I/O Port B Data Direction register, if C003$_{16}$, bit 2 = 0
	I/O Port B Data buffer, if C003$_{16}$, bit 2 = 1
C003$_{16}$	I/O Port B Control register

If you read from an I/O port data buffer, you input from the I/O port to the CPU; if you write to an I/O port data buffer, you output from the CPU to the I/O port.

The Data Direction registers identify each pin of an I/O port as being dedicated to either input or output. These are write only registers. You must write a control word into each Data Direction register; a 0 in a bit position configures the corresponding I/O port pin as an input, while a 1 results in an output:

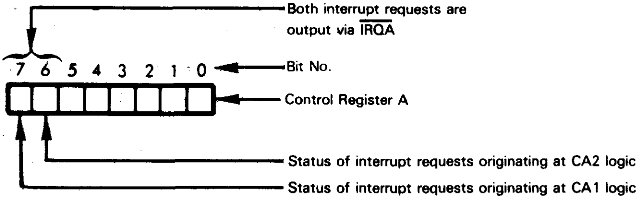


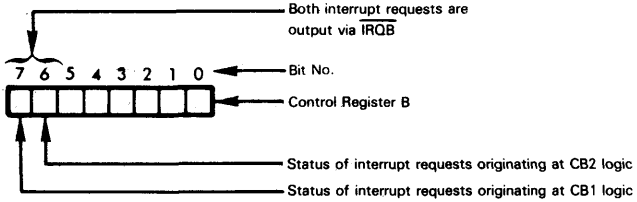
Observe that I/O Ports A and B will both be configured as 8-bit input ports when the MC6820 is reset, since $\overline{\text{RESET}}$ clears all internal registers.

Control register interpretation is quite complex.

MC6820 CONTROL CODES

The two high order bits of each Control register are read only locations, which record the status of interrupt requests which may originate from either of two control lines associated with an I/O port:


Both interrupt requests are output via $\overline{\text{IRQA}}$
7 6 5 4 3 2 1 0
Bit No.
Control Register A
Status of interrupt requests originating at CA2 logic
Status of interrupt requests originating at CA1 logic


Both interrupt requests are output via $\overline{\text{IRQB}}$
7 6 5 4 3 2 1 0
Bit No.
Control Register B
Status of interrupt requests originating at CB2 logic
Status of interrupt requests originating at CB1 logic

The remaining six control bits may be written into or read; they define the way in which the I/O port will operate.

Figures 8-22 and 8-23 describe the Control register interpretation for I/O Ports A and B respectively; since the two Control register interpretations are very similar, the points of difference are shaded so that they are easy to spot.

Let us clarify the functions enabled by the two Control registers.

MC6820 INTERRUPT LOGIC

Each I/O port has its own interrupt request signal: $\overline{\text{IRQA}}$ for I/O Port A and $\overline{\text{IRQB}}$ for I/O Port B. Each interrupt request signal has two separate sets of request logic, based on an interrupt request originating with a CA1/CB1 signal transition, or a CA2/CB2 signal transition.

Control register bit 0 enables or disables $\overline{\text{IRQA}}$/$\overline{\text{IRQB}}$, based on signal CA1/CB1 transitions only. Quite independently, Control register bit 3 enables or disables $\overline{\text{IRQA}}$/$\overline{\text{IRQB}}$ based on transitions of signal CA2/CB2. However, Control register bit 3 has an alternative interpretation; the one we have just described only applies if Control register bit 5 is 0.

Interrupt requests are triggered by the "active transitions" of a control signal. The active transitions of control signals may be a high-to-low, or a low-to-high transition. For CA1/CB1, the active transition is selected by Control register bit 1. For CA2/CB2, the active transition is selected by Control register bit 4, but only if Control register bit 5 is 0.

Irrespective of whether interrupt request signals $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$ have been enabled or disabled, Control register bits 6 and 7 will report the interrupt request as a status, that is to say, if a condition exists where CA1/CB1 makes an interrupt requesting active transition, then Control register bit 7 will be set to 1. Similarly, if control signal CA2/CB2 makes an interrupt requesting transition, then Control register bit 6 will be set to 1. Once set, Control register bits 6 and 7 will remain set until a Read operation addresses the Control register; at that time Control register bits 6 and 7 will both be reset to 0, while other bits of the Control register are left unaltered.

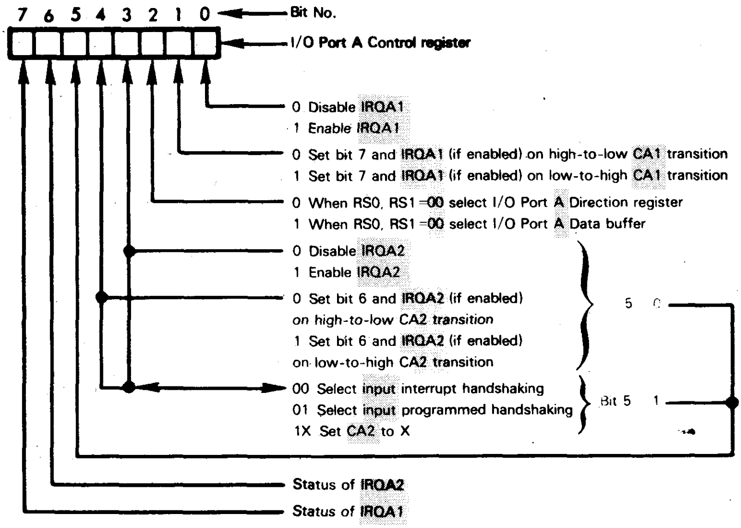

7 6 5 4 3 2 1 0
Bit No.
I/O Port A Control register
0 Disable IRQA1
1 Enable IRQA1
0 Set bit 7 and IRQA1 (if enabled) on high-to-low CA1 transition
1 Set bit 7 and IRQA1 (if enabled) on low-to-high CA1 transition
0 When RS0, RS1 =00 select I/O Port A Direction register
1 When RS0, RS1 =00 select I/O Port A Data buffer
0 Disable IRQA2
1 Enable IRQA2
0 Set bit 6 and IRQA2 (if enabled) on high-to-low CA2 transition
1 Set bit 6 and IRQA2 (if enabled) on low-to-high CA2 transition
5 0
00 Select input interrupt handshaking
01 Select input programmed handshaking
1X Set CA2 to X
Bit 5 1
Status of IRQA2
Status of IRQA1

Figure 8-22. I/O Port A Control Register Interpretation

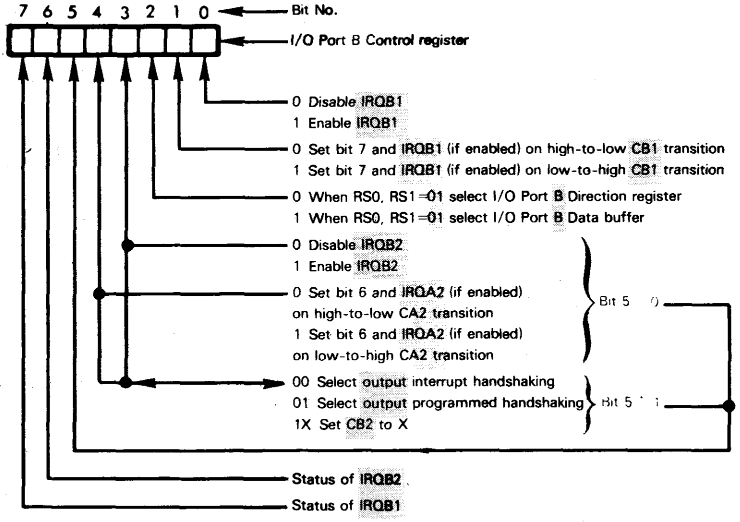

7 6 5 4 3 2 1 0
Bit No.
I/O Port B Control register
0 Disable IRQB1
1 Enable IRQB1
0 Set bit 7 and IRQB1 (if enabled) on high-to-low CB1 transition
1 Set bit 7 and IRQB1 (if enabled) on low-to-high CB1 transition
0 When RS0, RS1 =01 select I/O Port B Direction register
1 When RS0, RS1 =01 select I/O Port B Data buffer
0 Disable IRQB2
1 Enable IRQB2
0 Set bit 6 and IRQA2 (if enabled) on high-to-low CA2 transition
1 Set bit 6 and IRQA2 (if enabled) on low-to-high CA2 transition
Bit 5 0
00 Select output interrupt handshaking
01 Select output programmed handshaking
1X Set CB2 to X
Bit 5 1
Status of IRQB2
Status of IRQB1

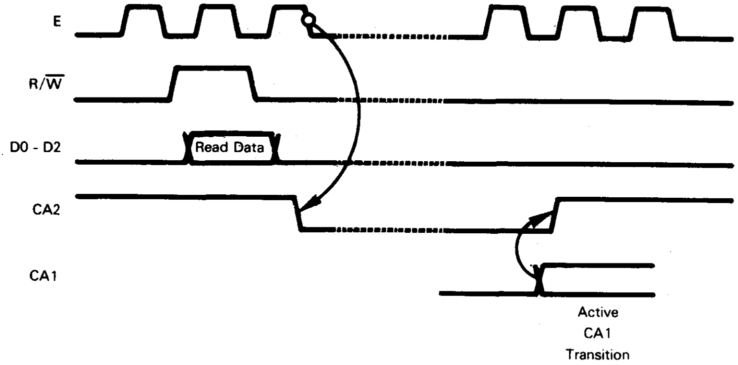
Figure 8-23. I/O Port B Control Register Interpretation

If Control register bit 5 is 1, then Control register bits 4 and 3 take on a second interpretation. If Control register bits 5 and 4 are both 1, then control signal CA2/CB2 will be output at all times with the level of control bit 3.

MC6820 AUTOMATIC HANDSHAKING

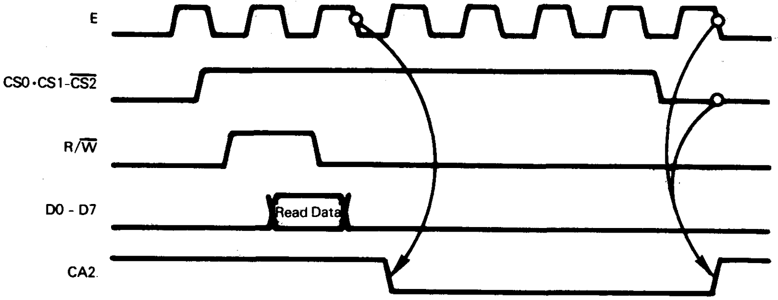
If Control register bits 5 and 4 are 1 and 0 respectively, then Control register bit 3 specifies an automatic handshaking signal sequence. Let us describe these signal sequences.

Input interrupt handshaking applies to I/O Port A only, and may be illustrated as follows:



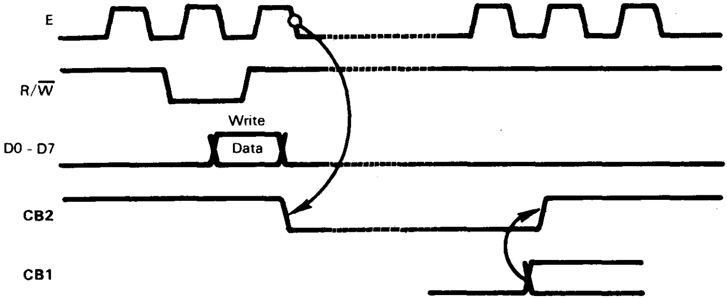
CA2 is output on the trailing edge of E, after the CPU has read the contents of the I/O Port A data buffer; this tells external logic that previously input data has been read and new data may now be input. External logic receives CA2 low, and upon transmitting new data to I/O Port A, must cause an active interrupt requesting transition of input control signal CA1. What constitutes an active transition will be determined by I/O Port A Control register bit 1. When external logic requests an interrupt via signal CA1, CA2 will be set high again.

Input programmed handshaking applies only to I/O Port A, and may be illustrated as follows:



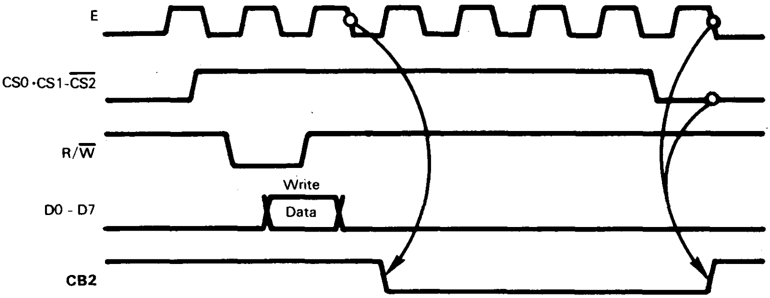
Once again control signal CA2 is output low when I/O Port A data buffer contents are read by the CPU. This tells external logic that previously input data has been read and new data may be input. External logic does not have to identify newly transmitted data with an interrupt request; rather, CA2 will be reset as soon as the MC6820 is deselected. Using programmed handshaking, external logic may use the CA2 low pulse as a Write strobe, causing new data to be input to I/O Port A.

Output interrupt handshaking applies only to I/O Port B, and may be illustrated as follows:



In this instance, control signal CB2 is output low on the high-to-low transition of E following a Write to I/O Port A Data buffer. In other words, CB2 tells external logic that new data has been output to I/O Port B and is ready to be read. External logic tells the MC6820 that I/O Port B contents have been read by making an interrupt requesting active transition of the CB1 signal. Once again, I/O Port B Control register bit 1 will determine what constitutes an active transition of the CB1 signal. Program logic can use an interrupt to branch to a program which outputs the next byte of data to I/O Port B.

Output programmed handshaking applies only to I/O Port B, and may be illustrated as follows:



CB2 makes a high-to-low transition when data is written into the I/O Port B data buffer, just as occurred with output interrupt handshaking. However, CB2 will automatically be set to 1 as soon as the MC6820 is deselected. External logic can use the CB2 low pulse as a strobe, causing it to read the contents of I/O Port B.

Many other handshaking protocols may be created under program control. The four automatic protocols described above are simply four situations which can be specified, and which will subsequently occur without further program intervention. But remember, you can modify the level of control signal CA2/CB2 any time by outputting a Control code with bits 5 and 4 both set to 1; CA2/CB2 will then take the level of Control code bit 3. You can also determine the conditions which will cause an interrupt request as a result of any control signal transition.

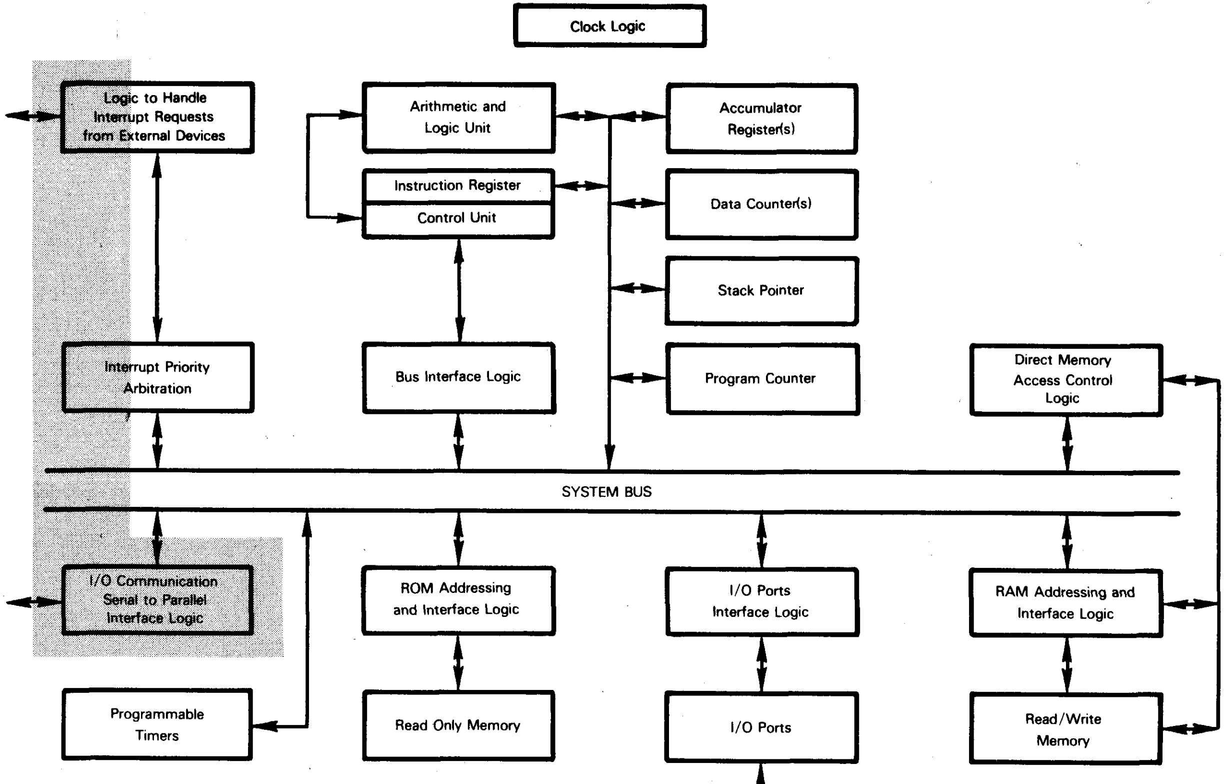

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 8-24. Logic Of The MC6850 ACIA Or XC6852 SSDA Devices

THE MC6850 ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER (ACIA)

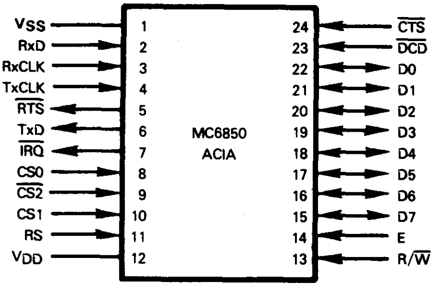
The MC6800 microcomputer system provides separate devices supporting synchronous and asynchronous serial I/O. The MC6850, which we are about to describe, provides asynchronous serial I/O. The XC6852, which we will describe next, supports synchronous serial I/O.

Taken together, the MC6850 and XC6852 devices are approximately equivalent to the 8251 USART which is used in 8080A microcomputer systems.

Figure 8-24 illustrates that part of our general microcomputer system logic which is provided by the MC6850 and XC6852 devices.

Having separate synchronous and asynchronous serial I/O devices has advantages and disadvantages, when compared to the 8251 USART which provides both sets of logic on a single device. In a microcomputer system that uses either asynchronous or synchronous serial I/O, but not both, separate devices are better, because they come in smaller packages and require less space on a PC card. If your microcomputer system uses both synchronous and asynchronous serial I/O, then a single device will be more economical.

When comparing the MC6850 with the 8251, you will find that the 8251 offers more asynchronous serial I/O options, but it is harder to program. In fact, you must program the 8251 defensively; 8251 statuses and control signals simply prompt your program logic, but actually do nothing within the 8251 USART itself. When using the MC6850 and XC6852, that is not the case; **these two devices are much easier to program.**



Pin Name	Description	Type
D0 - D7	Data Bus to CPU	Tristate, bidirectional
CS0, CS1, CS2	Chip Select	Input
E	Internal synchronization	Input
RS	Register Select	Input
R/$\overline{W}$	Read/Write control	Input
TxCLK	Transmit Clock	Input
TxD	Transmit Data	Output
RxCLK	Receive Clock	Input
RxD	Receive Data	Input
$\overline{CTS}$	Clear To Send	Input
$\overline{RTS}$	Request To Send	Output
$\overline{DCD}$	Data Carrier Detect	Input
$\overline{IRQ}$	Interrupt request	Output
V_{DD}, V_{SS}	Power and Ground	

Figure 8-25. MC6850 ACIA Signals And Pin Assignments

The MC6850 ACIA is packaged as a 24-pin DIP. It is fabricated using N-channel silicon gate technology.

A single +5V power supply is required.

In the discussion of the MC6850 that follows we will frequently refer to the 8251 USART description in Chapter 4. **If you are unfamiliar with asynchronous serial I/O devices in general, see Chapter 5 of Volume I, then read the description of the 8251 USART which is given in Chapter 4 of this book.**

THE MC6850 ACIA PINS AND SIGNALS

MC6850 ACIA pins and signals are illustrated in Figure 8-25. Signals may be divided into the following four categories:

1) CPU interface and control signals
2) Serial input
3) Serial output
4) Modem control

We will first consider CPU interface and control signals.

D0 - D7 constitutes an 8-bit bidirectional Data Bus connecting the MC6850 with the CPU.

When data is output to the MC6850 by the CPU, either a byte of parallel data or a Control code will be transmitted.

A byte of parallel data will be serialized and transmitted according to the protocol which has been selected under program control.

Either data or status may be input from the MC6850 ACIA to the CPU via the Data Bus. Data consists of an 8-bit parallel data unit extracted from the serial input data stream. Status consists of the contents of the ACIA Status register.

The Status register of the MC6850 ACIA is very important, because **the MC6850 uses status flags where the 8251 uses control signals to monitor serial data transfer logic.**

The MC6850 ACIA is accessed by the CPU as two memory locations. **MC6850 select logic consists of the three chip select signals CS0, CS1 and $\overline{\text{CS2}}$; manufacturers' literature also refers to the enable signal E as being part of the chip select logic;** however, E is more accurately visualized as an internal synchronization signal.

For the MC6850 ACIA to be selected, CS0 and CS1 must be input high while $\overline{\text{CS2}}$ is simultaneously input low. Once selected, **the register select signal RS determines which of the two addressable locations within the MC6850 ACIA will be accessed.** When RS is low, a Read will access the ACIA Status register, while a Write will access the ACIA Control register. When RS is high, ACIA data buffers will be addressed.

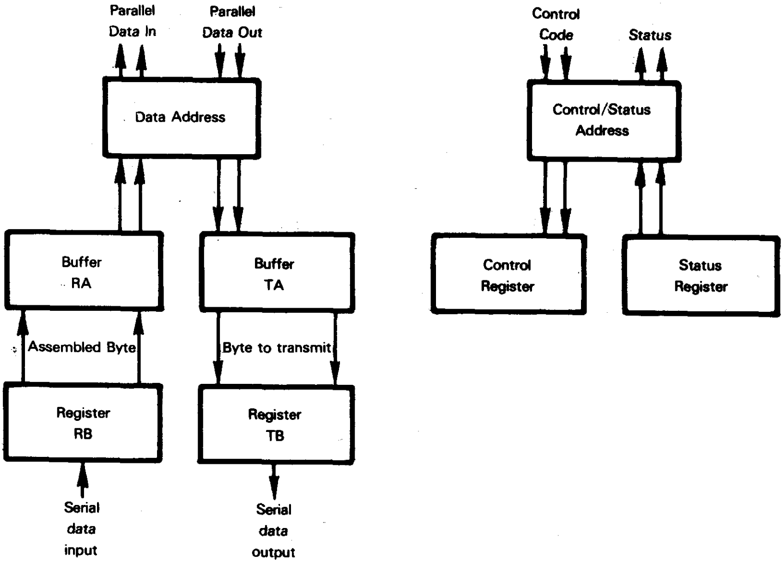
While the MC6850 ACIA is selected, internal logic is synchronized on the trailing edge of the E signal. E is a standard output of the various MC6870 clock devices used to synchronize support logic throughout an MC6800 microcomputer system.

R/$\overline{\text{W}}$ is the control input which determines whether a Read or Write operation is in progress. When R/$\overline{\text{W}}$ is high, the CPU is reading data out of the MC6850. When R/$\overline{\text{W}}$ is low, the CPU is writing data to the MC6850.

The MC6850 has no RESET input; a Control code is used as a master Reset. When power is first detected within the MC6850, internal logic automatically initiates a Reset sequence. Subsequently, before initializing the MC6850 for serial data transfer you should again reset the device by inputting a Reset Control code.

MC6850 DATA TRANSFER AND CONTROL OPERATIONS

There are a number of buffers through which data flows in and out of the MC6850 ACIA. These data flows may be illustrated as follows:



Buffer names in the illustration above conform with terminology used for the 8251 in Chapter 4; this will make it easier for you to compare the two devices.

Like the 8251, the MC6850 has double buffered serial input and output logic. As described for the 8251, **while a data byte is being serialized and output from Buffer TB, you must simultaneously write the next data byte to Buffer TA. Also, while a serial data byte is being assembled in Buffer RB, you must read the previously assembled data byte out of Buffer RA.**

Unlike the 8251, **the MC6850 has a separate Control register.** You can therefore write Control codes and read status at any time without fear of scrambling data waiting to be transmitted.

MC6850 SERIAL I/O DATA AND CONTROL SIGNALS

As compared to the 8251, the MC6850 has very elementary serial I/O logic.

TxCLK is an externally provided clock signal which times the serial, asynchronous data stream which is output via TxD.

Similarly, RxCLK is an externally provided clock signal which times the serial, asynchronous data stream which is input via RxD.

There are no control signals accompanying serial I/O data; rather, a single interrupt request signal is shared by all transmit and receive conditions. You have to write an interrupt service routine which reads the contents of the MC6850 Status register, and thus determine which one of the many serial data transfer interrupt request conditions has occurred.

The fact that you must execute instructions to duplicate the logic which the 8251 provides with its TxRDY, RxRDY and TxE signals will certainly make an MC6800 microcomputer system less attractive in an application that makes heavy use of serial I/O. Conversely, the MC6800 system will appear more attractive in simple applications, since you have less interface circuitry to be concerned with.

MC6850 MODEM CONTROL SIGNALS

Three modem control signals are provided: Clear To Send ($\overline{\mathrm{CTS}}$), Request To Send ($\overline{\mathrm{RTS}}$), and Data Carrier Detect ($\overline{\mathrm{DCD}}$). $\overline{\mathrm{CTS}}$ and $\overline{\mathrm{RTS}}$ are identical to the signals with the same names described in Volume I, Chapter 5 for the general case, and in Chapter 4 of this book for the 8251.

$\overline{\mathrm{RTS}}$ is output by the MC6850 under program control when the MC6850 is ready to transmit data. A full duplex line turns $\overline{\mathrm{RTS}}$ around and sends it back as $\overline{\mathrm{CTS}}$; a half duplex line returns $\overline{\mathrm{CTS}}$ after two milliseconds.

$\overline{\mathrm{DCD}}$ is equivalent to the standard $\overline{\mathrm{DTR}}$ signal. The MC6850 has no Data Set Ready ($\overline{\mathrm{DSR}}$) signal; this is the signal which many serial I/O devices transmit to modems or any external receiving logic when ready to commence with serial data communications. When using an MC6850, $\overline{\mathrm{RTS}}$ must serve double duty, additionally substituting for $\overline{\mathrm{DSR}}$.

Even though the MC6850 has only three of the normal four control signals, these signals work hard within the MC6850.

The $\overline{\mathrm{DCD}}$ input must be low for serial transmit logic within the MC6850 to be enabled. This is true also of the equivalent 8251 $\overline{\mathrm{DSR}}$ signal; however, if the $\overline{\mathrm{DCD}}$ signal makes a low-to-high transition, the MC6850 will generate an interrupt request, thus effectively halting serial data output. A low-to-high $\overline{\mathrm{DCD}}$ transition implies that the modem has, for some reason, disconnected itself; any further data transfer will be lost. In the case of the 8251, if a modem disconnects itself and $\overline{\mathrm{DSR}}$ goes high, this condition will be reflected in a Status register flag, but unless the CPU executes instructions to read the Status register and test for this condition, the 8251 will continue transmitting data — even though the receiving end is dead.

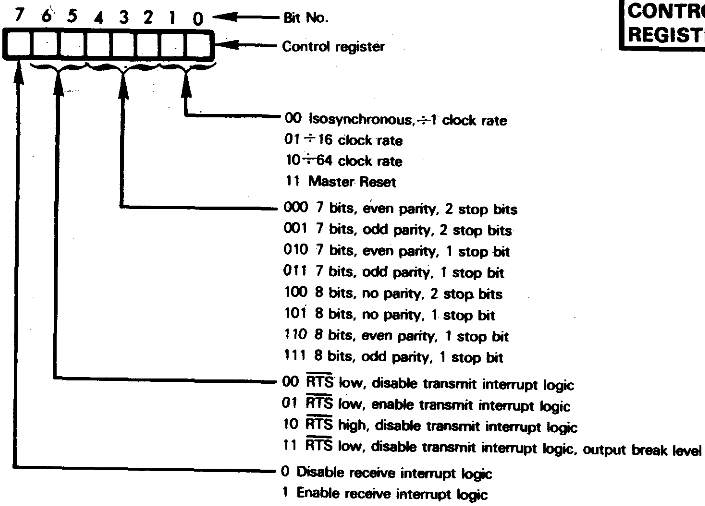
The MC6850 uses $\overline{\mathrm{CTS}}$ high to prevent the Status register from reporting a "Transmit Register Empty" condition. The MC6800 CPU determines when to send another byte of data to the MC6850 by testing the Status register, and looking for a "Transmit Register Empty" condition. If this condition never gets reported, no data will ever be uselessly transmitted. Contrast this with 8251 logic, where a misprogrammed 8251 can and will continue to transmit data after $\overline{\mathrm{CTS}}$ has gone high.

MC6850 ACIA CONTROL CODES AND STATUS FLAGS

Let us now examine the way in which the MC6850 Control and Status registers are interpreted.

MC6850 CONTROL REGISTER

Here is the Control register interpretation:



00 Isosynchronous, ÷1 clock rate
01 ÷16 clock rate
10 ÷64 clock rate
11 Master Reset

000 7 bits, even parity, 2 stop bits
001 7 bits, odd parity, 2 stop bits
010 7 bits, even parity, 1 stop bit
011 7 bits, odd parity, 1 stop bit
100 8 bits, no parity, 2 stop bits
101 8 bits, no parity, 1 stop bit
110 8 bits, even parity, 1 stop bit
111 8 bits, odd parity, 1 stop bit

00 $\overline{\text{RTS}}$ low, disable transmit interrupt logic
01 $\overline{\text{RTS}}$ low, enable transmit interrupt logic
10 $\overline{\text{RTS}}$ high, disable transmit interrupt logic
11 $\overline{\text{RTS}}$ low, disable transmit interrupt logic, output break level

0 Disable receive interrupt logic
1 Enable receive interrupt logic

The CPU neither sends nor receives the parity bit. The MC6850 adds the parity bit to transmitted data and strips or resets the parity bit in received data before it goes to the CPU.

Control register bits 0 and 1 determine the data transfer clock rate. Recall that serial data is usually transmitted or received at 1/16th or 1/64th of the clock rate, TxCLK or RxCLK. Transferring serial data at the exact clock rate is referred to as isosynchronous data transfer.

MC6850 SYSTEM RESET

The master reset Control code substitutes for the normal reset input signal, which the MC6850 lacks. A master reset clears all MC6850 registers, with the exception of Status register bit 3, which is unaltered.

Control register bits 2, 3 and 4 identify data bit, stop bit and parity options. Compared to the 8251, MC6850 options are somewhat limited; five and six data bits are not provided and you cannot select 1.5 stop bits.

MC6850 SERIAL I/O CONTROL LOGIC

Control register bits 5 and 6 are transmit logic control bits. Control register bit 7 is a receive logic control bit.

Transmit logic consists of the $\overline{\text{RTS}}$ modem control and various transmit conditions that can cause an interrupt request.

Receive control logic consists of various receive conditions that can cause an interrupt request.

MC6850 INTERRUPT LOGIC

Interrupt logic of the MC6850 is an integral part of status logic. Conditions that can result in an interrupt request are therefore summarized below along with a definition of Status register bits. A "T" is placed in those bit positions that can result in an interrupt request from transmit logic. An "R" is placed in those bit positions that can result in an interrupt request from receive logic. Status register bit positions that have neither a "T" nor an "R" identify conditions that do not result in interrupt requests.

In those bit positions containing a "T" or an "R", a 1 causes an interrupt request to occur. $\overline{\text{DCD}}$ (bit 3) is an exception; here it is the transition from 0 to 1 that causes an interrupt request. In each case, the interrupt request will only occur if interrupt logic has been enabled. If you look back at the Control register, you will see that transmit and receive interrupt logic can be enabled and disabled separately. Control register bits 5 and 6 determine whether transmit interrupt logic is enabled, while Control register bit 7 determines whether receive interrupt logic is enabled. Note that the condition of Status register bit 3 can also disable a TDRE interrupt request.

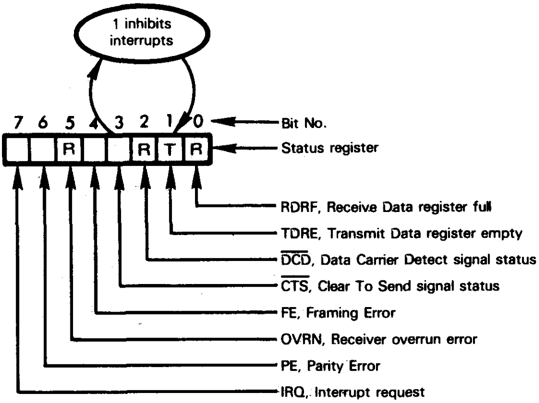
When an interrupt request occurs, the requesting condition is cleared in various ways depending upon where the request originated.

An RDRF interrupt request will be cleared if the CPU reads data from the MC6850, or if a reset Control code is output.

A TDRE interrupt request will be cleared by writing data to the MC6850 or by issuing a reset Control code.

Interrupts requested by $\overline{\text{DCD}}$ or OVRN are cleared by reading the Status register after the error condition has occurred, and then reading the Data register. A Master Reset will also clear these interrupt requests.

Let us now take a closer look at the Status register itself. **This is how register bits are interpreted:**


1 inhibits interrupts
7 6 5 4 3 2 1 0
Bit No.
R R T R
Status register
RDRF, Receive Data register full
TDRE, Transmit Data register empty
$\overline{\text{DCD}}$, Data Carrier Detect signal status
$\overline{\text{CTS}}$, Clear To Send signal status
FE, Framing Error
OVRN, Receiver overrun error
PE, Parity Error
IRQ, Interrupt request

(1 in a bit position represents "true" condition for bits 7, 6, 5, 4, 1 and 0.)

Status register bit 0, Receive Data Register Full, goes to 1 when a byte of assembled data is transferred from Receive register RB to Receive register RA. Bit 0 is cleared as soon as the CPU reads the contents of Register RA. The $\overline{\text{DCD}}$ modem control signal, when high, forces Status register bit 0 to stay low so that the CPU will not attempt to read nonexistent data.

Status register bit 1, Transmit Data Register Empty, goes from 0 to 1 as soon as data is transferred from Register TA to Register TB. This bit is reset to 0 as soon as the CPU writes another bit of data into Register TA.

Status register bit 2, Data Carrier Detect, is used by the MC6800 to determine the status of external logic communicating with the MC6850. When $\overline{\text{DCD}}$ makes a low-to-high transition, an interrupt request is generated and Status register bit 2 goes high. Bit 2 remains high until the Status register contents are read by the CPU after $\overline{\text{DCD}}$ has gone low again. A Reset will also set Status register bit 2 to 0. If the CPU reads the Status register while $\overline{\text{DCD}}$ is high, then subsequently Status register bit 2 will track the $\overline{\text{DCD}}$ level; however, another interrupt will not be requested. It is the actual low-to-high transition of the $\overline{\text{DCD}}$ signal which causes an interrupt request, not a high level of Status register bit 2.

Status register bit 3, Clear To Send, tracks the $\overline{\text{CTS}}$ modem control input. MC6850 logic uses Status register bit 3 to inhibit serial data transfer when external receiving logic is not ready to receive the serial data. When $\overline{\text{CTS}}$ is high, Status register bit 1 will be held low. A TDRE interrupt request cannot occur, and program logic which tests Status register bit 1 will not transmit another data byte to Register TA until it detects a 1 in Status register bit 1. Thus, for as long as $\overline{\text{CTS}}$ is high, serial transmit logic will be inhibited.

Status register bits 4, 5 and 6 report framing, overrun and parity errors, respectively. Recall that a framing error is reported when start and stop bits do not correctly frame a data character; a framing error refers to the data byte currently waiting to be read out of RA. An overrun error is reported if the CPU does not read Register RA contents before a byte of data is transferred from Register RB to Register RA. A parity error is reported if parity has been enabled by Control register bits 2, 3 and 4, but the wrong parity is detected.

A framing or parity error is automatically reset as soon as the erroneous data is read out of Register RA, or is overwritten.

An overrun error is cleared by reading data from the MC6850.

Status register bit 7, Interrupt Request, is 1 whenever there is an unacknowledged interrupt request pending at the MC6850 device. One method that an MC6800 will use to determine the source of an interrupt request is to read device Status registers. If the MC6850 has no other method of identifying itself to the CPU when requesting an interrupt, then the CPU determines whether the MC6850 was the requesting device by reading the contents of the MC6850 Status register and testing the condition of bit 7.

THE XC6852 SYNCHRONOUS SERIAL DATA ADAPTER (SSDA)

The XC6852 SSDA provides MC6800 microcomputer systems with synchronous serial I/O logic.

The XC6852 SSDA may be looked upon as a companion device to the MC6850 ACIA which we have just described. Taken together, these two devices provide MC6800 microcomputer systems with total serial I/O capability.

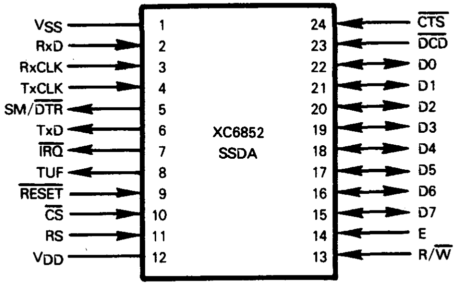
Figure 8-24 illustrates that part of our general microcomputer system logic which is provided by the MC6850 and XC6852 devices.

The most striking difference between the MC6850 and the XC6852 is their respective capabilities. Whereas the MC6850 offers fewer asynchronous serial I/O options than the 8251 USART (described in Chapter 4), the XC6852 offers significantly more synchronous serial I/O options. Moreover, the XC6852 provides additional serial I/O options without the penalty of defensive programming which is demanded by the 8251 USART.

The XC6852 SSDA is packaged as a 24-pin DIP. It is fabricated using N-channel silicon gate technology.

A single +5V power supply is required.

In the discussion of the XC6852 that follows, we will frequently refer to the 8251 USART description given in Chapter 4. If you are unfamiliar with synchronous serial I/O devices in general, see Chapter 5 of Volume I, then read the description of the 8251 USART which is given in Chapter 4 of this book.



Pin Name	Description	Type
D0 - D7	Data Bus to CPU	Tristate, bidirectional
$\overline{CS}$	Chip Select	Input
E	Internal synchronization	Input
RS	Register Select	Input
R/$\overline{W}$	Read/Write control	Input
TxCLK	Transmit Clock	Input
TxD	Transmit Data	Output
RxCLK	Receive Clock	Input
RxD	Receive Data	Input
$\overline{RESET}$	Master Reset	Input
$\overline{DCD}$	Data Carrier Detect	Input
$\overline{CTS}$	Clear To Send	Input
SM/$\overline{DTR}$	Sync Match/Data Terminal Ready	Output
TUF	Transmitter Underflow	Output
$\overline{IRQ}$	Interrupt request	Output
VDD, VSS	Power and Ground	

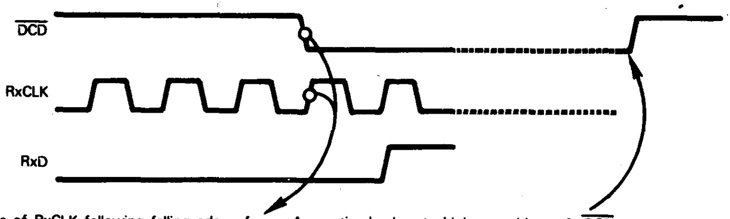
Figure 8-26. XC6852 SSDA Signals And Pin Assignments

XC6852 SSDA PINS AND SIGNALS

XC6852 SSDA pins and signals are illustrated in Figure 8-26. Most of these signals are identical to those illustrated in Figure 8-25 for the MC6850, therefore we will only describe four signals which differ.

The XC6852 has a master Reset input, which, when input low, logically resets the XC6852. We will define how a Reset occurs after describing the XC6852 controls and status flags affected by a Reset.

The Data Carrier Detect ($\overline{DCD}$) modem control input performs two functions. The normal function of $\overline{DCD}$ is to serve as a control signal transmitted by an external data carrier which is ready to transmit serial data to the XC6852 SSDA. Both the high-to-low and the low-to-high transitions of $\overline{DCD}$ have additional significance. The high-to-low signal transition can optionally be used as an external synchronization indicator, while a subsequent low-to-high transition is an error indicator, signalling an unexpected disconnect:



Rising edge of RxCLK following falling edge of $\overline{\text{DCD}}$ can serve as external synchronization, marking the start of data bits incoming on RxD.

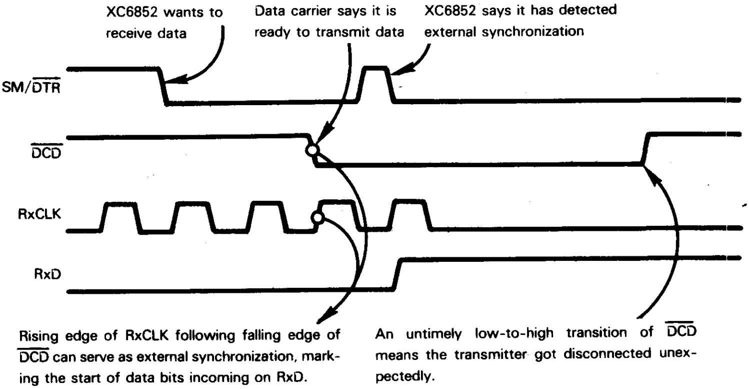
An untimely low-to-high transition of $\overline{\text{DCD}}$ means the transmitter got disconnected unexpectedly.

Using the high-to-low $\overline{\text{DCD}}$ pulse for external synchronization is a programmable option. The error condition reported if $\overline{\text{DCD}}$ makes an unexpected low-to-high transition is not a programmable option; it is a permanent part of the XC6852 error detection logic.

Clear To Send ($\overline{\textbf{CTS}}$) is the modem control signal which is normally input by external receiving logic, indicating that the XC6852 may begin transmitting serial data. Like $\overline{\text{DCD}}$, the $\overline{\text{CTS}}$ high-to-low transition can be used to synchronize the beginning of data transmission; the low-to-high transition of $\overline{\text{CTS}}$ is an error indicator. Once again, using the high-to-low $\overline{\text{CTS}}$ pulse to provide external transmit synchronization is a programmable option. However, an untimely low-to-high transition of $\overline{\text{CTS}}$ is an error indicator only if internal synchronization is being used. Therefore, if the high-to-low $\overline{\text{CTS}}$ transition is active, then the low-to-high subsequent transition must be inactive; conversely, if the high-to-low $\overline{\text{CTS}}$ transition is inactive, then a subsequent low-to-high transition will be active. This is because the high-to-low transition, if active, means that external synchronization has been selected — in which case the disconnect error logic is inactive.

Note that whereas the $\overline{\text{CTS}}$ signal low-to-high transition is only active during internal synchronization operations, the $\overline{\text{DCD}}$ low-to-high transition is active at all times. This means that **external logic disconnecting itself during a serial transmit operation will only cause an error to be indicated if external synchronization has been selected. On the other hand, during a serial receive operation, if external logic disconnects itself, an error will be indicated whether internal or external synchronization has been selected.**

Since $\overline{DCD}$ and $\overline{CTS}$ can both be used for external synchronization, as we might expect, **$\overline{DTR}$ also serves a double function.** Under normal circumstances, $\overline{DTR}$ will be output low by the XC6852 when it is ready either to transmit, or to receive serial data. If the XC6852 has output $\overline{DTR}$ low before transmitting serial data, then the receiving data carrier will turn $\overline{DTR}$ around and send back a high-to-low $\overline{DCD}$ pulse as we illustrated. If you have selected external synchronization under program control, then you can additionally program $\overline{DTR}$ to output a single high pulse as soon as synchronization has been detected. This may be illustrated as follows:



XC6852 wants to receive data

Data carrier says it is ready to transmit data

XC6852 says it has detected external synchronization

Rising edge of RxCLK following falling edge of $\overline{DCD}$ can serve as external synchronization, marking the start of data bits incoming on RxD.

An untimely low-to-high transition of $\overline{DCD}$ means the transmitter got disconnected unexpectedly.

Because $\overline{DTR}$ also acts as a Sync Match acknowledge, it is referred to as SM/$\overline{DTR}$.

XC6852 SERIALIZATION SEQUENCE

When the XC6852 transmits serial data, it transmits the least significant bit first. The XC6852 also expects to receive the least significant bit first when receiving serial data.

Transmitter Underflow (TUF) is the fourth unique XC6852 signal. This signal is output when an underflow condition occurs during serial synchronous data transmission. Recall that during serial synchronous data transmission, if serial transmit logic finds no data ready to be output, then in order to maintain synchronization, a break character or a Sync character will be output. A break character is a continuous high level, equivalent to FF_{16}. A Sync character will have some predefined binary pattern. Providing you have programmed the XC6852 to output Sync characters when no valid data is ready for serial transmission, the XC6852 will precede each Sync character with a high TUF pulse. External receive logic can use a high TUF pulse as an indicator that the next received character is a Sync and can be discarded.

XC6852 DATA TRANSFER AND CONTROL OPERATIONS

Like the MC6850, the XC6852 SSDA is accessed via two memory addresses; however, these two memory addresses are shared by seven locations within the XC6852, which results in a complex set of data flows, as illustrated in Figure 8-27.

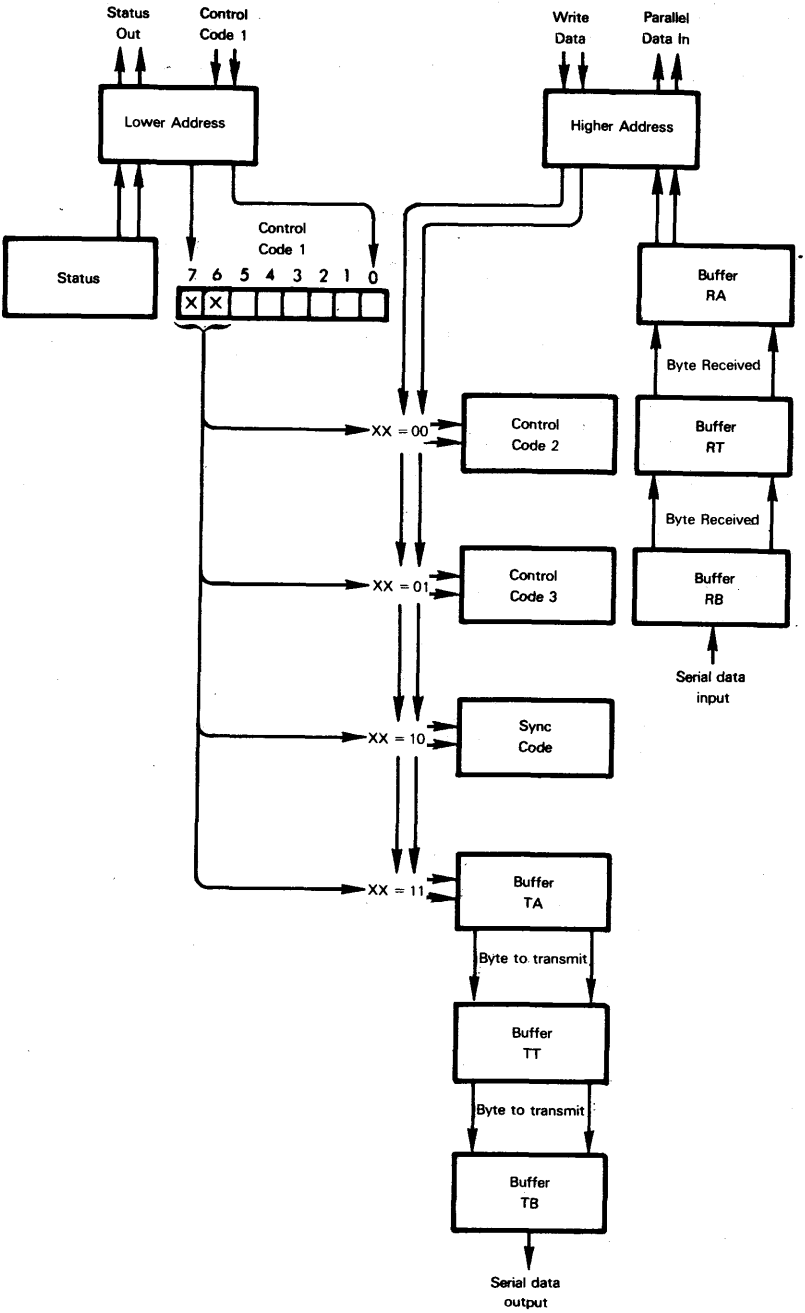


Figure 8-27. Data Flows Within An XC6852 SSDA

These are the seven addressable locations of the XC6852:

1) Data input — a read only location.

2) Data output — a write only location.

3) Status register — a read only location.

4) Sync Code register — a write only location.

5, 6, and 7) Three Control registers — all are write only locations.

Data input and data output are self-evident; apart from being triple buffered — and we will discuss the implications of triple buffering shortly — there is nothing unusual about XC6852 data input or output.

The Status register is absolutely standard.

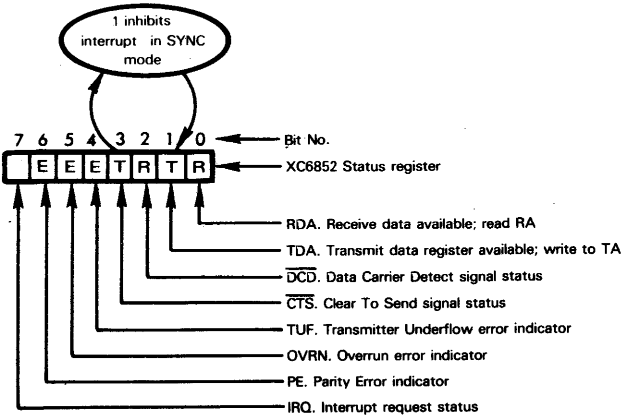
The three 8-bit Control registers provide the XC6852 with a substantial variety of control options, as compared to the MC6850, which was somewhat limited in this respect.

The Sync Code register stores the 8-bit synchronization character code; this is the character which must appear at the beginning of any synchronous serial data stream and may also be transmitted when data is unavailable during a normal transmit sequence.

Of the seven addressable locations, two are read only, while five are write only. **Each memory address can access two locations, providing one is exclusively read only, while the other is exclusively write only.** Since there are just two read only locations, one is assigned to each memory address. Since there are five write only locations, one (Control Code 1) is assigned to the lower address, which leaves four assigned to the higher address; the two high order bits of Control Code 1 are used to select one of the four write only locations assigned to the higher address. While this may look like a complex scheme, in reality it is not; all it means is that you have to observe a rigid programming sequence when using an XC6852. In fact, understanding the XC6852 depends completely on understanding the Control and Status registers; therefore we will describe these registers first, then look at data transfer sequences.

XC6852 STATUS REGISTER

The XC6852 Status register may be illustrated as follows:


1 inhibits interrupt in SYNC mode
7 6 5 4 3 2 1 0
Bit No.
E E E T R T R
XC6852 Status register
RDA. Receive data available; read RA
TDA. Transmit data register available; write to TA
$\overline{\text{DCD}}$. Data Carrier Detect signal status
$\overline{\text{CTS}}$. Clear To Send signal status
TUF. Transmitter Underflow error indicator
OVRN. Overrun error indicator
PE. Parity Error indicator
IRQ. Interrupt request status

(1 in a bit position represents "true" condition for bits 7, 6, 5, 4, 1 and 0.)

Conditions that may generate interrupts are marked with letters in appropriate Status register bit positions. An interrupt request initiated by an error condition is represented by the letter E. Interrupt requests originating at transmit or receive logic are represented by the letters T and R, respectively.

XC6852 TRIPLE DATA BUFFERS

Status register bit 0 (RDA) indicates when the XC6852 Status register has a byte of data ready to be read. Similarly Status register bit 1 (TDA) indicates when the XC6852 is ready to receive another byte of data which will be output as a serial data stream.

As indicated in Figure 8-27, XC6852 transmit and receive logic is triple buffered. This differs from the MC6850 which uses double buffering.

You can use the triple buffering of the XC6852 in one of two ways which you select using appropriate Control register codes.

You can select a single byte option, in which case as soon as a single byte of data can be written to Buffer TA or read from Buffer RA, the appropriate status flag will be set — and if interrupts are enabled, an interrupt request will be made to the CPU. The program controlling XC6852 operation must respond by reading or writing a single byte of data. A byte of data written to Buffer TA will automatically be rippled through Buffer TT to Buffer TB, whence it will output as a serial data stream. Data arriving at Buffer RB will be rippled through Buffer RT to Buffer RA, whence it must be read by the CPU.

If you select the two byte option under program control, then no status flags will be set, nor will interrupt requests occur until two of the three 8-bit buffers are empty. Thus, status bit 0 will be set and a receive interrupt request will occur when Buffers RA and RT are both full. Under program control you must, at this time, read two bytes of data. So long as a single pulse of the timing E signal separates the two read commands, XC6852 logic will transfer Buffer RT contents to Buffer RA so that the second read accesses what had been in Buffer RT. In fact, you should read RA contents, then status, then RA contents again. If there are errors associated with the data byte in RT, they will not be reported until RT contents have been transferred to RA.

When using the two byte option with transmit logic, Status register bit 1 will not be set and the appropriate interrupt request will not occur until Buffers TA and TT are both empty. At this time the executing program must write two bytes of data to the higher XC6852 address, while Control code 1, bits 7 and 6 are both 1. The first byte of data written to the higher XC6852 address will store data in Buffer TA. The next pulse of the E clock will transfer the contents of Buffer TA to Buffer TT. The second write will again load Buffer TA whose previous contents are now in Buffer TT.

Status register bits 2 and 3 are associated with signals $\overline{DCD}$ and $\overline{CTS}$, respectively. If $\overline{DCD}$ or $\overline{CTS}$ makes a low-to-high transition, then its corresponding Status register bit will latch high — that is, it will maintain a level of 1 until it is reset by the CPU. Once bit 2 (or 3) has been reset, it will track $\overline{DCD}$ (or $\overline{CTS}$) until the next low-to-high transition.

Note that in Sync mode, if Status register bit 3 is 1, then Status register bit 1 will be held at 0; this is how the XC6852 suppresses subsequent transmit logic.

Status register bits 4, 5 and 6 indicate Underflow, Overrun or Parity errors, respectively.

An Underflow error occurs when transmit logic does not have a byte of data ready to transmit and has to insert a Sync character. The Underflow error is reported just before the Sync character is transmitted. When Status register bit 4 is set, the TUF signal is simultaneously pulsed high.

An Overrun error occurs when a byte of data is written into Buffer RA before prior buffer contents have been read. An Overrun error therefore indicates that a single byte of data has been lost.

A Parity error indicates that a Parity option has been selected, but the wrong Parity was detected for the data byte currently in Buffer RA.

These three error conditions are completely standard; however, the way they are handled within the XC6852 is not standard. When any one of these error conditions occurs, the appropriate Status register bit will be set and simultaneously an interrupt request will be generated, providing you have enabled these three error interrupts.

An error status is not cleared automatically. To clear Status register bits 4, 5 or 6, you have to read Status register contents, then issue an appropriate Control code to reset the selected bit.

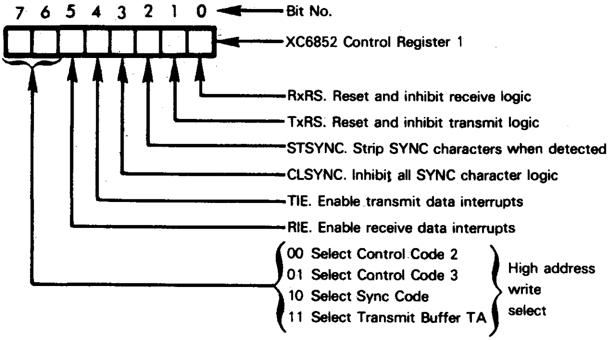
We can summarize the functions performed by XC6852 Status register bits by looking at the manner in which each bit is set or reset; then we can separately examine the way in which interrupt logic is associated with each status bit position.

Table 8-6 summarizes the conditions which cause each bit to be set and then reset. Table 8-7 summarizes interrupt requests associated with each status bit, indicating the way the interrupt is enabled or disabled and the way in which an interrupt request occurs. You will find Table 8-7 following the three Control registers' description, because interrupt logic is equally dependent upon the Status register's contents and the three Control registers' contents.

THE XC6852 CONTROL REGISTERS

Now consider the three XC6852 Control registers.

Control register 1 is normally the first to be accessed and has to be written into in order to select any other write only XC6852 location. **Control register 1 format may be illustrated as follows:**


7 6 5 4 3 2 1 0
Bit No.
XC6852 Control Register 1
RxRS. Reset and inhibit receive logic
TxRS. Reset and inhibit transmit logic
STSYNC. Strip SYNC characters when detected
CLSYNC. Inhibit all SYNC character logic
TIE. Enable transmit data interrupts
RIE. Enable receive data interrupts
00 Select Control Code 2
01 Select Control Code 3
10 Select Sync Code
11 Select Transmit Buffer TA
High address write select

(1 in a bit position represents "true" condition for bits 5, 4, 3, 2, 1 and 0.)

Control register 1, bits 0 and 1 reset and inhibit receive and transmit logic, respectively. You use these two Control register bits in order to disable transmit and receive logic while modifying the contents of any Control register or the Sync register.

Control register 1, bits 0 and 1 are very important. It is easy to miss the significance of these two control bits. If you always inhibit transmit and receive logic before modifying the contents of Control or Sync registers, you can make sure that spurious data is never transmitted or received. The 8251 USART described in Chapter 4, does not have any inhibit logic of this type; and as a result, you have to adopt elaborate precautions to avoid data transmission errors.

Table 8-6. XC6852 Status Register Bit Set/Reset Conditions

STATUS	SET	RESET
RDA - Bit 0	1) If Control register 2 bit 2 is 1, when Buffer RA is full. 2) If Control register 2 bit 2 is 0, when Buffers RA and RT are full.	1) Write 1 in Control register 1 bit 0. 2) Read Buffer RA contents.
TDA - Bit 1	1) If Control register 2 bit 2 is 1 when Buffer TA is empty. 2) If Control register 2 bit 2 is 0 when Buffers TA and TT are empty.	1) 1 occurs in Status register bit 5, together with 0 in Control register 3 bit 0. 2) Write 1 in Control register 1 bit 1. 3) Write into Buffer TA.
$\overline{\text{DCD}}$ - Bit 2	A low-to-high $\overline{\text{DCD}}$ input transition when Control register 1 bit 0 is 0.	1) Read Status register, then read Buffer RA. Status will subsequently go low when $\overline{\text{DCD}}$ input goes low. 2) Write 1 into Control register 1 bit 0. Status will subsequently go low when $\overline{\text{DCD}}$ input goes low.
$\overline{\text{CTS}}$ - Bit 3	A low-to-high $\overline{\text{CTS}}$ input transition when Control register 1 bit 1 is 0.	1) Write 1 to Control register 3 bit 2. Status will subsequently go low when $\overline{\text{CTS}}$ input goes low. 2) Write 1 into Control register bit 1. Status will subsequently go low when $\overline{\text{CTS}}$ input goes low.
TUF - Bit 4	Underflow when Control register 3 bit 0 is 0 and Control register 2 bit 6 is 1.	1) Write 1 into Control register 3 bit 3. 2) Write 1 into Control register 1 bit 1.
OVRN - Bit 5	Buffer RT contents is transferred to Buffer RA before Buffer RA contents is read by CPU.	1) Read Status register, then read Buffer RA 2) Write 1 into Control register 1 bit 0.
FE - Bit 6	Parity error for data in RA, providing Control register 2 bits 3, 4 and 5 identify a parity option.	1) Read data out of Buffer RA. 2) Write 1 into Control register 1 bit 0.
IRQ - Bit 7	Any interrupt request occurs.	No active interrupt requests exist.

While transmit and receive logic is inhibited, Status register bits 2 and 3 will still track the $\overline{\text{DCD}}$ and $\overline{\text{CTS}}$ signals; however, no data transfers will occur and interrupts associated with the inhibited logic will be disabled.

Using Control register 1, bits 0 and 1 to inhibit transmit and/or receive logic also affects Status register bits and interrupt requests, as summarized in Tables 8-6 and 8-7.

Control register 1, bit 5 allows you to enable or disable receive data interrupt logic. Control register 1, bit 4 allows you to enable or disable transmit data interrupt logic.

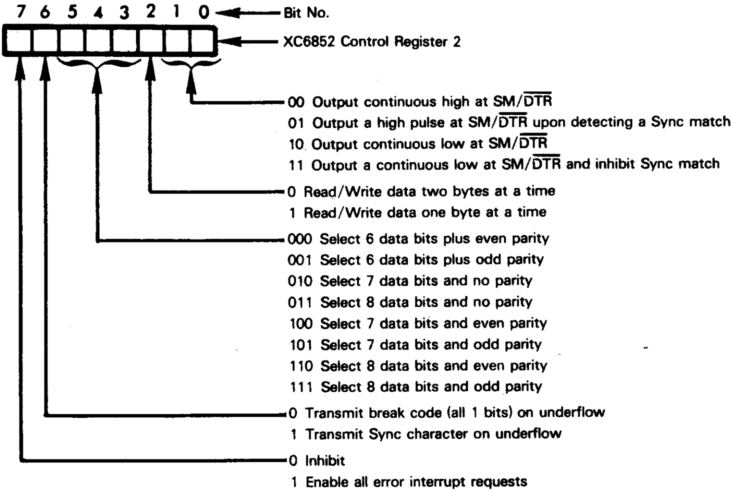
There is no connection between Control register 1, bits 0 and 1, and Control register 1, bits 4 and 5. Obviously, if transmit or receive logic has been inhibited, then it makes no difference whether interrupt logic has been enabled or disabled; in either case an interrupt cannot occur. However, if transmit or receive logic is enabled, then interrupt logic may be separately enabled or disabled.

Control register 1, bits 2 and 3 determine the way the Sync character will be handled. If Control register bit 2 is high, then all Sync characters in a serial receive data stream will be stripped, so that only non-Sync characters are read by the CPU. If Control register 1, bit 2 is low, then the entire data stream will be transmitted to the CPU, including data and Sync characters. Note that the initial Sync character is always stripped.

Control register 1, bit 3 allows you to completely inhibit all Sync character logic. Now the Sync character will be treated as any other character, and the XC6852 must use external synchronization.

Control register bits 6 and 7 determine which write only location will be accessed when the CPU writes to the higher memory location of the XC6852.

Now consider Control registers 2 and 3, which are best looked upon as a single 12-bit control unit. These two Control registers may be illustrated as follows:



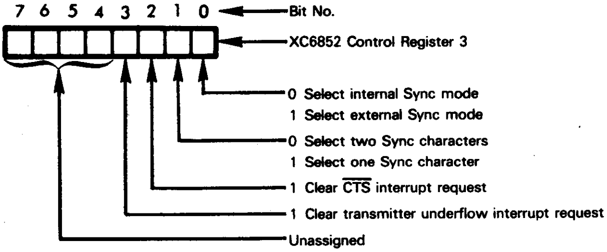
00 Output continuous high at SM/$\overline{\text{DTR}}$
01 Output a high pulse at SM/$\overline{\text{DTR}}$ upon detecting a Sync match
10 Output continuous low at SM/$\overline{\text{DTR}}$
11 Output a continuous low at SM/$\overline{\text{DTR}}$ and inhibit Sync match

0 Read/Write data two bytes at a time
1 Read/Write data one byte at a time

000 Select 6 data bits plus even parity
001 Select 6 data bits plus odd parity
010 Select 7 data bits and no parity
011 Select 8 data bits and no parity
100 Select 7 data bits and even parity
101 Select 7 data bits and odd parity
110 Select 8 data bits and even parity
111 Select 8 data bits and odd parity

0 Transmit break code (all 1 bits) on underflow
1 Transmit Sync character on underflow

0 Inhibit
1 Enable all error interrupt requests



0 Select internal Sync mode
1 Select external Sync mode

0 Select two Sync characters
1 Select one Sync character

1 Clear $\overline{\text{CTS}}$ interrupt request

1 Clear transmitter underflow interrupt request

Unassigned

Control register 2, bits 0 and 1, and Control register 3, bits 0, 1, 2 and 3 are used to define synchronization logic.

Control register 3, bit 0 is used to determine whether internal or external synchronization will be employed. If internal synchronization is selected, then Control register 3, bit 1 determines whether one or two Sync characters must precede a serial data stream for initial synchronization to occur.

Control register 2, bits 0 and 1 must now be set so that SM/$\overline{\text{DTR}}$ logic conforms to the synchronization options selected by Control register 3, bits 0 and 1. You also use Control register 2, bits 0 and 1 to select the signal level that will be output for a standard DTR modem control.

Control register 2, bits 2, 3, 4, 5 and 6 define the data transfer options.

Recall that when the CPU reads received data, or writes data to be transmitted, data may be read and written one byte at a time, or two bytes at a time. We discussed this option when describing Status register bits 0 and 1. **You select the one byte or two byte mode via Control register 2, bit 2.**

Control register 2, bits 3, 4 and 5 allow you to define the number of data bits per word, and parity options. These are standard selections which have been described in detail in Volume I, Chapter 5. Notice that the XC6852 provides a much wider variety of data and parity options than the MC6850.

Control register 2, bit 6 determines the response of XC6852 transmit logic when no data is ready to be transmitted. If Control register 2, bit 6 is 0, then a break code will be output on underflow; if this bit is 1, then a Sync character code will be output on underflow. Remember an underflow error will be reported in the Status register only if you transmit Sync character codes on underflow. Therefore, Control register 2, bit 6 must be 1 if underflow errors are to be reported in the Status register. Recall that an underflow error is reported before a Sync character is transmitted; also, the underflow error status is accompanied by a high TUF output signal pulse.

Table 8-7. XC6852 Interrupt Summary

INTERRUPT	ENABLE	REQUEST
RDA — Read Buffer RA or Buffers RA and RT contents	Control register 1 bits 0 and 5 must be 0 and 1 respectively	Status register bit 0 = 1
TDA — Write into Buffer TA or RA and TT	Control register 1 bits 1 and 4 must be 0 and' 1 respectively.	Status register bit 1 = 1. This will not occur if Status register bit 3 = 1.
$\overline{\text{DCD}}$ — Transmitting data carrier disconnected	Control register 2 bit 7 must be 1	On low-to-high transition of $\overline{\text{DCD}}$.
$\overline{\text{CTS}}$ — Receiving external logic disconnected	Control register 2 bit 7 must be 1.	On low-to-high transition of $\overline{\text{CTS}}$.
TUF — Transmit underflow has occurred	Control register 2 bit 7 must be 1.	Status register bit 4 = 1.
OVRN — Receive overrun error has occurred	Control register 2 bit 7 must be 1.	Status register bit 5 = 1.
PE — Parity Error	Control register 2 bit 7 must be 1	Status register bit 6 = 1.

XC6852 INTERRUPT LOGIC

Along with Control register 1, bits 4 and 5, which we have already described, Control register 2, bit 7 and Control register 3, bits 2 and 3 apply to XC6852 interrupt logic. XC6852 interrupt logic is quite complex. There are a number of interrupt sources and no standard procedure for enabling, disabling, acknowledging or processing different interrupt requests. Rather than describing the Control register bits that pertain to interrupts, therefore, **various interrupt options provided by the XC6852 are summarized in Table 8-7.**

PROGRAMMING THE XC6852

Let us now look at the normal sequence of events when programming the XC6852.

First the XC6852 must be initialized. Initialization begins by resetting the XC6852 using the $\overline{\text{RESET}}$ control input. **When the XC6852 is reset this is what happens:**

XC6852 RESET OPERATION

1) Control Register 1, bits 0 and 1 are set to 1, inhibiting transmit and receive logic.
2) Control register 2, bits 0 and 1 are reset to 0, causing SM/$\overline{\text{DTR}}$ to be output high.
3) Control register 2, bit 7 is reset to 0, disabling $\overline{\text{DCD}}$ and $\overline{\text{CTS}}$ interrupt requests, and all error interrupt requests.
4) Control register 3, bit 0 is reset to 0, selecting internal synchronous mode.
5) Status register bit 1 is cleared and held low so that the CPU never reads a status that requests data be written to the XC6852.

Control register bits affected by the $\overline{\text{RESET}}$ control input cannot be modified until $\overline{\text{RESET}}$ goes high again.

Following device Reset, you must load Control registers 1, 2 and 3 and the Sync Code register. The only caution concerns Control register 1; remember, Control register 1, bits 6 and 7 must be modified so that you can access Control registers 2 and 3 and the Sync Code register. When modifying Control register bits 6 and 7, be sure not to inadvertently modify the remaining six bits of Control register 1.

Once the XC6852 has been initialized, you are ready to start transmitting or receiving data.

The only complications associated with transmitting or receiving data involve the way in which you select the programmable options of this device. There is nothing intrinsically different or complicated about the XC6852, as compared to any other synchronous serial I/O device. **These are the only rules to observe:**

1) Always inhibit transmit and receive logic via Control register 1, bits 0 and 1 before modifying the contents of any Control register or the Sync register.
2) Unless you have enabled error interrupts, always precede any data read or write operation by reading the contents of the Status register and checking for errors.
3) **Remember, the XC6852 transmits serial data least significant bit first. This is the inverse of IBM format; and it is up to you to invert the data stream when using an XC6852 with external IBM protocol logic.**

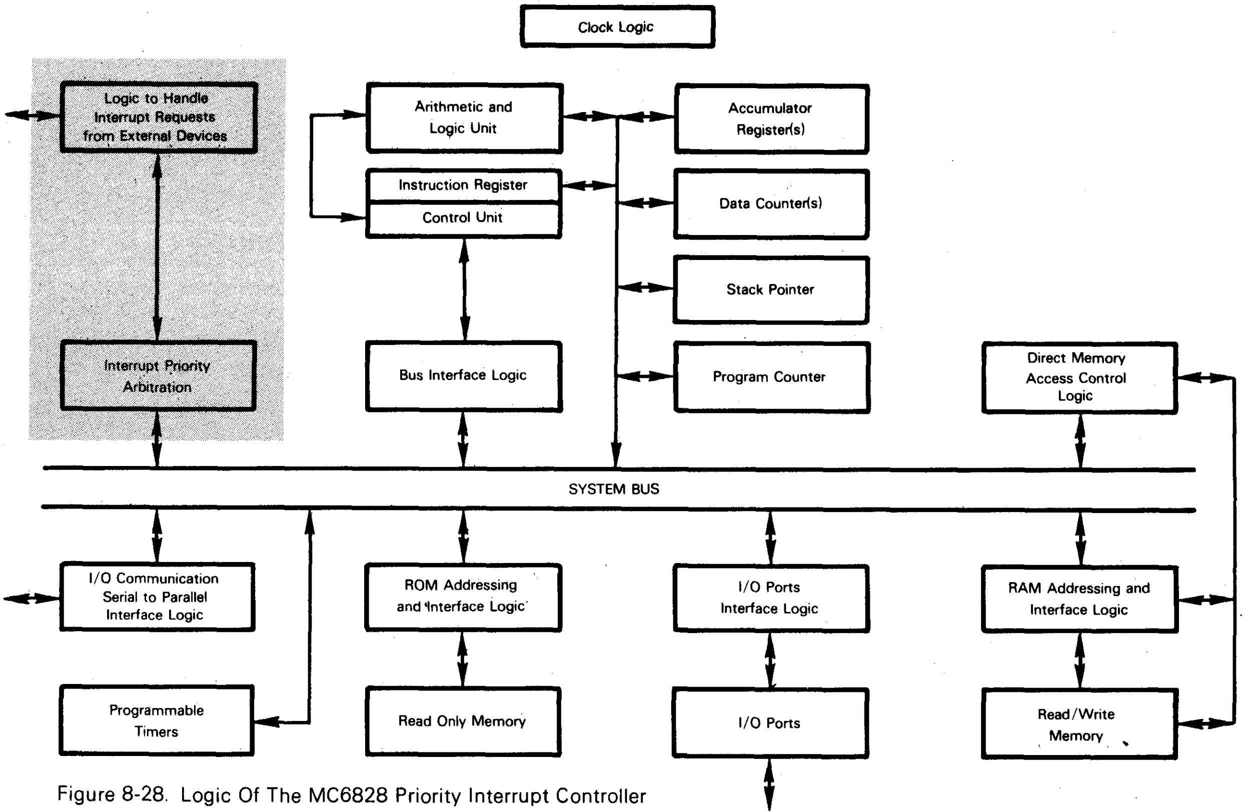

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Interrupt Priority Arbitration
Arithmetic and Logic Unit
Instruction Register
Control Unit
Bus Interface Logic
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
Programmable Timers
ROM Addressing and Interface Logic
Read Only Memory
I/O Ports Interface Logic
I/O Ports
RAM Addressing and Interface Logic
Read/Write Memory

Figure 8-28. Logic Of The MC6828 Priority Interrupt Controller

THE MC8507 (OR MC6828) PRIORITY INTERRUPT CONTROLLER (PIC)

This Priority Interrupt Controller has two part numbers, identifying the fact that it is a bipolar part, and also compatible with the NMOS family of the MC6800 microcomputer devices. We will use the part identification MC6828 in the discussion that follows.

The MC6828 Priority Interrupt Controller processes up to eight external interrupt requests, creating a vectored response to an interrupt acknowledge. Interrupt priorities are determined by pin connections, but under program control you can set a priority level below which all interrupts are inhibited. You cannot have more than one MC6828 in a microcomputer system without resorting to complex multiplexing logic.

Figure 8-28 illustrates that part of our general microcomputer system logic which is provided by the MC6828 PIC.

The MC6828 PIC cannot be compared to the 8259 PICU which is available with 8080A microcomputer systems. The briefest inspection of the two devices will indicate that the 8259 offers a significantly wider range of options — which can be a good thing or a bad thing. As we have often stated, an excessive dependence on interrupt processing in microcomputer applications is hard to justify; in all probability the more limited capabilities of the MC6828 will adequately serve the needs of any reasonable microcomputer application.

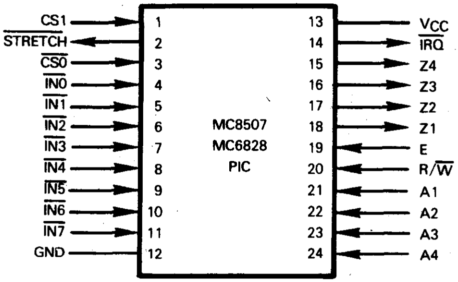
The MC6828 is packaged as a 24-pin DIP. It is fabricated using bipolar LSI technology.

A single +5V power supply is required.

MC6828 PINS AND SIGNALS

MC6828 pins and signals are illustrated in Figure 8-29.

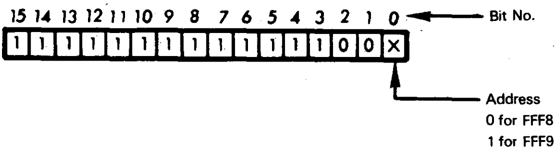
In order to understand this device, you must first look at the way in which it is used within an MC6800 microcomputer system.



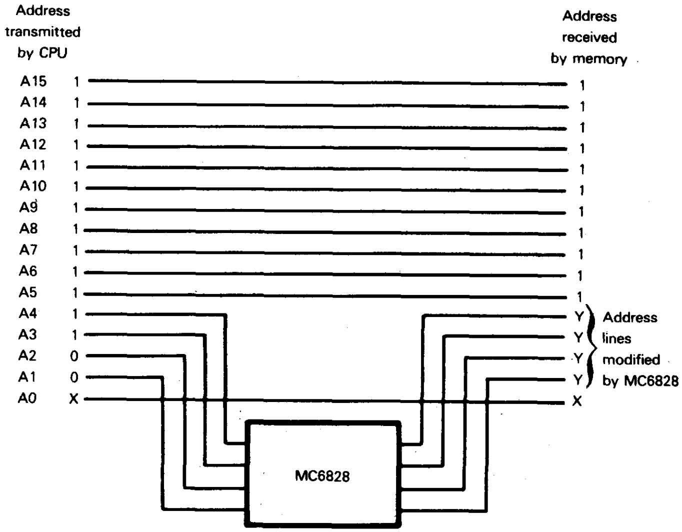
Pin Name	Description	Type
A1 - A4	Termination of system Address Bus lines A1-A4	Input
Z1 - Z4	Continuation of system Address bus lines A1-A4	Output
$\overline{IN0}$ - $\overline{IN7}$	External interrupt requests	Input
$\overline{CS0}$, CS1	Device Select	Input
R/$\overline{W}$	Read/Write control	Input
E	Device Enable	Input
$\overline{STRETCH}$	Clock stretching signal	Output
$\overline{IRQ}$	Interrupt request	Output
V_{CC}, GND	Power and Ground	

Figure 8-29. MC6828 Signals And Pin Assignments

Recall that when any standard external interrupt is acknowledged by an MC6800 CPU, the CPU will fetch the starting address for the interrupt service routine from memory locations $FFF8_{16}$ and $FFF9_{16}$. These two addresses may be illustrated as follows:



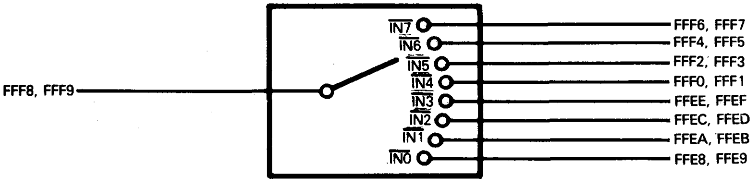
The MC6828 PIC is positioned serially, preceding the external memory device which is to be selected by the addresses $FFF8_{16}$ and $FFF9_{16}$. Address lines A1, A2, A3 and A4 terminate at the MC6828. Logic within the MC6828 appropriately manipulates these four address lines and outputs some value which may differ from the input value. This may be illustrated as follows:



Thus, what the MC6828 does is extend the two addresses $FFF8_{16}$ and $FFF9_{16}$ into 16 addresses, $FFE8_{16}$ through $FFF7_{16}$.

The CPU knows nothing about the address manipulation which is taking place within the MC6828. So far as the CPU is concerned, upon acknowledging an external interrupt, it reads two bytes of data from memory locations $FFF8_{16}$ and $FFF9_{16}$; the fact that there are eight possible responses to these two addresses is of no concern to the CPU.

Conceptually, the MC6828 is acting as an 8-way switch. The CPU addresses the switch by its "stem", via a single address. The actual conduit for the transfer of two bytes of data depends on the switch position at the time the CPU accesses the switch stem; and the switch position is going to be determined by the highest priority active interrupt request. This may be illustrated as follows:



The only stipulation made by the MC6828 is that memory addresses $FFE0_{16}$ through $FFFF_{16}$ never access read/write memory.

Let us now look at the device pins and signals.

A1 - A4 represents the termination of System Address Bus lines A1 - A4 at the MC6828.

The continuation of the four address lines is via pins Z1 - Z4.

The eight external interrupt requests are connected to $\overline{IN0}$ - $\overline{IN7}$. Interrupt priorities are in ascending level, from $\overline{IN0}$ which has lowest priority through $\overline{IN7}$ which has highest priority.

Device select logic consists of $\overline{CS0}$ and CS1. For this device to be selected, $\overline{CS0}$ must be low while CS1 is high. **There are additional select requirements** that depend on the operation being performed, as we will describe shortly.

R/$\overline{W}$ is the read/write control output by the MC6800 CPU.

E is the standard enable signal required by all support devices of an MC6800 microcomputer system. You can extend the response time available to the MC6828 by extending the E input.

A $\overline{STRETCH}$ output is created and can be connected directly to the clock device of the microcomputer system in order to provide as much response time as needed by the MC6828.

The actual interrupt request which generates the entire response process occurs via the $\overline{IRQ}$ output from the MC6828. This output will normally be connected to the MC6800 $\overline{IRQ}$ input.

THE INTERRUPT ACKNOWLEDGE PROCESS

When any one of the eight interrupt request lines $\overline{IN0}$ - $\overline{IN7}$ is low, an interrupt request is generated via $\overline{IRQ}$. This interrupt request is passed on to the MC6800 CPU.

As is normal, the MC6800, upon acknowledging the interrupt request, will perform two read operations; during these read operations the contents of memory locations $FFF8_{16}$ and $FFF9_{16}$ are read. The MC6800 CPU interprets the contents of these two memory locations as a 16-bit address, identifying the beginning of the interrupt service routine which is to be executed following the acknowledge.

When the MC6800 CPU is reading the contents of memory locations FFF8$_{16}$ and FFF9$_{16}$, these are the signal levels for the control and select inputs to the MC6828:

R/$\overline{W}$	$\overline{CS0}$	CS1	A4	A3	A2	A1
1	0	1	1	1	0	0

The MC6828 interprets the signal combination R/W·$\overline{CS0}$·CS1·$\overline{A1}$·$\overline{A2}$·A3·A4, as a special select, causing it to output binary data on the Z1, Z2, Z3 and Z4 pins representing the highest priority active interrupt request occurring on any of the interrupt request pins $\overline{IN0}$ - $\overline{IN7}$. Table 8-8 defines the binary data output corresponding to each interrupt level.

If R/$\overline{W}$ is high, $\overline{CS0}$ is low and CS1 is high, but A1, A2, A3, A4 are not 0011, then the MC6828 will simply output, via Z1 - Z4, whatever is being input via A1 - A4. Thus, the presence of the MC6828 on the A1 - A4 address lines of the Address Bus will be transparent until either the address FFF8$_{16}$ or the address FFF9$_{16}$ appears on the Address Bus.

In order to guarantee that the MC6828 remains synchronized with the rest of the MC6800 microcomputer system, **logic internal to the MC6828 uses the E synchronization signal as part of internal enable logic.** The way in which the E synchronization signal is used is of no particular concern to you, as an MC6828 user. Providing the E synchronization signal which drives the rest of the MC6800 microcomputer system also drives the MC6828, problems will not arise.

Table 8-8. MC6828 Address Vectors Created For Eight Priority Interrupt Requests

PRIORITY	PIN	Z4	Z3	Z2	Z1	EFFECTIVE ADDRESSES
Highest 7	$\overline{IN7}$	1	0	1	1	FFF6 and FFF7
6	$\overline{IN6}$	1	0	1	0	FFF4 and FFF5
5	$\overline{IN5}$	1	0	0	1	FFF2 and FFF3
4	$\overline{IN4}$	1	0	0	0	FFF0 and FFF1
3	$\overline{IN3}$	0	1	1	1	FFEE and FFEF
2	$\overline{IN2}$	0	1	1	0	FFEC and FFED
1	$\overline{IN1}$	0	1	0	1	FFEA and FFEB
Lowest 0	$\overline{IN0}$	0	1	0	0	FFE8 and FFE9

INTERRUPT PRIORITIES

Table 8-8 defines the priorities that will be applied to simultaneous interrupt requests occurring at pins $\overline{IN0}$ - $\overline{IN7}$. This table also indicates the exact memory addresses which will be created by the MC6828 in response to each of the interrupt requests. In order to use the MC6828 PIC in an MC6800 microcomputer system, 16 bytes of PROM or ROM, selected by the addresses given in Table 8-8 must be connected to the MC6828. Within these 16 bytes of PROM or ROM, you must store the starting addresses for the eight interrupt service routines which are going to be executed following acknowledgement of each possible external interrupt request. For example, suppose that interrupt requests arriving at the $\overline{IN5}$ pin of the MC6828 must be serviced by an interrupt service routine whose first executable instruction is stored in memory location 2E00$_{16}$. The value 2E00$_{16}$ must then be stored in the two PROM or ROM bytes selected by memory addresses FFF2$_{16}$ and FFF3$_{16}$. Remember, the high order byte of an address is always stored at the lower address. Thus 2E$_{16}$ will be stored in memory location FFF2$_{16}$ while 00$_{16}$ is stored in memory location FFF3$_{16}$.

INTERRUPT INHIBIT LOGIC

The MC6828 provides a very elementary level of interrupt inhibit logic. You can output a mask to the MC6828 identifying a priority level below which all interrupts will be inhibited.

Now the mask is written out to the MC6828 in a very unusual way.

Recall that the MC6828 requires memory addresses $FFE8_{16}$ through $FFF9_{16}$ to access PROM or ROM. Any attempt to write into these memory addresses will be ignored. The MC6828 takes advantage of this fact by trapping attempts to write into memory locations $FFE8_{16}$ through $FFF9_{16}$.

That is to say, when $R/\overline{W}$ is low while $\overline{CS0}$ is low and CS1 is high, the MC6828 considers itself selected, but it interprets the four address lines A1, A2, A3, A4 as data, defining the mask level below which interrupts will be inhibited. **Table 8-9 defines the way in which the mask specified by address lines A1, A2, A3 and A4 will be interpreted.**

Table 8-9. MC6828 Interrupt Masks — Their Creation And Interpretation

Write anything to this address:	and Address Bus lines A1-A4 will have this value:	Which will inhibit all interrupts, including and below:
FFE0 or FFE1	0000	All interrupts enabled
FFE2 or FFE3	0001	IN1
FFE4 or FFE5	0010	IN2
FFE6 or FFE7	0011	IN3
FFE8 or FFE9	0100	IN4
FFEA or FFEB	0101	IN5
FFEC or FFED	0110	IN6
FFEE or FFEF	0111	IN7
FFF0 through FFFF	1000 through 1111	All interrupts disabled

DATA SHEETS

The following section contains data on electrical characteristics and specific delay times for the MC6800 and other devices described in this chapter.

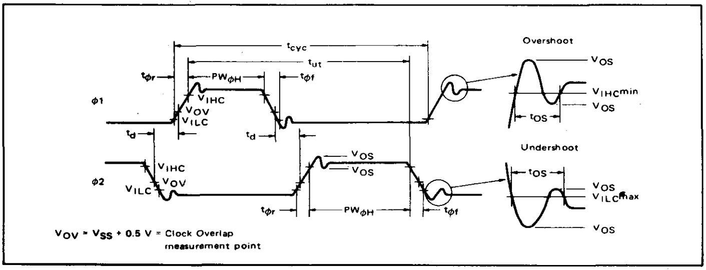
ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 V ± 5%, V_{SS} = 0, T_A = 0 to 70°C unless otherwise noted.)

Characteristic		Symbol	Min	Typ	Max	Unit
Input High Voltage	Logic φ1,φ2	V_{IH} V_{IHC}	$V_{SS}+2.0$ $V_{CC}-0.3$	– –	V_{CC} $V_{CC}+0.1$	Vdc
Input Low Voltage	Logic φ1,φ2	V_{IL} V_{ILC}	$V_{SS}-0.3$ $V_{SS}-0.1$	– –	$V_{SS}+0.8$ $V_{SS}+0.3$	Vdc
Clock Overshoot/Undershoot – Input High Level – Input Low Level		V_{OS}	$V_{CC}-0.5$ $V_{SS}-0.5$	– –	$V_{CC}+0.5$ $V_{SS}+0.5$	Vdc
Input Leakage Current (V_{in} = 0 to 5.25 V, V_{CC} = max) (V_{in} = 0 to 5.25 V, V_{CC} = 0.0 V)	 Logic* φ1,φ2	I_{in}	 – –	 1.0 –	 2.5 100	μAdc
Three-State (Off State) Input Current (V_{in} 0.4 to 2.4 V, V_{CC} = mak)	D0-D7 A0-A15,R/W	I_{TSI}	– –	2.0 –	10 100	μAdc
Output High Voltage (I_{Load} = −205 μAdc, V_{CC} = min) (I_{Load} = −145 μAdc, V_{CC} = min) (I_{Load} = −100 μAdc, V_{CC} = min)	 D0-D7 A0-A15,R/W,VMA BA	V_{OH}	 $V_{SS}+2.4$ $V_{SS}+2.4$ $V_{SS}+2.4$	 – – –	 – – –	Vdc
Output Low Voltage (I_{Load} = 1.6 mAdc, V_{CC} = min)		V_{OL}	–	–	$V_{SS}+0.4$	Vdc
Power Dissipation		P_D	–	0.600	1.2	W
Capacitance # (V_{in} = 0, T_A = 25°C, f = 1.0 MHz)	φ1,φ2 TSC DBE D0-D7 Logic Inputs	C_{in}	80 – – – –	120 – 7.0 10 6.5	160 15 10 12.5 8.5	pF
	A0-A15,R/W,VMA	C_{out}	–	–	12	pF
Frequency of Operation		f	0.1	–	1.0	MHz
Clock Timing (Figure 1) Cycle Time		t_{cyc}	1.0	–	10	μs
Clock Pulse Width (Measured at V_{CC} – 0.3 V)	 φ1 φ2	$PW_{\phi H}$	 430 450	 – –	 4500 4500	ns
Total φ1 and φ2 Up Time		t_{ut}	940	–	–	ns
Rise and Fall Times (Measured between V_{SS} + 0.3 V and V_{CC} – 0.3 V)	φ1,φ2	$t_{\phi r}$, $t_{\phi f}$	5.0	–	50	ns
Delay Time or Clock Separation (Measured at V_{OV} = V_{SS} + 0.5 V)		t_d	0	–	9100	ns
Overshoot Duration		t_{OS}	0	–	40	ns

*Except $\overline{IRQ}$ and $\overline{NMI}$, which require 3 kΩ pullup load resistors for wire-OR capability at optimum operation.

#Capacitances are periodically sampled rather than 100% tested.

FIGURE 1 – CLOCK TIMING WAVEFORM



MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	Vdc
Input Voltage	V_{in}	–0.3 to +7.0	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature Range	T_{stg}	–55 to +150	°C
Thermal Resistance	θ_{JA}	70	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high impedance circuit.

READ/WRITE TIMING

Figures 2 and 3, f = 1.0 MHz, Load Circuit of Figure 6.

Characteristic	Symbol	Min	Typ	Max	Unit
Address Delay	t_{AD}	–	220	300	ns
Peripheral Read Access Time $t_{acc} = t_{ut} - (t_{AD} + t_{DSR})$	t_{acc}	–	–	540	ns
Data Setup Time (Read)	t_{DSR}	100	–	–	ns
Input Data Hold Time	t_H	10	–	–	ns
Output Data Hold Time	t_H	10	25	–	ns
Address Hold Time (Address, R/W, VMA)	t_{AH}	50	75	–	ns
Enable High Time for DBE Input	t_{EH}	450	–	–	ns
Data Delay Time (Write)	t_{DDW}	–	165	225	ns
Processor Controls*					
Processor Control Setup Time	t_{PCS}	200	–	–	ns
Processor Control Rise and Fall Time	t_{PCr}, t_{PCf}	–	–	100	ns
Bus Available Delay	t_{BA}	–	–	300	ns
Three State Enable	t_{TSE}	–	–	40	ns
Three State Delay	t_{TSD}	–	–	700	ns
Data Bus Enable Down Time During ϕ1 Up Time (Figure 3)	t_{DBE}	150	–	–	ns
Data Bus Enable Delay (Figure 3)	t_{DBED}	300	–	–	ns
Data Bus Enable Rise and Fall Times (Figure 3)	t_{DBEr}, t_{DBEf}	–	–	25	ns

*Additional information is given in Figures 12 through 16 of the Family Characteristics – see pages 17 through 20.

FIGURE 2 – READ DATA FROM MEMORY OR PERIPHERALS

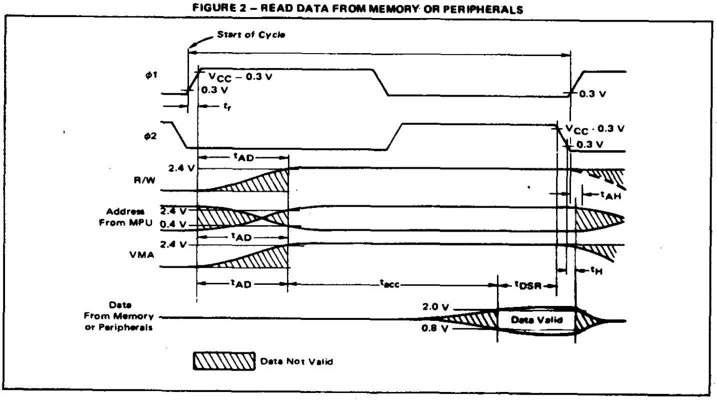


FIGURE 3 – WRITE IN MEMORY OR PERIPHERALS

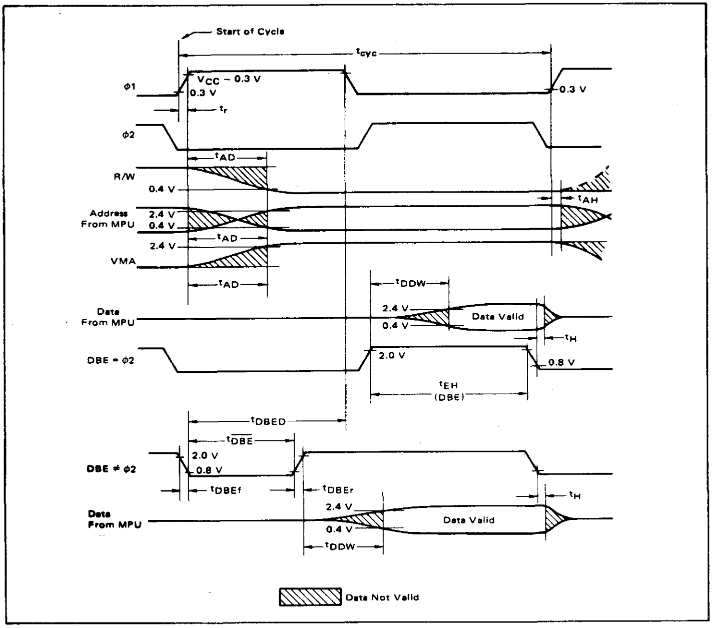


FIGURE 4 – TYPICAL DATA BUS OUTPUT DELAY versus CAPACITIVE LOADING

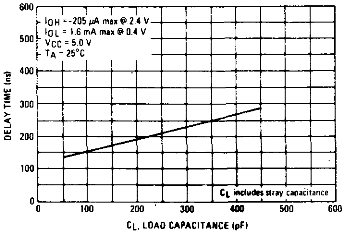


FIGURE 5 – TYPICAL READ/WRITE, VMA, AND ADDRESS OUTPUT DELAY versus CAPACITIVE LOADING

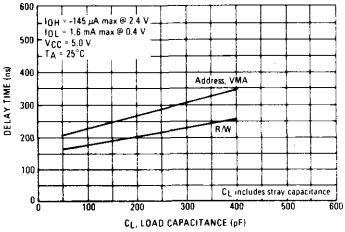
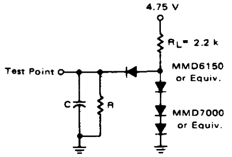


FIGURE 6 – BUS TIMING TEST LOAD



C = 130 pF for D0-D7
= 90 pF for A0-A15, R/W, and VMA
= 30 pF for BA
R = 11.7 kΩ for D0-D7
= 16.5 kΩ for A0-A15, R/W, and VMA
= 24 kΩ for BA

TYPICAL POWER SUPPLY CURRENT

FIGURE 7 – VARIATIONS WITH FREQUENCY

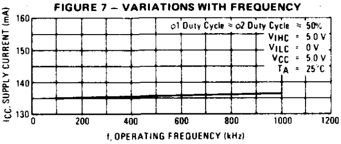
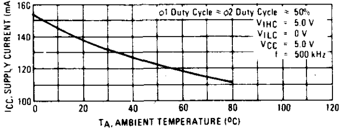
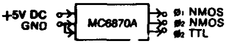


FIGURE 8 – VARIATIONS WITH TEMPERATURE

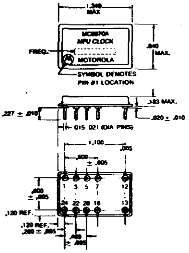


MC6870A

limited function microprocessor clock
250 kHz to 2.5 MHz



dimensions



PIN	CONNECTION
1	GND
3	NC
5	Ø₂ TTL
7	V_{cc} (+5VDC)
12	Ø₂ NMOS
13	Ø₁ NMOS
18	GND
20	NC
22	NC
24	NC

Note: All dimensions are in inches

specifications

Rating	Symbol	Value	Unit
Supply Voltage	V_{cc}	5.00±5%	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature	T_{stg}	−55 to +125	°C
Power Supply Drain (max.)	I_{pd}	100	mA

ELECTRICAL CHARACTERISTICS (V_{cc} = 5.0 ± 5%, V_{ss} = O,T_A = 0° to 70°C, unless otherwise noted)

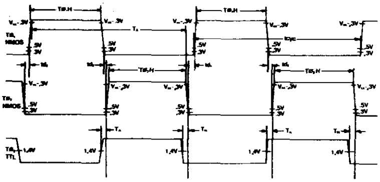
Characteristic	Symbol	Min	Typ	Max	Unit
Frequency					
Operating Frequency	f_c	.250		2.5	MHz
Frequency stability (inclusive of calibration tolerance at +25°C, operating temperature, input voltage change, load change, aging, shock and vibration)			±.01		%
NMOS Outputs at 1.0 MHz Operation**					
Pulse Width (meas. at V_{cc}= −.3V dc level)	TØ₁H TØ₂H	430 450			ns ns
Logic Levels	V_{OLC} V_{OHC}	V_{ss}-.1 V_{cc}-.3	− −	V_{ss}+.3 V_{cc}+.1	Vdc Vdc
Rise and Fall Times	t_r t_f	5 5	12 12	50 50	ns ns
*Overshoot/Undershoot Logic "1" Logic "0"	V_{OS}	V_{cc}-.5 V_{ss}-.5		V_{cc}+.5 V_{ss}+.5	Vdc Vdc
Pulse duration of any over-shoot or undershoot	T_{OS}			40	ns
Period @ 0.3V dc Level	t_{cyc}		1.00		us
Edge Timing @ V_{cc}=0.3V dc	Tx	940			ns
NMOS Relationship @ +0.5V dc Level	t_{d1} t_{d2}	0 0		8.0	us
TTL Outputs					
In ref. to Ø₂ NMOS @ 0.3V dc					
Ø₂ TTL @ +1.4V dc	T_A T_H	15 10	30 25	45 40	ns ns
Logic Levels	V_{OH} V_{OL}	2.4	3.2 .3	.4	Vdc Vdc
Rise and Fall Times .4V and 2.4V 2.4V and .4V	t_r t_f			15 15	ns ns
Logic "0" Sink (/Gate)	I_{OL}			−1.6	mA
Logic "1" Source (/Gate)	I_{OH}			+40	uA
Current Output Shorted	I_{SC}	−18		−57	mA
Load					
NMOS—Load Capacity Ø₁, Ø₂	C_{NMOS}	80	120	160	pf
TTL—No. of Loads				5	ttl
TTL—Load Capacity	C_{TTL}			50	pf

*Into specified test load

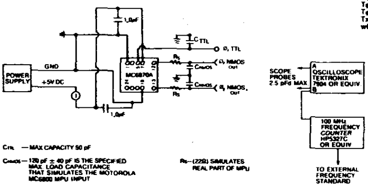
**Apply the following parameters for frequencies other than 1.0 MHz:
TφₗH=0.5 (P-140) ns
Tφ₂H=0.5 (P-100) ns
Tx=(P-60) ns
where P=desired period of operation in nanoseconds

WAVEFORM TIMING

(ALL TIME IN NANOSECONDS)



TEST CIRCUIT


1.0μF
C TTL
Ø₂ TTL
Rs
C NMOS
Ø₂ NMOS OUT
Ø₁ NMOS OUT
MC6870A
POWER SUPPLY
GND
+5V DC
1.0μF
SCOPE PROBES 2.5 pFd MAX
A
OSCILLOSCOPE TEKTRONIX 7904 OR EQUIV.
B
100 MHz FREQUENCY COUNTER HP5327C OR EQUIV
TO EXTERNAL FREQUENCY STANDARD

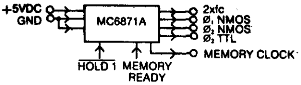
C_{TTL} —MAX CAPACITY 50 pF

C_{NMOS}—120 pF ± 40 pF IS THE SPECIFIED MAX. LOAD CAPACITANCE THAT SIMULATES THE MOTOROLA MC6800 MPU INPUT

Rs—(22Ω) SIMULATES REAL PART OF MPU

MC6871A

full.function microprocessor clock
850 kHz to 2.5 MHz



specifications

Rating	Symbol	Value	Unit
Supply Voltage	V_{cc}	5.00±5%	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature	T_{stg}	−55 to +125	°C
Power Supply Drain (max.)	I_{dd}	100	mA

ELECTRICAL CHARACTERISTICS (V_{cc} = 5.0 ± 5%, V_{ss} = O,T_A = 0° to 70°C, unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
Frequency					
Operating Frequency	f_c	.850		2.5	MHz
Frequency stability (inclusive of calibration tolerance at +25°C, operating temperature, input voltage change, load change, aging, shock and vibration)			±.01		%
NMOS Outputs at 1.0 MHz Operation* **					
Pulse Width (meas. at V_{cc}= −.3V dc level)	TØ₁H TØ₂H	430 450			ns ns
Logic Levels	V_{OLC} V_{OHC}	V_{ss}-.1 V_{cc}-.3	— —	V_{ss}+.3 V_{cc}+.1	Vdc Vdc
Rise and Fall Times	t_r t_f	5 5	12 12	50 50	ns ns
*Overshoot/Undershoot Logic "1" Logic "0"	V_{OS}	V_{cc}-.5 V_{ss}-.5		V_{cc}+.5 V_{ss}+.5	Vdc Vdc
Pulse duration of any overshoot or undershoot	T_{OS}			40	ns
Period @ 0.3V dc Level	t_{cyc}		1.00		us
Edge Timing @ V_{cc}=0.3V dc	Tx	940			ns
NMOS Relationship @ +0.5V dc Level	$t_{\phi 1}$ $t_{\phi 2}$	0 0		8.0	us
TTL Outputs					
In ref. to Ø₁ NMOS @ 0.3V dc					
Ø₂ TTL @ 1.4V dc	T_A T_H	15 10	30 25	45 40	ns ns
Memory Clock @ 1.4V dc	T_C T_J	30 20	50 40	70 60	ns ns
2xfc @ 1.4V dc	T_B	40	80	120	ns
Logic Levels	V_{OH} V_{OL}	2.4	3.2 .3	.4	Vdc Vdc
Rise and Fall Times .4V and 2.4V 2.4V and .4V	t_r t_f			15 15	ns ns
Logic "0" Sink (/Gate)	I_{OL}			−1.6	mA
Logic "1" Source (/Gate)	I_{OH}			+40	uA
Current Output Shorted.	I_{SC}	−18		−57	mA
Load					
NMOS—Load Capacity Ø₁, Ø₂	C_{NMOS}	80	120	160	pf
TTL—No. of Loads				5	ttl
TTL—Load Capacity	C_{TTL}			50	pf
Logic Inputs ("0" Level Applies HOLD or MEMORY READY)**					
Holds Ø₁ NMOS 'High', Ø₂ NMOS 'Low', Ø₂ TTL 'Low'	$\overline{\text{HOLD 1}}$	−.2		+.4	Vdc
Holds Ø₁ NMOS 'Low', Ø₂ NMOS 'High', Ø₂ TTL 'High', and MEMORY CLOCK 'High'	MEMORY READY	−.2		+.4	Vdc

*Into specified test load

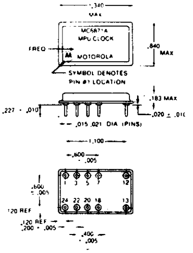
**Must be externally held at "1" level (2.4V min., 5.0V max.) if not used

***Apply the following parameters for frequencies other than 1 MHz
TØ₁H=0.5 (P-140) ns
TØ₂H=0.5 (P-100) ns
Tx=(P-60) ns
where P=desired period of operation in nanoseconds

PIN	CONNECTION
1	GND
3	MEMORY CLOCK
5	Ø₂ TTL
7	V_{cc} (+5VDC)
12	Ø₂ NMOS
13	Ø₁ NMOS
18	GND
20	$\overline{\text{HOLD 1}}$
22	MEMORY READY
24	2xfc

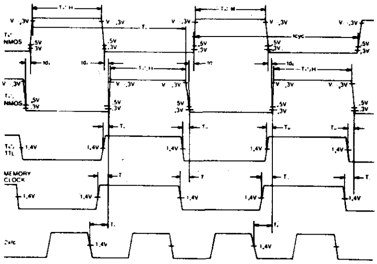
Note All dimensions are in inches

DIMENSIONS

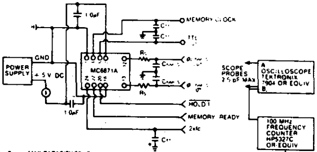


WAVEFORM TIMING

(ALL TIME IN NANOSECONDS)



TEST CIRCUIT



C_{TTL} - MAX CAPACITY 50 pF

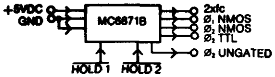
C_{NMOS} - 120 pF ± 40 pF IS THE SPECIFIED MAX. LOAD CAPACITANCE THAT SIMULATES THE MOTOROLA MC6800 MPU INPUT

R_S (22Ω) SIMULATES REAL PART OF MPU

HOLD AND MEMORY READY MUST BE EXTERNALLY HELD AT "1" LEVEL (2.4VDC MIN. 5.0VDC MAX) WHEN NOT USED

MC6871B

alternate function microprocessor clock
250 kHz to 2.5 MHz



specifications

Rating	Symbol	Value	Unit
Supply Voltage	V_{cc}	5.00±5%	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature	T_{stg}	−55 to +125	°C
Power Supply Drain (max.)	I_{pd}	100	mA

ELECTRICAL CHARACTERISTICS (V_{cc} = 5.0 ± 5%, V_{ss} = 0, T_A = 0° to 70°C, unless otherwise noted)

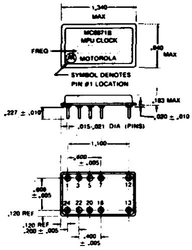
Characteristic	Symbol	Min	Typ	Max	Unit
Frequency					
Operating Frequency Frequency stability (inclusive of calibration tolerance at +25°C, operating temperature, input voltage change, load change, aging, shock and vibration)	f_c	250	 ±.01	2.5	MHz %
NMOS Outputs at 1.0 MHz Operation***					
Pulse Width (meas. at V_{cc} = −.3V dc level)	TØ₁H TØ₂H	430 450			ns ns
Logic Levels	V_{OLC} V_{OHC}	V_{ss}-.1 V_{cc}-.3	– –	V_{ss}+.3 V_{cc}+.1	Vdc Vdc
Rise and Fall Times	t_r t_f	5 5	12 12	50 50	ns ns
*Overshoot/Undershoot Logic "1" Logic "0"	 V_{OS}	 V_{cc}-.5 V_{ss}-.5		 V_{cc}+.5 V_{ss}+.5	 Vdc Vdc
Pulse duration of any overshoot or undershoot	T_{OS}			40	ns
Period @ 0.3V dc Level	t_{cyc}		1.00		us
Edge Timing @ V_{cc}=0.3V dc	Tx	940			ns
NMOS Relationship @ +0.5V dc	t_{d1} t_{d2}	0 0		 8.0	 us
TTL Outputs					
In ref. to Ø₂ NMOS @ 0.3V dc					
Ø₂ TTL @ 1.4V dc	T_A T_H	15 10	30 25	45 40	ns ns
Ø₂ Ungated @ 1.4V dc	T_C T_J	30 20	50 40	70 60	ns ns
2xfc @ 1.4V dc	T_B	40	80	120	ns
Logic Levels	V_{OH} V_{OL}	2.4	3.2 .3	 .4	Vdc Vdc
Rise and Fall Times .4V and 2.4V 2.4V and .4V	 t_r t_f			 15 15	 ns ns
Logic "0" Sink (/Gate)	I_{OL}			−1.6	mA
Logic "1" Source (/Gate)	I_{OH}			+40	uA
Current Output Shorted	I_{SC}	−18		−57	mA
Load					
NMOS—Load Capacity Ø₁, Ø₂	C_{NMOS}	80	120	160	pf
TTL—No. of Loads				5	ttl
TTL—Load Capacity	C_{TTL}			50	pf
Logic Inputs ("0"-Level applies HOLD)**					
Holds Ø₁ NMOS 'High', Ø₂ NMOS 'Low', Ø₂ TTL 'Low'	$\overline{\text{HOLD 1}}$	−.2		+.4	Vdc
Holds Ø₁ NMOS 'Low', Ø₂ NMOS 'High', Ø₂ TTL 'High'	$\overline{\text{HOLD 2}}$	−.2		+.4	Vdc

*Into specified test load

**Must be externally held at "1" level (2.4V min., 5.0V max.) if not used

***Apply the following parameters for frequencies other than 1 MHz
To₁H=0.5 (P-140) ns
To₂H=0.5 (P-100) ns
Tx=(P-60) ns
where P=desired period of operation in nanoseconds

DIMENSIONS



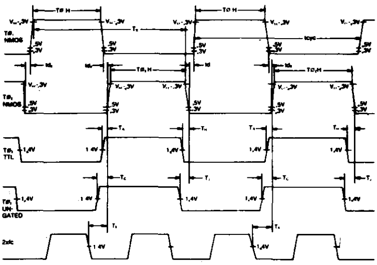
PIN	CONNECTION
1	GND
3	Ø₂ TTL UNGATED
5	Ø₂ TTL
7	V_{cc} (+5VDC)
12	Ø₂ NMOS
13	Ø₁ NMOS
18	GND
20	$\overline{\text{HOLD 1}}$
22	$\overline{\text{HOLD 2}}$
24	2xfc

Note: 4xfc available on request

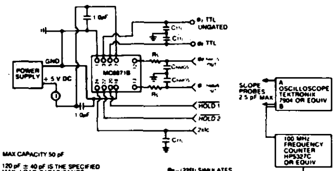
Note: All dimensions are in inches

WAVEFORM TIMING.

ALL-TIME IN NANOSECONDS.



TEST DIAGRAM



TO EXTERNAL FREQUENCY STANDARD

C_{TTL} MAX CAPACITY 50 pF

C_{NMOS} - 120 pF ± 40 pF IS THE SPECIFIED MAX. LOAD CAPACITANCE THAT SIMULATES THE MOTOROLA MC6800 MPU INPUT

R_S - (22Ω) SIMULATES REAL PART OF MPU

*$\overline{\text{HOLD 1}}$ AND $\overline{\text{HOLD 2}}$ MUST BE EXTERNALLY HELD AT "1" LEVEL (2.4VDC MIN. 5.0VDC MAX.) WHEN NOT USED

ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 V ±5%, V_{SS} = 0, T_A = 0 to 70°C unless otherwise noted.)

Characteristic	Symbol	Min	Typ	Max	Unit
Input High Voltage Enable Other Inputs	V_{IH}	V_{SS} + 2.4 V_{SS} + 2.0	– –	V_{CC} V_{CC}	Vdc
Input Low Voltage Enable Other Inputs	V_{IL}	V_{SS} -0.3 V_{SS} -0.3	– –	V_{SS} + 0.4 V_{SS} + 0.8	Vdc
Input Leakage Current R/W, $\overline{Reset}$, *RS0, RS1, CS0, CS1*, $\overline{CS2}$, CA1, (V_{in} = 0 to 5.25 Vdc) CB1, Enable	I_{in}	–	1.0	2.5	µAdc
Three-State (Off State) Input Current D0-D7, PB0-PB7, CB2 (V_{in} = 0.4 to 2.4 Vdc)	I_{TSI}	–	2.0	10	µAdc
Input High Current PA0-PA7, CA2 (V_{IH} = 2.4 Vdc)	I_{IH}	-100	-250	–	µAdc
Input Low Current PA0-PA7, CA2 (V_{IL} = 0.4 Vdc)	I_{IL}	–	-1.0	-1.6	mAdc
Output High Voltage (I_{Load} = -205 µAdc, Enable Pulse Width < 25 µs) D0-D7 (I_{Load} = –100 µAdc, Enable Pulse Width <25 µs) Other Outputs	V_{OH}	 V_{SS} + 2.4 V_{SS} + 2.4	 – –	 – –	Vdc
Output Low Voltage (I_{Load} = 1.6 mAdc, Enable Pulse Width < 25 µs)	V_{OL}	–	–	V_{SS} + 0.4	Vdc
Output High Current (Sourcing) (V_{OH} = 2.4 Vdc) D0-D7 Other Outputs (V_O = 1.5 Vdc, the current for driving other than TTL, e.g., Darlington Base) PB0-PB7, CB2	I_{OH}	 -205 -100 -1.0	 – – -2.5	 – – -10	 µAdc µAdc mAdc
Output Low Current (Sinking) (V_{OL} = 0.4 Vdc)	I_{OL}	1.6	–	–	mAdc
Output Leakage Current (Off State) $\overline{IRQA}$, $\overline{IRQB}$ (V_{OH} = 2.4 Vdc)	I_{LOH}	–	1.0	10	µAdc
Power Dissipation	P_D	–	–	650	mW
Input Capacitance Enable (V_{in} = 0, T_A = 25°C, f = 1.0 MHz) D0-D7 PA0-PA7, PB0-PB7, CA2, CB2 R/W, $\overline{Reset}$, RS0, RS1, CS0, CS1, $\overline{CS2}$, CA1, CB1	C_{in}	– – – –	– – – –	20 12.5 10 7.5	pF
Output Capacitance $\overline{IRQA}$, $\overline{IRQB}$ (V_{in} = 0, T_A = 25°C, f = 1.0 MHz) PB0-PB7	C_{out}	– –	– –	5.0 10	pF
Peripheral Data Setup Time (Figure 1)	t_{PDSU}	200	–	–	ns
Delay Time, Enable negative transition to CA2 negative transition (Figure 2, 3)	t_{CA2}	–	–	1.0	µs
Delay Time, Enable negative transition to CA2 positive transition (Figure 2)	t_{RS1}	–	–	1.0	µs
Rise and Fall Times for CA1 and CA2 input signals (Figure 3)	t_r, t_f	–	–	1.0	µs
Delay Time from CA1 active transition to CA2 positive transition (Figure 3)	t_{RS2}	–	–	2.0	µs
Delay Time, Enable negative transition to Peripheral Data valid (Figures 4, 5)	t_{PDW}	–	–	1.0	µs
Delay Time, Enable negative transition to Peripheral CMOS Data Valid (V_{CC} – 30% V_{CC}, Figure 4; Figure 12 Load C) PA0-PA7, CA2	t_{CMOS}	–	–	2.0	µs
Delay Time, Enable positive transition to CB2 negative transition (Figure 6, 7)	t_{CB2}	–	–	1.0	µs
Delay Time, Peripheral Data valid to CB2 negative transition (Figure 5)	t_{DC}	20	–	–	ns
Delay Time, Enable positive transition to CB2 positive transition (Figure 6)	t_{RS1}	–	–	1.0	µs
Rise and Fall Time for CB1 and CB2 input signals (Figure 7)	t_r, t_f	–	–	1.0	µs
Delay Time, CB1 active transition to CB2 positive transition (Figure 7)	t_{RS2}	–	–	2.0	µs
Interrupt Release Time, $\overline{IRQA}$ and $\overline{IRQB}$ (Figure 8)	t_{IR}	–	–	1.6	µs
Reset Low Time* (Figure 9)	t_{RL}	2.0	–	–	µs

*The Reset line must be high a minimum of 1.0 µs before addressing the PIA.

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	Vdc
Input Voltage	V_{in}	–0.3 to +7.0	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature Range	T_{stg}	–55 to +150	°C
Thermal Resistance	θ_{JA}	82.5	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high impedance circuit.

BUS TIMING CHARACTERISTICS

READ (Figures 10 and 12)

Characteristic	Symbol	Min	Typ	Max	Unit
Enable Cycle Time	t_{cycE}	1.0	–	–	µs
Enable Pulse Width, High	PW_{EH}	0.45	–	25	µs
Enable Pulse Width, Low	PW_{EL}	0.43	–	–	µs
Setup Time, Address and R/W valid to Enable positive transition	t_{AS}	160	–	–	ns
Data Delay Time	t_{DDR}	–	–	320	ns
Data Hold Time	t_H	10	–	–	ns
Address Hold Time	t_{AH}	10	–	–	ns
Rise and Fall Time for Enable input	t_{Er}, t_{Ef}	–	–	25	ns

WRITE (Figures 11 and 12)

Characteristic	Symbol	Min	Typ	Max	Unit
Enable Cycle Time	t_{cycE}	1.0	–	–	µs
Enable Pulse Width, High	PW_{EH}	0.45	–	25	µs
Enable Pulse Width, Low	PW_{EL}	0.43	–	–	µs
Setup Time, Address and R/W valid to Enable positive transition	t_{AS}	160	–	–	ns
Data Setup Time	t_{DSW}	195	–	–	ns
Data Hold Time	t_H	10	–	–	ns
Address Hold Time	t_{AH}	10	–	–	ns
Rise and Fall Time for Enable input	t_{Er}, t_{Ef}	–	–	25	ns

FIGURE 1 – PERIPHERAL DATA SETUP TIME
(Read Mode)

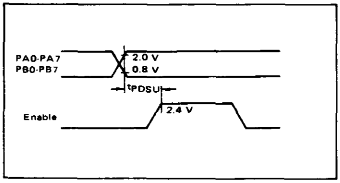
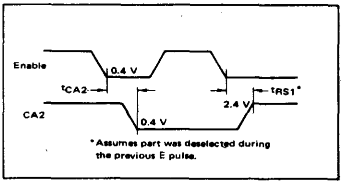


FIGURE 2 – CA2 DELAY TIME
(Read Mode; CRA-5 = CRA-3 = 1, CRA-4 = 0)



*Assumes part was deselected during the previous E pulse.

FIGURE 3 – CA2 DELAY TIME
(Read Mode; CRA-5 = 1, CRA-3 = CRA-4 = 0)

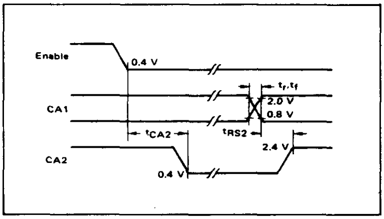


FIGURE 4 – PERIPHERAL CMOS DATA DELAY TIMES
(Write Mode; CRA-5 = CRA-3 = 1, CRA-4 = 0)

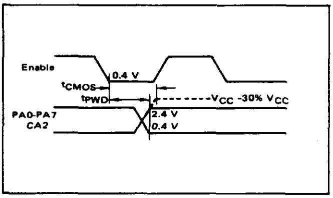
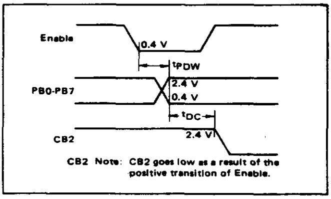
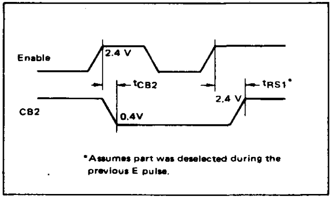


FIGURE 5 – PERIPHERAL DATA AND CB2 DELAY TIMES
(Write Mode; CRB-5 = CRB-3 = 1, CRB-4 = 0)



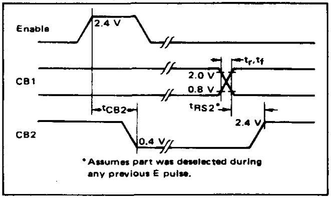
CB2 Note: CB2 goes low as a result of the positive transition of Enable.

FIGURE 6 – CB2 DELAY TIME
(Write Mode; CRB-5 = CRB-3 = 1, CRB-4 = 0)



*Assumes part was deselected during the previous E pulse.

FIGURE 7 – CB2 DELAY TIME
(Write Mode; CRB-5 = 1, CRB-3 = CRB-4 = 0)



*Assumes part was deselected during any previous E pulse.

FIGURE 8 – $\overline{\text{IRQ}}$ RELEASE TIME

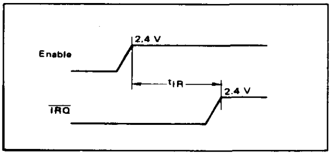
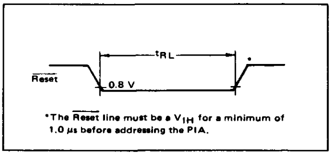


FIGURE 9 – $\overline{\text{RESET}}$ LOW TIME



*The $\overline{\text{Reset}}$ line must be a V_{IH} for a minimum of 1.0 μs before addressing the PIA.

FIGURE 10 – BUS READ TIMING CHARACTERISTICS
(Read Information from PIA)

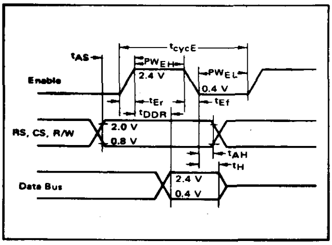


FIGURE 11 – BUS WRITE TIMING CHARACTERISTICS
(Write Information into PIA)

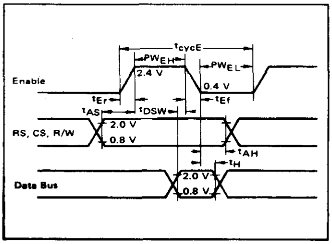
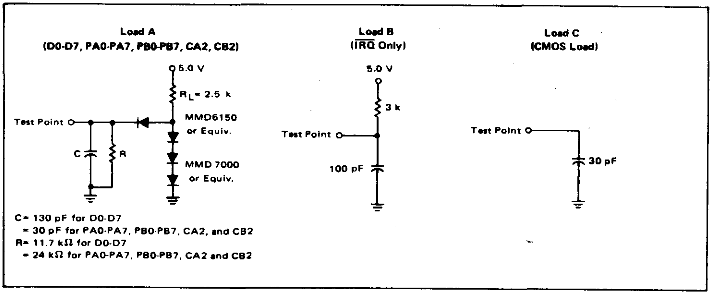


FIGURE 12 – BUS TIMING TEST LOADS



C = 130 pF for D0-D7
= 30 pF for PA0-PA7, PB0-PB7, CA2, and CB2
R = 11.7 kΩ for D0-D7
= 24 kΩ for PA0-PA7, PB0-PB7, CA2 and CB2

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	-0.3 to +7.0	Vdc
Input Voltage	V_{in}	-0.3 to +7.0	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature Range	T_{stg}	-55 to +150	°C
Thermal Resistance	θ_{JA}	82.5	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high-impedance circuit.

ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 V ±5%, V_{SS} = 0, T_A = 0 to 70°C unless otherwise noted.)

Characteristic	Symbol	Min	Typ	Max	Unit
Input High Voltage	V_{IH}	V_{SS} + 2.0	–	V_{CC}	Vdc
Input Low Voltage	V_{IL}	V_{SS} -0.3	–	V_{SS} + 0.8	Vdc
Input Leakage Current R/W,CS0,CS1,$\overline{CS2}$,Enable (V_{in} = 0 to 5.25 Vdc)	I_{in}	–	1.0	2.5	µAdc
Three-State (Off State) Input Current D0-D7 (V_{in} = 0.4 to 2.4 Vdc)	I_{TSI}	–	2.0	10	µAdc
Output High Voltage D0-D7 (I_{Load} = -205 µAdc, Enable Pulse Width <25 µs) (I_{Load} = -100 µAdc, Enable Pulse Width <25 µs) Tx Data, $\overline{RTS}$	V_{OH}	 V_{SS} + 2.4 V_{SS} + 2.4	 – –	 – –	Vdc
Output Low Voltage (I_{Load} = 1.6 mAdc, Enable Pulse Width <25 µs)	V_{OL}	–	–	V_{SS} + 0.4	Vdc
Output Leakage Current (Off State) $\overline{IRQ}$ (V_{OH} = 2.4 Vdc)	I_{LOH}	–	1.0	10	µAdc
Power Dissipation	P_D	–	300	525	mW
Input Capacitance (V_{in} = 0, T_A = 25°C, f = 1.0 MHz) D0-D7 E, Tx Clk, Rx Clk, R/W, RS, Rx Data, CS0, CS1, $\overline{CS2}$, $\overline{CTS}$, $\overline{DCD}$	C_{in}	 – –	 10 7.0	 12.5 7.5	pF
Output Capacitance $\overline{RTS}$, Tx Data (V_{in} = 0, T_A = 25°C, f = 1.0 MHz) $\overline{IRQ}$	C_{out}	– –	– –	10 5.0	pF
Minimum Clock Pulse Width, Low (Figure 1) ÷16, ÷64 Modes	PW_{CL}	600	–	–	ns
Minimum Clock Pulse Width, High (Figure 2) ÷16, ÷64 Modes	PW_{CH}	600	–	–	ns
Clock Frequency ÷1 Mode ÷16, ÷64 Modes	f_C	– –	– –	500 800	kHz
Clock-to-Data Delay for Transmitter (Figure 3)	t_{TDD}	–	–	1.0	µs
Receive Data Setup Time (Figure 4) ÷1 Mode	t_{RDSU}	500	–	–	ns
Receive Data Hold Time (Figure 5) ÷1 Mode	t_{RDH}	500	–	–	ns
$\overline{\text{Interrupt Request}}$ Release Time (Figure 6)	t_{IR}	–	–	1.2	µs
$\overline{\text{Request-to-Send}}$ Delay Time (Figure 6)	t_{RTS}	–	–	1.0	µs
Input Transition Times (Except Enable)	t_r,t_f	–	–	1.0*	µs

*1.0 µs or 10% of the pulse width, whichever is smaller.

BUS TIMING CHARACTERISTICS

READ (Figures 7 and 9)

Characteristic	Symbol	Min	Typ	Max	Unit
Enable Cycle Time	t_{cycE}	1.0	–	–	µs
Enable Pulse Width, High	PW_{EH}	0.45	–	25	µs
Enable Pulse Width, Low	PW_{EL}	0.43	–	–	µs
Setup Time, Address and R/W valid to Enable positive transition	t_{AS}	160	–	–	ns
Data Delay Time	t_{DDR}	–	–	320	ns
Data Hold Time	t_H	10	–	–	ns
Address Hold Time	t_{AH}	10	–	–	ns
Rise and Fall Time for Enable input	t_{Er}, t_{Ef}	–	–	25	ns

WRITE (Figure 8 and 9)

Characteristic	Symbol	Min	Typ	Max	Unit
Enable Cycle Time	t_{cycE}	1.0	–	–	µs
Enable Pulse Width, High	PW_{EH}	0.45	–	25	µs
Enable Pulse Width, Low	PW_{EL}	0.43	–	–	µs
Setup Time, Address and R/W valid to Enable positive transition	t_{AS}	160	–	–	ns
Data Setup Time	t_{DSW}	195	–	–	ns
Data Hold Time	t_H	10	–	–	ns
Address Hold Time	t_{AH}	10	–	–	ns
Rise and Fall Time for Enable input	t_{Er}, t_{Ef}	–	–	25	ns

FIGURE 1 – CLOCK PULSE WIDTH, LOW-STATE

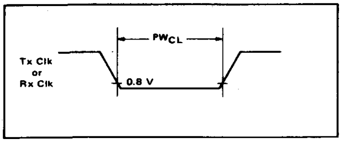


FIGURE 2 – CLOCK PULSE WIDTH, HIGH-STATE

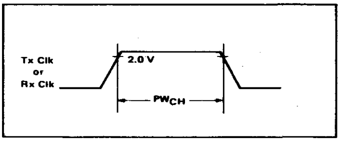


FIGURE 3 – TRANSMIT DATA OUTPUT DELAY

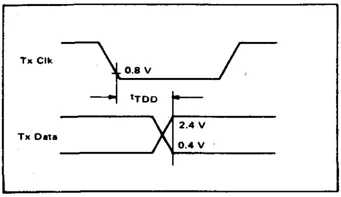


FIGURE 4 – RECEIVE DATA SETUP TIME
(÷1 Mode)

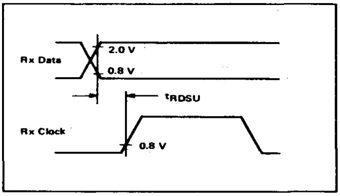


FIGURE 5 – RECEIVE DATA HOLD TIME
(÷1 Mode)

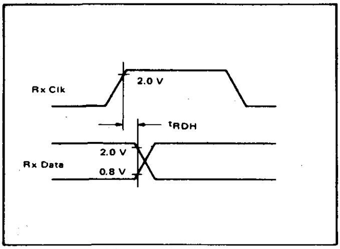


FIGURE 6 – $\overline{\text{REQUEST-TO-SEND}}$ DELAY AND
$\overline{\text{INTERRUPT-REQUEST}}$ RELEASE TIMES

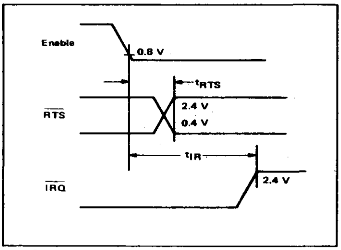


FIGURE 7 – BUS READ TIMING CHARACTERISTICS
(Read information from ACIA)

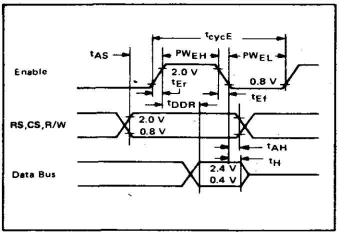


FIGURE 8 – BUS WRITE TIMING CHARACTERISTICS
(Write information into ACIA)

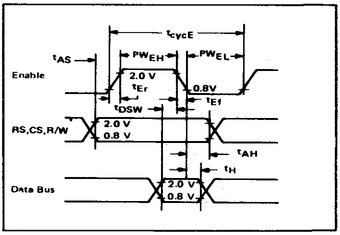
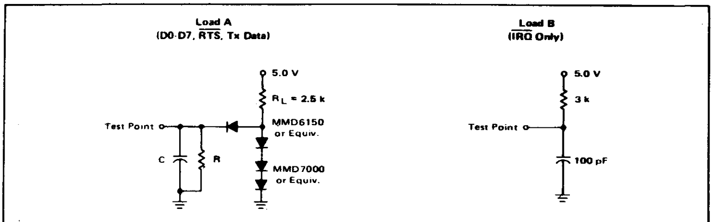


FIGURE 9 – BUS TIMING TEST LOADS



C = 130 pF for D0-D7
= 30 pF for $\overline{RTS}$ and Tx Data

R = 11.7 kΩ for D0-D7
= 24 kΩ for $\overline{RTS}$ and Tx Data

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	Vdc
Input Voltage	V_{in}	–0.3 to +7.0	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature Range	T_{stg}	–55 to +150	°C
Thermal Resistance	θ_{JA}	70	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, is is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high-impedance circuit.

ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 V ±5%, V_{SS} = 0, T_A = 0 to 70°C unless otherwise noted.)

Characteristic	Symbol	Min	Typ	Max	Unit
Input High Voltage	V_{IH}	V_{SS} + 2.0	–	–	Vdc
Input Low Voltage	V_{IL}	–	–	V_{SS} + 0.8	Vdc
Input Leakage Current (V_{in} = 0 to 5.25 Vdc) Tx Clk, Rx Clk, Rx Data, Enable, $\overline{\text{Reset}}$, RS, R/W, $\overline{\text{CS}}$, $\overline{\text{DCD}}$, $\overline{\text{CTS}}$	I_{in}	–	1.0	2.5	µAdc
Three-State (Off State) Input Current (V_{in} = 0.4 to 2.4 Vdc, V_{CC} = 5.25 Vdc) D0–D7	I_{TSI}	–	2.0	10	µAdc
Output High Voltage	V_{OH}				Vdc
(I_{Load} = –205 µAdc, Enable Pulse Width <25 µs) D0–D7		V_{SS} + 2.4	–	–	
(I_{Load} = –100 µAdc, Enable Pulse Width <25 µs) Tx Data, $\overline{\text{DTR}}$, TUF		V_{SS} + 2.4	–	–	
Output Low Voltage (I_{Load} = 1.6 mAdc, Enable Pulse Width <25 µs)	V_{OL}	–	–	V_{SS} + 0.4	Vdc
Output Leakage Current (Off State) (V_{OH} = 2.4 Vdc) $\overline{\text{IRQ}}$	I_{LOH}	–	1.0	10	µAdc
Power Dissipation	P_D	–	300	525	mW
Input Capacitance (V_{in} = 0, T_A = 25°C, f = 1.0 MHz)	C_{in}				pF
D0–D7		–	–	12.5	
All Other Inputs		–	–	7.5	
Output Capacitance (V_{in} = 0, T_A = 25°C, f = 1.0 MHz) Tx Data, SM/$\overline{\text{DTR}}$, TUF	C_{out}	–	–	10	pF
$\overline{\text{IRQ}}$				5.0	
Minimum Clock Pulse Width, Low (Figure 1)	PW_{CL}	700	–	–	ns
Minimum Clock Pulse Width, High (Figure 2)	PW_{CH}	700	–	–	ns
Clock Frequency	f_C	–	–	600	kHz
Receive Data Setup Time (Figure 3, 7)	t_{RDSU}	350	–	–	ns
Receive Data Hold Time (Figure 3)	t_{RDH}	350	–		ns
Sync Match Delay Time (Figure 3)	t_{SM}	–	–	1.0	µs
Clock-to-Data Delay for Transmitter (Figure 4)	t_{TDD}	–	–	1.0	µs
Transmitter Underflow (Figure 4,6)	t_{TUF}	–	–	1.0	µs
$\overline{\text{DTR}}$ Delay Time (Figure 5)	t_{DTR}	–	–	1.0	µs
Interrupt Request Release Time (Figure 5)	t_{IR}	–	–	1.2	µs
$\overline{\text{Reset}}$ Minimum Pulse Width	t_{Res}	1.0	–	–	µs
$\overline{\text{CTS}}$ Setup Time (Figure 6)	t_{CTS}	–	–	200	ns
$\overline{\text{DCD}}$ Setup Time (Figure 7)	t_{DCD}	–	–	500	ns
Input Rise and Fall Times (except Enable) (0.8 V to 2.0 V)	t_r, t_f	–	–	1.0*	µs

*1.0 µs or 10% of the pulse width, whichever is smaller.

FIGURE 1 – CLOCK PULSE WIDTH, LOW-STATE

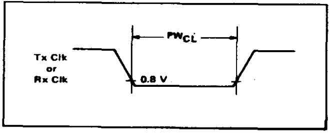
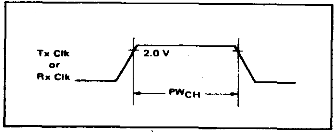


FIGURE 2 – CLOCK PULSE WIDTH, HIGH-STATE



BUS TIMING CHARACTERISTICS

READ (Figures 8 and 10)

Characteristic	Symbol	Min	Typ	Max	Unit
Enable Cycle Time	t_{cycE}	1.0	–	–	µs
Enable Pulse Width, High	PW_{EH}	0.45	–	25	µs
Enable Pulse Width, Low	PW_{EL}	0.43	–	–	µs
Setup Time, Address and R/W valid to Enable positive transition	t_{AS}	160	–	–	ns
Data Delay Time	t_{DDR}	–	–	320	ns
Data Hold Time	t_H	10	–	–	ns
Address Hold Time	t_{AH}	10	–	–	ns
Rise and Fall Time for Enable input	t_{Er}, t_{Ef}	–	–	25	ns

WRITE (Figures 9 and 10)

Characteristic	Symbol	Min	Typ	Max	Unit
Enable Cycle Time	t_{cycE}	1.0	–	–	µs
Enable Pulse Width, High	PW_{EH}	0.45	–	25	µs
Enable Pulse Width, Low	PW_{EL}	0.43	–	–	µs
Setup Time, Address and R/W valid to Enable positive transition	t_{AS}	160	–	–	ns
Data Setup Time	t_{DSW}	195	–	–	ns
Data Hold Time	t_H	10	–	–	ns
Address Hold Time	t_{AH}	10	–	–	ns
Rise and Fall Time for Enable input	t_{Er}, t_{Ef}	–	–	25	ns

FIGURE 3 – RECEIVE DATA SETUP AND HOLD TIMES AND SYNC MATCH DELAY TIME

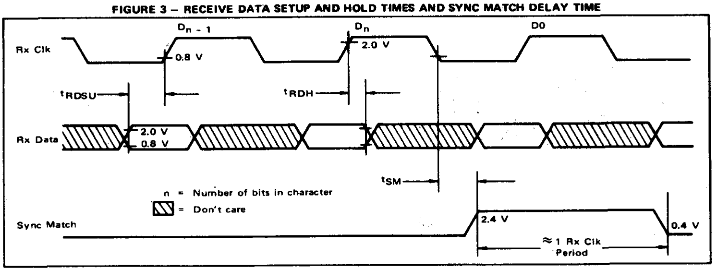


FIGURE 4 – TRANSMIT DATA OUTPUT DELAY AND TRANSMITTER UNDERFLOW DELAY TIME

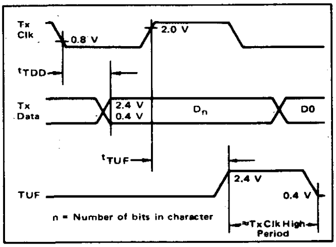


FIGURE 5 – DATA TERMINAL READY AND INTERRUPT REQUEST RELEASE TIMES

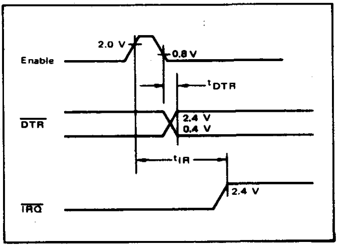


FIGURE 6 – $\overline{\text{CLEAR-TO-SEND}}$ SETUP TIME

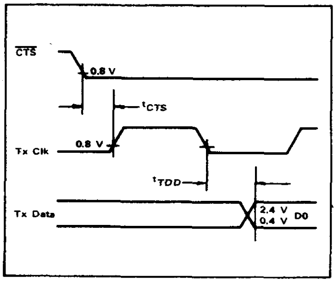


FIGURE 7 – $\overline{\text{DATA CARRIER DETECT}}$ SETUP TIME

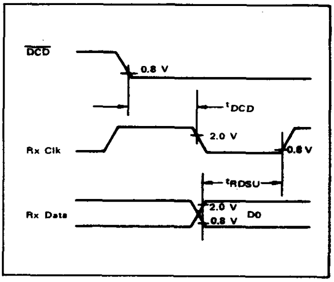


FIGURE 8 – BUS READ TIMING CHARACTERISTICS
(Read information from SSDA)

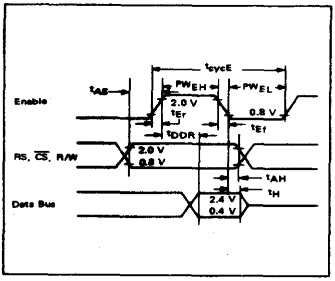


FIGURE 9 – BUS WRITE TIMING CHARACTERISTICS
(Write information into SSDA)

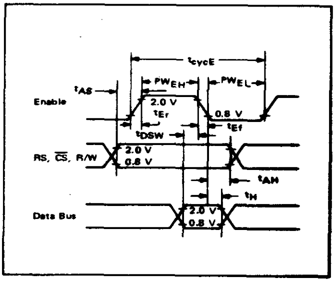
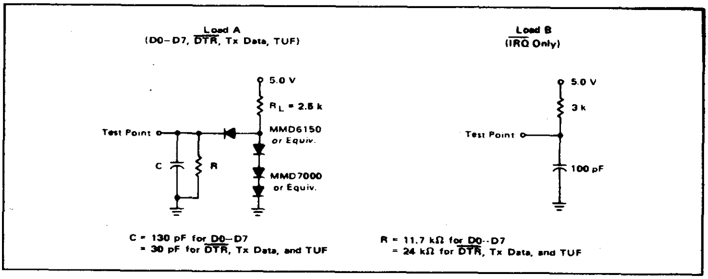


FIGURE 10 – BUS TIMING TEST LOADS



C = 130 pF for D0–D7
= 30 pF for $\overline{\text{DTR}}$, Tx Data, and TUF

R = 11.7 kΩ for D0–D7
= 24 kΩ for $\overline{\text{DTR}}$, Tx Data, and TUF

Chapter 9
THE MOS TECHNOLOGY MCS6500

In many ways the MCS6500 microcomputer systems can be compared to the Zilog Z80, which we described in Chapter 7. Just as the Z80 is an enhancement of the 8080A, which is described in Chapter 4, so MOS Technology's products are enhancements of the MC6800, which we described in Chapter 8.

But there are some interesting conceptual differences between the way MOS Technology went about enhancing the MC6800, as against the product enhancement philosophy adopted by Zilog.

The Z80 is indeed an enhancement of the 8080A, but only to the extent that the 8080A instruction set is a subset of the Z80 instruction set; there are architectural similarities between the Z80 and the 8080A, but System Bus philosophies are markedly different. It would be hard to look upon the Z80 as simply another member of the 8080A family of microcomputer devices.

The MCS6500 product line, by way of contrast, can be looked upon as a CPU whose philosophical concepts agree closely with the MC6800 product line — without being in any way compatible, either in terms of instruction set or System Bus philosophy. While on the surface it may appear as though MCS6500 CPUs represent some form of an MC6800 superset, this is not the case. System Busses are sufficiently different that you could not consider replacing an MC6800 CPU with an MCS6500 equivalent, leaving other logic unaltered. Instruction sets are similar, but deceptively so. In reality, the instruction sets are sufficiently different that converting an MC6800 source program to its MCS6500 equivalent is no simple task. It would be completely impossible to take an MC6800 program ROM and use it to drive an MCS6500 CPU. Recall that you can take an 8080A program ROM and use it to drive a Z80 CPU.

Since this chapter is devoted to the MOS Technology product line, let us begin by summarizing the components of this product line, and the principal CPU enhancements that have been made.

The MOS Technology devices described in this chapter consist of nine CPUs, plus two support circuits. A third support circuit is described in Chapter 8.

The nine CPUs share the same instruction set and addressing modes, but have minor differences in packaging and system interface. Table 9-1 summarizes the nine CPUs.

The two support circuits which are described in this chapter are the MCS6522 Peripheral Interface Adapter, and the MCS6530 combination logic device. The MCS6520 PIA is identical to the MC6852 PIA; for a description of this device see Chapter 8.

In order to enhance the MC6800 CPU, MOS Technology made a number of useful, yet obvious instruction set changes; they also made a number of subjective architectural changes which might have significant impact in particular applications, but in general, result in products that adhere quite closely to MC6800 philosophy.

The most important enhancement that MOS Technology made is to develop a whole family of CPU devices.

The second most important feature of the MCS6500 line of CPU devices is the fact that the MCS650X series CPUs contain on-chip clock logic; therefore, when using these CPUs, you do not need an MC6870 series clock device. However, you will need an external crystal oscillator or RC network — which is typical of any microprocessor with on-chip clock logic.

Another important feature of all MCS6500 series CPUs is that you cannot float the Address and Data Busses separately during Φ1 high and Φ1 low clock pulses and there is no HALT condition. Also, you cannot stretch clock pulses. Slow memories are accommodated in the more traditional manner, by allowing you to insert extra machine cycles, equivalent to 8080A Wait states.

If you are making extensive use of clock stretching, or DMA data transfers during Halt states in an MC6800 microcomputer system, switching to an MCS6500 CPU will require considerable system redesign.

In order to perform Direct Memory Access or dynamic memory refresh operations using an MCS6500 CPU, you must again "steal" machine cycles by inserting Wait states, as you would for slow memories.

MOS Technology, the principal manufacturer of the MCS6500 product line, is located at:

MOS TECHNOLOGY INC
950 Rittenhouse Road
Norristown, PA 19401

The second source is:

SYNERTEK INC
1901 Old Middlefield Way
Mountain View, CA 94043

The MCS6500 devices use a single +5V power supply. Using a 1 microsecond clock, instruction execution times range from 2 to 12 microseconds.

All MCS6500 devices have TTL compatible signals.

N-channel, silicon gate, depletion load MOS technology is used for MCS6500 devices.

THE MCS6500 SERIES CPUS

Functions implemented on any of the MCS6500 CPUs are illustrated in Figure 9-1. As this figure would imply, capabilities offered by the various MCS6500 CPUs differ in scope rather than function.

Table 9-1. A Comparison Of MCS6500 Series And The MC6800 CPU Devices

CPU	CPU PINS AND SIGNALS													PINS	COMMENTS
	ADDRESS BUS	DATA BUS	$\Phi 0$	$\Phi 1$	$\Phi 2$	RDY	$\overline{IRQ}$	$\overline{NMI}$	SYNC	$\overline{RES}/\overline{RESET}$*	SO	$R/\overline{W}$	DBE		
6502	A0-A15	D0-D7	I	O	O	I	I	I	O	I	I	O		40	This is the on-chip-clock version of the 6512.
6503	A0-A11	D0-D7	I		O		I	I		I		O		28	This is the on-chip-clock version of the 6513.
6504	A0-A12	D0-D7	I		O		I			I		O		28	This is the on-chip-clock version of the 6514.
6505	A0-A11	D0-D7	I		O	I	I			I		O		28	This is the on-chip-clock version of the 6515.
6506	A0-A11	D0-D7	I	O	O		I			I		O		28	On-chip-clock version, 4K memory, $\overline{IRQ}$, $\Phi 1$ (out) and $\Phi 2$ (out).
6512	A0-A15	D0-D7		I	I/O	I	I	I	O	I	I	O	I	40	This CPU is most like the MC6800. The HALT, VMA, TSC and BA signals are not present. SYNC, SO, $\Phi 2$ (out) and RDY are added.
6513	A0-A11	D0-D7		I	I		I	I		I		O		28	4K memory with $\overline{IRQ}$ and $\overline{NMI}$.
6514	A0-A11	D0-D7		I	I		I			I		O		28	8K memory with $\overline{IRQ}$.
6515	A0-A11	D0-D7		I	I	I	I			I		O		28	4K memory with $\overline{IRQ}$ and RDY.
MC6800	A0-A15	D0-D7		I	I		I	I		I		O	I	40	The MC6800 TSC, VMA, BA and HALT signals are not implemented on any MCS6500 CPU.

*The second name is the name used by MC6800 literature for the same signal.

Within CPU PINS AND SIGNALS columns, I identifies an input signal present, O identifies an output signal present, I/O identifies a signal that appears twice, at two pins, one as an input, the other as an output.

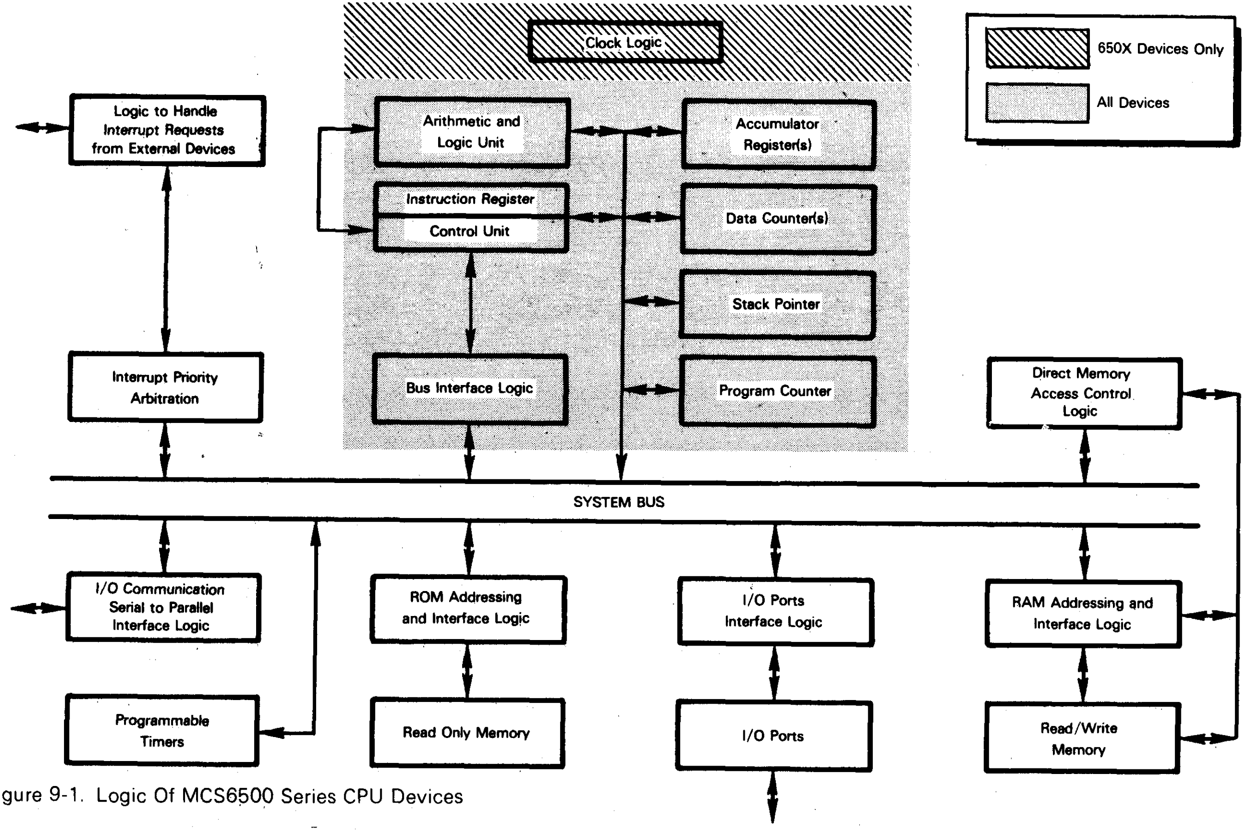
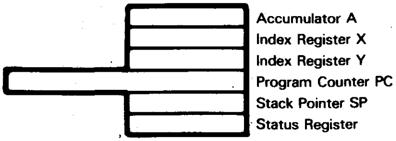

Clock Logic
650X Devices Only
All Devices
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 9-1. Logic Of MCS6500 Series CPU Devices

MCS6500 SERIES CPU PROGRAMMABLE REGISTERS

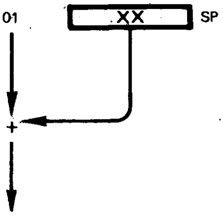
The MCS6500 series CPUs all have the same programmable registers; they may be illustrated as follows:



The MC6800 has two Accumulators; the MCS6500 has just one.

The Index register represents a significant departure from the MC6800. The MCS6500 breaks one 16-bit Index register into two 8-bit Index registers.

The MCS6500 Stack Pointer also represents a significant departure from MC6800 architecture. The MC6800 Stack Pointer is 16 bits wide, which means that the Stack may be located anywhere in memory, and may be of any length. **The MCS6500 Stack Pointer is 8 bits wide, which means that maximum Stack length is 256 bytes.** The CPU always appends 01_{16} as the high order byte of any Stack address, which means that memory locations 0100_{16} through $01FF_{16}$ are permanently assigned to the Stack:



01XX is the Stack address

There is nothing very significant about the shorter MCS6500 Stack Pointer if you are using this CPU as a stand-alone product. A 256-byte Stack is usually sufficient for any typical microcomputer application; and its location in early memory simply means that low memory addresses must be implemented as read/write memory. **If you are transferring from an MC6800 to an MCS6500, however, there are two very important consequences of the shorter MCS6500 Stack Pointer.**

The first and most important consequence is that you are unlikely to be so lucky as to have implemented the MC6800 Stack within the address space that the MCS6500 requires. Therefore, you will have to reassemble MC6800 programs, repartitioning memory in order to run the same programs in an MCS6500 microcomputer system.

A less obvious consequence of a shorter MCS6500 Stack Pointer is the fact that many MC6800 programs use the Stack Pointer as an alternate Index register. If you have used the Stack Pointer in this way when writing programs for an MC6800 microcomputer system, the program conversion, when going to an MCS6500 system, could be significant.

The MCS6500 Program Counter is a typical program counter, identical to the MC6800 implementation.

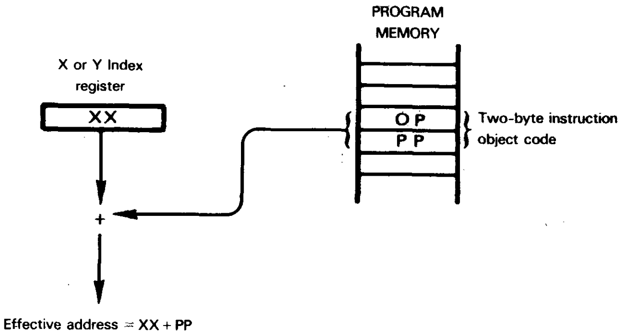
MCS6500 MEMORY ADDRESSING MODES

MCS6500 memory reference instructions use direct addressing, indexed addressing, and indirect addressing. The MC6800 has no indirect addressing and different indexed addressing.

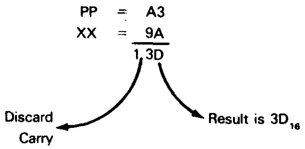
The MC6800 and MCS6500 have identical direct addressing. Three-byte instructions use the second and third bytes of the object code to provide a direct, 16-bit address; therefore, 65,536 bytes of memory can be addressed directly. The commonly used memory reference instructions also have a two-byte object code variation, where the second byte directly addresses one of the first 256 bytes of memory.

MCS6500 direct indexed addressing differs markedly from MC6800 indexed addressing.

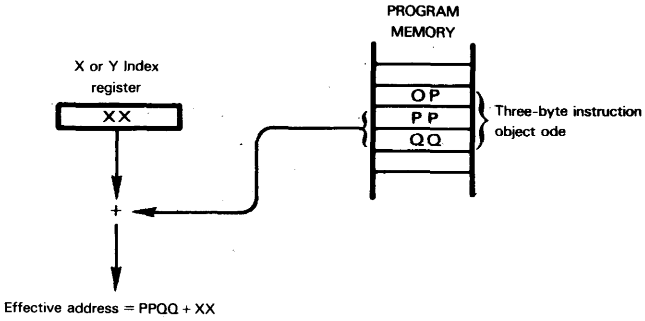
The MCS6500 offers base page, indexed addressing. In this case, the instruction has two bytes of object code. The contents of either the X or Y Index registers are added to the second object code byte in order to compute a memory address. This may be illustrated as follows:



Base page, indexed addressing, as illustrated above, is wraparound — which means that there is no carry. If the sum of the Index register and second object code byte contents is more than FF_{16}, the carry bit will be discarded. This may be illustrated as follows:



Absolute indexed addressing is also provided. In this case, the contents of either the X or the Y Index register are added to a 16-bit direct address provided by the second and third bytes of an instruction's object code. This may be illustrated as follows:

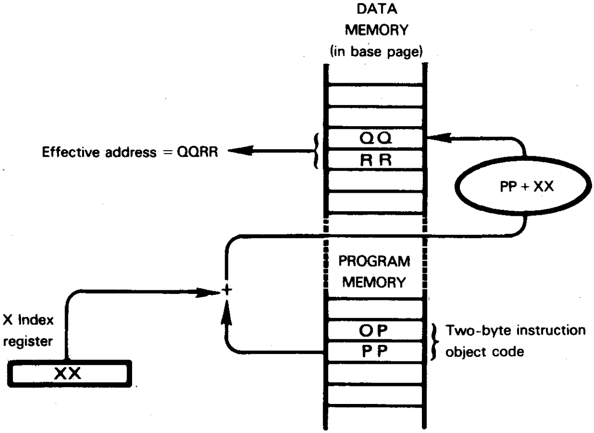


Indirect addressing represents a feature of the MCS6500 which the MC6800 does not have. Instructions that use simple indirect addressing have three bytes of object code. The second and third object code bytes provide a 16-bit address; therefore, the indirect address can be located anywhere in memory. This is straightforward indirect addressing, as described in Volume I, Chapter 6.

MCS6500 indirect, indexed addressing comes in two forms: there is pre-indexed indirect addressing and there is post-indexed indirect addressing.

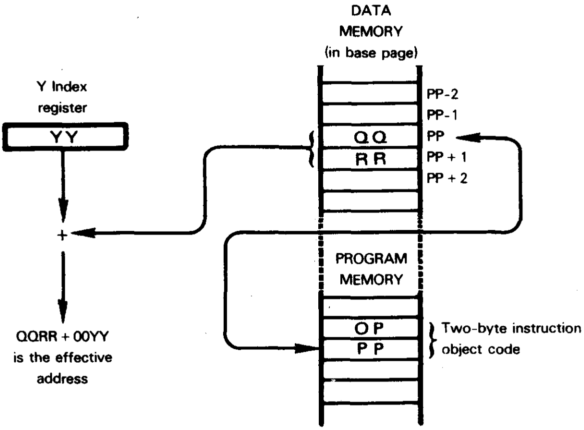
In each case the instruction object code is two bytes long and the second object code byte provides an 8-bit address.

Instructions with pre-indexed indirect addressing add the contents of the X Index register and the second object code byte to access a memory location in the first 256 bytes of memory, where the indirect address will be found:



When using pre-indexed indirect addressing, once again wraparound addition is used, which means that when the X Index register contents are added to the second object code byte, any carry will be discarded. Note that only the X Index register can be used with pre-indexed indirect addressing.

The Y Index register is used for post-indexed indirect addressing; now the second object code byte identifies a location in the first 256 bytes of memory where an indirect address will be found. The contents of the Y Index register are added to this indirect address. This may be illustrated as follows:



Note that only the Y Index register can be used with post-indexed indirect addressing.

MCS6500 Branch and Branch-on-Condition instructions use program relative, direct addressing as described for the MC6800. These instructions have two bytes of object code. The second object code byte is treated as an 8-bit, signed binary number, which is added to the Program Counter after the Program Counter contents have been incremented to address the next sequential instruction. This allows displacements in the range +129 through -126 bytes.

The MCS6500 literature uses the term implied addressing, as Motorola's MC6800 literature does, to describe instructions that identify one of the programmable registers. **The MCS6500 does not have implied addressing as the term is used in this book.**

MCS6500 STATUS FLAGS

The MCS6500 has a Status register which maintains six status flags and a master interrupt control bit. These are the six status flags:

Carry (C)
Zero (Z)
Overflow (O)
Sign (S)
Decimal Mode (D)
Break (B)

Statuses are assigned bit positions within the Status register as follows:

7	6	5	4	3	2	1	0	← Bit No.
S	O		B	D	I	Z	C	← Status Register MCS6500

7	6	5	4	3	2	1	0	← Bit No.
		AC	I	S	Z	O	C	← Status Register MC6800

In the illustration above, MCS6500 statuses and status bit assignments that differ from MC6800 equivalents have been shaded.

The Carry, Zero and Sign statuses are absolutely standard, and are identical to the MC6800.

Carry represents any carry out of bit 7 during arithmetic or logical operations.

Zero is set to 1 when any arithmetic or logical operation results in a 0 value. Zero is set to 0 otherwise.

The Sign status will acquire the value of the high order (Sign) bit of any arithmetic operation result. Thus, a Sign status value of 1 identifies a negative result and a Sign status of 0 identifies a positive result. The Sign status will be set or reset on the assumption that you are using signed binary arithmetic. If you are not using signed binary arithmetic, you can ignore the Sign status, or you can use it to identify the value of the high order result bit.

The Decimal Mode and Break statuses have no MC6800 equivalent.

The Decimal Mode status, when set, causes the Add-with-Carry and Subtract-with-Carry instructions to perform BCD operations. Thus, when the Decimal Mode status is set and an Add-with-Carry or Subtract-with-Carry instruction is executed, CPU logic assumes that both source 8-bit values are valid BCD numbers — and a result generated will also be a valid BCD number. Because MCS6500 CPUs perform decimal addition and subtraction, there is no need for an Intermediate Carry status. This status is used for decimal adjust operations only, as described in Volume I.

The Break status pertains to software interrupts. MCS6500 supports software interrupts, just as the MC6800 does. When a software interrupt is executed, however, MCS6500 CPU logic will set the Break status flag.

I is a standard master interrupt enable/disable flag. When I equals 1, interrupts are disabled; when I equals 0, interrupts are enabled.

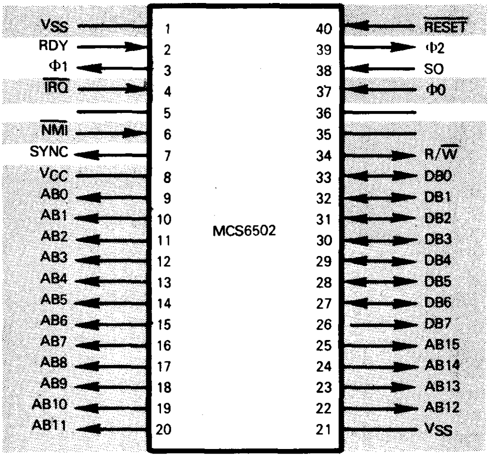
The Overflow status is a typical overflow, except that it can also be used as a control input. Recall that an Overflow status represents a carry when performing signed binary arithmetic. The Overflow status has been discussed in detail in Volume I; it equals the exclusive-OR of carries out of bits 6 and 7 when performing arithmetic operations. Some MCS6500 CPUs allow external logic to set or reset the Overflow status, in which case it can be used subsequently as a general logic indicator; you must be very careful when using the Overflow status in this way, since the same status flag will be modified by arithmetic instructions. It is up to you, as a programmer, to make sure that an instruction which modifies the Overflow status is not executed in between the time external logic sets or resets this status, and subsequent program logic tests it.

MCS6500 CPU PINS AND SIGNALS

Figures 9-2 through 9-10 illustrate pins and signals for the nine CPUs of the MCS6500 family. Shaded pins in Figures 9-2 and 9-7 identify signals which are identical to the MC6800, both in pin location and signal type. Most of the 28-pin MCS6500 series CPUs have signals which are identical to those of the MC6800; however, between a 40-pin DIP and a 28-pin DIP, it is impossible to talk about pin compatibility.

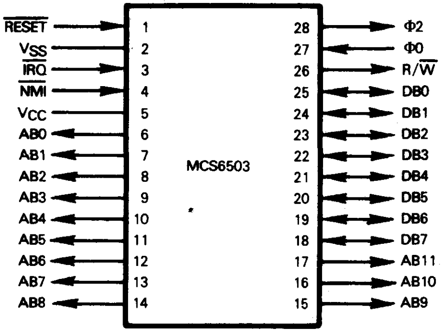
MCS6500 signals may be divided between those that have MC6800 equivalents and those that do not. We are going to describe all of the MCS6500 series signals, as a group. **In order to determine which signals are available on the different MCS6500 CPUs, see Table 9-1.**

Let us begin with the signals which are direct reproductions of MC6800 signals.



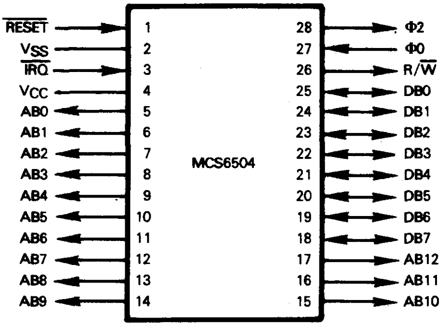
Pin Name	Description	Type
$R/\overline{W}$	Read/Write control	Output
$\overline{\text{IRQ}}$	Interrupt request	Input
$\overline{\text{NMI}}$	Non-maskable interrupt	Input
$\overline{\text{RESET}}$	Reset	Input
Φ0	CPU clock	Input
Φ1, Φ2	System clocks	Output
DB0 - DB7	Data Bus	Tristate, bidirectional
AB0 - AB15	Address Bus	Output
RDY	Single cycle control	Input
SO	Set Overflow flag	Input
SYNC	Identify op code fetch cycle	Output
VCC, VSS	Power and Ground	

Figure 9-2. MCS6502 Signals And Pin Assignments



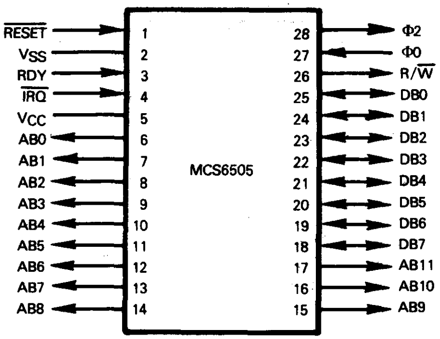
Pin Name	Description	Type
$R/\overline{W}$	Read/Write control	Output
$\overline{IRQ}$	Interrupt request	Input
$\overline{NMI}$	Non-maskable interrupt	Input
$\overline{RESET}$	Reset	Input
Φ0	CPU clock	Input
Φ2	System clock	Output
DB0 - DB7	Data Bus	Tristate, bidirectional
AB0 - AB11	Address Bus	Output
V_{CC}, V_{SS}	Power and Ground	

Figure 9-3. MCS6503 Signals And Pin Assignments



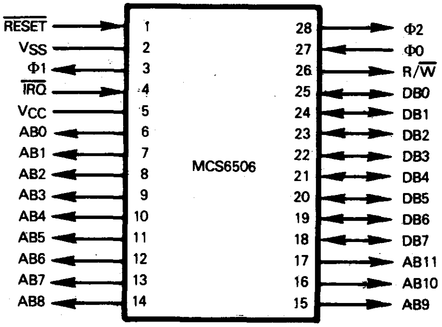
Pin Name	Description	Type
$R/\overline{W}$	Read/Write control	Output
$\overline{IRQ}$	Interrupt request	Input
$\overline{RESET}$	Reset	Input
Φ0	CPU clock	Input
Φ2	System clock	Output
DB0 - DB7	Data Bus	Tristate, bidirectional
AB0 - AB12	Address Bus	Output
V_{CC}, V_{SS}	Power and Ground	

Figure 9-4. MCS6504 Signals And Pin Assignments



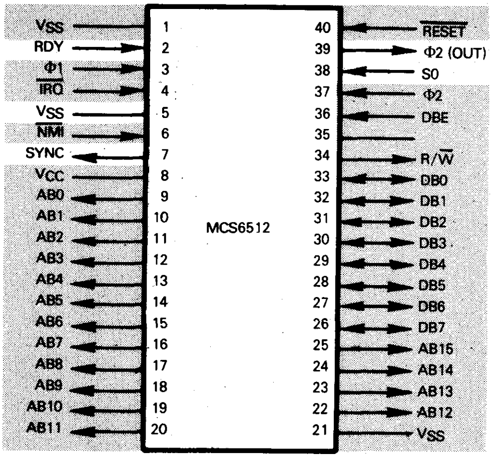
Pin Name	Description	Type
$R/\overline{W}$	Read/Write control	Output
$\overline{IRQ}$	Interrupt request	Input
$\overline{RESET}$	Reset	Input
Φ0	CPU clock	Input
Φ2	System clock	Output
DB0 - DB7	Data Bus	Tristate, bidirectional
AB0 - AB11	Address Bus	Output
RDY	Single cycle control	Input
V_{CC}, V_{SS}	Power and Ground	

Figure 9-5. MCS6505 Signals And Pin Assignments



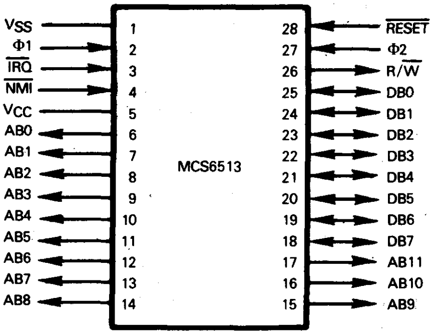
Pin Name	Description	Type
$R/\overline{W}$	Read/Write control	Output
$\overline{IRQ}$	Interrupt request	Input
$\overline{RESET}$	Reset	Input
Φ0	CPU clock	Input
Φ1, Φ2	System clocks	Output
DB0 - DB7	Data Bus	Tristate, bidirectional
AB0 - AB11	Address Bus	Output
V_{CC}, V_{SS}	Power and Ground	

Figure 9-6. MCS6506 Signals And Pin Assignments



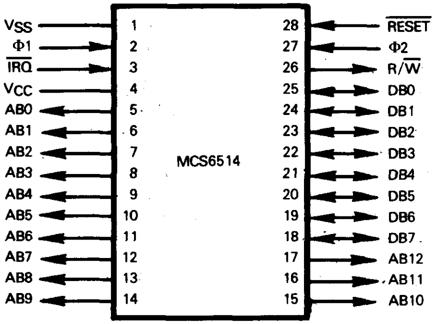
Pin Name	Description	Type
DBE	Data Bus Enable	Input
R/$\overline{\text{W}}$	Read/Write control	Output
$\overline{\text{IRQ}}$	Interrupt request	Input
$\overline{\text{NMI}}$	Non-maskable interrupt	Input
$\overline{\text{RESET}}$	Reset	Input
Φ1, Φ2	CPU clocks	Input
Φ2 (OUT)	System clock	Output
DB0 - DB7	Data Bus	Tristate, bidirectional
AB0 - AB15	Address Bus	Output
RDY	Single cycle control	Input
SO	Set Overflow flag	Input
SYNC	Identify op code fetch cycle	Output
V_{CC}, V_{SS}	Power and Ground .	

Figure 9-7. MCS6512 Signals And Pin Assignments



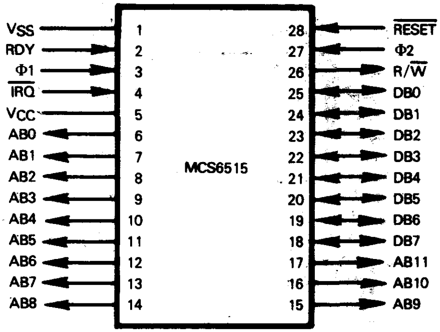
Pin Name	Description	Type
R/$\overline{W}$	Read/Write control	Output
$\overline{IRQ}$	Interrupt request	Input
$\overline{NMI}$	Non-maskable interrupt	Input
$\overline{RESET}$	Reset	Input
Φ1, Φ2	CPU clocks	Input
DB0 - DB7	Data Bus	Tristate, bidirectional
AB0 - AB11	Address Bus	Output
VCC, VSS	Power and Ground	

Figure 9-8. MCS6513 Signals And Pin Assignments



Pin Name	Description	Type
R/$\overline{W}$	Read/Write control	Output
$\overline{IRQ}$	Interrupt request	Input
$\overline{RESET}$	Reset	Input
Φ1, Φ2	CPU clocks	Input
DB0 - DB7	Data Bus	Tristate, bidirectional
AB0 - AB12	Address Bus	Output
VCC, VSS	Power and Ground	

Figure 9-9. MCS6514 Signals And Pin Assignments



Pin Name	Description	Type
$R/\overline{W}$	Read/Write control	Output
$\overline{IRQ}$	Interrupt request	Input
$\overline{RESET}$	Reset	Input
Φ1, Φ2	CPU clocks	Input
DB0 - DB7	Data Bus	Tristate, bidirectional
AB0 - AB11	Address Bus	Output
RDY	Single cycle control	Input
V_{CC}, V_{SS}	Power and Ground	

Figure 9-10: MCS6515 Signals And Pin Assignments

DATA BUS ENABLE (DBE). Only the MCS6512 CPU supports this signal. This signal is input low in order to float the Data Bus. DBE is frequently tied to the Φ2 clock input, in which case Φ2 and DBE are identical signals.

READ/WRITE ($R/\overline{W}$). When high, this signal indicates that the CPU wishes to read data off the Data Bus; when low, this signal indicates that the CPU is outputting data on the Data Bus. The normal standby state for this signal is "read" (high).

INTERRUPT REQUEST ($\overline{IRQ}$). This signal is used by external logic to request an interrupt. If interrupts have been enabled, then the CPU will acknowledge an interrupt at the end of the currently executing instruction. There is a small difference between MCS6500 and MC6800 interrupt acknowledge logic. The MC6800 cannot acknowledge an interrupt while it is in the Halt state. The MCS6500 has no Halt state, therefore this situation cannot arise.

NONMASKABLE INTERRUPT ($\overline{NMI}$). This signal differs from $\overline{IRQ}$ in that it cannot be inhibited. Typically this input is used for catastrophic interrupts such as power failure.

$\overline{RESET}$ This is a typical RESET signal. Reset logic within an MCS6500 microcomputer system is identical to Reset logic within an MCS6800 microcomputer system.

Next consider MC6800 signals which are the same on some MCS6500 CPUs, but not on others.

The clock signals Φ1 and Φ2 are identical to MC6800 clock signals for the MCS651X series CPUs. These CPUs require external clock signals whose waveforms are identical to the MC6800. **The MCS650X series CPUs have clock logic on the CPU chip; these CPUs output Φ2;** the MCS6502 and the MCS6506 output Φ1 as well.

The Data Bus of the MCS6500 series CPUs is identical to that of the MC6800. The Data Bus is a tristate, 8-bit bidirectional bus via which data is transferred between memory and all MCS6500 microcomputer system devices. **However, only the**

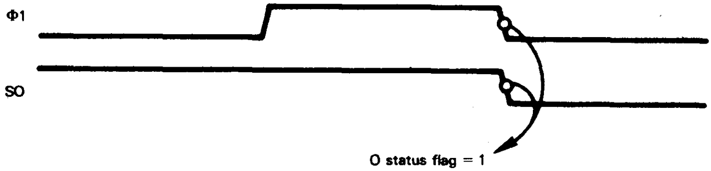
MCS6512 has a DBE input for external control of the bus. On MCS6500 CPUs other than the MCS6512, an internal Data Bus Enable is connected to Φ2; in these devices the Data Bus is always floated during the first part of a machine cycle.

We will now look at the CPU signals which are unique to the MCS6500 microcomputer system.

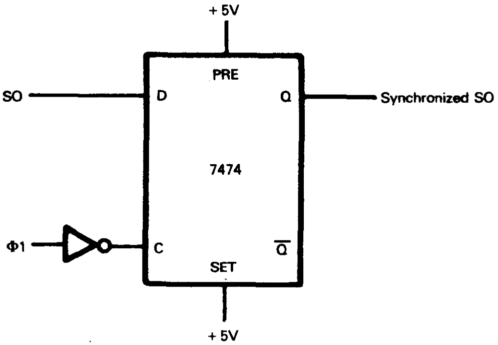
The Address Bus in MCS6500 microcomputer systems is not a tristate bus and cannot be floated. Also, the 28-pin MCS6500 series CPUs have either 12 or 13 Address Bus lines, allowing a total memory space of either 4K or 8K bytes. The Address Bus is used in the normal way by the CPU to output memory addresses.

READY (RDY) is an input control signal which, in MCS6500 microcomputer systems, performs the task of MC6800 TSC DBE and $\overline{\text{HALT}}$ signals. The RDY input causes the equivalent of a Wait machine cycle to be inserted within the normal machine cycle sequence. In order to generate a Wait machine cycle, RDY must make a high-to-low transition during a Φ1 high clock pulse in any machine cycle other than a write. We will illustrate the use of the RDY signal, and discuss a number of its non-obvious ramifications, following this summary description of MCS6500 signals.

The Set Overflow flag (SO) signal can be used to set to 1 the Overflow bit of the Status register. The SO input must make a high-to-low transition on the trailing edge of the Φ1 pulse in order for the Overflow bit of the Status register to be set to 1. This may be illustrated as follows:



You cannot use the SO input signal in order to reset the Overflow bit of the Status register to 0. Note that external logic must use the Φ1 clock signal in order to synchronize the SO high-to-low transition. A simple 7474 flip-flop can be used for this purpose:

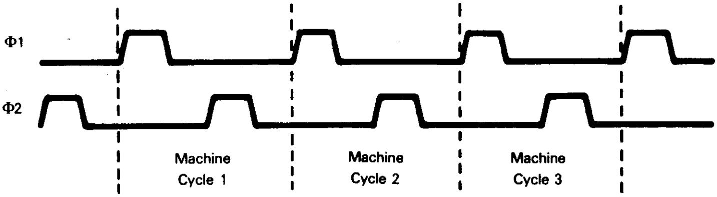


The SYNC signal is used to identify instruction fetch machine cycles. There are a number of important uses for this signal which we will discuss along with general instruction timing.

MCS6500 TIMING AND INSTRUCTION EXECUTION

MCS6500 CPUs execute instructions using exactly the same clock signals, machine cycles and machine cycle types as described for the MC6800 in Chapter 8.

Recall that the two clock signals, Φ1 and Φ2, define machine cycles as follows:

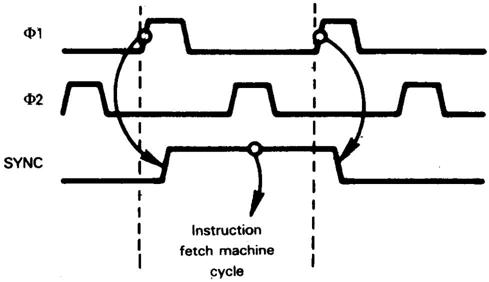


So far as external logic is concerned, there are only three types of machine cycles which can occur during an instruction's execution:

1) **A read operation** during which a byte of data must be input to the CPU.
2) **A write operation** during which a byte of data is output by the CPU.
3) **An internal operation** during which no activity occurs on the System Bus.

As was the case with the MC6800, all MCS6500 instructions have timing which is a simple concatenation of the three basic machine cycle types. See Figures 8-3 and 8-4 and the accompanying text in Chapter 8 for a description of these three basic machine cycles.

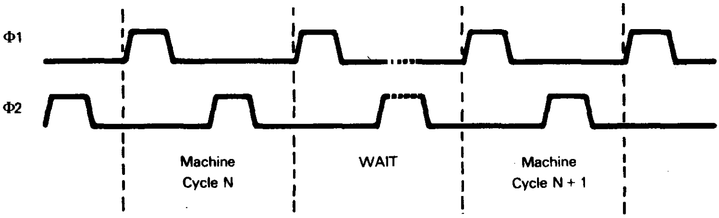
Instruction execution differences between the MC6800 and MCS6500 arise only when we depart from simple instruction execution logic. The MCS6500 SYNC signal is also a difference to be noted; the SYNC signal identifies MCS6500 machine cycles during which any instruction object code is being fetched. SYNC timing may be illustrated as follows:



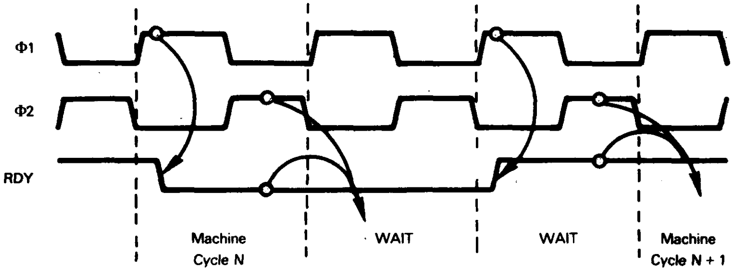
MCS6500 CPUs do not allow the Φ1 and Φ2 clocks to be stretched, nor do they allow the Data and Address Busses to be floated; also, there is no Halt state. **The single RDY signal is used to interface slow memories, to refresh dynamic memories or to perform Direct Memory Access operations.**

MCS6500 WAIT STATE

What the RDY input signal does is allow you to insert one or more Wait machine cycles in between two normal instruction execution machine cycles:



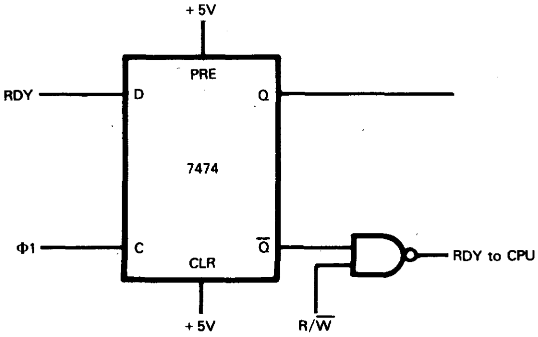
The RDY input allows Wait machine cycles to be inserted within any instruction's normal sequence of machine cycles. For Wait machine cycles to occur, the RDY input must make a high-to-low transition during a Φ1 high clock pulse. This transition may occur during any nonwrite machine cycle. Timing may be illustrated as follows:



Wait machine cycles will be inserted until RDY is sensed high during a Φ2 high pulse.

If an RDY high-to-low transition occurs during a write machine cycle, then the Wait states will still be inserted, but the insertion will occur following the next nonwrite machine cycle.

A non-obvious feature of the MCS6500 RDY signal is the fact that there is no acknowledge response from the CPU to external logic. This can be a problem. To guarantee that the machine cycle following the RDY high-to-low transition will be a Wait, you must make sure that RDY never makes a high-to-low transition during a write cycle. Fortunately, you can use the R/$\overline{W}$ output to detect write cycles and thus generate a safe RDY input. Here is simple sample logic:



Since the same $\Phi 1$ clock pulse that triggers the 7474 flip-flop also triggers any change in $R/\overline{W}$ signal level, $R/\overline{W}$ is NANDed with $\overline{Q}$ after taking the 7474 settling delay — which also gives $R/\overline{W}$ time to acquire its new level.

If you are interfacing slow memories, performing Direct Memory Access or refreshing dynamic memories, in each case the extra time provided for the secondary operation is the Wait state generated via the RDY input, as we have just described.

MCS6500 SLOW MEMORY INTERFACE

When interfacing slow memories, the logic of the Wait state is self-evident. The slow memory simply has additional machine cycles in which to respond to the memory access, and memory select logic holds RDY low for any required time delay.

When using a Wait state to perform Direct Memory Access or dynamic memory refresh operations, there is a further complication. During the Wait state, the Data and Address Busses are not floated. Alternate Data and Address Busses must therefore be provided, connected via a tristate buffer to any memory device which is being accessed.

INTERRUPT PROCESSING AND SYSTEM RESET

The MCS6500 microcomputer system handles interrupts and resets exactly as the MC6800. For a discussion of this subject, therefore, see Chapter 8 — with the following provisos:

1) Neither the MCS6500 nor the MC6800 will acknowledge an interrupt if the interrupt enable status bit has been set to 1. Additionally, the MC6800 will not acknowledge an interrupt while in the Halt state. The MCS6500 has no Halt state, but Wait states induced by the RDY line may be looked upon as equivalent. If an interrupt request occurs while Wait states are being created by an MCS6500 CPU in response to the RDY control input, then the interrupt acknowledge process will begin with the first non-Wait machine cycle.
2) When the MCS6500 executes a software interrupt, the Break status is set. The MC6800 has no such status flag.
3) The MCS6500 Stack is 256 bytes long and is implemented in memory locations 0100_{16} through $01FF_{16}$. The MC6800 Stack can have any length within the allowed memory space, and can be located anywhere in memory.

One technique you can use in order to disconnect memory or other external logic from an MCS6500 CPU, is to address a nonexistent memory location. Clearly if an invalid address is on the Address Bus, then no device can consider itself selected and all devices will remain in an inactive state.

MCS6500 CPU CLOCK LOGIC

Clock logic required by the MCS651X series of CPUs is identical to that which has already been described for the MC6800 in Chapter 8. Indeed, you can use any of the MC6870 series clock devices in order to create timing inputs.

The MCS650X series CPUs have on-chip logic; all they need is an external crystal or RC network. A number of possible circuits, described in MOS Technology literature, are reproduced in Figure 9-11.

MCS6500 CPU INTERFACE LOGIC

Look again at Table 9-1 and you will see that the 28-pin CPUs are remarkable because they output so few control signals; in fact, the MCS6513, MCS6514, and MCS6515 output just one control signal: $R/\overline{W}$. The remaining 28-pin CPUs additionally output clock signals only. There is no interrupt acknowledge, no synchronization output, nor any control signal which external logic can use to determine what is going on within the CPU. **Of all the microprocessors described in this book, none provides so few**

control output signals. So long as you are building relatively straightforward microcomputer systems, this does not present a problem. The Address and Data Busses are never floated by 28-pin CPUs; therefore, external logic, upon detecting a select address on the Address Bus, will simply respond by reading or writing — depending upon the level of the R/$\overline{W}$ signal. The fact that this signal is high in its idle state, indicating a read, simply means that the selected external logic will place the contents of its addressed memory location on the Data Bus. If the R/$\overline{W}$ signal is really in its standby state, then the CPU will ignore the Data Bus contents and no harm is done. Thus, for simple microcomputer systems, the MCS6500 series CPUs are remarkably simple devices to work with. If a microcomputer system becomes complex, however, problems may arise. **DMA logic must account for the fact that there is no detectable standby state for memory or I/O devices to detect; any device selected by the address of the Address Bus is continuously responding to a read or write command.**

We conclude that when designing microcomputer systems around an MCS6500 CPU, if you are going to share the System Bus in any way, you must be very cautious about ensuring that you have accounted for the passive role of support logic surrounding the CPU.

THE MCS6500 INSTRUCTION SET

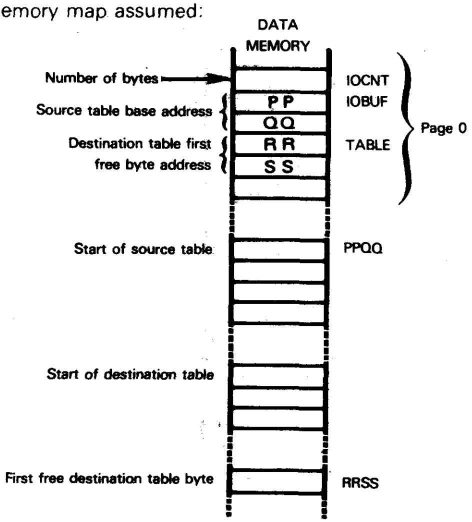
Table 9-2 summarizes the MCS6500 instruction set. This instruction set follows the philosophy of the MC6800 very closely.

THE BENCHMARK PROGRAM

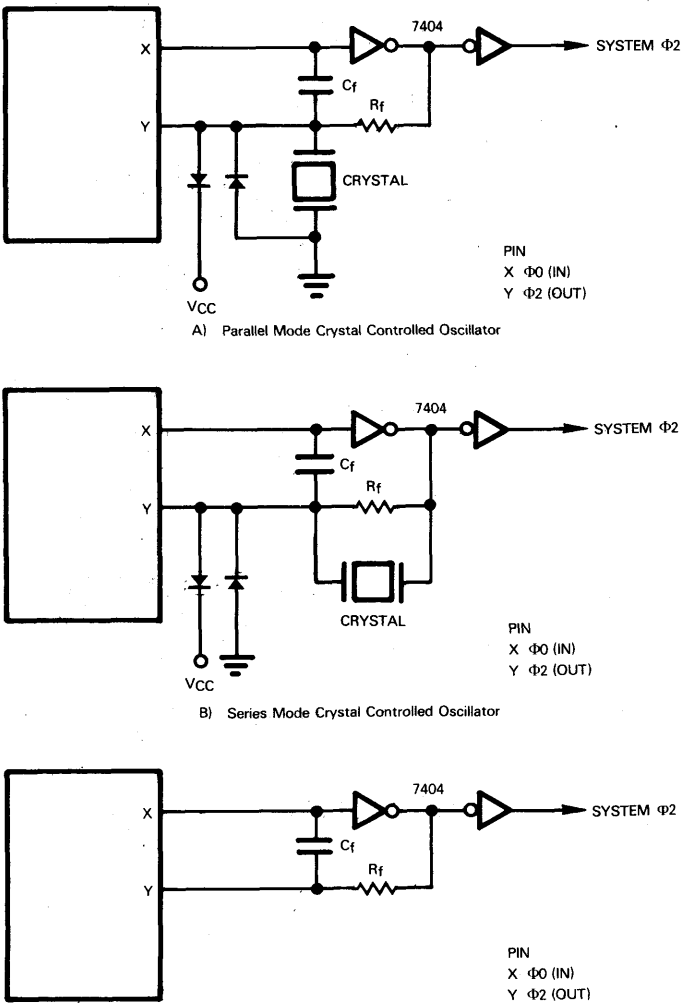
The benchmark program is coded for the MCS6500 as follows:

```
        LDY     IOCNT       LOAD BUFFER LENGTH INTO Y INDEX
LOOP    LDA     (IOBUF),Y   LOAD NEXT SOURCE BYTE
        STA     (TABLE),Y   STORE IN NEXT DESTINATION BYTE
        DEY                 DECREMENT Y
        BNE     LOOP        RETURN FOR MORE BYTES
        LDA     IOCNT       AT END ADD NUMBER OF BYTES
        CLC                 TO CURRENT TABLE BASE ADDRESS
        ADC     TABLE +1
        STA     TABLE +1
```

This is the memory map assumed:



DATA
MEMORY
Number of bytes
Source table base address
Destination table first free byte address
PP
QQ
RR
SS
IOCNT
IOBUF
TABLE
Page 0
Start of source table
PPQQ
Start of destination table
First free destination table byte
RRSS



A) Parallel Mode Crystal Controlled Oscillator

B) Series Mode Crystal Controlled Oscillator

C) Time Base Generator — RC Network

X is pin 39 for the MCS6502, or pin 28 for any other MCS650X CPU
Y is pin 37 for the MCS6502, or pin 27 for any other MCS650X CPU

Figure 9-11. Time Base Generation For MCS650X CPU Input Clocks

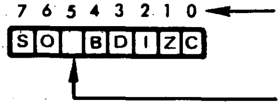
The programming example illustrated above makes use of indirect addressing. Somewhere in the first 256 bytes of memory we store the number of bytes to be transferred, the beginning address for the source table, and the address for the first free destination table byte. By loading the byte count into the Y Index register, we can use this register both as an index for moving data from source to destination, and as a counter.

After moving the block of data, we must add the number of moved data bytes to the destination table first free byte address; this accounts for the fact that the destination table has been incrementally filled.

When comparing the MCS6500 with the MC6800, we see that we have indeed reduced the number of instructions from 11 to 9; the number of instructions within the iterative loop has been reduced from 5 to 4. We cannot make a more substantial reduction in the number of instructions because the MC6800 program uses the Stack Pointer as an Index register — which is not an option with the MCS6500. We might argue that the MCS6500 has an advantage by not immobilizing the Stack while the instruction sequence is executed; however, the MCS6500 has the disadvantage of requiring both the source and destination tables to have a maximum length of 256 bytes; the MC6800 program makes no such demand.

Symbols are used in Table 9-2 as follows:

Registers: A Accumulator
X Index Register X
Y Index Register Y
PC Program Counter
SP Stack Pointer
SR Status register, with bits assigned as follows:



Statuses: S Sign status
Z Zero status
C Carry status
O Overflow status

Symbols in the column labeled STATUSES:

(blank) operation does not affect status
X operation affects status
0 operation clears status
1 operation sets status
6 status reflects bit 6 of memory location
7 status reflects bit 7 of memory location

ADR 8 bits of immediate or base address

ADR16 16 bits of immediate or base address

a8 Any of the following operands and addressing modes:
ADR Base Page Direct
ADR,X Base Page Indexed via Register X
(ADR,X) Pre-Indexed Indirect
(ADR),Y Post-Indexed Indirect

a16 Any of the following operands and addressing modes:
ADR16 Extended Direct
ADR16,X Absolute Indexed via Register X
ADR16,Y Absolute Indexed via Register Y

B	Break status
D	Decimal Mode status
DATA	8 bits of immediate data
DISP	An 8-bit, signed address displacement
I	Interrupt disable status
LABEL	16-bit immediate address, destination of Jump-on-Subroutine call
M ( )	The memory location addressed via the mode specified in parenthesis
PC(HI)	The most significant 8 bits of the Program Counter
PC(LO)	The least significant 8 bits of the Program Counter
[ ]	Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.
[[ ]]	Implied memory addressing; the contents of the memory location designated by the contents of a register or address calculation.
∧	Logical AND
∨	Logical OR
∀	Logical Exclusive-OR
← →	Data is transferred in the direction of the arrow
⟷	Data is exchanged between the two locations designated on either side of the arrow

Table 9-2. A Summary Of The MCS6500 Microcomputer Instruction Set

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES	OPERATION PERFORMED
				S Z C O	
I/O AND PRIMARY MEMORY REFERENCE	LDA	ADR	2	X X	[A]←[ADR] or
		ADR,X } a8	2		[A]←[[X] + ADR] or
		(ADR,X)	2		[A]←[[[X] + ADR]] or
		(ADR),Y	2		[A]←[[ADR + 1,ADR] + [Y]] or
		ADR16	3		[A]←[ADR16] or
		ADR16,X } a16	3		[A]←[ADR16 + [X]] or
		ADR16,Y	3		[A]←[ADR16 + [Y]] Load Accumulator from memory using any of the following addressing modes: Base page direct Base page indexed (X register) Pre-indexed indirect Post-indexed indirect Extended direct Absolute indexed (Register X or Register Y)
	STA	a8	2		M(a8) ←[A] or M(a16) ←[A]
		a16	3		Store Accumulator to memory using any of the addressing modes permitted with LDA.
	LDX	ADR or ADR,Y	2	X X	[X]←[ADR] or [X]←[ADR16] or
		ADR16 or ADR16,Y	3		[X]←[[Y] + ADR] or [X]←[ADR16 + [Y] Load Index Register X from memory using direct, extended, base page indexed or absolute indexed addressing, indexing through Register Y.
	STX	ADR or ADR,Y	2		[ADR]←[X] or [ADR16]←[X] or
		ADR16	3		[[Y] + ADR]←[X] Store Index Register X to memory using direct, extended or base page indexed addressing, indexing through Register Y.
	LDY	ADR or ADR,X	2	X X	[Y]←[ADR] or [Y]←[ADR16] or
		ADR16 or ADR16,X	3		[Y]←[[X] + ADR] or [Y]←[ADR16 + [X]] Load Index Register Y from memory using direct, extended, base page indexed or absolute indexed addressing, indexing through Register Y.

Table 9-2. A Summary Of The MCS6500 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
I/O AND PRIMARY MEMORY REF. (CONTINUED)	STY	ADR or ADR,X ADR16	2					[ADR]←[Y] or [ADR16]←[Y] or [[X] + ADR]←[Y] Store Index Register Y to memory using direct, extended, or base page indexed addressing, indexing through Register X.
MEMORY OPERATE	ADC	a8 a16	2 3	X	X	X	X	[A]←[A] + M(a8) + C or [A]←[A] + M(a16) + C Add contents of memory location, with carry, to those of Accumulator, using any of the addressing modes permitted with LDA. Zero flag is not valid in Decimal Mode.
	AND	a8 a16	2 3	X	X			[A]←[A] ∧ M(a8) or [A]←[A] ∧ M(a16) AND contents of Accumulator with those of memory location addressed via any of the modes permitted with LDA.
	BIT	ADR8 ADR16	2 3	7	X		6	[A] ∧ [ADR8] or [A] ∧ [ADR16] AND contents of Accumulator with those of memory location. Only the status bits are affected. Direct or extended addressing modes may be used.
	CMP	a8 a16	2 3	X	X	X		[A] - M(a8) or [A] - M(a16) Compare contents of Accumulator with those of memory location, affecting status bit only. Any of the addressing modes permitted with LDA may be used.
	EOR	a8 a16	2 3	X	X			[A]←[A] ⊻ M(a8) or [A]←[A] ⊻ M(a16) Exclusive-OR contents of Accumulator with those of memory location, using any of the addressing modes permitted with LDA.
	ORA	a8 a16	2 3	X	X			[A]←[A] V M(a8) or [A]←[A] V M(a16) OR contents of Accumulator with those of memory location, using any of the addressing modes permitted with LDA.
	SBC	a8 a16	2 3	X	X	X	X	[A]←[A] - M(a8) - $\overline{C}$ or [A]←[A] - M(a16) - $\overline{C}$ Subtract contents of memory location, with borrow, from contents of Accumulator. Any addressing mode permitted with LDA may be used. Note that Carry reflects the complement of the borrow.

Table 9-2. A Summary Of The MCS6500 Microcomputer Instruction Set (Continued)

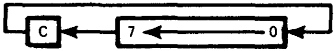
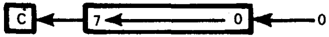
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES S	Z	C	O	OPERATION PERFORMED
SECONDARY MEMORY REFERENCE (MEMORY OPERATE) (CONTINUED)	INC	ADR8 or ADR,X ADR16 or ADR16,X	2 3	X	X			[ADR]←[ADR] + 1 or [ADR16]←[ADR16] + 1 or [[X] + ADR]←[[X] + ADR] + 1 or [ADR16 + [X]]←[ADR16 + [X]] + 1 Increment contents of memory location using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.
	DEC	ADR or ADR,X ADR16 or ADR16,X	2 3	X	X			[ADR]←[ADR] - 1 or [ADR16]←[ADR16] - 1 or [[X] + ADR]←[[X] + ADR] - 1 or [ADR16 + [X]]←[ADR16 + [X]] - 1 Decrement contents of memory location using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.
	CPX	ADR ADR16	2 3	X	X	X		[X] - [ADR] or [X] - [ADR16] Compare contents of X register with those of memory location, using direct or extended addressing. Only the status flags are affected.
	CPY	ADR ADR16	2 3	X	X	X		[Y] - [ADR] or [Y] - [ADR16] Compare contents of Y register with those of memory location using direct or extended addressing. Only the status flags are affected.
	ROL	ADR or ADR,X ADR16 or ADR16,X	2 3	X	X	X		 [ADR] or [ADR16] or [[X] + ADR] or [ADR16 + [X]] Rotate contents of memory location left through Carry, using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.
	ASL	ADR or ADR,X ADR16 or ADR16,X	2 3	X	X	X		 [ADR] or [ADR16] or [[X] + ADR] or [ADR16 + [X]] Arithmetic shift left contents of memory location using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.

Table 9-2. A Summary Of The MCS6500 Microcomputer Instruction Set (Continued)

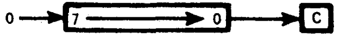
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
MEMORY OPERATE (CONTINUED)	LSR	ADR or ADR,X ADR16 or ADR16,X	2 3	0	X	X		 [ADR] or [ADR16] or [[X] + ADR] or [ADR16 + [X]] Logical shift right contents of memory location, using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.
IMMEDIATE	LDA	DATA	2	X	X			[A]←DATA Load Accumulator with immediate data.
	LDX	DATA	2	X	X			[X]←DATA Load Index Register X with immediate data.
	LDY	DATA	2	X	X			[Y]←DATA Load Index Register Y with immediate data.
IMMEDIATE OPERATE	ADC	DATA	2	X	X	X	X	[A]←[A] + DATA + C Add immediate, with Carry, to Accumulator. The Zero flag is not valid in Decimal Mode.
	AND	DATA	2	X	X			[A]←[A] Λ DATA AND immediate with Accumulator.
	CMP	DATA	2	X	X	X		[A] - DATA Compare immediate with Accumulator. Only the status flags are affected.
	EOR	DATA	2	X	X			[A]←[A] ⊻DATA Exclusive-OR immediate with Accumulator.
	ORA	DATA	2	X	X			[A]←[A] V DATA OR immediate with Accumulator.
	SBC	DATA	2	X	X	X	X	[A]←[A] - DATA - $\overline{C}$ Subtract immediate, with borrow, from Accumulator. Note that Carry reflects the complement of the borrow.
	CPX	DATA	2	X	X	X		[X] - DATA Compare immediate with Index Register X. Only the status flags are affected.
	CPY	DATA	2	X	X	X		[Y] - DATA Comoare immediate with Index Register Y. Only the status flags are affected.

Table 9-2. A Summary Of The MCS6500 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES S	 Z	 C	 O	OPERATION PERFORMED
JUMP	JMP	LABEL (LABEL)	3					[PC]←LABEL or [PC]←[LABEL] Jump to new location, using extended or indirect addressing.
	JSR	LABEL	3					[[SP]]←[PC(HI)], [[SP]-1]←[PC(LO)], [SP]←[SP]-2, [PC]←LABEL Jump to subroutine beginning at address given in bytes 2 and 3 of the instruction.
BRANCH ON CONDITION	BCC	DISP	2					If C = 0, then [PC]←[PC] + 1 + DISP Branch relative if Carry flag is cleared.
	BCS	DISP	2					If C = 1, then [PC]←[PC] + 1 + DISP Branch relative if Carry flag is set.
	BEQ	DISP	2					If Z = 1, then [PC]←[PC] + 1 + DISP Branch relative if result is equal to zero.
	BMI	DISP	2					If S = 1, then [PC]←[PC] + 1 + DISP Branch relative if result is negative.
	BNE	DISP	2					If Z = 0, then [PC]←[PC] + 1 + DISP Branch relative if result is not zero.
	BPL	DISP	2					If S = 0, then [PC]←[PC] + 1 + DISP Branch relative if result is positive.
	BVC	DISP	2					If O = 0, then [PC]←[PC] + 1 + DISP Branch relative if Overflow flag is cleared.
	BVS	DISP	2					If O = 1, then [PC]←[PC] + 1 + DISP Branch relative if Overflow flag is set.
REGISTER-REGISTER MOVE	TAX		1	X	X			[A]→[X] Move Accumulator contents to Index Register X.
	TXA		1	X	X			[X]→[A] Move contents of Index Register X to Accumulator.

Table 9-2. A Summary Of The MCS6500 Microcomputer Instruction Set (Continued)

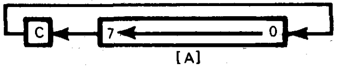
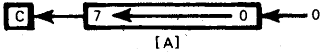
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
REGISTER-REGISTER MOVE (CONTINUED)	TAY		1	X	X			[A]→[Y] Move Accumulator contents to Index Register Y.
	TYA		1	X	X			[Y]→[A] Move contents of Index Register Y to Accumulator.
	TSX		1	X	X			[SP]→[X] Move contents of Stack Pointer to Index Register X.
	TXS		1					[X]→[SP] Move contents of Index Register X to Stack Pointer.
REGISTER OPERATE	DEX		1	X	X			[X]←[X]-1 Decrement contents of Index Register X.
	DEY		1	X	X			[Y]←[Y]-1 Decrement contents of Index Register Y.
	INX		1	X	X			[X]←[X]+1 Increment contents of Index Register X.
	INY		1	X	X			[Y]←[Y]+1 Increment contents of Index Register Y.
	ROL	A	1	X	X	X		 Rotate contents of Accumulator left through Carry.
	ASL	A	1	X	X	X		 Arithmetic shift left contents of Accumulator.

Table 9-2. A Summary Of The MCS6500 Microcomputer Instruction Set (Continued)

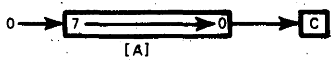
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES S Z C O	OPERATION PERFORMED
REGISTER OPERATE (CONTINUED)	LSR	A	1	0 X X	 Logical shift right contents of Accumulator.
STACK	PHA		1		[[SP]]←[A], [SP]←[SP]-1 Push Accumulator contents onto Stack.
	PLA		1	X X	[A]←[[SP] + 1], [SP]←[SP] + 1 Load Accumulator from top of Stack (PULL).
	PHP		1		[[SP]]←[SR], [SP]←[SP]-1 Push Status register contents onto Stack.
	PLP		1	X X X X	[SR]←[[SP] + 1], [SP]←[SP] + 1 Load Status register from top of Stack (PULL).
	RTS		1		[PC(LO)]←[[SP] + 1] [PC(HI)]←[[SP] + 2], [SP]←[SP] + 2, [PC]←[PC] + 1 Return from subroutine.
INTERRUPT	CLI		1		I←0 Enable interrupts by clearing interrupt disable bit of Status register.
	SEI		1		I←1 Disable interrupts.
	RTI		1	X X X X	[SR]←[[SP] + 1], [PC(LO)]←[[SP] + 2], [PC(HI)]←[[SP] + 3], [SP]←[SP] + 3, [PC]←[PC] + 1 Return from interrupt; restore Status register and Program Counter from top of Stack.

Table 9-2. A Summary Of The MCS6500 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
INTERRUPT (CONTINUED)	BRK		1					[[SP]]←[PC(HI)], [[SP]-1]←[PC(LO)], [[SP]-2]←[SR], [SP]←[SP]-3, [PC(LO)]←[FFFE], [PC(HI)]←[FFFF], I ← 1, B ← 1 Programmed interrupt. BRK cannot be disabled.
STATUS	CLC		1			0		C ←0 Clear Carry flag.
	SEC		1			1		C ←1 Set Carry flag.
	CLD		1					D ←0 Clear Decimal Mode.
	SED		1					D ←1 Set Decimal Mode.
	CLV		1			0		O ←0 Clear Overflow flag.
	NOP		1					No Operation.

The following symbols are used in the object codes in Table 9-3.

Address mode selection:

aaa		
	000	pre-indexed indirect — (ADR,X)
	001	direct — ADR
	010	immediate — DATA
	011	extended direct — ADR16
	100	post-indexed indirect — (ADR),Y
	101	base page indexed — ADR,X
	110	absolute indexed — ADR16,Y
	111	absolute indexed — ADR16,X
bb		
	00	direct — ADR
	01	extended direct — ADR16
	10	base page indexed — ADR,X
	11	absolute indexed — ADR16,X
bbb		
	001	direct — ADR
	010	accumulator — A
	011	extended direct — ADR16
	101	base page indexed — ADR,X
	111	absolute indexed — ADR16,X
cc		
	00	immediate — DATA
	01	direct — ADR
	11	extended direct — ADR16
ddd		
	000	immediate — DATA
	001	direct — ADR
	011	extended direct — ADR16
	101	base page indexed — ADR,Y in LDX; ADR,X in LDY
	111	absolute indexed — ADR16,Y in LDX; ADR16,X in LDY

pp the second byte of a two- or three-byte instruction.

qq the third byte of a three-byte instruction.

x one bit choosing the address mode.

Two numbers in the "Machine Cycles" column (for example, 2 - 6) indicate that execution time depends on the addressing mode.

Table 9-3. Summary Of MCS6500 Object Codes, With MC6800 Mnemonics

MNEMONIC	OPERAND(S)	OBJECT CODE	BYTES	MACHINE CYCLES	MC6800 INSTRUCTION
ADC		011aaa01			ADCA
	DATA or a8	pp	2	2-6	ADR8 or DATA
	a16	qq	3	4	ADR16
AND		001aaa01			ANDA
	DATA or a8	pp	2	2-6	ADR8 or DATA
	a16	qq	3	4	ADR16
ASL	A	000bbb10	1	2	ASL A
	ADR or ADR,X	pp	2	5-6	ADR8
	ADR16 or ADR16,X	qq	3	6-7	ADR16
BCC	DISP	90 pp	2	2	BCC DISP
BCS	DISP	B0 pp	2	2	BCS DISP
BEQ	DISP	F0 pp	2	2	BEQ DISP
BIT		0010x100			BITA
	ADR (x = 0)	pp	2	3	ADR8 or DATA
	ADR16 (x = 1)	qq	3	4	ADR16
BMI	DISP	30 pp	2	2	BMI DISP
BMI	DISP	30 pp	2	2	BMI DISP
BNE	DISP	D0 pp	2	2	BNE DISP
BPL	DISP	10 pp	2	2	BPL DISP
BRK		00	1	7	(SWI)
BVC	DISP	50 pp	2	2	BVC DISP
BVS	DISP	70 pp	2	2	BVS DISP
CLC		18	1	2	CLC
CLD		D8	1	2	
CLI		58	1	2	CLI
CLV		B8	1	2	CLV
CMP		110aaa01			CMPA
	DATA or a8	pp	2	2-6	ADR8 or DATA
	a16	qq	3	4	ADR16
CPX		1110cc00			CPX
	DATA or ADR	pp	2	2-3	ADR8
	ADR16	qq	3	4	DATA 16 or ADR16
CPY		1100cc00			
	DATA or ADR	pp	2	2-3	
	ADR16	qq	3	4	
DEC		110bb110			DEC
	ADR or ADR,X	pp	2	5-6	ADR8
	ADR16 or ADR16,X	qq	3	6-7	ADR16
DEX		CA	1	2	DEX
DEY		88	1	2	
EOR		010aaa01			EORA
	DATA or a8	pp	2	2-6	ADR8 or DATA
	a16	qq	3	4	ADR16
INC		111bb110			INC
	ADR or ADR,X	pp	2	5-6	ADR8
	ADR16 or ADR16,X	qq	3	6-7	ADR16
INX		E8	1	2	INX
INY		C8	1	2	
JMP	LABEL (x = 0)	01x01100	3	3-5	JMP ADR16
	or (LABEL)(x = 1)	ppqq			
JSR	LABEL	20 ppqq	3	6	JSR ADR16
LDA		101aaa01			LDAA
	DATA or a8	pp	2	2-6	ADR8 or DATA
	a16	qq	3	4	ADR16

Table 9-3. Summary Of MCS6500 Object Codes, With MC6800 Mnemonics (Continued)

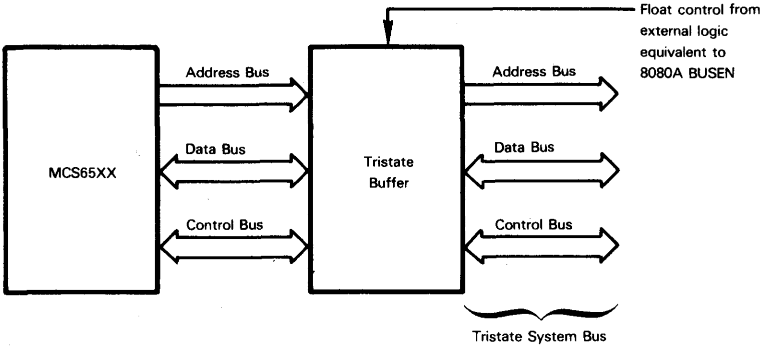
MNEMONIC	OPERAND(S)	OBJECT CODE	BYTES	MACHINE CYCLES	MC6800 INSTRUCTION
LDX	DATA or	101ddd10			LDX
	ADR or ADR,Y	pp	2	2-4	ADR8
	ADR16 or ADR16,Y	qq	3	4	ADR16 or DATA16
LDY	DATA or	101ddd00			
	ADR or ADR,X	pp	2	2-4	
	ADR16 or ADR16,Y	qq	3	4	
LSR	A	010bbb10	1	2	LSR A
	ADR or ADR,X	pp	2	5-6	ADR8
	ADR16 or ADR16,X	qq	3	6-7	ADR16
NOP		EA	1	2	NOP
ORA		000aaa01			ORA
	DATA or a8	pp	2	2-6	ADR8 or DATA
	a16	qq	3	4	ADR16
PHA		48	1	3	PSHA
PHP		08	1	3	
PLA		68	1	4	PULA
PLP		28	1	4	
ROL	A	001bbb10	1	2	ROL A
	ADR or ADR,X	pp	2	5-6	ADR8
	ADR16 or ADR16,X	qq	3	6-7	ADR16
RTI		40	1	6	RTI
RTS		60	1	6	RTS
SBC		111aaa01			SBCA
	DATA or a8	pp	2	2-6	ADR8 or DATA
	a16	qq	3	4	ADR16
SEC		38	1	2	SEC
SED		F8	1	2	
SEI		78	1	2	SEI
STA	(aaa=010)	100aaa01			STAA
	a8		2	3-6	ADR8
	a16		3	4-5	ADR16
STX	ADR(bb=00)	100bb110			STX
	or ADR,Y(bb=10)		2	3-4	ADR8
	ADR16 (bb=01)		3	4	ADR16
STY	ADR (bb=00)	100bb100			
	or ADR,X (bb=10)	pp	2	3-4	
	ADR16 (bb=01)	qq	3	4	
TAX		AA	1	2	
TAY		A8	1	2	
TSX		BA	1	2	TSX
TXA		8A	1	2	
TXS		9A	1	2	TXS
TYA		98	1	2	

SUPPORT DEVICES THAT MAY BE USED WITH THE MCS6500 SERIES MICROPROCESSORS

The MCS6500 and MC6800 microprocessors are similar enough for MC6800 support devices to be used with an MCS6500 series central processing unit.

The similarities between the MC6800 and MCS6500 extend also to the way in which you use other support devices with these two microprocessors. Thus comments regarding 8080A and Z80 support devices being used with the MC6800 apply for the most part to the MCS6500.

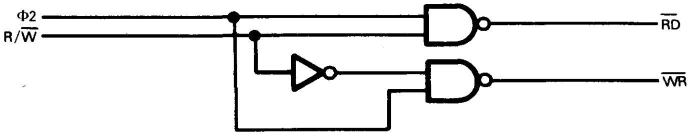
But the MCS6500 does have some limitations. The most prominent limitation is the fact that no MCS6500 microprocessor floats its System Bus. Only the MCS6512 has any bus floating capability at all; you can float its Data Bus. Within an MCS6500 microcomputer system, **if you wish to float the System Bus** or perform direct memory access operations, **you must have an external tristate buffer.** This tristate buffer receives as inputs the System Bus from the MCS6500; it creates as outputs the System Bus which will be used by support devices. This may be illustrated as follows:



Using a 40-pin MCS6500 series microprocessor, you **use the RDY control input to disable the CPU while floating the System Bus.** This control input causes the CPU to generate endless Wait states for as long as the RDY signal is low. In the unlikely event that you wish to float the System Bus of a 28-pin MCS6500 series microprocessor, you will have to generate an interrupt request, or use some other technique to enter a "No Operation" instruction loop for the duration of the floated System Bus.

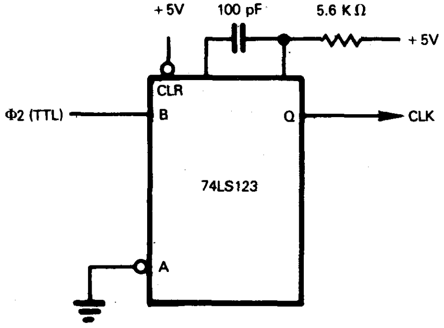
Like the MC6800, **the MCS6500 outputs no interrupt acknowledge signal. The MCS6500 also outputs no "Valid Memory Address" (VMA) signal.** You must therefore decode appropriate addresses off the Address Bus in order to generate an interrupt acknowledge signal, without the security of a conditioning VMA control signal. Your program logic will have to guarantee that the relevant addresses appear on the Address Bus only during the interrupt acknowledge process.

The read and write control signals required by 8080A support devices can be generated from the single MCS6500 R/$\overline{\text{W}}$ signal, together with the system clock signal, as follows:



The logic illustrated above is identical to that which was illustrated for the MC6800. As for the MC6800, if you wish to discriminate between I/O devices and memory, you will have to create appropriate control signals by decoding off the Address Bus addresses which have been assigned to your I/O space.

You can generate an 8080A compatible system clock from Φ2 (TTL) as follows:



The clock logic illustrated above is identical to that which we described for the MC6800.

THE MCS6522 PERIPHERAL INTERFACE ADAPTER

The MCS6522 PIA is an enhanced version of the MC6820, which is also manufactured by MOS Technology as the MCS6520 Peripheral Interface Adapter. As such, the MCS6522 PIA can be used interchangeably in MC6800 or MCS6500 microcomputer systems.

This description of the MCS6522 will concentrate on highlighting device enhancements, relying on the discussion of the MC6820, given in Chapter 8, for a detailed explanation of functions common to both parts.

The MCS6522 PIA is a general purpose I/O device which, like the MC6820 PIA provides 16 I/O pins, configured as two 8-bit I/O ports. As compared to the MC6820 PIA the MCS6522 provides more handshaking logic associated with parallel data transfers occurring via I/O Port A. Counter/timer and elementary serial I/O logic have been added to MCS6522 Port B.

Figure 9-12 illustrates that part of our general purpose microcomputer system logic which has been implemented on the MCS6522 PIA.

The MCS6522 PIA is packaged as a 40-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL compatible. I/O Port A and B pins are also CMOS logic compatible. I/O Port B pins may be used as a power source to directly drive the base of a transistor switch.

The device is implemented using N-channel, silicon gate MOS technology.

THE MCS6522 PIA PINS AND SIGNALS

The MCS6522 PIA pins and signals are illustrated in Figure 9-13. Signals which are identical to the MC6820, both in function and pin assignment, are shaded.

We will summarize all signal functions, those which are unique to the MCS6522 as well as those which are common to the MC6820, before describing the various MCS6522 PIA operations which can be performed.

Consider first the various Data Busses.

D0 - D7 represents the bidirectional Data Bus via which all communications between the CPU and the MCS6522 occur. **This Data Bus is identical to that of the MC6820.** When the MCS6522 is not selected, the Data Bus buffer is placed in a high impedance state — which is absolutely necessary, since MCS6500 CPUs (with the exception of the MCS6512) cannot float the System Data Bus.

PA0 - PA7 and PB0 - PB7 represent Data Busses connecting I/O Ports A and B with external logic. In terms of simple data transfers, these two I/O ports are identical on the MCS6522 and MC6820 devices. In each case the 16 I/O port pins may be looked upon as 16 individual signal lines, or as two 8-bit I/O busses. Each I/O port pin can be individually assigned to input or output, but an individual pin cannot support bidirectional data transfers.

There are differences between I/O Ports A and B. Some of these differences are found in MC6800 I/O ports; others represent enhancements of the MCS6522. Let us first look at I/O port differences which are common to the MC6820 as well as the MCS6522:

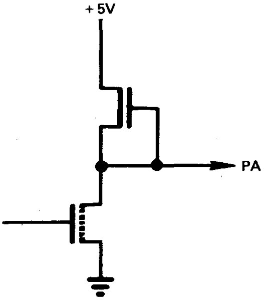
1) An I/O Port B pin which has been assigned to output will enter a tristate condition during an input operation; this is not the case for an I/O Port A pin. This means that loads placed on I/O Port B pins will not modify data waiting to be read by the CPU.

2) I/O Port A pins will register logical 1 when +2V or more are input; logical 0 results from an input of +0.4V or less. I/O Port B pins will register logical 1 when power levels below +2V are input.

3) As outputs, I/O Port B pins may be used as a source of up to a milliampere, at +1.5V, to directly drive the base of a transistor switch. This is not feasible using I/O Port A pins.

The different I/O Port A and B characteristics are a function of port pin design.

I/O Port A pins contain "passive" pullups which are resistive and allow the output voltage to go to +5V for logic 1:



The PA pins can drive two standard TTL loads.

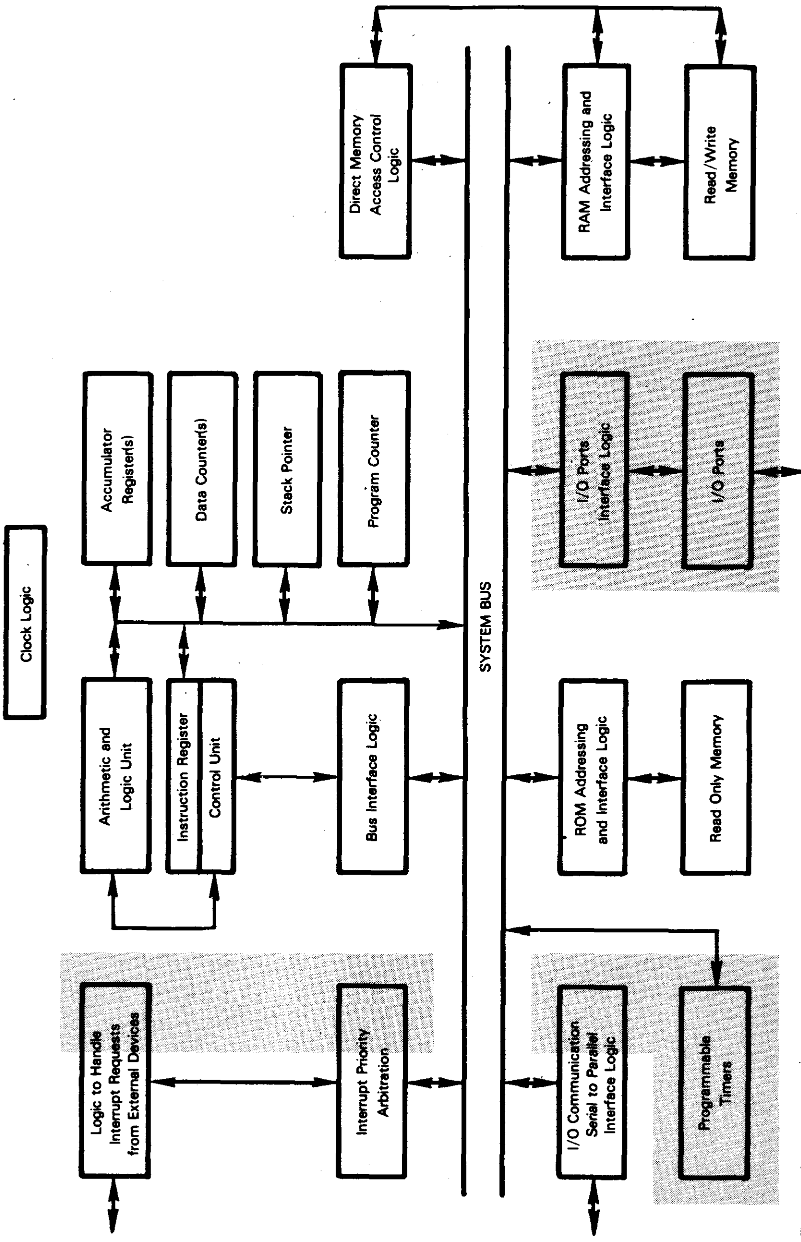
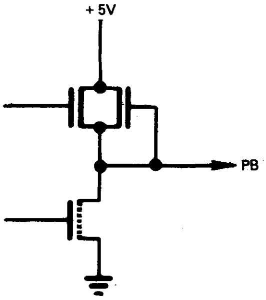
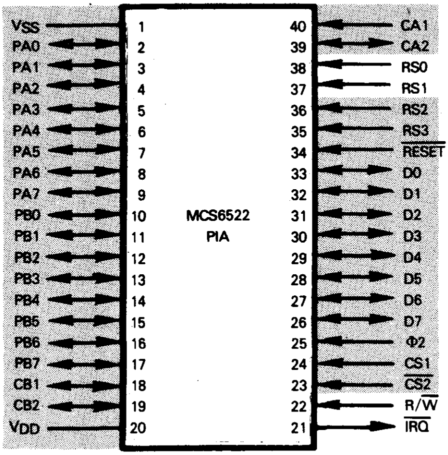

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 9-12. Logic Of The MCS6522 PIA

I/O Port B pins are push-pull devices; the pullup is switched "off" in the 0 state and "on" for a logic 1:



The pullup can source up to 3 ma at 1.5V; that is why an I/O Port B pin can drive a diode, LED or similar device.



Pin Name	Description	Type
D0 - D7	Data Bus to CPU	Tristate, bidirectional
PA0 - PA7	Port A Peripheral Data Bus	Input or Output
PB0 - PB7	Port B Peripheral Data Bus	Tristate, Input or Output
CS1, $\overline{\text{CS2}}$	Chip Select	Input
RS0 - RS3	Register Select	Input
CA1	Interrupt input to Port A	Input
CA2	Interrupt input/Peripheral control output	Input or Output
CB1	Interrupt input/Shift register access	Input or Output
CB2	Interrupt input/Peripheral control/Shift register access	Input or Output
Φ2	Device synchronization	Input
R/$\overline{\text{W}}$	Read/Write control	Input
$\overline{\text{IRQ}}$	Interrupt request	Output
$\overline{\text{RESET}}$	Reset	Input
V_{DD}, V_{SS}	Power and Ground	

Figure 9-13. MCS6522 PIA Signals And Pin Assignments

Let us now look at differences between MCS6522 I/O Port A and B pins which are the result of MCS6522 logic enhancements:

1) There are two programmable counters connected to I/O Port B logic. The MC6820 has no counter logic.
2) There is an 8-bit Shift register associated with I/O Port B logic. The Shift register provides an elementary serial I/O capability which may be adequate for certain types of control logic, but falls short of what is needed to support serial data communications. The MC6820 has no serial I/O capability whatsoever.
3) I/O Port A provides CA2 as an output control signal when the CPU reads or writes data. I/O Port B provides CB2 as an output control signal when the CPU writes data only.

The MCS6522 PIA has six device select pins.

CS1 and $\overline{\text{CS2}}$ are two typical select signals, exactly equivalent to MC6820 signals bearing the same names. Note that the MCS6522 has no CS0 select. For the MCS6522 device to be selected, CS1 must receive a high input while $\overline{\text{CS2}}$ simultaneously receives a low input.

RS0, RS1, RS2 and RS3 address one of 16 locations within the MCS6522. Thus an MCS6522 device will appear to a programmer as 16 memory locations. Note that **the MC6820** has only two address lines, RS0 and RS1, and **appears to a programmer as four memory locations.**

MCS6522 ADDRESSING

Addressing logic associated with the MCS6522 is, in fact, quite simple. Combining the two chip select signals, CS1 and $\overline{\text{CS2}}$, with the four address select signals, R0, R1, R2 and R3, simply means that total device logic will be derived from six of the 16 Address Bus lines — and **to the programmer, the MCS6522 PIA will appear as 16 contiguous memory locations.** Table 9-4 identifies the 16 addressable locations of the MCS6522; for the moment it is not important that you understand the nature of these addressable locations; rather, let us concentrate on the select lines RS0 - RS3. Throughout this description of the MCS6522, we are going to identify addressable locations by their "select code", which consists of the signal levels given in the left-hand column of Table 9-4. To a programmer, a "select code" will simply become some index which must be added to a base address. Suppose, for example, that your interfacing logic will cause an MCS6522 to consider itself selected when any address is output in the range C000$_{16}$ through C00F$_{16}$. Select code 0000$_2$ now corresponds to memory address C000$_{16}$; select code 0111$_2$ now corresponds to memory address C007$_{16}$. That is the relationship between select code and memory address.

There are four timing and control signals which interface an MCS6522 with external logic. These four signals are CA1, CA2, CB1 and CB2. Superficially, these four signals are identical to their MC6820 equivalents. But there are some secondary differences.

CA1 and CA2 are control signals associated with I/O Port A. CA1 is an input signal whereas CA2 is bidirectional. CB1 and CB2 are equivalent signals associated with I/O Port B, however, CB1 is bidirectional, although it is used as an input by Shift register logic only.

There are two control signals associated with the MCS6522 CPU interface.

$\Phi 2$ is the phase two clock which is output by any of the MCS6500 CPUs. **The MCS6522 uses $\Phi 2$ as a standard synchronization signal, equivalent to the E signal used by the MC6820.** The trailing edge of each $\Phi 2$ pulse synchronizes all logic and timing within the MCS6522. $\Phi 2$ is used optionally by Shift register logic to clock serial input or output data.

R/$\overline{\text{W}}$ is the standard read/write control signal output by all MCS6500 CPUs. This signal is identical to that on the MC6820. Recall that when R/$\overline{\text{W}}$ is high, a read operation is specified and data transfer from the MCS6522 PIA to the CPU will occur. When R/$\overline{\text{W}}$ is low, a write operation is specified and data transfer from the CPU to the PIA will occur.

The MCS6522 has a single interrupt request signal $\overline{\text{IRQ}}$. In contrast, the MC6820 has two interrupt requests $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$. If you are simply going to wire-OR interrupt requests and connect them to the CPU $\overline{\text{IRQ}}$ pin, then having two requests, $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$, makes no sense; combining them is preferable. On the other hand, if you are going to include any type of interrupt priority arbitration logic, such as the MC6828, then by combining $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$ into a single interrupt request, you can no longer vector separately to interrupt requests arising at either I/O Port A or I/O Port B logic. You must vector a single interrupt request, arising from either of these ports; then you must execute instructions to test status bits and determine the exact interrupt source.

RESET is a standard Reset input. When input low, the contents of all MCS6522 registers will be set to 0. Reset logic of the MCS6522 and MC6820 is identical.

Table 9-4. Addressing MCS6522 Internal Registers

SELECT LINES RS3, RS2, RS1, RS0	ADDRESSED LOCATION
0000	Output register for I/O Port B
0001	Output register for I/O Port A, with handshaking
0010	I/O Port B Data Direction register
0011	I/O Port A Data Direction register
0100	Read Timer 1 Counter low order byte Write to Timer 1 Latch low order byte
0101	Read Timer 1 Counter high order byte Write to Timer 1 Latch high order byte and initiate count
0110	Access Timer 1 Latch low order byte
0111	Access Timer 1 Latch high order byte
1000	Read low order byte of Timer 2 and reset Counter interrupt Write to low order byte of Timer 2 but do not reset interrupt
1001	Access high order byte of Timer 2; reset Counter interrupt on write
1010	Serial I/O Shift register
1011	Auxiliary Control register
1100	Peripheral Control register
1101	Interrupt Flag register
1110	Interrupt Enable register
1111	Output register for I/O Port A, without handshaking

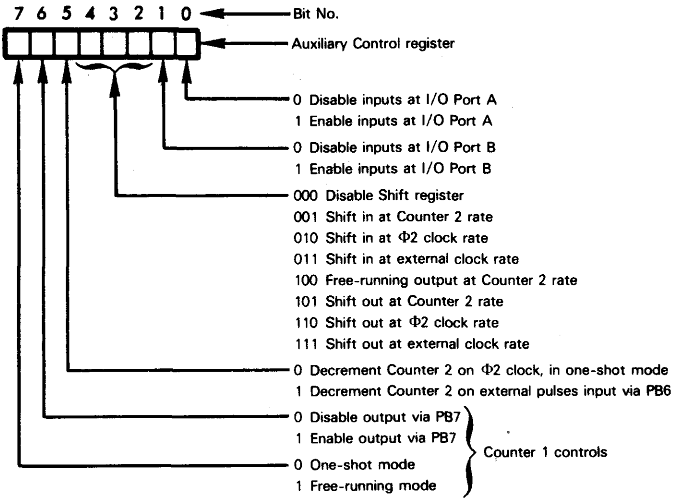

7 6 5 4 3 2 1 0
Bit No.
Auxiliary Control register
0 Disable inputs at I/O Port A
1 Enable inputs at I/O Port A
0 Disable inputs at I/O Port B
1 Enable inputs at I/O Port B
000 Disable Shift register
001 Shift in at Counter 2 rate
010 Shift in at Φ2 clock rate
011 Shift in at external clock rate
100 Free-running output at Counter 2 rate
101 Shift out at Counter 2 rate
110 Shift out at Φ2 clock rate
111 Shift out at external clock rate
0 Decrement Counter 2 on Φ2 clock, in one-shot mode
1 Decrement Counter 2 on external pulses input via PB6
0 Disable output via PB7
1 Enable output via PB7
0 One-shot mode
1 Free-running mode
Counter 1 controls

Figure 9-14. Auxiliary Control Register Bit Assignments

MCS6522 PARALLEL DATA TRANSFER OPERATIONS

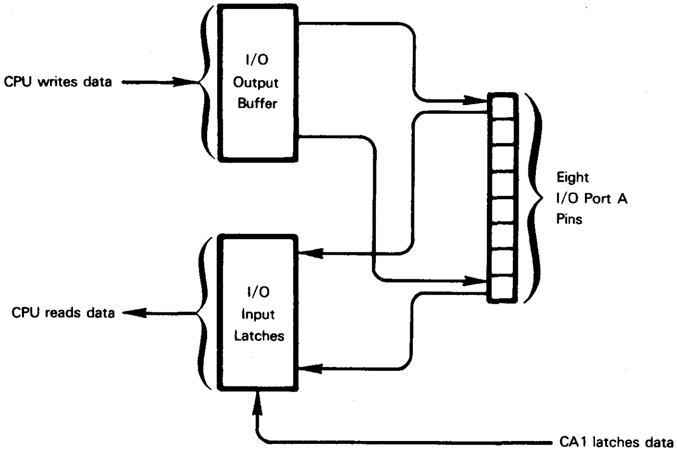
Because there are significant differences between the logic associated with MCS6522 I/O Ports A and B, we will begin by examining I/O Port A operations.

When you examine I/O Port A operations, **the first addressable location to look at is 0011 — the I/O Port A Data Direction register.** You must load a mask into this register in order to assign individual I/O port pins to input or output. A 0 in any bit of the Data Direction register will cause the corresponding I/O Port A pin to input data only. A 1 in any bit position will cause the corresponding I/O Port A pin to output data only.

**MCS6522
I/O PORT A
DATA TRANSFER**

You access I/O Port A, either to read or write data, via select code 0001_2 or 1111_2.

But before we discuss why I/O Port A has two select codes, **we must describe the way in which read and write operations occur — in conjunction with pins having been assigned to input or output.** Read and write logic is best illustrated as follows:



Data being output is written to the I/O Output buffer; signal levels are created immediately at those I/O pins which have been declared as output pins. I/O pins which have been declared as input pins are, in effect, disconnected from the I/O Output buffer — and are in no way affected by I/O Output buffer contents.

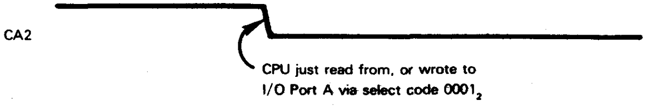
I/O Input latches will reflect the signal level of every I/O Port A pin, whether it has been assigned to input or output; I/O Input latches will acquire I/O Port A pin levels when latched by an active transition of the CA1 control input.

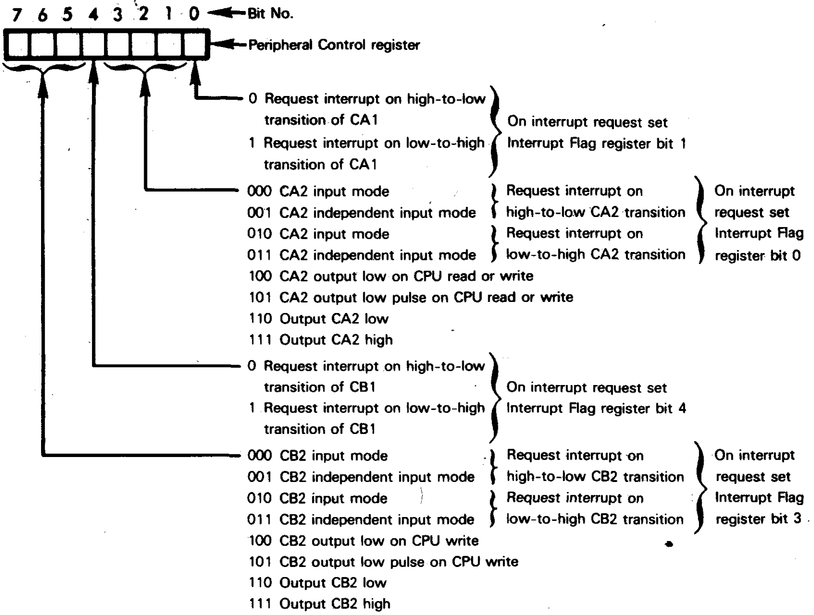
For the most part, this scheme is inconsequential to you as an MCS6522 user, since whatever you write to output pins will be output, and you will read whatever external logic inputs to input pins. The only caution is that you cannot read back what you write to output pins. Latch timing and transient signal levels at output pins can modify data as it travels from I/O Output buffers to I/O Input latches.

Irrespective of whether I/O Port A pins have been assigned to input or output, control signals CA1 and CA2 can be used to provide handshaking. External logic uses CA1 to communicate with the microcomputer system; CA2 may be a control input or a control output signal.

First you must enable I/O Port A by writing a 1 into bit 0 of the Auxiliary Control register (select code 1011), which is illustrated in Figure 9-14. Next you select your CA1 and CA2 control options by writing appropriate codes into bits 0 - 3 of the Peripheral Control register, which is illustrated in Figure 9-15.

When you access I/O Port A via select code 0001_2, then as soon as data is written into the I/O Port A buffer, the CA2 signal may output low, or it may pulse low; you determine how CA2 will respond by the code you load into the Peripheral Control register. Bits 1, 2 and 3 of the Peripheral Control register determine the way in which control signal CA2 will function. If these three bits are 100, then when you address I/O Port A via select code 0001_2, CA2 will go low as soon as the I/O Port is accessed:


CA2
CPU just read from, or wrote to I/O Port A via select code 0001_2



0 Request interrupt on high-to-low transition of CA1 } On interrupt request set Interrupt Flag register bit 1
1 Request interrupt on low-to-high transition of CA1

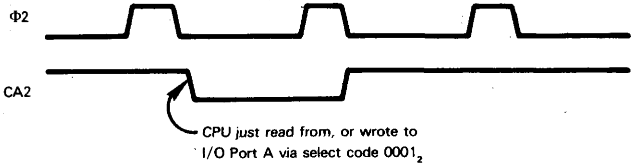
000 CA2 input mode } Request interrupt on high-to-low CA2 transition } On interrupt request set Interrupt Flag register bit 0
001 CA2 independent input mode
010 CA2 input mode } Request interrupt on low-to-high CA2 transition
011 CA2 independent input mode
100 CA2 output low on CPU read or write
101 CA2 output low pulse on CPU read or write
110 Output CA2 low
111 Output CA2 high

0 Request interrupt on high-to-low transition of CB1 } On interrupt request set Interrupt Flag register bit 4
1 Request interrupt on low-to-high transition of CB1

000 CB2 input mode } Request interrupt on high-to-low CB2 transition } On interrupt request set Interrupt Flag register bit 3
001 CB2 independent input mode
010 CB2 input mode } Request interrupt on low-to-high CB2 transition
011 CB2 independent input mode
100 CB2 output low on CPU write
101 CB2 output low pulse on CPU write
110 Output CB2 low
111 Output CB2 high

Figure 9-15. Peripheral Control Register Bit Assignments

If bits 3, 2 and 1 of the Peripheral Control register contain 101, then CA2 will pulse low for one clock period when you access the I/O Port via the select code 0001_2:


Φ2
CA2
CPU just read from, or wrote to
I/O Port A via select code 0001_2

If bits 3, 2 and 1 of the Peripheral Control register contain any other values, CA2 will not be affected by the CPU accessing I/O Port A via select code 0001_2.

If CA2 makes an active transition when you access I/O Port A, then any interrupts pending for CA1 or CA2 will be cleared.

If you access I/O Port A via the select code 1111_2, then CA2 is unaffected, whatever Peripheral Control register bits 3, 2 and 1 contain.

Notice that bits 3, 2 and 1 of the Peripheral Control register primarily determine whether control signal CA2 will be an input or an output control. We have seen two of the output control options. The remaining two output options force CA2 to be either output high or low.

Let us look at the CA2 input options, which are also specified via Peripheral Control register bits 3, 2 and 1. If any input option has been specified, then it makes no difference whether you access I/O Port A via the select code 0001_2 or 1111_2; since CA2 has been specified as input control, it cannot be output low or pulsed low when you access I/O Port A.

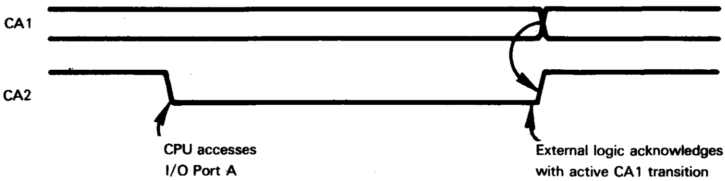
The CA2 input options available to you are as follows:

1) You can specify that a CA2 input high-to-low, or low-to-high transition will generate an interrupt request.

2) You can specify that any interrupt pending from a CA2 active transition will, or will not be cleared when I/O Port A is accessed via the select code 0001_2. Accessing I/O Port A via the select code 1111_2 will never affect any pending interrupt statuses. **In Figure 9-15, CA2 "input mode" means prior CA2 active transition interrupt requests are cleared when you access I/O Port A via select code 0001_2; no such interrupt reset occurs in "independent input" mode.**

Peripheral Control register bit 0 determines whether input control signal CA1 will generate an interrupt request on a high-to-low, or a low-to-high transition. One or the other transition will always cause an interrupt — and the only way of ignoring CA1 interrupts is to individually disable them; we will describe how this is done later when we discuss interrupt logic in general.

If you access I/O Port A via the select code 0001_2, and you cause CA2 to output low by storing 100 in bits 3, 2 and 1 of the Peripheral Control register, then CA2 will return high again when CA1 makes its active transition. This may be illustrated as follows:



While handshaking options available with I/O Port A may seem complex, in reality they are quite simple. For easy reference, options are summarized in Table 9-5.

Next consider I/O Port B.

**MCS6522
I/O PORT B
DATA TRANSFER**

If you look upon I/O Port B simply as a data transfer conduit, then it is very similar to I/O Port A, simply lacking a few I/O Port A features.

Like I/O Port A, I/O Port B has a Data Direction register (select code 0010_2), which you use to identify input and output pins. You must load a mask into this register in order to assign individual I/O port pins to input or output. A 0 in any bit of the Data Direction register will cause the corresponding I/O Port B pin to input data only. A 1 in any bit position will cause the corresponding I/O Port B pin to output data only.

You must enable I/O Port B by loading a 1 into bit 1 of the Auxiliary Control register, just as you had to enable I/O Port A.

Subsequently, you access I/O Port B via the single select code 0000_2.

You have to load an appropriate code into bits 4 - 7 of the Peripheral Control register to define the way in which control signals CB1 and CB2 will operate, just as you had to load a code into bits 3 - 0 of the Peripheral Control register to define control signal CA1 and CA2 operations. The only difference between control signals CB1 and CB2, as compared to control signals CA1 and CA2, pertains to codes 100 and 101 in bits 7, 6, 5 or 3, 2, 1 of the Peripheral Control register. Code 100 causes CA2 or CB2 to output low when appropriate conditions exist, while code 101 causes the signal to pulse. For I/O Port A, "appropriate conditions" consist of the CPU reading or writing, while selecting I/O Port A via the select code 0001_2. For I/O Port B, "appropriate conditions" consist of the CPU writing, but not reading, accessing I/O Port B via the select code 0000_2 — the only select code available for I/O Port B.

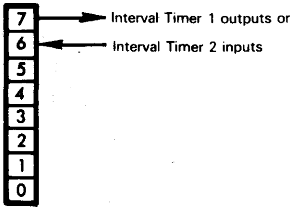
I/O Port B also has a simpler interface with the CPU Data Bus. Rather than having separate output buffer and input latches, there is a single output buffer, which is accessed by the CPU when reading from, or writing to I/O Port B. Coupled with the different pin configuration, which we have already described for I/O Port B, you can guarantee that bit levels written to I/O Port B output pins will subsequently be read back accurately.

The more limited capabilities of I/O Port B reflect the fact that pins 7 and 6 of this I/O port may be used by Interval Timer logic. Thus, the MCS6522 will frequently be configured with I/O Port A providing parallel I/O, while I/O Port B provides various types of control dialogue.

MCS6522 INTERVAL TIMER LOGIC

The most important point to note regarding the additional functions associated with I/O Port B is that they have logical priority over simple data transfers; what this means is that the Interval Timers and Shift register may, under some circumstances, use I/O Port B pins, control signals and interrupt logic. When Interval Timer or Shift register requirements are in conflict with simple data transfer, then Interval Timer or Shift register requirements will prevail.

Let us look at a specific example; **pins of I/O Port B are used by Interval Timer logic as follows:**



Suppose you have identified I/O Port B pin 7, via the Data Direction register, as an input pin; Interval Timer 1 uses this pin to output pulses or square waves and will override the Data Direction register.

It is a good idea not to use I/O Port B for parallel data transfer while you are using Interval Timer or Shift register logic. Also, exercise caution when using both Interval Timers, or when using the Serial Shift register in conjunction with Interval Timers.

Table 9-5. Summary Of I/O Port A Handshaking Control Signals

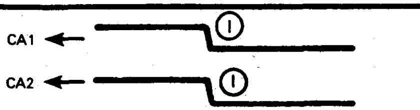
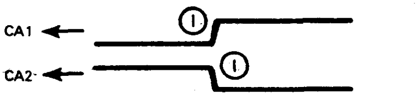
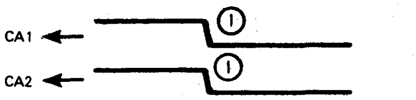
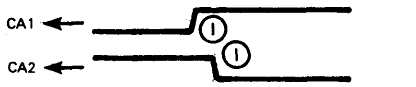
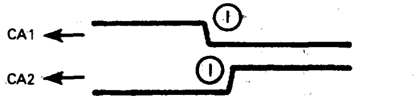
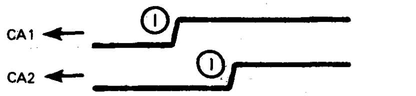
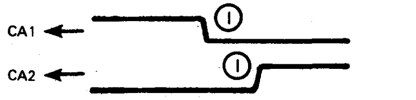
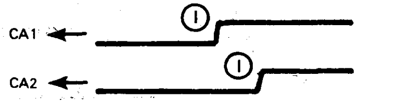
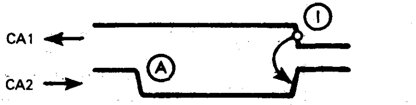
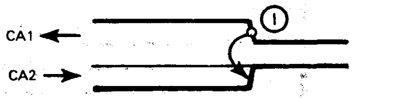
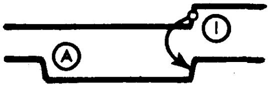
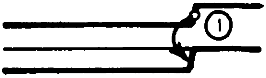
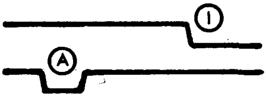
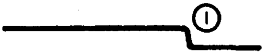
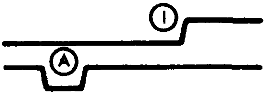
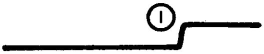
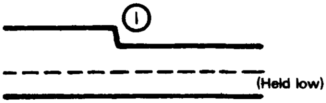
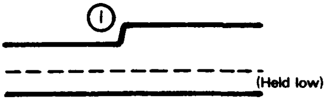
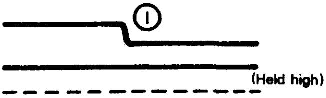
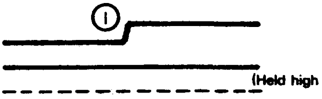
I/O Port A Select Code (Binary)	Peripheral Control Register Bits 3 2 1 0	CONTROL SIGNALS	Interrupt Reset
0001 or 1111	0 0 0 0		On 0001 select code access or programmed reset
0001 or 1111	0 0 0 1		On 0001 select code access or programmed reset
0001 or 1111	0 0 1 0		Programmed reset only
0001 or 1111	0 0 1 1		Programmed reset only
0001 or 1111	0 1 0 0		On 0001 select code access or programmed reset
0001 or 1111	0 1 0 1		On 0001 select code access or programmed reset
0001 or 1111	0 1 1 0		Programmed reset only
0001 or 1111	0 1 1 1		Programmed reset only
0001	1 0 0 0		At Ⓐ or programmed reset
1111	1 0 0 0		Programmed reset only

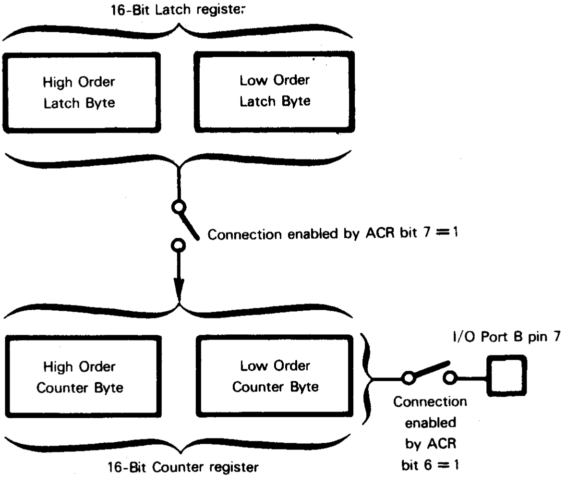
Table 9-5. Summary Of I/O Port A Handshaking Control Signals (Continued)

I/O Port A Select Code (Binary)	Peripheral Control Register Bits 3 2 1 0	CONTROL SIGNALS	Interrupt Reset
0001	1 0 0 1	CA1 ← CA2 → 	At Ⓐ or programmed reset
1111	1 0 0 1	CA1 ← CA2 → 	Programmed reset only
0001	1 0 1 0	CA1 ← CA2 → 	At Ⓐ or programmed reset
1111	1 0 1 0	CA1 ←  CA2 → unaffected	Programmed reset only
0001	1 0 1 1	CA1 ← CA2 → 	At Ⓐ or programmed reset
1111	1 0 1 1	CA1 ←  CA2 → unaffected	Programmed reset only
0001 or 1111	1 1 0 0	CA1 ← CA2 → 	On 0001 select code access or programmed reset
0001 or 1111	1 1 0 1	CA1 ← CA2 → 	On 0001 select code access or programmed reset
0001 or 1111	1 1 1 0	CA1 ← CA2 → 	On 0001 select code access or programmed reset
0001 or 1111	1 1 1 1	CA1 ← CA2 → 	On 0001 select code access or programmed reset

Ⓘ Interrupt request   Ⓐ CPU access

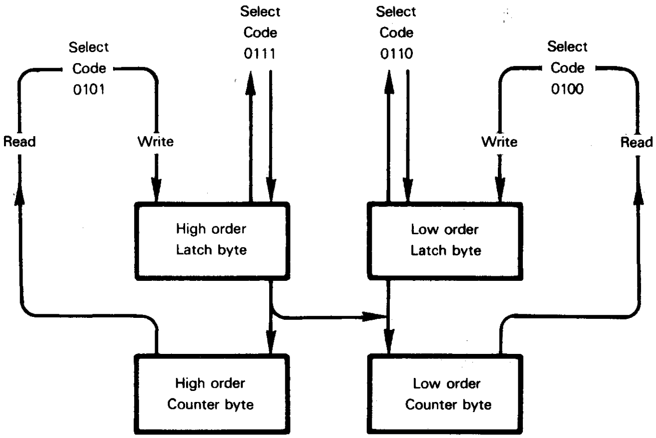
Let us first examine Interval Timer 1. This is the more versatile of the two MCS6522 Interval Timers; it is most easily understood if visualized as follows:

MCS6522 INTERVAL TIMER 1



You select from among its many functions by appropriately loading bits 7 and 6 of the Auxiliary Control register (ACR).

Interval Timer 1 addressing via select codes may be illustrated as follows:



Select codes 0110_2 and 0111_2 are quite straightforward. The former accesses the low order Latch byte to read or write; the latter accesses the high order Latch byte to read or write.

Select codes 0100_2 and 0101_2 are not so straightforward. If you access the MCS6522 PIA with select code 0100_2, you will write into the lower order Latch byte, but you will read the contents of the iow order Counter byte.

If you access the MCS6522 PIA with select code 0101_2, you will read the contents of the high order Counter byte; but upon writing, you will access the high order Latch byte and the high order Counter byte, while simultaneously transferring the low order Latch byte contents to the low order Counter byte. This allows a clean method of loading 16 bits of data into the Counter byte following the execution of a single instruction.

Writing to select code 0101_2 will also initiate a new Timer interval.

The two Counter registers constitute a 16-bit entity which is decremented on the trailing edges of the Φ2 clock pulse. The initial value loaded into the Counter registers identifies the interval of the Counter. An active time-out of the Counter is marked by an interrupt request.

If the Counter is connected to pin 7 of I/O Port B, then an active time-out will also cause the signal output at pin 7 of I/O Port B to invert or pulse low, depending on the mode in which the Interval Timer is operating.

A 1 in bit 6 of the Auxiliary Control register will connect Counter logic to pin 7 of I/O Port B. A 0 in bit 6 of the Auxiliary Control register disconnects Counter logic from pin 7.

Via bit 7 of the Auxiliary Control register, you can connect or disconnect Counter and Latch logic. A 0 in bit 7 of the Auxiliary Control register is a disconnect, whereas a 1 is a connect.

Referring to Figure 9-14, "One-Shot Mode" refers to disconnected Latch and Counter logic, while "Free Running Mode" refers to connected Latch and Counter logic.

If Counter logic is disconnected from the Latch registers, then following Counter initiation there will be one active time-out, after which the Counter will continuously redecrement from 0000_{16}, through $FFFF_{16}$, and back to 0000_{16}. Subsequent counts are inactive — which means that no interrupt will be requested, and if connected to pin 7 of I/O Port B, no signal changes will be output.

If Counter logic is connected to the two Latch registers, then every time the Counter times out, it is immediately reloaded with the contents of the Latch registers — and begins another active time out. Under these circumstances, every Counter time out is active — and will be marked by an interrupt request, plus a signal level change at pin 7 of I/O Port B, if this pin is connected to Counter logic.

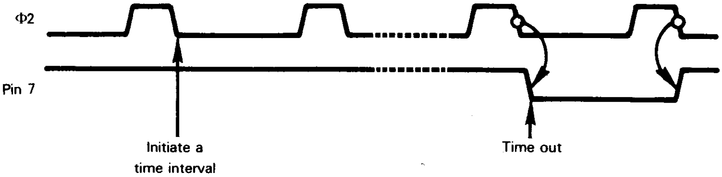
While the Interval Timer 1 options may appear complicated, in fact they are very simple.

To you, as a programmer, there is only one option that you must define when using Interval Timer 1 of the MCS6522: do you want the Interval Timer to operate in one-shot or free running mode?

MCS6522 INTERVAL TIMER 1 ONE-SHOT MODE

Let us first consider One-Shot Mode, which is selected by having a 0 in bit 7 of the Auxiliary Control register.

Recall that in One-Shot Mode the Counter is disconnected from the Latch registers. For practical reasons, however, this disconnection is not complete; you have to initiate a time out by loading an initial value into the high order and low order Counter bytes; but the Counter is continuously running. Were you to load the low order byte, and then the high order byte to the Counter register, problems could arise, because the low order byte would start decrementing before you had completed loading the high order byte. To resolve this problem, you initially load the low order Counter register byte value into the low order Latch register byte; then you directly load the high order Counter register byte. You do this by writing into the memory addresses associated with select codes 0100_2 and 0101_2. When you write into select code 0100_2, you load the low order byte of the initial Counter value into the low order Latch register byte. When you write into select code 0101_2, you load the high order Latch register byte, but immediately the 16 Latch register bits are loaded into the Counter, which starts decrementing. As soon as the Counter times out, an interrupt is requested; and if, via Auxiliary Control register bit 6, you have connected I/O port pin 7 to the Counter, then a low pulse will be output via pin 7. The low pulse will have a width of one Φ2 clock period:



Note that when using an MCS6522, the onus is upon you to make sure that all programmable signal levels are at their correct level. In the illustration above, Φ2 is not a programmable signal, so you can ignore it. The pin 7 level is programmable; it is up to you to make sure that a high level is being output at pin 7, or else a low pulse will not occur.

What we are saying is that Interval Timer 1 logic will not insure that pin 7 is normally outputting a high level. You must first define pin 7 as an output by writing a 0 into bit 7 of the I/O Port B Data Direction register. Then you must output a 1 to bit 7 of I/O Port B. Having thus established a continuous high level being output at pin 7, you can be sure of a low pulse marking an active time out.

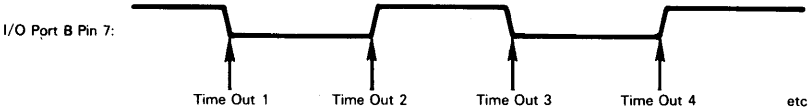
Following a time out in the One-Shot Mode, the Counter decrements continuously via $FFFF_{16}$ to 0000_{16}. On subsequent time outs no interrupt request occurs and no low pulse is output via pin 7 of I/O Port B.

MCS6522 INTERVAL TIMER 1 FREE RUNNING MODE

If you have specified the free running mode by loading 1 into bit 7 of the Auxiliary Control register, then as soon as the Counter times out, Latch register contents are immediately transferred to the Counter register, which again decrements to an active time out. Thus a sequence of interrupt requests, with optional signal output via pin 7 of I/O Port B will occur — but there are some differences.

When using Interval Timer 1 in free running mode, you initialize exactly as you do for the one-shot mode; you load the low order and high order Counter bytes via select codes 0100_2 and 0101_2. As soon as you write into select code 0101_2, the Latch contents are transferred to the Counter, which starts decrementing. While the Counter is decrementing you can reset the next Counter initial value by writing into the Latch register using select codes 0110_2 and 0111_2. Now as soon as the Counter times out, the new value you have loaded into the Latch register becomes the next initial Counter value.

If you have connected I/O Port B pin 7 to the Counter by storing 1 in Auxiliary Control register bit 6, then each time the Counter times out, the signal output via pin 1 of I/O Port B is inverted, generating a square wave; this may be illustrated as follows:



Remember, you can, at any time, read the contents of Interval Timer 1 Counter or Latch registers. This gives you a complete ability to test and modify Timer intervals in any way, under program control, while Interval Timer 1 is operating.

Now consider Interval Timer 2.

MCS6522 INTERVAL TIMER 2

MCS6522 Interval Timer 2 has logic which is markedly different from Interval Timer 1, which we have just described. Interval Timer 2 offers two modes of operation:

1) One-shot mode with no signal output.

2) Pulse counting mode.

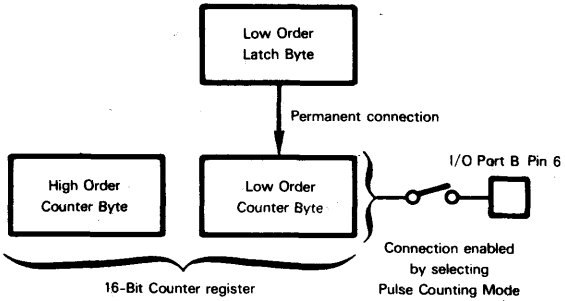
You select one of the two Interval Timer 2 options by appropriately setting bit 5 of the Auxiliary Control register, as illustrated in Figure 9-14.

One-shot mode, with no signal output, is identical in operation to one-shot mode with no signal output, as described for Interval Timer 1.

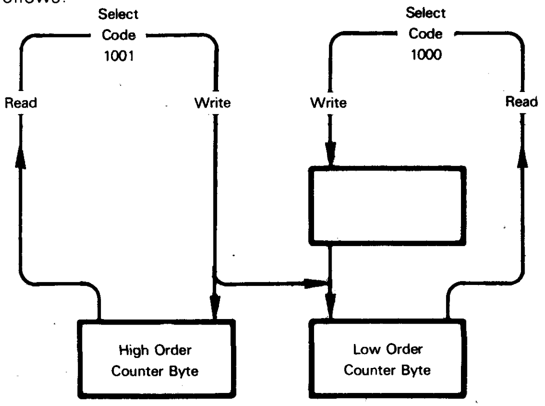
Pulse counting mode is an alternative one-shot mode; the Interval Timer 2 Counter decrements on high-to-low transitions of signal input via pin 6 of I/O Port B. Thus, in the pulse count mode, Interval Timer 2 will count out after the number of high-to-low transitions specified by the initial Counter value. For example, if you initially load 2000_{16} into the Interval Timer 2 Counter, then after 8192 high-to-low transitions of the signal input via pin 6, an active time out will occur.

Following an active time out, an interrupt is requested. Subsequently, Interval Timer 2 continues to decrement continuously from 0000_{16} through $FFFF_{16}$ and back to 0000_{16}; on subsequent time outs however, no interrupt request is generated. Subsequent time outs are passive.

Since the logic capabilities of Interval Timer 2 differ from Interval Timer 1, as we might expect, the register organization and addressing logic associated with Interval Timer 2 also differs. It may be illustrated as follows:



Interval Timer 2 is accessed via two select codes, 1000_2 and 1001_2; addressing may be illustrated as follows:



Since Interval Timer 2 has no free running option, there is no need for a high order Latch register byte; the sole purpose of such a location is to store a high order Counter byte, waiting to be loaded into the Counter register when it times out. You do need a low order Latch register byte,, because when loading the Counter register, you still have to make two accesses. You cannot load the low order Counter byte, and then load the high order Counter byte; the Counter is continuously decrementing and would start decrementing the low order Counter byte while you were loading the high order Counter byte.

The initiation procedure for Interval Timer 2, whether you are in one-shot mode or pulse counting mode, is to write the low order Counter byte to select code 1000_2, then the high order Counter byte to select code 1001_2. As soon as you write the high order Counter byte to select code 1001_2, Interval Timer 2 logic transfers the contents of the low order Latch byte to the low order Counter byte — and initiates decrementing.

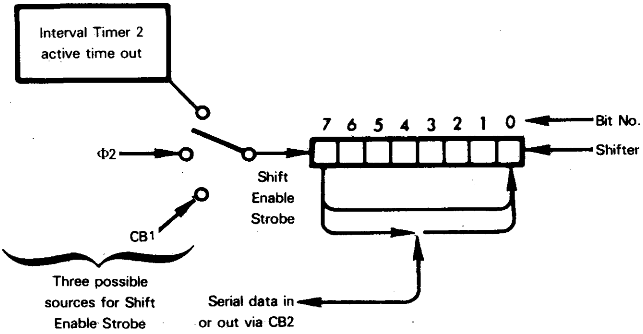
If you are in one-shot mode, the Counter register is decremented on each high-to-low transition of the Φ2 clock pulse.

If you are in pulse counting mode, the Counter decrements on each high-to-low transition of a signal input via pin 6 of I/O Port B.

That is the only difference between the two modes.

MCS6522 SHIFTER LOGIC

MCS6522 Shifter logic may be illustrated as follows:



As illustrated above, serial data may be shifted into bit 0 or out of the Shift register bit 7. Serial data is transferred via control signal CB2.

When you shift into bit 0 the data transfer is accompanied by a one-bit left shift of the Shifter contents. When you shift out of bit 7, the data transfer is accompanied by a one-bit left rotate of the Shifter contents.

Every serial bit data transfer is enabled by a strobe signal. The strobe may be derived from:

1) A signal input by external logic via CB1.
2) The Φ2 clock signal.
3) Interval Timer 2 active time-outs.

If the enable strobe is derived from external logic via CB1 or from Φ2, then the high-to-low transition of either signal triggers the enable strobe.

If the shift enable strobe is derived from Interval Timer 2, then only the low order eight Counter bits for Interval Timer 2 are decremented.

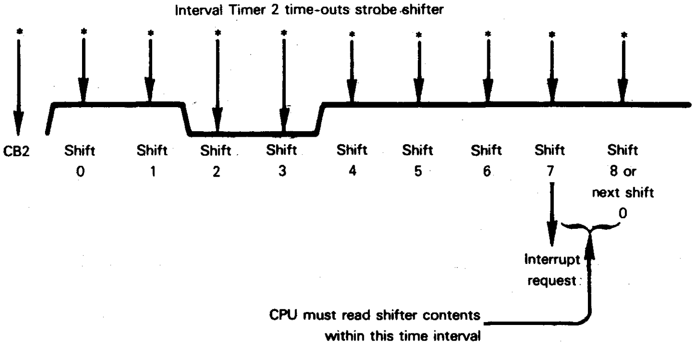
There are seven modes in which the Shifter can be operated; three are input modes and four are output modes. You select an appropriate mode by the code loaded into bits 5, 4 and 3 of the Auxiliary Control register. Let us examine the response of Shifter logic to the eight possible Auxiliary Control register bit combinations.

Mode 000; disable Shift register. When Auxiliary Control register bits 5, 4 and 3 are 000, the Shift register is disabled. Control signals CB1 and CB2 respond as defined by bits 7, 6 and 5 of the Peripheral Control register. While the Shift register is disabled, the CPU can still write into it and read from it; you, as a programmer, can therefore use it as a storage location for a single data byte.

Mode 001; input under Interval Timer 2 strobe. Auxiliary Control register bits 5, 4 and 3 set to 001 specify serial data shifted in, as timed by Interval Timer 2. However, only the low order byte of Interval Timer 2 is active, which means that 256 is the maximum initial Interval Timer 2 count which can be used. A low pulse with a width of one Φ2 clock is output via CB1 on each Interval Timer 2 time-out, as a signal that external logic must provide the next serial data bit to be input. Interrupts are generated, as usual, following each time-out; an additional interrupt is generated after eight bits in the Shift register have been serially output.

When Interval Timer 2 is being used to strobe the Shift register in Mode 001, then it operates in a unique mode which is not available at any other time.

Whenever Interval Timer 2 times-out, the contents of the low order Latch byte are immediately transferred to the low order Counter byte — and decrementing resumes. Thus, Interval Timer 2 is operating in a free-running mode, with only the low order Counter byte active. As this would imply, you must initiate Interval Timer 2 by loading the appropriate initial count into the low order Timer 2 Latch byte — before enabling the Shift register in Mode 001. Following a time-out you can, of course, reload the Interval Timer 2 low order Latch byte to modify the next time interval. Timing may be illustrated as follows:

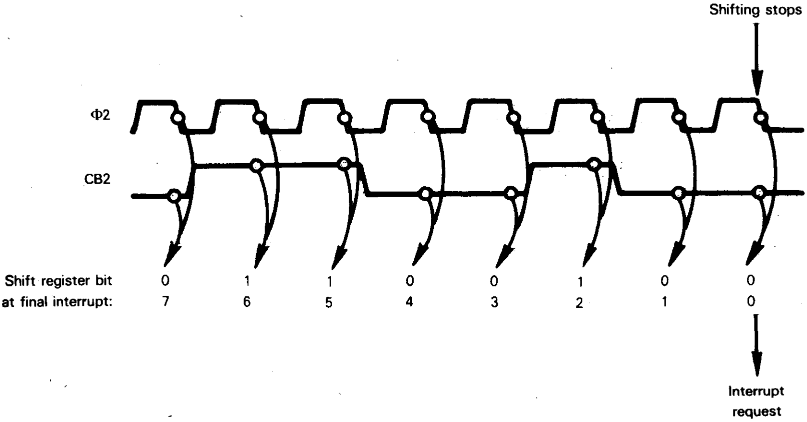


Note that it is your responsibility as a programmer to ensure that all logic needed by the Shifter has been appropriately set for operations illustrated above. This means that you must program Interval Timer 2 to redecrement following each time-out by writing a 0 into select code 1001_2, the high order Timer 2 Counter byte.

Since control signals CB1 and CB2 are being used by the Shift register in this mode of operation, Shift register requirements will override any CB1 and CB2 control signal specifications that have been made via bits 7, 6, 5, and 4 of the Peripheral Control register.

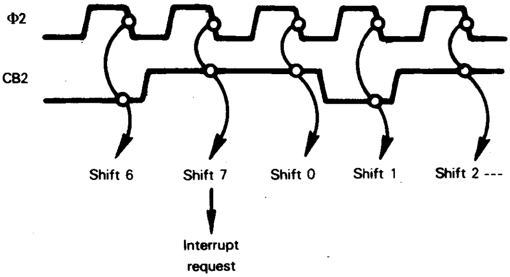
Mode 010; input under Φ2 clock strobe. This mode is specified by 010 in bits 5, 4 and 3 of the Auxiliary Control register.

In Mode 010, and in all other Shift register modes that are clocked by Φ2, shifting stops on the eighth shift — which is marked by an interrupt request. Timing may be illustrated as follows:



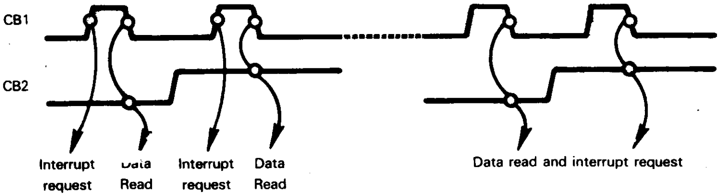
Mode 011; input under external pulse strobe. This mode is specified by 011 in bits 5, 4 and 3 of the Auxiliary Control register. This mode is equivalent to the standard serial input found in most serial I/O devices, where external logic provides the clocking signal which is used to time in serial data. In this case, external logic provides a clocking signal via CB1; a high-to-low transition of CB1 is interpreted by the Shift register as a strobe to input the next serial data bit from CB2.

Timing may be illustrated as follows:



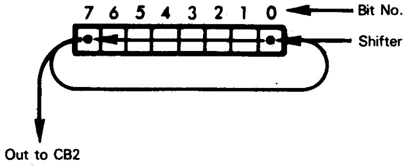
As was the case with Mode 001, shifting is continuous. So far as external logic is concerned it is shifting in an endless stream of serial data bits. Shifter logic generates an interrupt request every eighth shift so that the CPU will know when to read the contents of the Shifter. The CPU has the time interval between a Shifter interrupt and the next high-to-low transition of CB1 within which to read Shifter register contents. If the CPU does not read Shifter register contents in this time interval then an error will occur but no error status will be reported.

Shift register use of control signals CB1 and CB2 overrides specifications made for these signals via bits 7, 6, 5 and 4 of the Peripheral Control register; however, the policy of overriding adopted by the designers of the MCS6522 is somewhat subtle. Since control signal CB2 is used as a serial data input signal, any specifications made for this signal via the Peripheral Control register are totally ignored. Specifications made for control signal CB1, however, remain. If you have enabled I/O Port B via bit 1 of the Auxiliary Control register, then the active transition for control signal CB1 which is specified by bit 4 of the Peripheral Control register will apply. Thus you will generate an interrupt whenever CB1 makes an active transition in the process of clocking in serial data. The two possibilities may be illustrated as follows:



You can disable interrupts occurring as a result of active CB1 transitions via the Interrupt Enable register, which we have yet to describe.

Let us now look at the output modes of the Shift register. In all output modes, the Shift register transfers the contents of bit 7 to control signal CB2. Simultaneously, bit 7 contents are shifted back into bit 0. This may be illustrated as follows:



Depending upon the serial output option you choose, CB1 may or may not be used as a companion control signal.

Mode 100; free-running output under Interval Timer 2 strobe. This mode is selected via 100 in bits 5, 4 and 3 of the Auxiliary Control register. Data is shifted out of Shift register bit 7, clocked by Interval Timer 2, as described for input mode 001. Data shifted out appears on CB2. Shifting is continuous, which means that the bit pattern in the Shift register will output endlessly.

Mode 101; output under Interval Timer 2 strobe. This mode is specified by 101 in bits 5, 4 and 3 of the Auxiliary Control register. It differs from Mode 100, which we have just described, in that once eight bits have been shifted out of the Shifter, an interrupt is requested and shifting halts.

You can output continuously under Mode 101 by making appropriate use of Shift register interrupts and Interval Timer 2. The Shift register interrupt occurs on the eighth shift out of the Shifter; but within the time it takes for Interval Timer 2 to again time-out, you can reload the Shifter. If you reload the Shifter during this time interval, then on the next time-out of Interval Timer 2, shifting will begin again, and thus become an uninterrupted bit stream on signal CB2.

Mode 110; shift out under Φ2 pulse. This mode is selected via 110 in bits 5, 4 and 3 of the Auxiliary Control register. In this mode eight bits are shifted out of the Shift register, clocked by Φ2. Then shifting ceases.

These are the steps you must adopt when using the Shifter in Mode 110:

1) Disable the Shifter by loading 000 into bits 5, 4 and 3 of the Auxiliary Control register.
2) Load a byte of data into the Shifter. Remember the data you load will be shifted high order bit first.
3) Enable the Shifter by loading 110 into bits 5, 4 and 3 of the Auxiliary Control register.
4) Again disable the Shifter by loading 000 into bits 5, 4 and 3 of the Auxiliary Control register.

In Mode 110, data will be shifted out on every high-to-low transition of the Φ2 clock pulse. Thus the entire shift operation will be completed in eight clock pulses.

Mode 111; shift out under external pulse strobe. This mode is identical to Mode 101, except that instead of output being timed by Interval Timer 2, external logic provides the output timing pulse via control signal CB1. As was the case for input mode 011, the high-to-low transition of the external timing signal input via CB1 causes serial data to be shifteo out of the Shift register. Once again, unless you have disabled CB1 interrupts via the Interrupt Enable register, the condition of bit 4 in the Peripheral Control register will cause the interrupts to be requested each time control signal CB1 makes a high-to-low or a low-to-high transition.

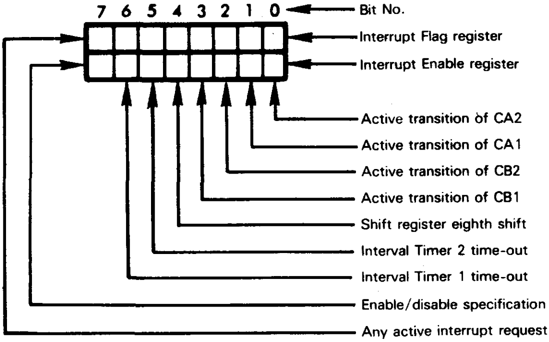
MCS6522 INTERRUPT LOGIC

Interrupt logic is one of the first things you must initialize when starting to use an MCS6522. It is the last subject we describe, because in order to understand MCS6522 interrupts, you must first be aware of the numerous ways in which interrupt requests may originate within this device.

There are two addressable locations within the MCS6522 dedicated to interrupt logic:

1) The Interrupt Flag register, selected by 1101_2.
2) The Interrupt Enable register, selected by 1110_2.

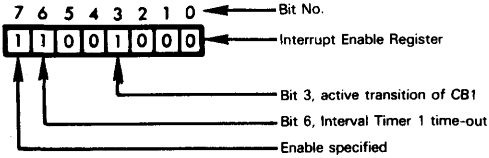
These two registers have individual bits assigned to the different interrupt requesting sources as follows:



7 6 5 4 3 2 1 0
Bit No.
Interrupt Flag register
Interrupt Enable register
Active transition of CA2
Active transition of CA1
Active transition of CB2
Active transition of CB1
Shift register eighth shift
Interval Timer 2 time-out
Interval Timer 1 time-out
Enable/disable specification
Any active interrupt request

The Interrupt Flag register identifies those interrupts which are active. A 1 in any bit position indicates an active interrupt, whereas a 0 indicates an inactive interrupt.

You can selectively enable or disable individual interrupts via the Interrupt Enable register. You enable individual interrupts by writing to the Interrupt Enable register with a 1 in bit 7. Thus you could enable "time-out for Timer 1" and "active transitions of signal CB1" by outputting $C8_{16}$ to the Interrupt Enable register:



7 6 5 4 3 2 1 0
Bit No.
1 1 0 0 1 0 0 0
Interrupt Enable Register
Bit 3, active transition of CB1
Bit 6, Interval Timer 1 time-out
Enable specified

You selectively disable interrupts by writing to the Interrupt Enable register with bit 7 set to 0. Thus you would disable time-outs from Timer 1 and active transitions of signal CB1 by outputting 48_{16} to the Interrupt Enable register.

If an active interrupt exists in the Interrupt Flag register for an interrupt which has been enabled via the Interrupt Enable register, then bit 7 of the Interrupt Flag register will be set — and an interrupt request will be passed on to the CPU by setting IRQ low. The interrupt service routine executed in response to an interrupt request from the MCS6522 must read the contents of the Interrupt Flag register in order to determine the source of the interrupt, and thus the manner in which the interrupt must be serviced.

You can clear any bit in the Interrupt Flag register, except bit 7, by writing a 1 to that bit. Writing a 0 to a bit has no effect. Thus, if interrupt requests were being made from time-out of Timer 1 and an active transition on CA1:

7	6	5	4	3	2	1	0	← Bit No.
1	1	0	0	0	0	1	0	← Interrupt Flag register

Writing either 82_{16} or 02_{16} to select code 1101_2 would clear the interrupt due to an active transition on CA1 (bit 1); however, bits 7 and 6 would remain set.

There are a number of ways in which interrupt requests are automatically cleared, and the corresponding Interrupt Flag register bits get reset. These are summarized in Table 9-6.

Table 9-6. A Summary Of MCS6522 Interrupt Setting And Resetting

	SET	CLEARED BY
6	Time-out of Timer 1	Reading Timer 1 Low Order Counter or writing T1 High Order Latch
5	Time-out of Timer 2	Reading Timer 2 Low Order Counter or writing T2 High Order Counter
4	Completion of eight shifts	Reading or writing the Shift register
3	Active transition of the signal on CB1	Reading from or writing to I/O Port B
2	Active transition of the signal on CB2 (input mode)	Reading from or writing to I/O Port B in input mode only
1	Active transition of the signal on CA1	Reading from or writing to I/O Port A using address 0001_1
0	Active transition of the Signal on CA2 (input mode)	Reading from or writing to I/O Port A Output register (ORA) using address 0001_1 in input mode only

THE MCS6530 MULTIFUNCTION SUPPORT LOGIC DEVICE

This is a device which appears to have been designed by MOS Technology as an answer to the F8, which is summarized in Chapter 2. Recall that the F8 3850 CPU and 3851 PSU combine to provide a CPU, 1K bytes of ROM, 64 bytes of RAM, four 8-bit I/O ports, a programmable Interval Timer and an interrupt. This is a formidable amount of logic to obtain with just two devices.

In order to compete in low-end, high volume, price sensitive markets, MOS Technology came up with the MCS6530 which provides 1K bytes of ROM, 64 bytes of RAM, two I/O ports, a Programmable Interval Timer and interrupt logic. The realities of the MCS6530 are such that if you use the Interval Timer and interrupt logic, one of the I/O ports is only partially functional; nevertheless, an MCS6530 multifunction support device, together with an MCS6500 series CPU can compete with a two-device F8 system more effectively than most 8-bit microprocessors described in this book.

Specifically comparing a two-chip F8 with a two-chip MCS6500 microcomputer system, we conclude that the F8 offers more logic — specifically three more I/O ports, while the MCS6500 offers a more powerful instruction set — thus more effective ROM utilization.

But the introduction of a single-chip F8 equivalent of the 3850 CPU and 3851 PSU makes the comparative analysis moot.

If we look at the MCS6530 simply as a member of the MCS6500 microcomputer family of devices, it is best visualized as a memory device which, in addition, provides a significant subset of the MCS6522 logic capabilities. The MCS6530 may be compared with the Fairchild 3851 PSU, described in Chapter 2.

Figure 9-16 illustrates that part of our general purpose microcomputer logic which has been implemented on the MCS6530 multifunction logic device. Figure 9-16 also applies to the MCS6532, which we will describe next.

The MCS6530 is packaged as a 40-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL compatible. I/O Port A and B pins are also CMOS compatible. PA0 and PB0 may be used as a power source to directly drive the base of a transistor switch.

The devices are implemented using N-channel silicon gate MOS technology.

Figure 9-17 illustrates the logic provided by an MCS6530 multifunction logic device.

THE MCS6530 MULTIFUNCTION DEVICE PINS AND SIGNALS

The MCS6530 multifunction device pins and signals are illustrated in Figure 9-18.

These signals are identical to signals with the same names which we have already described for the MCS6522:

D0 - D7	the bidirectional Data Bus
$\Phi 2$	the system clock input
$\text{R}/\overline{\text{W}}$	the Read/Write control output by the CPU
$\overline{\text{RESET}}$	which is a standard reset input

I/O port pins PA0 - PA7 and PB0 - PB7 are functionally similar to equivalent I/O port pins of the MCS6522, but there are some differences.

Pin 17 may be specified, when you order the MCS6530, **as $\overline{\text{IRQ}}$ only, PB7 only, or as the programmable dual function pin $\overline{\text{IRQ}}$/PB7.**

Electrical characteristics of all 16 MCS6530 I/O port pins are equivalent to MCS6522 I/O Port B pins, rather than I/O Port A pins.

MCS6530 pins 18 and 19 may implement I/O Port B pins PB6 and PB5, or they may serve as chip select pins. Note carefully that these are not programmable dual function pins. Each pin will either have one function or the other; and when ordering the part, you must indicate which function the pin is to serve. Pins 18 and 19 are logically independent, and the function assigned to one in no way restricts the choices available to you when assigning functions to the other pins.

If pins 18 and/or 19 have been assigned to chip select logic, then they contribute to device addressing in a unique way.

MCS6530 ADDRESSING LOGIC

The MCS6530 has ten address lines, A0 - A9; this is sufficient to address 1024 bytes of ROM. In addition, the MCS6530 has 64 bytes of RAM plus assorted I/O and Interval Timer logic which needs to be addressed. RS0, CS1 and CS2 are used to discriminate between ROM addresses, RAM addresses and additional logic addresses. But there is

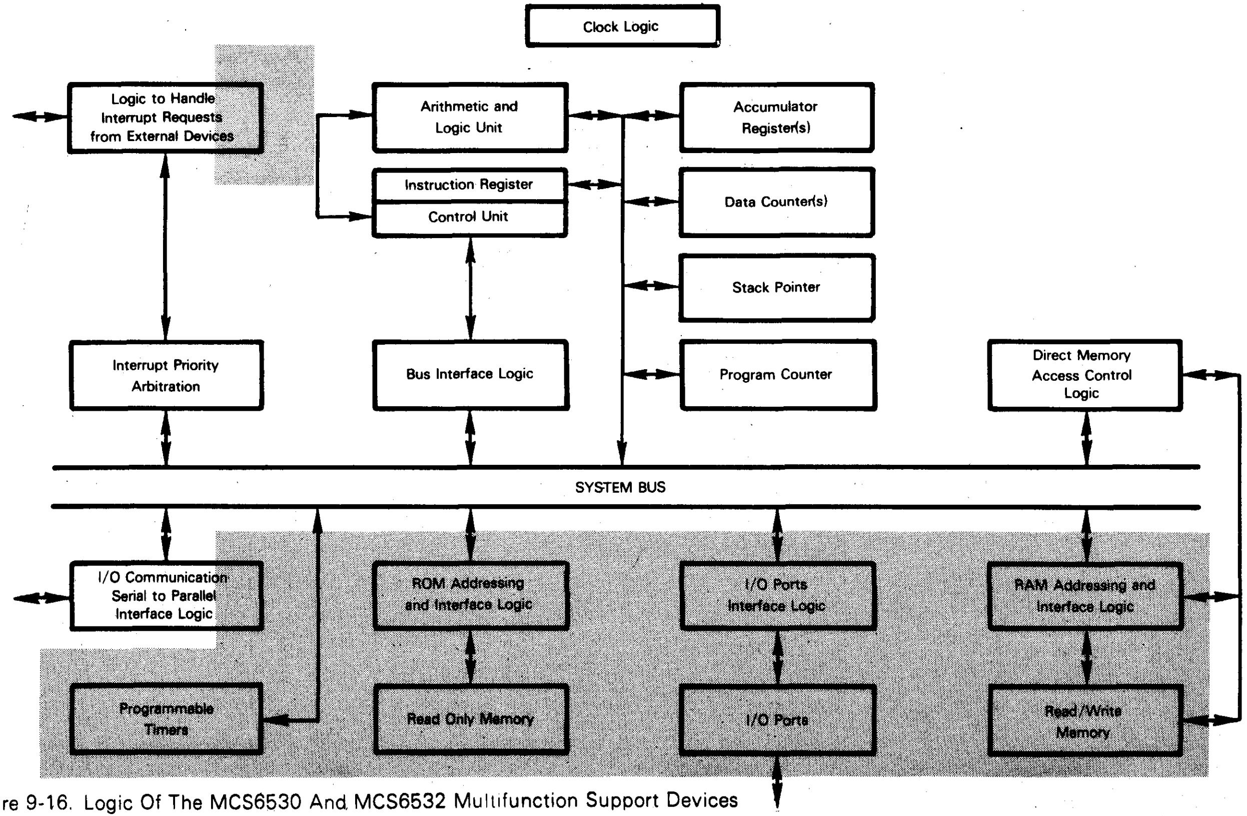

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Instruction Register
Control Unit
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 9-16. Logic Of The MCS6530 And MCS6532 Multifunction Support Devices

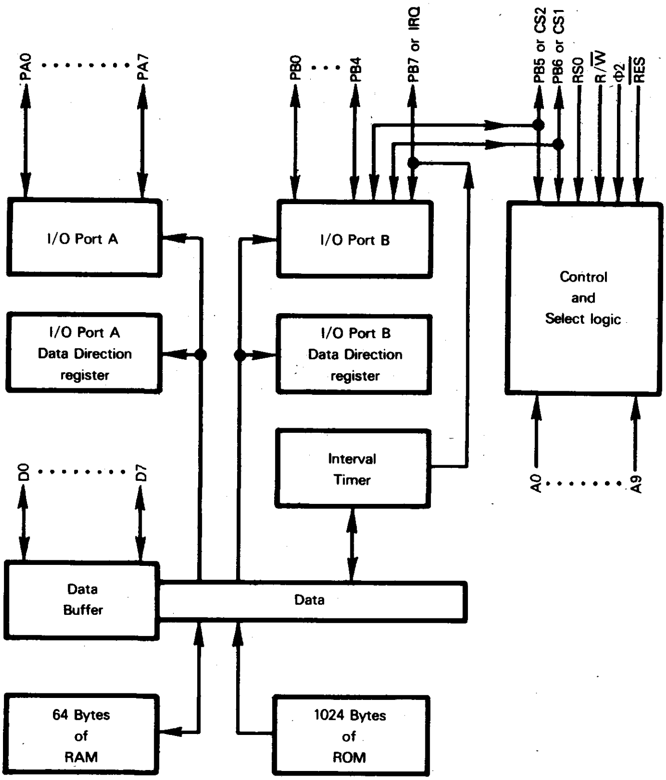


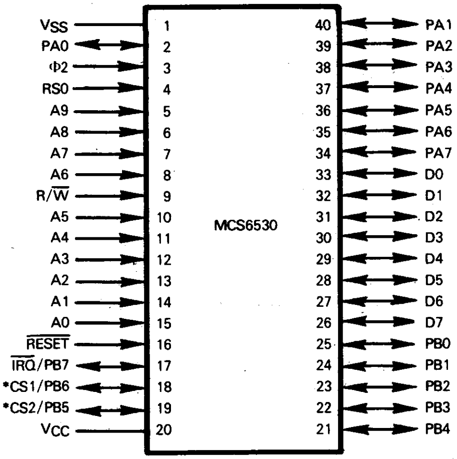
Figure 9-17. Logic Provided By The MCS6530 Multifunction Device

no predefined way in which the different addressable locations of the MCS6530 will be accessed — which is only to be expected since CS1 and CS2 are not permanent features of every MCS6530 device. **When RS0 is high, ROM will always be selected. When RS0 is low, RAM or additional logic may be accessed** — and the way in which the access works is entirely up to you.

RAM and additional logic each have an internal master select; and what you specify is the way in which these master selects will be derived. As you will see upon examining Table 9-7, master selects for RAM and additional logic each will consist of the following:

1) RS0 set to 0.
2) Address lines A4 - A9 with specific values which you define.
3) CS1 and CS2, if implemented, with specific values which you define.

As seen by a programmer, the address space of an MCS6530 can be divided in many flexible ways.

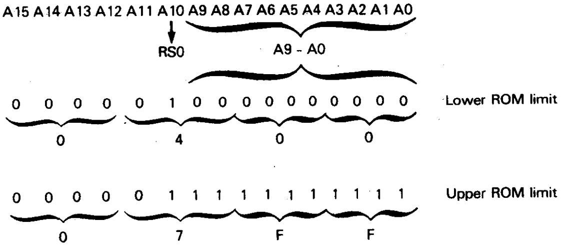


Pin Name	Description	Type
D0 - D7	Data Bus to CPU	Tristate, bidirectional
Φ2	System Clock	Input
R/$\overline{\text{W}}$	Read/Write control	Input
$\overline{\text{RESET}}$	Reset	Input
PA0 - PA7	Port A Peripheral Data Bus	Tristate, Input or Output
PB0 - PB7	Port B Peripheral Data Bus	Tristate, Input or Output
$\overline{\text{IRQ}}$	Interrupt from Interval Timer; special function of input pin PB7	Input
CS1, CS2	Chip Select	Input
A0 - A9	Address lines	Input
RS0	ROM Select	Input
VCC, VSS	Power and Ground	

*Mutually exclusive functions. One or the other must be specified when the chip is ordered.

Figure 9-18. MCS6530 Multifunction Device Signals And Pin Assignments

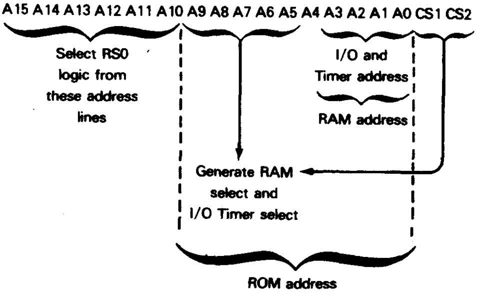
Usually RS0 will be connected to a high order address line; let us assume it is A10, so that we can develop real examples. Now ROM will be accessed by addresses in the range 0400_{16} through $07FF_{16}$:



RAM may respond to any 64 contiguous addresses in the range 0000_{16} through $03FF_{16}$.

Similarly, I/O and timer logic will be selected by 16 contiguous memory addresses in the same address space.

In summary, we may illustrate addressing and select options as follows:


A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0 CS1 CS2
Select RS0 logic from these address lines
I/O and Timer address
RAM address
Generate RAM select and I/O Timer select
ROM address

There are a number of aspects to MCS6530 addressing which need clarification.

First of all, you may well ask why pins 18 and 19 can optionally be assigned as additional chip select inputs. After all, with RS0 low, you have more than enough address lines to access RAM plus I/O and timer logic. The purpose of having CS1 and CS2, as additional chip selects, is to allow a number of MCS6530 devices to interface with a single CPU — without requiring complex device select logic. If the additional chip select signals CS1 and CS2 are not available, you can still have more than one MCS6530 connected to a CPU, but additional support logic must selectively suppress Φ2 for all but one MCS6530 device. Remember, RS0, R/$\overline{W}$ and the Address Bus are all signals with two active and no passive states. These signals are always selecting some MCS6530 location.

Since the whole purpose of the MCS6530 is to support very low cost, simple microcomputer configurations, the ability to minimize device select logic becomes very important.

Observe that address logic is used not only to access individual addressable locations within the MCS6530, but also to perform certain programming functions. We will describe these programming functions in greater detail later. It is interesting to note that both the MC6800 and MCS6500 microcomputer devices use address logic to provide control functions in support devices. In contrast, 8080A devices will be very spartan when it comes to device addressing, frequently having two I/O or memory addresses to access numerous different locations — with complex sequencing schemes determining how locations will be accessed.

MCS6530 PARALLEL DATA TRANSFER OPERATIONS

Parallel data transfer operations, when using the MCS6530 are exactly as described for the MCS6522 I/O Port B.

Each I/O port of the MCS6530 has a Data Direction register. Into this register you load a mask which has a 1 in every bit position corresponding to an output I/O port pin and a 0 corresponding to an input I/O port pin. Subsequently the CPU reads and writes data by accessing the assigned I/O port address.

MCS6530 INTERVAL TIMER AND INTERRUPT LOGIC

MCS6530 Interval Timer logic differs significantly from MCS6522 logic. The MCS6530 Interval Timer is a single 8-bit register which can be loaded with any initial value. The initial value decrements on high-to-low transitions of the Φ2 clock pulse, or multiples of the Φ2 clock pulse; and on decrementing to 0, an interrupt request is generated. Thus the largest time interval is generated by loading 0 into the Interval Timer register.

Table 9-7. Addressing The MCS6530 Multifunction Support Logic Device

<table>
<tr><th colspan="3">PRIMARY SELECT</th><th colspan="5" rowspan="2">ACCESSED LOCATIONS</th></tr>
<tr><th>RSO</th><th>RAM SELECT*</th><th>I/O TIMER SELECT*</th></tr>
<tr><td>1</td><td>X</td><td>X</td><td colspan="5">A0 - A9 directly address one of 1024 ROM bytes</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td colspan="5">A0 - A5 directly address one of 64 RAM bytes</td></tr>
<tr><td></td><td></td><td></td><th colspan="4">SECONDARY SELECT</th><th rowspan="2">INTERPRETATION</th></tr>
<tr><td></td><td></td><td></td><th>A3</th><th>A2</th><th>A1</th><th>A0</th></tr>
<tr><td>0</td><td>0</td><td>1</td><td>X</td><td>0</td><td>0</td><td>0</td><td>Access I/O Port A</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>X</td><td>0</td><td>0</td><td>1</td><td>Access I/O Port A Data Direction register</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>X</td><td>0</td><td>1</td><td>0</td><td>Access I/O Port B</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>X</td><td>0</td><td>1</td><td>1</td><td>Access I/O Port B Data Direction register</td></tr>
<tr><td>0</td><td>0</td><td>1W</td><td>0</td><td>1</td><td>X</td><td>X</td><td>Disable $\overline{IRQ}$</td></tr>
<tr><td>0</td><td>0</td><td>1W</td><td>1</td><td>1</td><td>X</td><td>X</td><td>Enable $\overline{IRQ}$</td></tr>
<tr><td>0</td><td>0</td><td>1W</td><td>X</td><td>1</td><td>0</td><td>0</td><td>Write to timer, then decrement every Φ2 pulse</td></tr>
<tr><td>0</td><td>0</td><td>1W</td><td>X</td><td>1</td><td>0</td><td>1</td><td>Write to timer, then decrement every 8 Φ2 pulses</td></tr>
<tr><td>0</td><td>0</td><td>1W</td><td>X</td><td>1</td><td>1</td><td>0</td><td>Write to timer, then decrement every 64 Φ2 pulses</td></tr>
<tr><td>0</td><td>0</td><td>1W</td><td>X</td><td>1</td><td>1</td><td>1</td><td>Write to timer, then decrement every 1024 Φ2 pulses</td></tr>
<tr><td>0</td><td>0</td><td>1R</td><td>X</td><td>1</td><td>X</td><td>0</td><td>Read timer</td></tr>
<tr><td>0</td><td>0</td><td>1R</td><td>X</td><td>1</td><td>X</td><td>1</td><td>Read interrupt flag</td></tr>
</table>

* RAM select and I/O select are "true" if 1, or "false" if 0; true and false are functions of your specification.
X represents "don't care". Bits may be 0 or 1.
1R represents Select during a read.
1W represents Select during a write.

As defined in Table 9-7, the Interval Timer has four addresses which you can use when loading an initial timer value. Each address specifies a different decrement interval. The four decrement intervals are 1, 8, 64 or 1024 Φ2 clock pulses.

Suppose the MCS6500 microcomputer system is being driven by a 500 nanosecond clock. The four decrement options mean that the Interval Timer may be decremented once every 1/2, 4, 32 or 512 microseconds. The time-out will occur anywhere from 1 to 256 decrements following the write into the Interval Timer.

Following a time-out, an interrupt will be requested. When an interrupt request occurs, the interrupt flag will be set. This flag may be read by the CPU using the address shown in Table 9-7.

The interrupt request will appear as a low level on pin 17 if the following conditions are met:

1) Address line A3 is 1 when reading from or writing to the timer.
2) PB7 has been programmed as an input by loading a 0 into bit 7 of the I/O Port B Data Direction register. (This is not necessary if the pin is factory masked to be $\overline{IRQ}$ only.)

The interrupt to pin 17 is disabled when address line A3 is 0 on a timer read or write.

The interrupt request is cleared (that is, $\overline{IRQ}$ returns high) the next time the timer is written or read.

Once the Interval Timer has timed out, it will decrement once more, from 0 back to 0. Then it will stop. Post-interrupt decrementing occurs on every Φ2 clock cycle, regardless of whether pre-interrupt decrements occurred every 1, 8, 64 or 1024 Φ2 clock cycles.

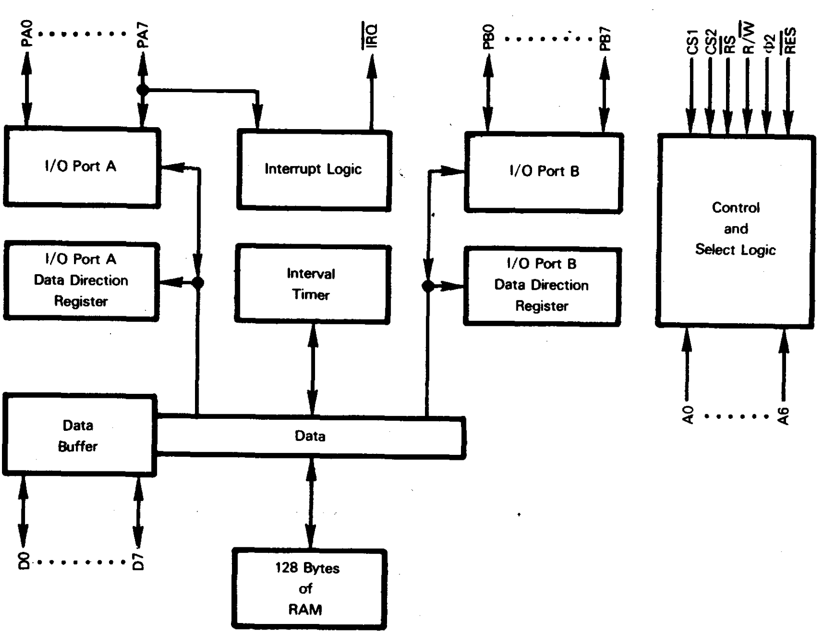


Figure 9-19. Logic Provided By The MCS6532 Multifunction Device

THE MCS6532 MULTIFUNCTION SUPPORT LOGIC DEVICE

This device is a variation of the MCS6530 which we have just described.

The MCS6532 provides no ROM memory, but twice the RAM — 128 bytes.

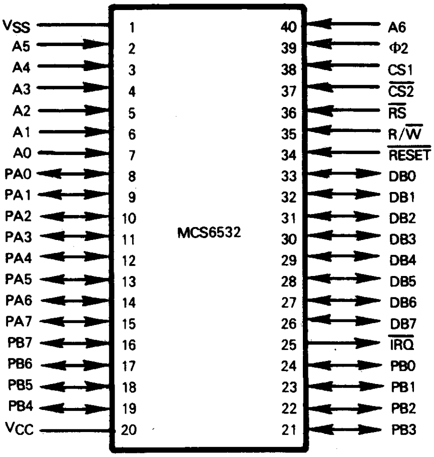
External logic can request an interrupt via the MCS6532 using a control signal which may be likened to the MCS6522 CA1 or CB1 control input.

The mask defined addressing options of the MCS6530 have been removed from the MCS6532; otherwise the balance of logic on the two devices is identical.

Figure 9-16 also illustrates that part of our general purpose microcomputer system logic which has been implemented on the MCS6532 multifunction device. Figure 9-19 illustrates the logic functions provided by the MCS6532.

The MCS6532 multifunction device is packaged as a 40-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL compatible. I/O Port A and B pins are also CMOS logic compatible. Pins of I/O Port B may be used as a power source to directly drive the base of a transistor switch.

The device is implemented using N-channel silicon gate MOS technology.



Pin Name	Description	Type
DB0 - DB7	Data Bus to CPU	Tristate, Bidirectional
Φ2	System Clock	Input
R/$\overline{\text{W}}$	Read/Write control	Input
$\overline{\text{RESET}}$	Reset	Input
PA0 - PA7	Port A Peripheral Data Bus	Tristate, Input or Output
PB0 - PB7	Port B Peripheral Data Bus	Tristate, Input or Output
$\overline{\text{IRQ}}$	Interrupt Request	Output
CS1, $\overline{\text{CS2}}$, $\overline{\text{RS}}$	Device or internal register select	Input
A0 - A6	Address lines	Input
Vcc, Vss	Power and Ground	

Figure 9-20. MCS6532 Multifunction Device Signals And Pin Assignments

MCS6532 MULTIFUNCTION DEVICE PINS AND SIGNALS

The MCS6532 multifunction device pins and signals are illustrated in Figure 9-20. These are the only differences between MCS6532 and MCS6530 signals:

1) $\overline{\text{IRQ}}$, CS1 and $\overline{\text{CS2}}$ are assigned unique pins by the MCS6532; the MCS6530 requires you to choose individually between these three signals and the three high order bits of I/O Port B.

2) For the MCS6532 to be selected, $\overline{\text{RS}}$ and $\overline{\text{CS2}}$ must be low while CS1 is high. Recall that with the MCS6530 RS0 is a signal which discriminates between ROM and other addressable locations; you define the way in which CS1 and CS2, if present, will function when you order an MCS6530 part.

MCS6532 ADDRESSING

Addressing the MCS6532 is a good deal simpler than addressing the MCS6530, since the MCS6532 has no ROM present and it has separate Chip Select signals. You still must define RAM select and I/O timer select as a function of $\overline{\text{RS}}$, CS1 and $\overline{\text{CS2}}$ and address lines A0 - A6. By connecting RS, CS1 and $\overline{\text{CS2}}$ to higher address lines, you can assign RAM or I/O timer logic various address spaces. This ability to define RAM and I/O Timer select as a mask option is a convenience, where with the MCS6530 it was frequently a necessity. With the MCS6532 you can accept whatever standard "off-the-shelf" option is being provided, and still have enough flexibility using RS, CS1 and CS2 to include a number of MCS6532 devices in a microcomputer configuration.

Table 9-8. Addressing The MCS6532 Multifunction Support Logic Device

PRIMARY SELECT		SECONDARY SELECT					INTERPRETATION
RAM SELECT	I/O TIMER SELECT	A4	A3	A2	A1	A0	
1	0	X	X	X	X	X	A0 - A6 directly addresses one of 128 RAM bytes
0	1	X	X	0	0	0	Access I/O Port A
0	1	X	X	0	0	1	Access I/O Port A Data Direction register
0	1	X	X	0	1	0	Access I/O Port B
0	1	X	X	0	1	1	Access I/O Port B Data Direction register
0	1W	1	0	1	X	X	Disable $\overline{IRQ}$
0	1W	1	1	1	X	X	Enable $\overline{IRQ}$
0	1W	1	X	1	0	0	Write to timer, then decrement every Φ2 pulse
0	1W	1	X	1	0	1	Write to timer, then decrement every 8 Φ2 pulses
0	1W	1	X	1	1	0	Write to timer, then decrement every 64 Φ2 pulses
0	1W	1	X	1	1	1	Write to timer, then decrement every 1024 Φ2 pulses
0	1R	X	X	1	X	0	Read timer
0	1R	X	X	1	X	1	Read interrupt flags
0	1W	0	X	1	X	0	Request interrupt on high-to-low PA7 transition
0	1W	0	X	1	X	1	Request interrupt on low-to-high PA7 transition
0	1W	0	X	1	0	X	Enable PA7 interrupt request
0	1W	0	X	1	1	X	Disable PA7 interrupt request

X represents "don't care". Bits may be 0 or 1.

1R represents Read access. 1W represents Write access.

MCS6532 LOGIC FUNCTIONS

Table 9-8 summarizes the way in which addressing is used both to access locations within the MCS6532 and to provide various logic functions.

The only logic of the MCS6532 which differs from the MCS6530 and needs to be described is the external interrupt request capability.

External logic requests interrupts via I/O Port A pin PA7. I/O Port A pin PA7 must be declared an input pin by loading 0 into bit 7 of the I/O Port A Data Direction register. Data Direction registers have been described in conjunction with the MCS6522. A low-to-high or high-to-low transition on a signal input to PA7 will generate the interrupt request. An interrupt request will be accompanied by bit 6 of the Interrupt Flag register being set. Table 9-8 defines the way in which you select interrupt options.

MCS6532 interrupt acknowledge logic requires the CPU to read the Interrupt Flags register. This read operation resets MCS6532 interrupt logic.

DATA SHEETS

The following section contains electrical characteristics and specific timing data for the MCS6500 series CPUs and the MCS6530 Multifunction Device.

COMMON CHARACTERISTICS

MAXIMUM RATINGS

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	Vcc	-0.3 to +7.0	Vdc
INPUT VOLTAGE	Vin	-0.3 to +7.0	Vdc
OPERATING TEMPERATURE	T_A	0 to +70	°C
STORAGE TEMPERATURE	T_{STG}	-55 to +150	°C

This device contains input protection against damage due to high static voltages or electric fields; however, precautions should be taken to avoid application of voltages higher than the maximum rating.

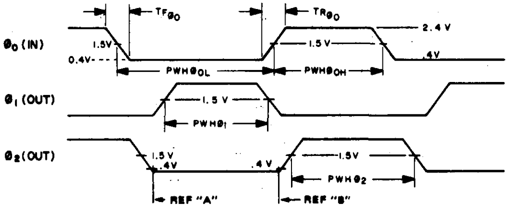
ELECTRICAL CHARACTERISTICS **(Vcc = 5.0V ± 5%, Vss = 0, T_A = 25° C)**

ϕ_1, ϕ_2 applies to MCS6512, 13, 14, 15, ϕ_o (in) applies to MCS6502, 03, 04, 05 and 06

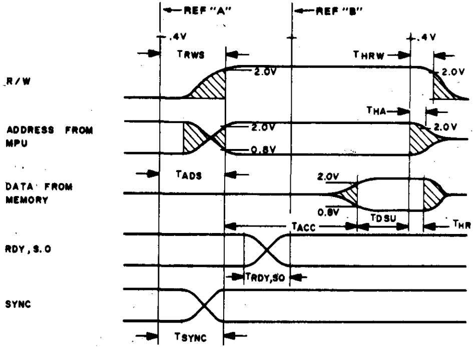
CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
Input High Voltage Logic, ϕ_o(in) ϕ_1, ϕ_2	V_{IH}	Vss + 2.4 Vcc - 0.2	- -	Vcc Vcc + 0.25	Vdc
Input Low Voltage Logic, ϕ_o(in) ϕ_1, ϕ_2	V_{IL}	Vss - 0.3 Vss - 0.3	- -	Vss + 0.4 Vss + 0.2	Vdc
Input High Threshold Voltage $\overline{RES}$, $\overline{NMI}$, RDY, $\overline{IRQ}$, Data, S.O.	V_{IHT}	Vss + 2.0	-	-	Vdc
Input Low Threshold Voltage $\overline{RES}$, $\overline{NMI}$, RDY, $\overline{IRQ}$, Data, S.O.	V_{ILT}	-	-	Vss + 0.8	Vdc
Input Leakage Current (V_{in} = 0 to 5.25V, Vcc = 0) Logic (Excl. RDY, S.O.) ϕ_1, ϕ_2 ϕ_o(in)	I_{in}	- - -	- - -	2.5 100 10.0	μA μA μA
Three-State (Off State) Input Current (V_{in} = 0.4 to 2.4V, Vcc = 5.25V) Data Lines	I_{TSI}	-	-	10	μA
Output High Voltage (I_{LOAD} = -100μAdc, Vcc = 4.75V) SYNC, Data, A0-A15, R/W	V_{OH}	Vss + 2.4	-	-	Vdc
Output Low Voltage (I_{LOAD} = 1.6mAdc, Vcc = 4.75V) SYNC, Data, A0-A15, R/W	V_{OL}	-	-	Vss + 0.4	Vdc
Power Dissipation	P_D	-	.25	.70	W
Capacitance (V_{in} = 0, T_A = 25°C, f = 1MHz)	C				pF
Logic	C_{in}	-	-	10	
Data A0-A15, R/W, SYNC	C_{out}	- -	- -	15 12	
ϕ_o(in)	$C_{\phi_o(in)}$	-	-	15	
ϕ_1	C_{ϕ_1}	-	30	50	
ϕ_2	C_{ϕ_2}	-	50	80	

Note: $\overline{IRQ}$ and $\overline{NMI}$ require 3K pull-up resistors.

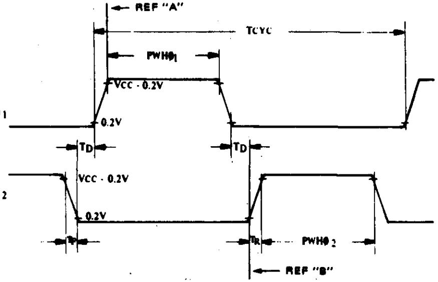
COMMON CHARACTERISTICS



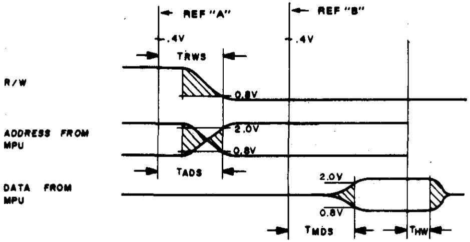
Clock Timing – MCS6502, 03, 04, 05, 06



Timing for Reading Data from Memory or Peripherals



Clock Timing – MCS6512, 13, 14, 15



Timing for Writing Data to Memory or Peripherals

Note: "REF." means Reference Points on clocks.

1 MH$_Z$ TIMING

Clock Timing – MCS6512, 13, 14, 15

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
Cycle Time	T_{CYC}	1000	---	---	nsec
Clock Pulse Width Ø1 (Measured at Vcc - 0.2v) Ø2	PWH Ø1 PWH Ø2	430 470	---	---	nsec
Fall Time (Measured from 0.2v to Vcc - 0.2v)	T_F	---	---	25	nsec
Delay Time between Clocks (Measured at 0.2v)	T_D	0	---	---	nsec

CLOCK TIMING -MCS6502, 03, 04, 05, 06

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNITS
Cycle Time	T_{CYC}	1000	--	--	ns
$\phi_{o(IN)}$ Pulse Width (measured at 1.5V)	$PWH\phi_o$	460	--	520	ns
$\phi_{o(IN)}$ Rise, Fall Time	$TR\phi_o, TF\phi_o$	--	--	10	ns
Delay Time Between Clocks (measured at 1.5V)	T_D	5	--	--	ns
$\phi_{1(OUT)}$ Pulse Width (measured at 1.5V)	$PWH\phi_1$	$PWH\phi_{oL}-20$	--	$PWH\phi_{oL}$	ns
$\phi_{2(OUT)}$ Pulse Width (measured at 1.5V)	$PWH\phi_2$	$PWH\phi_{oH}-40$	--	$PWH\phi_{oH}-10$	ns
$\phi_{1(OUT)}$, $\phi_{2(OUT)}$ Rise, Fall Time (measured .8V to 2.0 V) (Load = 30pf + 1 TTL)	T_R, T_F	--	--	25	ns

READ/WRITE TIMING

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNITS
Read/Write Setup Time from **MCS6500**	T_{RWS}	--	100	300	ns
Address Setup Time from **MCS6500**	T_{ADS}	--	100	300	ns
Memory Read Access Time	T_{ACC}	--	--	575	ns
Data Stability Time Period	T_{DSU}	100	--	--	ns
Data Hold Time - Read	T_{HR}	10	--	--	ns
Data Hold Time - Write	T_{HW}	30	60	--	ns
Data Setup Time from **MCS6500**	T_{MDS}	--	150	200	ns
RDY, S.O. Setup Time	T_{RDY}	100	--	--	ns
SYNC Setup Time from **MCS6500**	T_{SYNC}	--	--	350	ns
Address Hold Time	T_{HA}	30	60	--	ns
R/W Hold Time	T_{HRW}	30	60	--	ns

2 MH$_Z$ TIMING

Clock Timing – MCS6512, 13, 14, 15, 16

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
Cycle Time	T_{CYC}	500	---	---	nsec
Clock Pulse Width Ø1 (Measured at Vcc - 0.2v) Ø2	PWH Ø1 PWH Ø2	215 235	---	---	nsec
Fall Time (Measured from 0.2v to Vcc - 0.2v)	T_F	---	---	12	nsec
Delay Time between Clocks (Measured at 0.2v)	T_D	0	---	---	nsec

CLOCK TIMING - MCS6502, 03, 04, 05, 06

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNITS
Cycle Time	T_{CYC}	500	--	--	ns
$\phi_{o(IN)}$ Pulse Width (measured at 1.5V)	$PWH\phi_o$	240	--	260	ns
$\phi_{o(IN)}$ Rise, Fall Time	$TR\phi_o, TF\phi_o$	--	--	10	ns
Delay Time Between Clocks (measured at 1.5V)	T_D	5	--	--	ns
$\phi_{1(OUT)}$ Pulse Width (measured at 1.5V)	$PWH\phi_1$	$PWH\phi_{oL}-20$	--	$PWH\phi_{oL}$	ns
$\phi_{2(OUT)}$ Pulse Width (measured at 1.5V)	$PWH\phi_2$	$PWH\phi_{oH}-40$	--	$PWH\phi_{oH}-10$	ns
$\phi_{1(OUT)}$, $\phi_{2(OUT)}$ Rise, Fall Time (measured .8V to 2.0 V) (Load = 30pf + 1 TTL)	T_R, T_F	--	--	25	ns

READ/WRITE TIMING

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNITS
Read/Write Setup Time from **MCS6500A**	T_{RWS}	--	100	150	ns
Address Setup Time from **MCS6500A**	T_{ADS}	--	100	150	ns
Memory Read Access Time	T_{ACC}	--	--	300	ns
Data Stability Time Period	T_{DSU}	50	--	--	ns
Data Hold Time - Read	T_{HR}	10	--	--	ns
Data Hold Time - Write	T_{HW}	30	60	--	ns
Data Setup Time from **MCS6500A**	T_{MDS}	--	75	100	ns
RDY, S.O. Setup Time	T_{RDY}	50	--	--	ns
SYNC Setup Time from **MCS6500A**	T_{SYNC}	--	--	175	ns
Address Hold Time	T_{HA}	30	60	--	ns
R/W Hold Time	T_{HRW}	30	60	--	ns

MAXIMUM RATINGS

RATING	SYMBOL	VOLTAGE	UNIT
Supply Voltage	VCC	-.3 to +7.0	V
Input/Output Voltage	V_{IN}	-.3 to +7.0	V
Operating Temperature Range	T_{OP}	0 to 70	°C
Storage Temperature Range	T_{STG}	-55 to +150	°C

All inputs contain protection circuitry to prevent damage due to high static charges. Care should be exercised to prevent unnecessary application of voltage outside the specification range.

ELECTRICAL CHARACTERISTICS (VCC = 5.0v ± 5%, VSS = 0v, T_A = 25° C)

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
Input High Voltage	V_{IH}	V_{SS}+2.4		VCC	V
Input Low Voltage	V_{IL}	V_{SS}-.3		V_{SS}+.4	V
Input Leakage Current; V_{IN} = V_{SS} + 5v AØ-A9, RS, R/W, $\overline{RES}$, Ø2, PB6*, PB5*	I_{IN}		1.0	2.5	µA
Input Leakage Current for High Impedance State (Three State); V_{IN} = .4v to 2.4v; DØ-D7	I_{TSI}		±1.0	±10.0	µA
Input High Current; V_{IN} = 2.4v PAØ-PA7, PBØ-PB7	I_{IH}	-100.	-300.		µA
Input Low Current; V_{IN} = .4v PAØ-PA7, PBØ-PB7	I_{IL}		-1.0	-1.6	MA
Output High Voltage VCC = MIN, I_{LOAD} ≤ -100µA(PAØ-PA7,PBØ-PB7,DØ-D7) I_{LOAD} ≤ -3 MA (PAØ,PBØ)	V_{OH}	VSS+2.4 VSS+1.5			V
Output Low Voltage VCC = MIN, I_{LOAD} ≤ 1.6MA	V_{OL}			VSS+.4	V
Output High Current (Sourcing); VOH ≥ 2.4v (PAØ-PA7,PBØ-PB7,DØ-D7) ≥ 1.5v Available for other than TTL (Darlingtons) (PAØ,PBØ)	I_{OH}	-100 -3.0	-1000 -5.0		µA MA
Output Low Current (Sinking); VOL ≤ .4v (PAØ-PA7)(PBØ-PB7)	I_{OL}	1.6			MA
Clock Input Capacitance	C_{Clk}			30	pf
Input Capacitance	C_{IN}			10	pf
Output Capacitance	C_{OUT}			10	pf
Power Dissipation	P_D		500	1000	MW

*When programmed as address pins
All values are D.C. readings

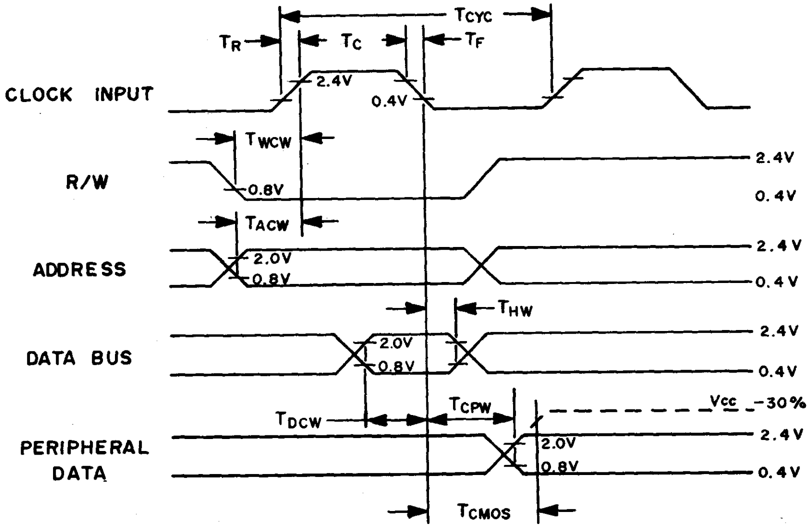
WRITE TIMING CHARACTERISTICS

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
Clock Period	T_{CYC}	1		10	µS
Rise & Fall Times	TR, TF			25	NS
Clock Pulse Width	TC	470			NS
R/W valid before positive transition of clock	TWCW	180			NS
Address valid before positive transition of clock	TACW	180			NS
Data Bus valid before negative transition of clock	TDCW	300			NS
Data Bus Hold Time	THW	10			NS
Peripheral data valid after negative transition of clock	TCPW			1	µS
Peripheral data valid after negative transition of clock driving CMOS (Level=VCC-30%)	TCMOS			2	µS

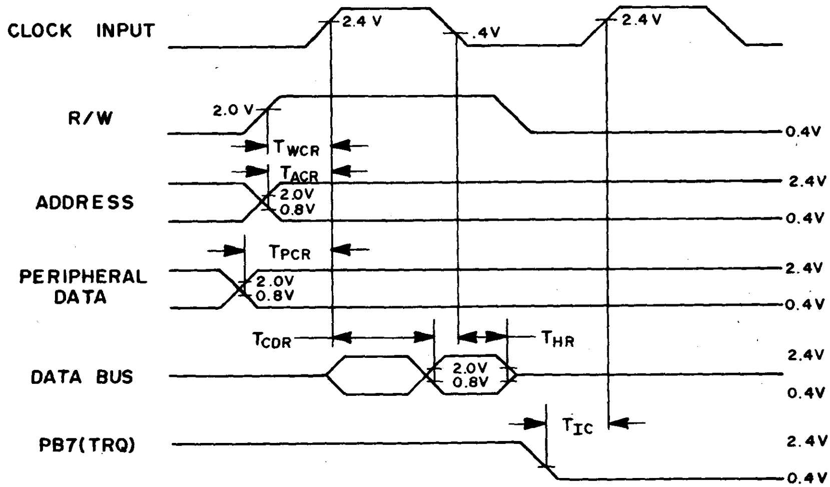
READ TIMING CHARACTERISTICS

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
R/W valid before positive transition of clock	TWCR	180			NS
Address valid before positive transition of clock	TACR	180			NS
Peripheral data valid before positive transition of clock	TPCR	300			NS
Data Bus valid after positive transition of clock	TCDR			395	NS
Data Bus Hold Time	THR	10			NS
$\overline{\text{IRQ}}$ (Interval Timer Interrupt) valid before positive transition of clock	TIC	200			NS

Loading = 30 pf + 1 TTL load for PAØ-PA7, PBØ-PB7
=130 pf + 1 TTL load for DØ-D7



WRITE TIMING CHARACTERISTICS



READ TIMING CHARACTERISTICS

CHAPTER 10
THE SIGNETICS 2650

Within the frame of reference of the microcomputers being described in this book, the Signetics 2650 is a very minicomputer-like device, comparable in design philosophy to the National Semiconductor products.

Like the National Semiconductor microcomputers, the Signetics 2650 has a wealth of memory addressing modes; a large number of CPU-generated control signals are aimed at allowing standard TTL logic to surround the microcomputer device itself, rather than requiring a family of support devices, as do most products described in this book. However, you will have very little trouble using support devices of the 8080A with the Signetics 2650 CPU. MC6800 support devices can be used with the Signetics 2650 — but with more difficulty.

Interesting features of the 2650, which are described on the following pages, are the imaginative use of status flags, a rich variety of very informative control signals, and the use of the second object code byte, in multibyte instructions, to encode memory addressing options.

Figure 10-1 illustrates the logical functions implemented on the 2650 CPU chip. Memory and other external logic will connect directly to the 2650 address, data and control lines, without need for interface devices (other than buffer amplifiers needed to meet signal loads).

The 2650 uses a single +5V power supply.

Using a clock with 0.8 microsecond period, instruction execution times vary between 4.8 and 9.6 microseconds.

All 2650 signals are TTL compatible.

THE 2650 CPU LOGIC

The 2650 CPU has a typical microcomputer organization. The Arithmetic and Logic Unit, the Control Unit and programmable registers are all implemented on the 2650 CPU.

The additions and omissions shown in Figure 10-1, as compared to typical CPU logic, need some preliminary explanation.

Although the 2650 has just one interrupt request line and one interrupt acknowledge line, CPU logic allows every interrupting device to force a vectored branch to its own, unique interrupt service routine; for this reason, logic to handle interrupt requests is shown as an integral part of CPU chip logic.

Standard ROM and RAM devices can be connected directly to 2650 bus lines; therefore, the 2650 is shown as providing complete memory interface logic. Note, however, that TTL load levels will almost certainly require that signal buffer amplifiers interface memory devices to the 2650 CPU.

I/O port interface logic is shown as only partially implemented on the 2650 CPU chip. A 2650-based microcomputer system with one or two I/O ports will require no special I/O port logic; control signals allow the Data Bus to be used either as a conduit to external devices or to memory. But if a 2650-based microcomputer system has more than two separately addressable I/O ports, external I/O port select logic must be added.

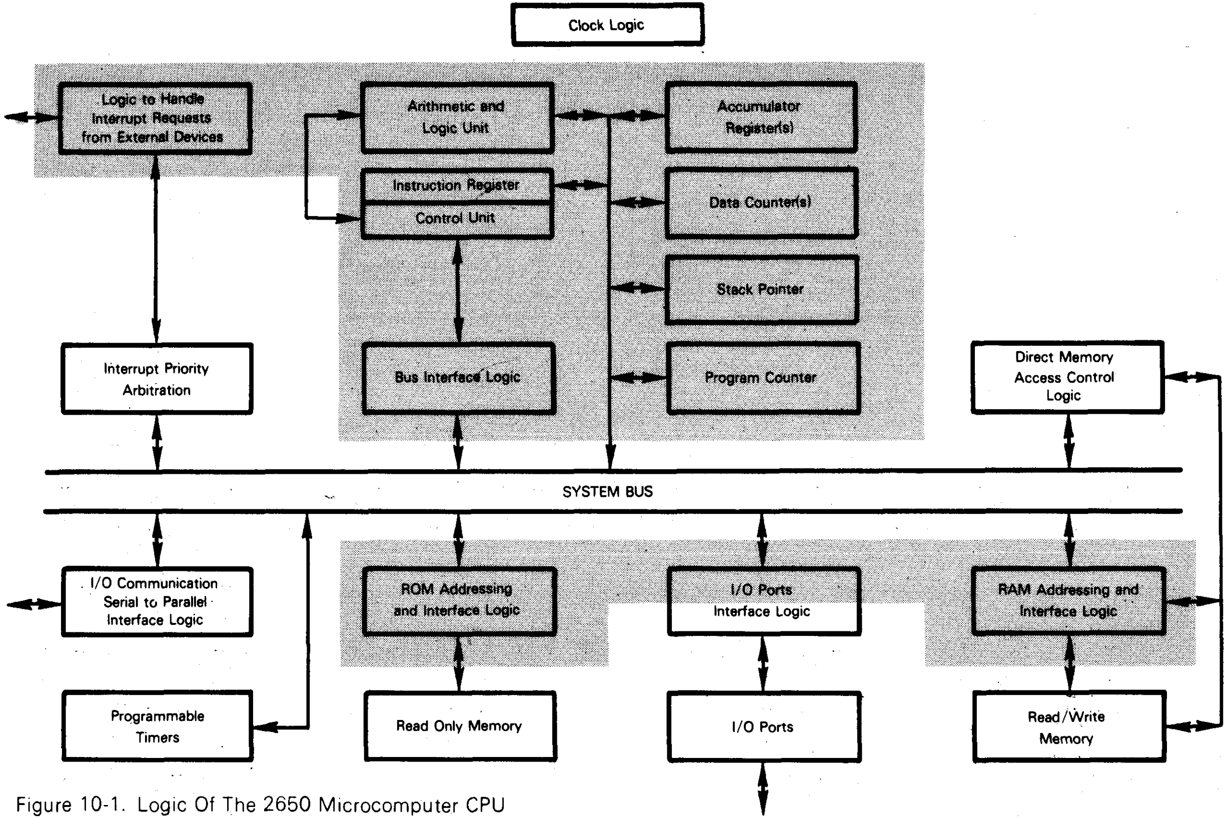
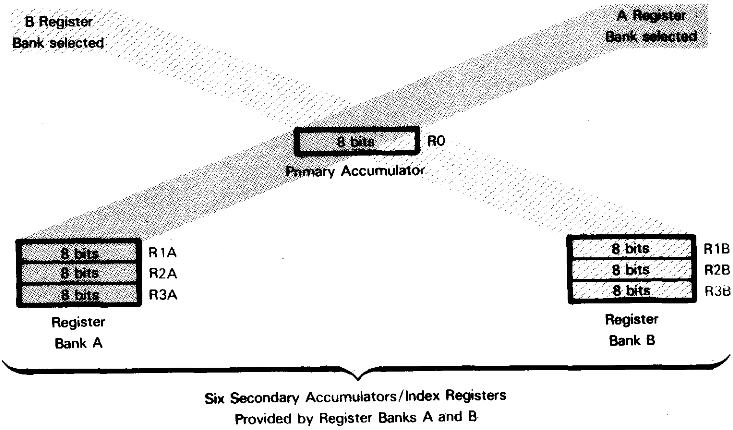

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 10-1. Logic Of The 2650 Microcomputer CPU

Figure 10-1 excludes clock logic from the CPU chip. The 2650 CPU does indeed require external logic to create its clock signal; however, a single TTL level clock signal, with relatively lax tolerances is required; therefore, external generation of the clock signal will be both inexpensive and free of problems.

2650 PROGRAMMABLE REGISTERS

In addition to a 15-bit Program Counter, the 2650 has seven 8-bit programmable registers which may be illustrated as follows:



R0 is a primary Accumulator. This register is always accessible.

2650 ACCUMULATOR

2650 INDEX REGISTERS

The remaining six 8-bit registers form two 3-register banks. A status bit (which will be described later) is used to identify one of the two register banks as accessible at any given time. Thus, depending on the status bit setting, Registers R0, R1A, R2A and R3A may be accessible, or else Registers R0, R1B, R2B and R3B may be accessible.

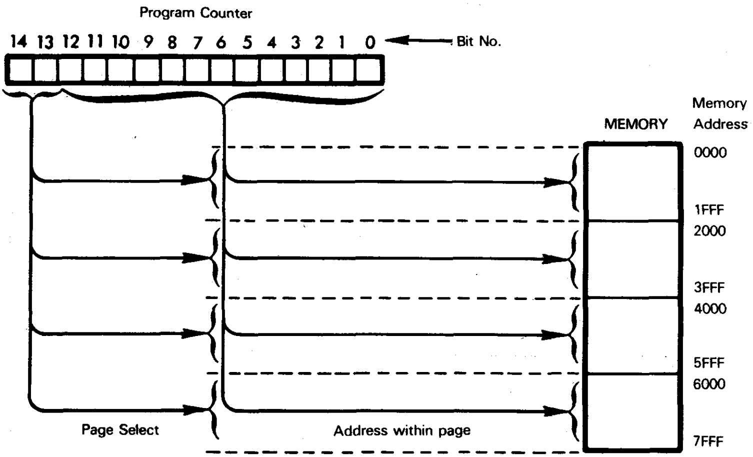
The six secondary registers serve as both secondary Accumulators and Index registers. The 2650 has no Data Counters, as do most microcomputers; rather, it uses the minicomputer philosophy of adding an index, out of an Index register, to a memory address which is computed from information provided by every Memory Reference instruction.

The Program Counter is 15 bits wide; therefore up to 32,768 bytes of memory may be addressed in the normal course of events.

2650 PROGRAM COUNTER

2650 MEMORY PAGES

The two high order bits of the Program Counter represent page select bits. 2650 memory is divided into four pages with 8192 bytes of memory per page; this scheme is illustrated as follows:



Pages are selected by Branch instructions; but we will defer to the discussion of addressing modes a description of how this is done.

2650 STACK

The 2650 has a primitive Stack, implemented on the CPU chip; this Stack is eight addresses deep, and its use is limited to storing subroutine return addresses and interrupt return addresses. Subroutines and interrupts may therefore be combined to a nested level of eight. There are no Push and Pop type instructions, and the Stack is indexed via three bits of a Status register.

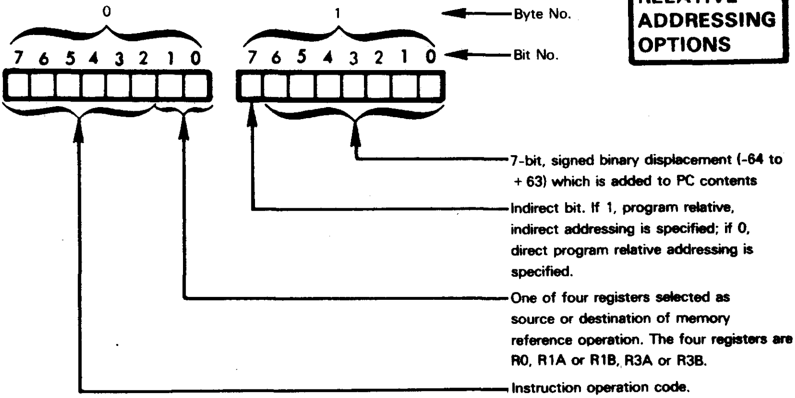
THE 2650 MEMORY ADDRESSING MODES

The 2650 has the most extensive and versatile range of addressing modes of any microcomputer described in this book — with the possible exception of the MCS6500 series microprocessors.

Primary and secondary memory referencing instructions each provide two sets of addressing options, one based on program relative addressing and a two-byte instruction code, the other based on direct addressing and a three-byte instruction code. These options are referred to in Table 10-1 as the program relative addressing options and the extended addressing options.

Instructions with program relative addressing options have the following object code:

2650 PROGRAM RELATIVE ADDRESSING OPTIONS

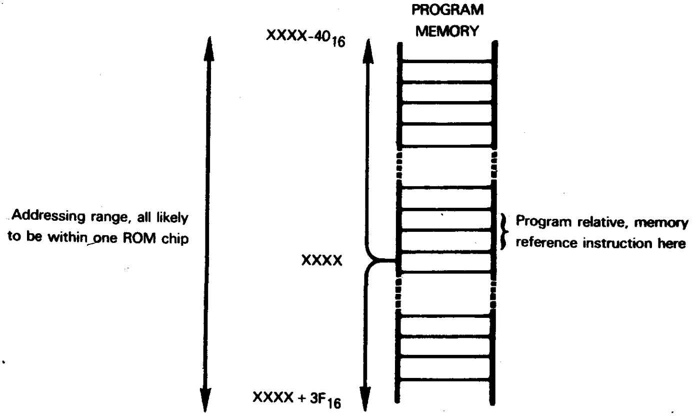

0
1
Byte No.
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
Bit No.
7-bit, signed binary displacement (-64 to + 63) which is added to PC contents
Indirect bit. If 1, program relative, indirect addressing is specified; if 0, direct program relative addressing is specified.
One of four registers selected as source or destination of memory reference operation. The four registers are R0, R1A or R1B, R3A or R3B.
Instruction operation code.

In the above illustration, the second byte of the instruction code provides a program relative displacement in the range +63 to -64. The displacement is provided as a 7-bit signed binary number, bit 6 is treated as the sign bit. The high order bit of the displacement byte specifies direct or indirect addressing.

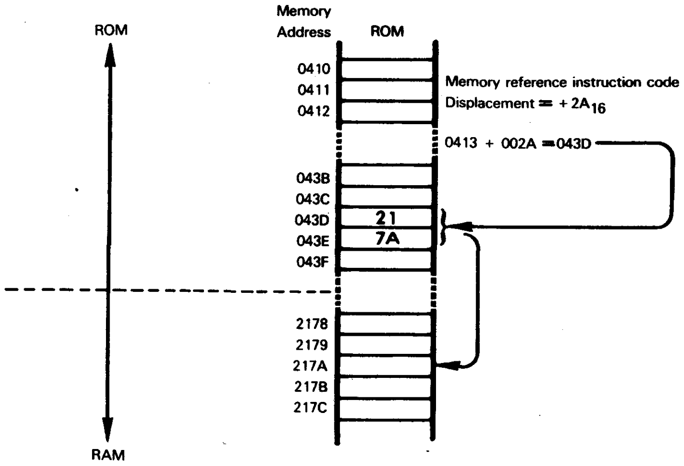
If direct, program relative addressing is specified, then the effective memory address is created by adding the 7-bit signed binary displacement to the Program Counter contents — after the Program Counter contents have been incremented. Direct and indirect program relative addressing have been described in Volume I, Chapter 6; 2650 program relative addressing differs only in the shorter displacement which is allowed.

If we are to relate the 2650 to our hypothetical microcomputer of Volume I, Chapter 7, or to any of the other microcomputers described in this book, then the task of specifying direct or indirect addressing should fall to a bit within the first object program byte; the fact that the 2650 uses a bit of the displacement byte to specify direct or indirect addressing means that, in effect, the 2650 instruction set has more than 256 object code options available to it. This feature of the 2650 allows it to have a much more powerful instruction set — in the minicomputer sense of the word — than any of the other devices described in this chapter.

In all probability, indirect, program relative addressing will be more commonly used than direct, program relative addressing. This is because microcomputer programs usually reside in read-only memory. If direct, program relative addressing is used, then data bytes must be located within 64 bytes of the memory reference instruction. That excludes having instructions in ROM and data in RAM; therefore, only unalterable constants can be addressed using program relative direct addressing.

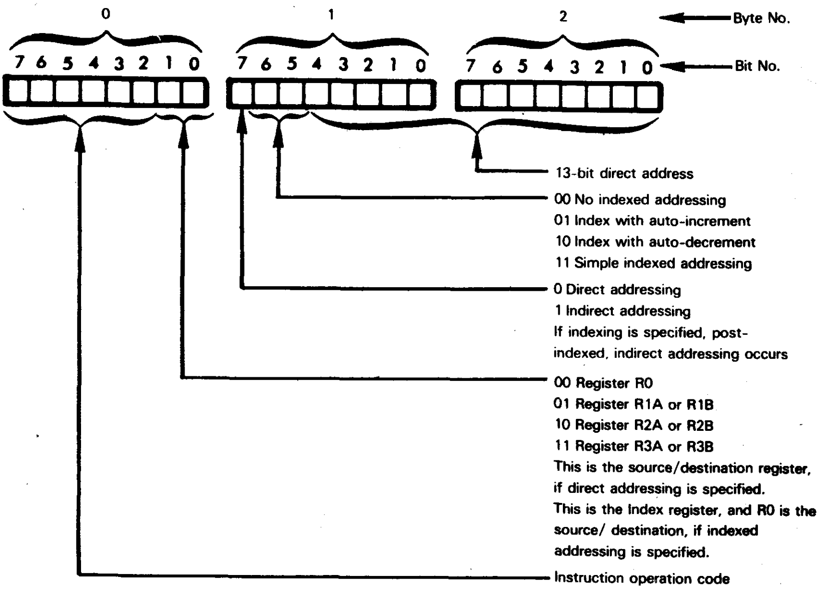

PROGRAM MEMORY
XXXX-40_{16}
Addressing range, all likely to be within one ROM chip
XXXX
Program relative, memory reference instruction here
XXXX + $3F_{16}$

Indirect, program relative addressing, on the other hand, only requires memory addresses to be positioned within 64 bytes of the memory reference instruction; this is illustrated as follows, using arbitrary memory addresses to make the illustration easier to understand:



Extended addressing options of the 2650 microcomputer may be illustrated as follows:

2650 EXTENDED ADDRESSING OPTIONS


0
1
2
Byte No.
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
Bit No.
13-bit direct address
00 No indexed addressing
01 Index with auto-increment
10 Index with auto-decrement
11 Simple indexed addressing
0 Direct addressing
1 Indirect addressing
If indexing is specified, post-indexed, indirect addressing occurs
00 Register R0
01 Register R1A or R1B
10 Register R2A or R2B
11 Register R3A or R3B
This is the source/destination register, if direct addressing is specified.
This is the Index register, and R0 is the source/ destination, if indexed addressing is specified.
Instruction operation code

All of the addressing options illustrated above have been described in Volume I, Chapter 6. To summarize, however, these are the addressing combinations which are allowed:

1) Direct addressing (absolute or program relative)

2) Direct indexed addressing

3) Direct indexed addressing with auto-increment

4) Direct indexed addressing with auto-decrement

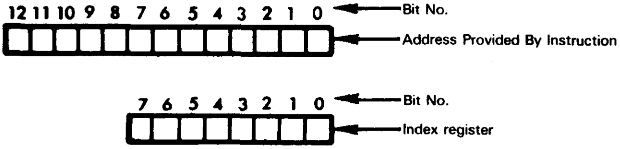
5) Indirect addressing

6) Indirect addressing with post-index

7) Indirect addressing with post-index and auto-increment

8) Indirect addressing with post-index and auto-decrement

There is a small difference between indexed addressing, as described in Volume I, Chapter 6, and indexed addressing, as implemented by the 2650. The 2650 memory reference instructions provide a 13-bit absolute address, which represent the full addressing range of any memory bank; an 8-bit index value is added to this displacement, as follows:

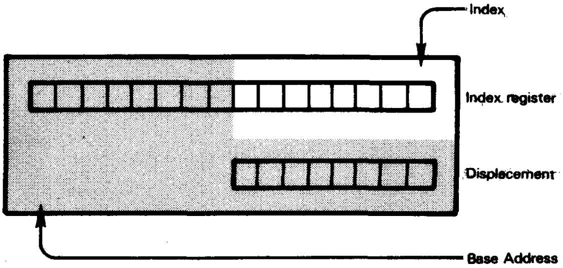


Effective address = 13-bit absolute address +8-bit·index.

If you are not clear on the difference between pre-indexed indirect addressing and post-indexed indirect addressing, refer again to Chapter 6, Volume I, before proceeding with this discussion of the 2650 microcomputer.

The fact that the 2650 has a 13-bit absolute address and an 8-bit index means that post-indexed, indirect addressing is very viable. The 13-bit absolute address identifies the memory location, anywhere within an 8192-byte program page, where an indirect address will be found. The indirect address becomes the base of a 256-byte table which may be indexed via any one of the six Index registers. The Index register contents are treated as an unsigned, binary number.

Now look again at indexed addressing the way it is in most microcomputers, and the way it is described in Chapter 6, Volume I. A 16-bit Index register indexes tables that are up to 65,536 bytes in length, and that is clearly ridiculous in microcomputers. The usual programming procedure, when using microcomputers that have a 16-bit Index register, is to use only the low order byte of the Index register for indexing. The base address is created out of the high order byte of the Index register, plus the displacement:

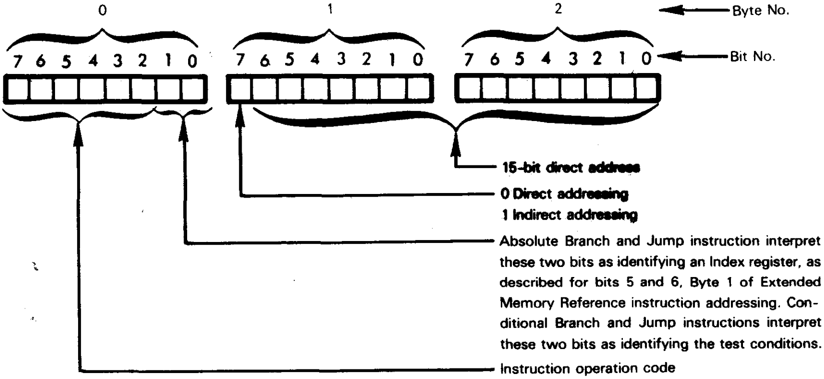


If the base address is created half out of an Index register and half out of a displacement, then clearly post-index indirect addressing is impossible.

Any minicomputer programmer will attest to the fact that post-indexed, indirect addressing is far more useful than pre-indexed, indirect addressing.

The 2650 has a wide variety of Branch and Branch-on-Condition instructions, which have the following object code and format:

2650 BRANCH INSTRUCTION ADDRESSING


0
1
2
Byte No.
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
Bit No.
15-bit direct address
0 Direct addressing
1 Indirect addressing
Absolute Branch and Jump instruction interpret these two bits as identifying an Index register, as described for bits 5 and 6, Byte 1 of Extended Memory Reference instruction addressing. Conditional Branch and Jump instructions interpret these two bits as identifying the test conditions.
Instruction operation code

Most 2650 Jump and Branch instructions are conditional; that means that only direct or indirect addressing may be used.

Notice that the branch direct address is 15 bits wide. Therefore, a Branch instruction may reference any byte within the maximum 32K-byte memory allowed by the 2650.

2650 MEMORY PAGE SELECTION

Branch instructions are, in fact, the means provided by the 2650 microcomputer to select a page of memory. The two high order bits of a Branch instruction's direct address select an 8K-byte memory bank, which remain selected until another Branch instruction modifies the selection.

The 2650 has two unconditional Branch instructions. These instructions also have a 15-bit direct address; therefore, they also select a memory page. In addition to allowing direct or indirect addressing, these two instructions allow indexed addressing to be specified, as described for the extended addressing options.

Since Branch instructions specify a 15-bit direct address, in the vast majority of cases simple direct addressing will be used. Indexed addressing will be valuable only in special logic sequences — such as branch tables. Branch instructions with indirect addressing will rarely have any justifiable value.

Conditional Branch instructions use bits 0 and 1 of byte 0 to determine if a test condition has been met. The way in which these two bits are used is discussed below, along with the description of 2650 Status registers.

THE 2650 STATUS FLAGS

The 2650 microcomputer has two 8-bit Status registers as follows:

7	6	5	4	3	2	1	0	← Bit No.
S	F	II	☒	☒	SP2	SP1	SP0	← Upper Program Status Word (PSU)

7	6	5	4	3	2	1	0	← Bit No.
CC1	CC0	IDC	RS	WC	O	COM	C	← Lower Program Status Word (PSL)

S and F represent a Sense Input bit and a Flag Output bit, both of which are connected directly to two CPU device pins. These two bits allow one input and one output signal to directly interface external devices to the CPU, under program control.

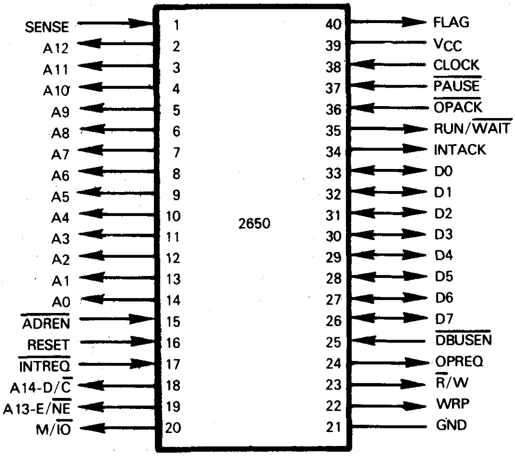
The Interrupt Inhibit bit is the master interrupt enable-disable flag for the 2650 microcomputer system.

SP0, SP1 and SP2 constitute a 3-bit Stack Pointer. Recall that the 2650 has a Stack eight addresses deep; the current top-of-Stack is addressed by this 3-bit Stack Pointer.

The two Condition Codes CC0 and CC1 report the condition of a data byte as zero, positive or negative. The zero condition represents a byte containing eight binary zeros. The positive condition represent a byte with 0 in the high order bit. The negative condition represents a byte with 1 in the high order bit. These Condition Codes are set following the execution of any instruction which loads a byte of data into a register, or modifies the register's contents. These two Condition bits represent a minor variation of the more common technique, which is for a conditional instruction to test a register's contents directly, at the time the conditional instruction is executed.

IDC is a standard intermediate Carry bit.

O, the Overflow bit, and C, the Carry/Borrow bit, are standard Overflow and Carry statuses as described in Volume I, Chapter 2.


SENSE 1
A12 2
A11 3
A10 4
A9 5
A8 6
A7 7
A6 8
A5 9
A4 10
A3 11
A2 12
A1 13
A0 14
$\overline{\text{ADREN}}$ 15
RESET 16
$\overline{\text{INTREQ}}$ 17
A14-D/$\overline{\text{C}}$ 18
A13-E/$\overline{\text{NE}}$ 19
M/$\overline{\text{IO}}$ 20
2650
40 FLAG
39 VCC
38 CLOCK
37 $\overline{\text{PAUSE}}$
36 $\overline{\text{OPACK}}$
35 RUN/$\overline{\text{WAIT}}$
34 INTACK
33 D0
32 D1
31 D2
30 D3
29 D4
28 D5
27 D6
26 D7
25 $\overline{\text{DBUSEN}}$
24 OPREQ
23 $\overline{\text{R}}$/W
22 WRP
21 GND

Pin Name	Description	Type
*A0-A12	Address Bus lines	Output
*A13-A14	Page Select lines	Output
*D0-D7	Data Bus lines	Bidirectional
*SENSE	Control input	Input
*FLAG	Control output	Output
*$\overline{\text{ADREN}}$	Address Bus float	Input
*$\overline{\text{DBUSEN}}$	Data Bus float	Input
*RESET	Reset	Input
*D/$\overline{\text{C}}$	Data/Control output	Output
*M/$\overline{\text{IO}}$	Memory/IO Reference	Output
*$\overline{\text{R}}$/W	Read/Write	Output
*OPREQ	Operation Request	Output
*$\overline{\text{OPACK}}$	Operation Acknowledge	Input
*E/$\overline{\text{NE}}$	I/O Instruction length	Output
*WRP	Write Pulse	Output
*$\overline{\text{INTREQ}}$	Interrupt Request	Input
*INTACK	Interrupt Acknowledge	Output
*RUN/$\overline{\text{WAIT}}$	Run status	Output
*$\overline{\text{PAUSE}}$	Wait	Input
CLOCK	Timing	Input
VCC, GND	Power and Ground	

*These signals become the System Bus.

Figure 10-2. 2650 CPU Signals And Pin Assignments

RS, the Register Bank Select bit, is used to select R1A, R2A and R3A, or R1B, R2B and R3B as the three currently selected Accumulator/Index registers bank.

Recall that addition, subtraction, shift and rotate instructions optionally may or may not include the Carry status; in other words a microcomputer may have an Add-with-Carry or an Add-without-Carry instruction; it may have a Rotate-simple or a Rotate-through-Carry instruction. The WC bit specifies whether the Carry will or will not be included in 2650 instructions of this type. If the C status is included in a rotate, the IDC status will also be included, operating as a branch carry out of bit 3. This is a unique 2650 feature.

The Compare status determines whether Compare instructions will treat data as signed or unsigned binary numbers. Consider an instruction which compares the contents of Register R0 with the contents of a memory byte. Clearly the result of the comparison will differ significantly, depending on whether the high order bit of each byte is being interpreted as a sign bit, or whether positive numbers only are being compared. If the COM status flag is set to I, the two bytes are assumed to be positive numbers. If the COM status is set to 0, the two bytes will be assumed to contain signed binary numbers.

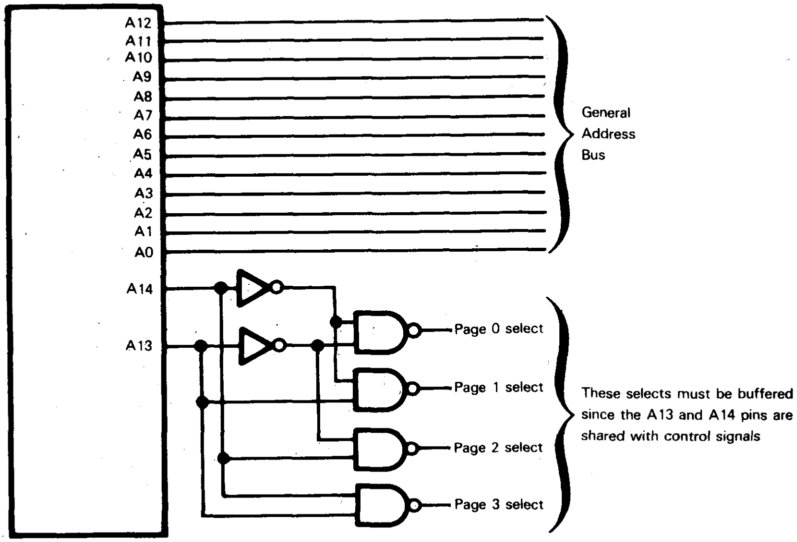
The WC and COM statuses of the 2650 microcomputer are very powerful features; their significance is that they double the available number of Arithmetic and Compare instructions, respectively, without increasing the number of instruction object codes.

THE 2650 CPU PINS AND SIGNALS

The 2650 CPU pins and signals are illustrated in Figure 10-2. A description of these signals will highlight the underlying philosophy of the 2650 chip design: that this device should be used with standard off-the-shelf TTL logic, rather than requiring a family of support devices. There are applications where the Signetics philosophy is viable and will work; there are other applications where the specialized devices provided by other microcomputer systems cannot be reproduced at equivalently low cost.

The Address Bus is 13 lines wide; it is used to address a single byte within 8192 bytes of memory. The low order eight address lines may also be used to address an external device.

A13 and 14 are page select lines. As described in the discussion of addressing modes, only Branch instructions provide 15-bit memory addresses. When a Branch instruction is executed, the two high order bits of the address, output on pins 18 and 19, are used by external memory to select or deselect 8K memory pages. Subsequent memory reference instructions that provide only a 13-bit memory address will reference the most recently selected 8K memory bank. This may be illustrated as follows:



Control lines of the 2650 microcomputer may be grouped into categories as follows:

1) **CPU execution control.**
2) **Data and Address Bus access control.**
3) **Data and Address Bus contents identification.**
4) **Interrupt processing.**
5) **Direct, external device interface.**

2650 CPU EXECUTION CONTROL SIGNALS

CPU execution control signals, being of primary importance, will be discussed first.

CLOCK is the master timing signal required by the 2650 CPU. Depending upon the way in which external logic is implemented, CLOCK may or may not be needed by other devices that surround the 2650; in most cases CLOCK will not be needed by other devices, since system control will normally be handled by 2650 control inputs and outputs.

RESET is the master reset input which every microcomputer has. As is standard for most microcomputers, when the CPU is reset, the Program Counter is cleared, with the result that the instruction stored in memory location 0 is executed. The CPU will typically be reset when first powered up.

PAUSE causes the CPU to enter a Wait state. PAUSE is an input signal which may be used by external direct memory access logic to stop the CPU while memory is being accessed. The Halt instruction also causes the CPU to enter the Wait state. A Wait state will be terminated by a Reset or by external logic removing its PAUSE input.

2650 BUS ACCESS CONTROL SIGNALS

There are two bus access control signals on the 2650, DBUSEN and $\overline{\text{ADREN}}$; these two signals float the Data and Address Busses, respectively. Observe that on the Address Bus, only the 13 Address Bus lines A0 - A12 are floated; the two page select lines A13 and A14 are not floated.

2650 BUS CONTENTS IDENTIFICATION SIGNALS

The most interesting feature of 2650 control signals is the scheme employed for identifying events on the Data and Address Busses.

The inception of any operation which will involve external devices is identified by OPREQ going high.

Normally the first step in any operation that involves external logic is for an address to be output on the Address Bus. **If memory is being accessed, then M/$\overline{\text{IO}}$ is output high. $\overline{\text{R}}$/W is output high to identify a write operation or low to identify a read operation.** As soon as memory has responded to the memory read or write operation, it inputs $\overline{\text{OPACK}}$ low. If $\overline{\text{OPACK}}$ low does not arrive in time for the CPU to continue processing the current instruction at the next clock cycle, then the CPU temporarily enters the Wait state and outputs RUN/$\overline{\text{WAIT}}$ low to indicate this condition. Now as soon as $\overline{\text{OPACK}}$ is input low, the Wait state will end and the CPU will continue execution.

The CPU will also output a write strobe, WRP, when writing to memory. This strobe is output when data is steady on the Data Bus.

When an I/O device is being accessed by one of the I/O instructions, M/$\overline{\text{IO}}$ is output low. You will see in Table 10-1 that the 2650 instruction set includes two sets of I/O instructions; one set does not identify an I/O port, and has a one-byte object code; the other set identifies an I/O port via a second byte of object code. Let us assume that the short I/O instructions will always reference I/O Port 0, while the long I/O instructions will specify one of 256 I/O ports. The E/$\overline{\text{NE}}$ signal, if low, identifies a short I/O instruction, therefore an instruction which accesses I/O Port 0; if high, this signal indicates that the current contents of the low order 8 address lines contains an I/O port address, and should be so decoded.

Once an I/O port has been selected, and external logic knows from the M/$\overline{IO}$ and E/$\overline{NE}$ controls which I/O port is selected, I/O logic needs to know whether an input or output I/O operation is to occur, and whether data or control/status information is to be transmitted. (Volume I, Chapter 5 discusses at length the difference between data, controls and status.) The $\overline{R}$/W control indicates whether data is being transmitted from the CPU to external devices, or whether external devices are supposed to transmit data to the CPU; then D/$\overline{C}$ identifies the output as either data or control information. Conversely, when $\overline{R}$/W identifies the CPU as requiring input from an I/O device, D/$\overline{C}$ indicates whether the input should be data or status.

When external device logic responds to the I/O request, it concludes by inputting $\overline{OPACK}$ low. Figure 10-3 illustrates how control signals may be used to interpret events on the Address and Data Busses.

2650 INTERRUPT CONTROL SIGNALS

2650 interrupt handling is very straightforward. An interrupt is requested by setting $\overline{INTREQ}$ low. The interrupt is acknowledged by the CPU outputting INACK high.

2650 EXTERNAL DEVICE CONTROL SIGNALS

The SENSE and FLAG signals allow the 2650 to directly control external devices. The condition of a SENSE input is immediately translated into a 0 or 1 within the Sense bit of the 2650 Status register. A 0 or 1 in the Flag bit of the 2650 Status register is immediately reflected by a low or high signal output at the Flag pin.

INTERFACING MEMORY TO THE 2650 MICROCOMPUTER

Given the wealth of control signals provided by the 2650 microcomputer, most types of memory can be interfaced with very little difficulty. The only peculiarities of the 2650 which external logic must be able to cope with is the fact that memory is paged into 8192-byte pages. Any memory device whose addressing range is smaller than a page must have select logic which takes into account not only high order address lines on the 13-line Address Bus but, in addition, the two page select lines. The two page select lines change status occasionally, when a new page is being selected; page select must be stored in an external buffer.

INTERFACING I/O DEVICES TO THE 2650 MICROCOMPUTER

The simplest way of interfacing external devices to the 2650 microcomputer is to use the microcomputer's I/O instructions, plus the control signals which identify I/O operations.

A very small microcomputer system may only have one I/O port. In this case the I/O port can connect directly to the Data Bus and can always consider itself selected. A larger system may have up to 256 8-bit ports, with select lines that simply connect to the Data Bus and use E/$\overline{NE}$ as a select enable signal.

THE 2650 MICROCOMPUTER INTERRUPT PROCESS

The 2650 has a single interrupt request line and a single interrupt acknowledge line. Interrupt priorities will therefore be handled via a daisy chain.

When the CPU acknowledges an interrupt, first it disables all further interrupts. Next, it pushes the contents of the Program Counter onto the address Stack and zeros the Program Counter.

The CPU will now insert the first byte of a ZBSR instruction code into the Instruction register; this instruction code is a Branch-to-Subroutine using program relative addressing. The interrupting device must submit a byte of data on the Data Bus, which will be interpreted as the second byte of the ZBSR instruction.

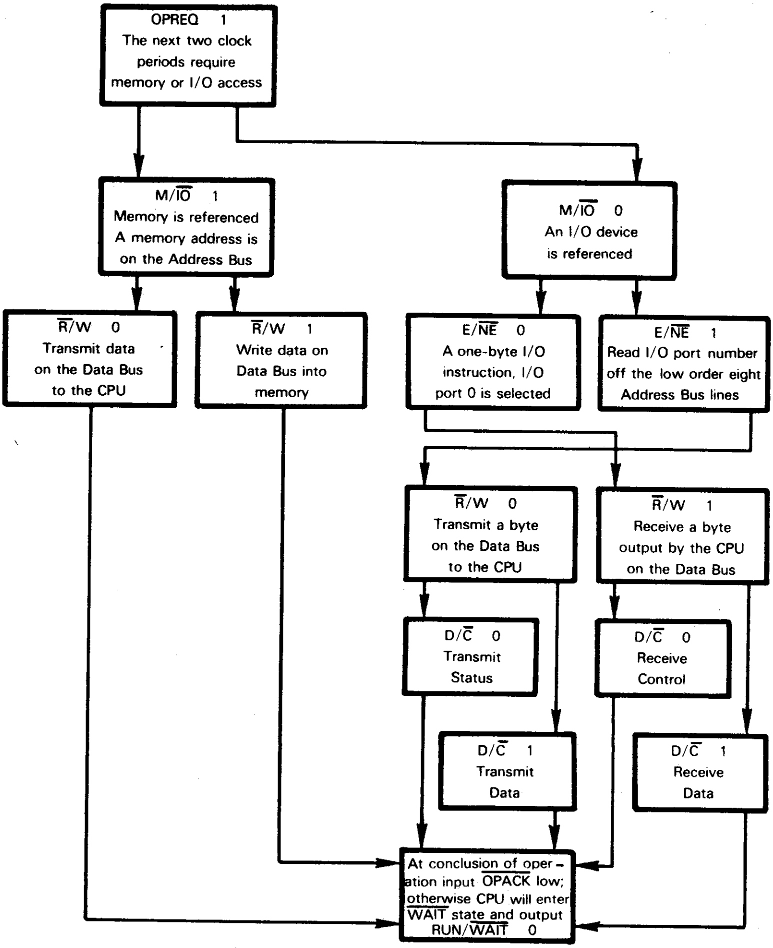
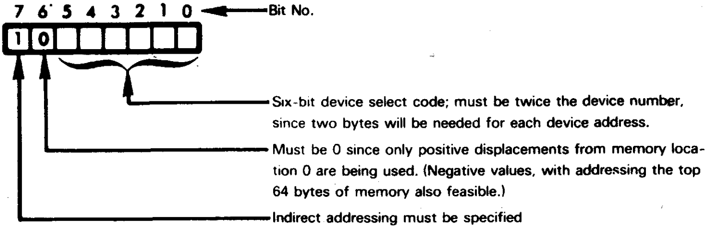

OPREQ 1
The next two clock periods require memory or I/O access
M/$\overline{\text{IO}}$ 1
Memory is referenced A memory address is on the Address Bus
M/$\overline{\text{IO}}$ 0
An I/O device is referenced
$\overline{\text{R}}$/W 0
Transmit data on the Data Bus to the CPU
$\overline{\text{R}}$/W 1
Write data on Data Bus into memory
E/$\overline{\text{NE}}$ 0
A one-byte I/O instruction, I/O port 0 is selected
E/$\overline{\text{NE}}$ 1
Read I/O port number off the low order eight Address Bus lines
$\overline{\text{R}}$/W 0
Transmit a byte on the Data Bus to the CPU
$\overline{\text{R}}$/W 1
Receive a byte output by the CPU on the Data Bus
D/$\overline{\text{C}}$ 0
Transmit Status
D/$\overline{\text{C}}$ 0
Receive Control
D/$\overline{\text{C}}$ 1
Transmit Data
D/$\overline{\text{C}}$ 1
Receive Data
At conclusion of operation input $\overline{\text{OPACK}}$ low; otherwise CPU will enter $\overline{\text{WAIT}}$ state and output RUN/$\overline{\text{WAIT}}$ 0

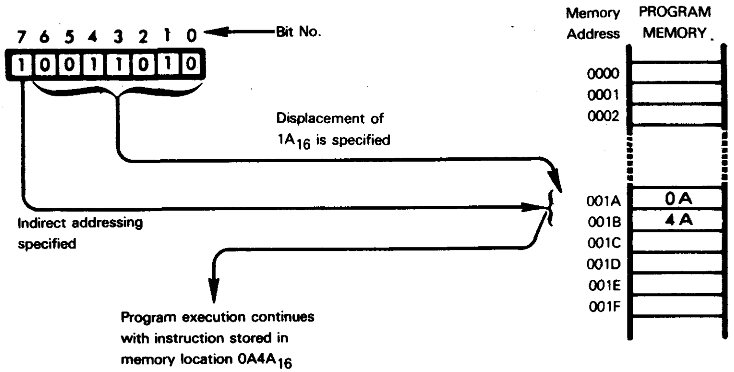
Figure 10-3. How Control Signals Identify Address And Data Bus Use For The 2650 Microcomputer

Look again at the discussion of 2650 addressing modes and you will see that with the Program Counter set to 0 the byte of data input by the interrupting device becomes a displacement vector.

Assume that each external device has the beginning address of its interrupt service routine stored somewhere within the first 64 bytes of the zero memory page. The interrupting device must input the following byte of data:


7 6 5 4 3 2 1 0
Bit No.
1 0
Six-bit device select code; must be twice the device number, since two bytes will be needed for each device address.
Must be 0 since only positive displacements from memory location 0 are being used. (Negative values, with addressing the top 64 bytes of memory also feasible.)
Indirect addressing must be specified

This byte of data causes an indirect program relative jump to the interrupting device's interrupt service routine, as follows:


7 6 5 4 3 2 1 0
Bit No.
1 0 0 1 1 0 1 0
Displacement of $1A_{16}$ is specified
Indirect addressing specified
Program execution continues with instruction stored in memory location $0A4A_{16}$
Memory Address
PROGRAM MEMORY
0000
0001
0002
001A
0A
001B
4A
001C
001D
001E
001F

2650 MICROCOMPUTER DIRECT MEMORY ACCESS

Direct memory access on a 2650 system is left up to external logic. Two schemes are possible.

External logic may stop the CPU, using the PAUSE input, and while the CPU is disabled, external logic may take control of Data and Address Busses to access memory in any way.

Alternatively, DMA logic may be implemented to operate in parallel with the CPU. The 2650 has periods when both the Data Bus and the Address Bus are floated. Handling DMA in parallel with normal instruction execution is made possible if you combine the OPREQ and OPACK handshake signals with normal timing sequences.

THE 2650 MICROCOMPUTER INSTRUCTION SET

The 2650 microcomputer instruction set is the most minicomputer-like of the microcomputers discussed in this book. It is particularly rich in addressing modes and memory reference instructions. The instruction set is listed in Table 10-1.

Memory reference instructions are shown as offering program relative addressing options, or extended addressing options. See the discussion of 2650 addressing options for a definition of these terms.

Note that in the statuses column, CC identifies the CC0 and CC1 statuses. These two statuses are used to test for a zero, positive or negative branch condition; these two statuses are described along with the 2650 Status registers.

The TMI Immediate Operate instruction compares a register's contents with a mask provided by the instruction operand. This instruction allows any bit combination to be tested for, in any CPU register.

The Decimal Adjust (DAR) instruction of the 2650 differs from the instructions with the same name as implemented on a number of other microcomputers. The Decimal Adjust instruction can be used to perform binary decimal arithmetic. Referring to the discussion of binary decimal arithmetic given in Volume I, the 2650 DAR instruction performs Step 3 of the binary-coded-decimal addition operation described in Chapter 3.

THE 2650 BENCHMARK PROGRAM

This is how the 2650 may implement our benchmark program:

```
        LODA,R1   TLENGTH       LOAD DISPLACEMENT TO FIRST FREE
                                TABLE BYTE
        LODA,R2   IOBFL         LOAD I/O BUFFER FILLED LENGTH
LOOP    LODA,R0   *IOBUF,R2     LOAD NEXT I/O BUFFER BYTE
        STRA,R0   *TABLE,R1, +  STORE IN TABLE, AUTO-INCREMENT R1
        BDRR,R2   LOOP          DECREMENT R2, RETURN TO LOOP ON NON-
                                ZERO
        STRA,R1   TLENGTH       AT END, RESTORE NEW TABLE LENGTH
```

The benchmark program, as illustrated for the 2650, assumes that both the data table and the I/O buffer have maximum lengths of 256 bytes.

The displacement to the first free byte of the data table is stored in a memory location identified by the label TLENGTH.

The number of filled I/O buffer bytes is stored in a memory location identified by the label IOBFL. It is assumed that the I/O buffer can be read backwards; in other words, the last I/O buffer byte becomes the first byte stored in the permanent data table.

The instruction with label LOOP begins by loading the last byte in the I/O buffer, using indirect, indexed addressing without auto-increment or auto-decrement. Subsequently, Index Register R2 is decremented and if it does not decrement to 0, execution returns to the instruction labeled LOOP.

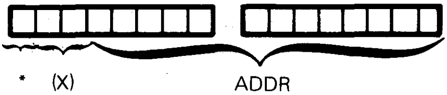
The instruction which stores data in TABLE uses indirect, post-indexed addressing, with the contents of the Index Register R1 auto-incremented. Thus, at the conclusion of data movement, Index Register R1 contains the displacement to the next free byte of TABLE.

Comparing the 2650 benchmark program with other benchmark programs shown in this book might suggest that the 2650 has the shortest, and therefore the fastest and most efficient benchmark program. This is not necessarily the case. Certainly the 2650 instruction set provides a source program which is likely to be shorter than any other microcomputer's source program, but that is because instructions are very minicomputer-like. The number of bytes required to implement the 2650 object program, and the time taken to execute the program, may bear no relationship to the length of the source program. For example, the program loop, although it contains only three instructions (LODA, STRA and BDRR), will require eight bytes of object program.

Once again, we caution against drawing fast conclusions from benchmark programs.

The following symbols are used in Table 10-1.

*ADDR(X) 16-bit extended addressing mode:

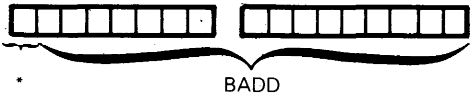


* 1 for indirection

(X) 00 for non-indexed
01 for indexed with auto-increment
10 for indexed with auto-decrement
11 for indexed only

ADDR
13-bit absolute address

*BADD 16-bit absolute addressing mode:

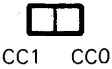


* is 1 for indirection

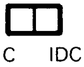
BADD
is a 15-bit absolute address

C Carry status

CC The two Condition Code bits CC1 and CC0



CIDC The Carry and Inter-Digit Carry

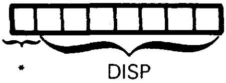


DATANE The non-extended data port

DATA2 2-bit data unit

DATA8 8-bit data unit

*DISP 8-bit relative addressing mode:



* is 1 for indirection

DISP is a 7-bit signed displacement

EAA Effective address generated by *BADD

EAD Effective address generated by *ADDR(X)

EAR PC relative address generated by *DISP

IDC Inter-Digit Carry status

O Overflow status

P An 8-bit port number

PC	Program Counter
PSU	Upper byte of Program Status Word
PSL	Lower byte of Program Status Word
r	One of the seven CPU registers
RAS(SP)	The Return Address Stack location indicated by the Stack Pointer.
R0	Accumulator
SP	Stack Pointer
Status NE	The Non-Extended status port
ZEA	A zero page relative address generated by DISP
x<y,z>	Bits y through z of the quantity x; for example, R0 <3,0> represents the lower 4 bits of the Accumulator
[ ]	Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If an I/O port number is enclosed within the brackets, then the I/O contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.
[[ ]]	Implied memory addressing; the contents of the memory location designated by the contents of a register.
∧	Logical AND
∨	Logical OR
∀	Logical Exclusive-OR
←	Data is transferred in the direction of the arrow
← →	Data is exchanged between the two locations designated on either side of the arrow.

Under the heading of STATUSES in Table 10-1, an X indicates statuses which are modified in the course of the instruction's execution. If there is no X, it means that the status maintains the value it had before the instruction was executed.

Table 10-1. Summary Of Signetics 2650 Instruction Set

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				C	O	IDC	CC	
I/O	REDD	,r	1				X	[r]←[dataNE] Read data at non-extended port into specified register.
	REDC	,r	1				X	[r]←[statusNE] Read non-extended status into specified register.
	REDE	,r P	2				X	[r]←[P] Read into specified register from Port P.
	WRTD	,r	1					[dataNE]←[r] Write specified register contents to non-extended data port.
	WRTC	,r	1					[statusNE]←[r] Write specified register contents to non-extended status port.
	WRTE	,r P	2					[P]←[r] Write specified register contents to Port P.
PRIMARY MEMORY REFERENCE	LODR	,r *DISP	2				X	[r]←[EAR] Load specified register from relative location.
	LODA	,r *ADDR(X)	3				X	[r]←[EAD] Load specified register from extended location.
	STRR	,r *DISP	2					[EAR]←[r] Store specified register contents in relative location.
	STRA	,r *ADDR(X)	3					[EAD]←[r] Store specified register contents in extended location.
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	ADDR	,r *DISP	2	X	X	X	X	[r]←[r] + [EAR] Add contents of relative location to specified register.
	ADDA	,r *ADDR(X)	3	X	X	X	X	[r]←[r] + [EAD] Add contents of extended location to specified register.
	SUBR	,r *DISP	2	X	X	X	X	[r]←[r] - [EAR] Subtract contents of relative location from specified register.
	SUBA	,r *ADDR(X)	3	X	X	X	X	[r]←[r] - [EAD] Subtract contents of extended location from specified register.

Table 10-1. Summary Of Signetics 2650 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				C	O	IDC	CC	
SECONDARY MEMORY REFERENCE (MEMORY OPERATE) (CONTINUED)	ANDR	,r *DISP	2				X	[r]←[r] ∧ [EAR] AND contents of relative location with those of specified register.
	ANDA	,r *ADDR(X)	3				X	[r]←[r] ∧ [EAD] AND contents of extended location with those of specified register.
	IORR	,r *DISP	2				X	[r]←[r] ∨ [EAR] OR contents of relative location with those of specified register.
	IORA	,r *ADDR(X)	3				X	[r]←[r] ∨ [EAD] OR contents of extended location with those of specified register.
	EORR	,r *DISP	2				X	[r]←[r] ⊻ [EAR] Exclusive-OR contents of relative location with those of specified register.
	EORA	,r *ADDR(X)	2				X	[r]←[r] ⊻ [EAD] Exclusive-OR contents of extended location with those of specified register.
	COMR	,r *DISP	2				X	If [r] > [EAR]; then CC = 01 If [r] = [EAR]; then CC = 00 If [r] < [EAR]; then CC = 10 Compare contents of relative location with those of specified register; set the CC accordingly.
	COMA	,r *ADDR(X)	2				X	If [r] > [EAD]; then CC = 01 If [r] = [EAD]; then CC = 00 If [r] < [EAD]; then CC = 10 Compare contents of extended location with those of specified register; set the CC accordingly.
IMMEDIATE	LODI	,r DATA8	2				X	[r]←DATA8 Load immediate into specified register.

Table 10-1. Summary Of Signetics 2650 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				C	O	IDC	CC	
IMMEDIATE OPERATE	ADDI	,r DATA8	2	X	X	X	X	[r]←[r] + DATA8 *Add immediate to specified register contents.*
	SUBI	,r DATA8	2	X	X	X	X	[r]←[r] - DATA8 Subtract immediate from specified registers contents.
	ANDI	,r DATA8	2				X	[r]←[r] ∧ DATA8 AND immediate with specified register contents.
	IORI	,r DATA8	2				X	[r]←[r] ∨ DATA8 OR immediate with specified register contents.
	EORI	,r DATA8	2				X	[r]←[r]] ⊻ DATA8 Exclusive-OR immediate with specified register contents.
	COMI	,r DATA8	2				X	If [r] > DATA8; [CC]←01 If [r] = DATA8; [CC]←00 If [r] < DATA8; [CC]←10 Compare immediate with specified register; set the CC accordingly.
	TMI	,r DATA8	2				X	If all selected bits are set, CC = 00; otherwise CC = 10 Test bits in specified register corresponding to 1s in immediate data. If all tested bits are 1s set CC accordingly.
JUMP	ZBRR	*DISP	2					[PC]←ZEA Branch to zero page address.
	BXA	*BADD	3					[PC]←EAA Branch to extended address.
	ZBSR	*DISP	2					[SP]←[SP] + 1 [RAS(SP)]←[PC] + 2 [PC]←ZEA *Call zero page subroutine.*
	BSXA	*BADD	3					[SP]←[SP] + 1 [RAS(SP)]←[PC] + 3 [PC]←EAA Call extended subroutine.

Table 10-1. Summary Of Signetics 2650 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				C	O	IDC	CC	
BRANCH ON CONDITION	BCTR	,DATA2 *DISP	2					If DATA2 = CC, then [PC]← EAR Branch relative if DATA2 equals CC.
	BCTA	,DATA2 *DISP	3					If DATA2 = CC, then [PC]← EAA Branch absolute if DATA2 equals CC.
	BCFR	,DATA2 *DISP	2					If DATA2 ≠ CC, then [PC]← EAR Branch relative if DATA2 is not equal to CC.
	BCFA	,DATA2 *BADD	3					If DATA2 ≠ CC, then [PC]← EAA Branch absolute if DATA2 is not equal to CC.
	BIRR	,r *DISP	2					[r]←[r] + 1 If [r] ≠ 0, [PC]← EAR Increment specified register. If nonzero result, branch relative.
	BIRA	,r *BADD	3					[r]←[r] + 1 If [r] ≠ 0, then [PC]← EAA Increment specified register. If nonzero result, branch absolute.
	BDRR	,r *DISP	2					[r]←[r] - 1 If [r] ≠ 0, then [PC]← EAR Decrement specified register. If nonzero result, branch relative.
	BDRA	,r *BADD	3					[r]←[r] - 1 If [r] ≠ 0; then [PC]← EAA Decrement specified register. If nonzero result, branch absolute.
	BRNR	,r *DISP	2					If [r] ≠ 0; then [PC]← EAR If specified register is nonzero, branch relative.
	BRNA	,r *BADD	3					If [r] ≠ 0; then [PC]← EAA If specified register is nonzero, branch absolute.
CONDITIONAL SUBROUTINE BRANCH	BSTR	,DATA2 *DISP	2					If DATA2 = CC; then [SP]←[SP] + 1 [RAS(SP)]←[PC] + 2 [PC]← EAR If DATA2 equals CC, then call subroutine at relative address.

Table 10-1. Summary Of Signetics 2650 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				C	O	IDC	CC	
CONDITIONAL SUBROUTINE BRANCH (CONTINUED)	BSTA	,DATA2 *BADD	3					If DATA2 = CC; then [SP]←[SP] + 1 [RAS(SP)]←[PC] + 3 [PC]←EAA If DATA2 equals CC, then call subroutine at absolute address.
	BSFR	,DATA2 *DISP	2					If DATA2 ≠ CC; then [SP]←[SP] + 1 [RAS(SP)]←[PC] + 2 [PC]←EAR If DATA2 not equal to CC, then call subroutine at relative address.
	BSFA	,DATA2 *BADD	3					If DATA2 ≠ CC; then [SP]←[SP] + 1 [RAS(SP)]←[PC] + 3 [PC]←EAA If DATA2 not equal CC, call subroutine at absolute address.
	BSNR	,r *DISP	2					If [r] ≠ 0; then [SP]←[SP] + 1 [RAS(SP)]←[PC] + 2 [PC]←EAR If specified register is nonzero, call subroutine at relative address.
	BSNA	,r *BADD	3					If [r] ≠ 0; then [SP]←[SP] + 1 [RAS(SP)]←[PC] + 3 [PC]←EAA If specified register is nonzero, call subroutine at absolute address.
	RETC	,DATA2	1					If DATA2 = CC, then [PC]←[RAS(SP)] [SP]←[SP] - 1 If DATA2 equals CC, then return from subroutine.
REGISTER-REGISTER MOVE	LODZ	r	1				X	[R0]←[r] Load Accumulator (Register 0) with specified register contents.
	STRZ	r	1				X	[r]←[R0] Store contents of Accumulator (Register 0) into specified register.

Table 10-1. Summary Of Signetics 2650 Instruction Set (Continued)

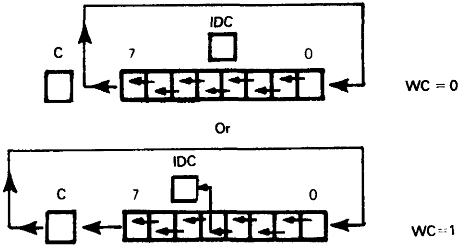
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				C	O	IDC	CC	
REGISTER-REGISTER OPERATE	ADDZ	,r	1	X	X	X	X	[R0]←[R0] + [r] Add specified register to Register 0.
	SUBZ	,r	1	X	X	X	X	[R0]←[R0] - [r] Subtract specified register from Register 0.
	ANDZ	,r	1				X	[R0]←[R0] Λ [r] AND specified register with Register 0.
	IORZ	,r	1				X	[R0]←[R0] V [r] OR specified register with Register 0.
	EORZ	,r	1				X	[R0]←[R0] ⊻ [r] Exclusive-OR specified register with Register 0.
	COMZ	,r	1				X	If [R0] > [r]; then CC = 01 If [R0] = [r]; then CC = 00 If [R0] < [r]; then CC = 10 Compare specified register with Register 0; set the CC accordingly.
REGISTER OPERATE	RRL	,r	1	X	X	X	X	 IDC C 7 0 WC = 0 Or IDC C 7 0 WC = 1 If WC is 0, rotate the specified register left. If WC is 1, rotate through Carry and Intermediate Carry.

Table 10-1. Summary Of Signetics 2650 Instruction Set (Continued)

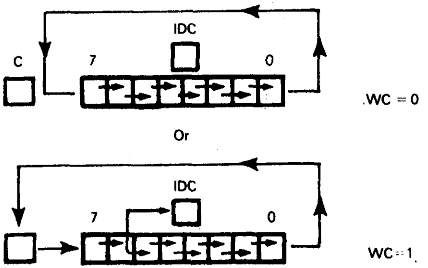
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				C	O	IDC	CC	
REGISTER OPERATE (CONTINUED)	RRR	,r	1	X	X	X	X	 IDC C 7 0 WC = 0 Or IDC 7 0 WC = 1 If WC is 0, rotate the specified register right. If WC is 1, rotate through Carry and Intermediate Carry.
	DAR	,r	1					Decimal adjust the specified register.
INTERRUPT	RTE	,DATA2	1					If DATA2 = CC; then [PC]←[RAS(SP)] [SP]←[SP] - 1 Enable interrupts If DATA2 equals CC, then return from subroutine and enable interrupts.
STATUS	LPSU		1					[PSU]←[R0] Load Register 0 into PSU.
	LPSL		1					[PSL]←[R0] Load Register 0 into PSL.
	SPSU		1					[R0]←[PSU] Load PSU into Register 0.

Table 10-1. Summary Of Signetics 2650 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES				OPERATION PERFORMED
				C	O	IDC	CC	
STATUS (CONTINUED)	SPSL		1					[R0]←[PSL] Load PSL into Register 0.
	PPSU	DATA8	2					If [DATA8<i>]=1; then [PSU<i>]←1 Set bits in PSU which correspond to 1s in immediate data.
	PPSL	DATA8	2					If [DATA8<i>]=1; then [PSL<i>]←1 Set bits in PSL which correspond to 1s in immediate data.
	CPSU	DATA8	2					If [DATA8<i>]=1 then [PSU<i>]←0 Clear bits of PSU which correspond to 1s in immediate data.
	CPSL	DATA8	2					If [DATA8<i>]=1 then [PSL<i>]←0 Clear bits of PSL which correspond to 1s in immediate data.
	TPSU	DATA8	2				X	If DATA8 = [PSU] then CC = 00, else CC = 10 Compare immediate with PSU; set CC accordingly.
	TPSL	DATA8	2				X	If DATA8 = [PSL]; then CC = 00, else CC = 10 Compare immediate with PSL; set CC accordingly.
	NOP		1					No Operation.
	HALT		1					Processor enters Wait state.

The following symbols are used in Table 10-2.

aa	Two bits which, in conjunction with the Register Bank Select bit in the PSL, choose the register.
b	One bit selecting the indirection option.
cc	Two bits choosing the indexing mode: 00 No indexing 01 Indexing with auto-increment 10 Indexing with auto-decrement 11 Indexing only
eeeeeee	7-bit signed address displacement
ff	2-bit test value
PP	8 bits of immediate data
q	One bit of absolute or extended address
Q	One byte (8 bits) of absolute or extended address

Table 10-2. Signetics 2650 Instruction Object Codes

INSTRUCTION	OBJECT CODE	BYTES	MACHINE CYCLES
ADDA,r *ADDR(X)	100011aa bccqqqqq QQ	3	4
ADDI,r DATA8	100001aa PP	2	2
ADDR,r *DISP	100010aa beeeeeee	2	3
ADDZ,r	100000aa	1	2
ANDA,r *ADDR(X)	010011aa bccqqqqq QQ	3	4
ANDI,r DATA8	010001aa PP	2	2
ANDR,r *DISP	010010aa beeeeeee	2	3
ANDZ,r	010000aa	1	2
BCFA,DATA2 *BADD	100111ff bqqqqqqq QQ	3	3
BCFR,DATA2 *DISP	100110ff beeeeeee	2	3
BCTA,DATA2 *BADD	000111ff bqqqqqqq QQ	3	3
BCTR,DATA2 *DISP	000110ff beeeeeee	2	3
BDRA,r *BADD	111111aa bqqqqqqq QQ	3	3
BDRR,r *DISP	111110aa beeeeeee	2	3
BIRA,r *BADD	110111aa bqqqqqqq QQ	3	3
BIRR,r *DISP	110110aa beeeeeee	2	3

Table 10-2. Signetics 2650 Instruction Object Codes (Continued)

INSTRUCTION	OBJECT CODE	BYTES	MACHINE CYCLES
BRNA,r *BADD	010111aa bqqqqqqq QQ	3	3
BRNR,r *DISP	010110aa beeeeeee	2	3
BSFA,DATA2 *BADD	101111ff bqqqqqqq QQ	3	3
BSFR,DATA2 *DISP	101110ff beeeeeee	2	3
BSNA,r *BADD	011111aa bqqqqqqq QQ	3	3
BSNR,r *DISP	011110aa beeeeeee	2	3
BSTA,DATA2 *BADD	001111ff bqqqqqqq QQ	3	3
BSTR,DATA2 *DISP	001110ff beeeeeee	2	3
BSXA *BADD	BF bqqqqqqq QQ	3	3
BXA *BADD	9F bqqqqqqq QQ	3	3
COMA,r *ADDR(X)	111011aa bccqqqqq QQ	3	4
COMI,r DATA8	111001aa PP	2	2
COMR,r *DISP	111010aa beeeeeee	2	3
COMZ,r	111000aa	1	2
CPSL DATA8	75 PP	2	3
CPSU DATA8	74 PP	2	3
DAR,r	100101aa	1	3
EORA,r *ADDR(X)	001011aa bccqqqqq QQ	3	4
EORI,r DATA8	001001aa PP	2	2
EORR,r *DISP	001010aa beeeeeee	2	3
EORZ,r	001000aa	1	2
HALT	40	1	2
IORA,r *ADDR(X)	011011aa bccqqqqq QQ	3	4
IORI,r DATA8	011001aa PP	2	2
IORR,r *DISP	011010aa beeeeeee	2	3

Table 10-2. Signetics 2650 Instruction Object Codes (Continued)

INSTRUCTION	OBJECT CODE	BYTES	MACHINE CYCLES
IORZ,r	011000aa	1	2
LODA,r *ADDR(X)	000011aa bccqqqqq QQ	3	4
LODI,r DATA8	000001aa PP	2	2
LODR,r *DISP	000010aa beeeeeee	2	3
LODZ,r	000000aa	1	2
LPSL	93	1	2
LPSU	92	1	2
NOP	C0	1	2
PPSL DATA8	77 PP	1	3
PPSU DATA8	76 PP	2	3
REDC,r	001100aa	1	2
REDD,r	011100aa	1	2
REDE,r P	010101aa PP	2	3
RETC,DATA2	000101ff	1	3
RETE,DATA2	001101ff	1	3
RRL,r	110100aa	1	2
RRR,r	010100aa	1	2
SPSL	13	1	2
SPSU	12	1	2
STRA,r *ADDR(X)	110011aa bccqqqqq QQ	3	4
STRR,r *DISP	110010aa beeeeeee	2	3
STRZ,r	110000aa	1	2
SUBA,r *ADDR(X)	101011aa bccqqqqq QQ	3	4
SUBI,r DATA8	101001aa PP	2	2
SUBR,r *DISP	101010aa beeeeeee	2	3
SUBZ,r	101000aa	1	2
TMI,r DATA8	111101aa PP	2	3
TPSL DATA8	B5 PP	2	3
TPSU DATA8	B4 PP	2	3
WRTC,r	101100aa	1	2
WRTD,r	111100aa	1	2
WRTE,r P	110101aa PP	2	3
ZBRR *DISP	9B beeeeeee	2	3
ZBSR *DISP	BB beeeeeee	2	3

SUPPORT DEVICES THAT MAY BE USED WITH THE 2650 MICROPROCESSOR

Interfacing the 2650 with 8080A support devices is very straightforward. Figure 10-4 shows how 8080A control signals may be generated from 2650 control signals. Figure 10-5 provides the same information for the MC6800.

But there are some ambiguities not immediately apparent when you look at Figure 10-4. To begin with, the 2650 uses a request/acknowledge handshaking control protocol which is alien to an 8080A-based system. Thus $\overline{\text{OPACK}}$, which is shown creating RDYIN in Figure 10-4, may well be grounded in a configuration that is not going to insert Wait states into 2650 instruction execution cycles. OPREQ will be used as a contributor to the chip select logic of 8080A support devices. M/$\overline{\text{IO}}$, which is shown discriminating between memory and I/O control signals in Figure 10-4, may alternatively be used as a contributor to chip select logic. **Figures 10-6 through 10-9 illustrate 8251 and 8255 devices connected to a 2650 CPU, being selected within memory or I/O spaces.** Note that where devices are selected within the 2650 I/O space, C/$\overline{\text{D}}$ could be generated from the 2650 D/$\overline{\text{C}}$ control output rather than using address line ADR0.

Figure 10-10 shows how 2650 priority interrupts may be generated using an 8214 Priority Interrupt Control Unit.

Interfacing MC6800 support devices to a 2650 CPU is again complicated by the synchronizing signal required by MC6800 support devices. But the 2650 is flexible enough to make this interface possible.

We must use OPREQ in order to generate the synchronizing enable signal for MC6800 support devices. Unfortunately, there is a significant variation in the leading edge of OPREQ. Therefore **logic to create an ENABLE synchronizing signal must have the following three parts:**

1) Create a continuous clock signal to substitute for the MC6800 ENABLE synchronizing signal.
2) Make sure that during a write cycle MC6800 device select logic is true across one pulse of the ENABLE signal. Chip select logic must be true from shortly before the beginning of the enable signal positive transition until shortly after the end of the negative transition.
3) During a read cycle, again make sure that chip select logic for the MC6800 support device is valid for one ENABLE cycle only; but this time stretch the ENABLE true pulse so that the 2650 CPU can latch the data on the negative transition of OPREQ before ENABLE goes low.

Timing for the above three conditions is illustrated in Figure 10-11. But note that since the minimum cycle time for MC6800 support devices is 1 microsecond, the 2650 CPU must also operate at this frequency — rather than using a 0.8 microsecond clock which is the fastest allowed.

Figure 10-2 illustrates a 2650 — 6850 ACIA interface. Figure 10-13 illustrates a 2650 — 6820 PIA interface.

Important aspects of 2650 interface timing are defined in Figure 10-14.

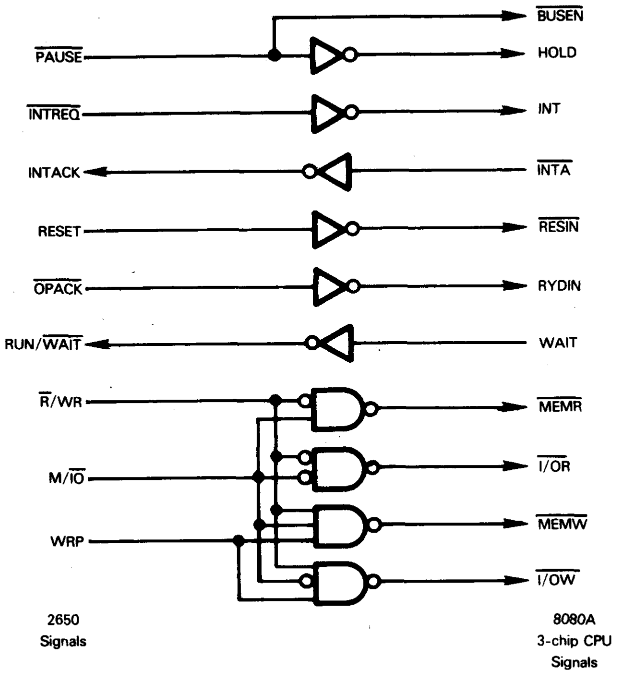


Figure 10-4. 2650 — 8080A Signal Equivalents

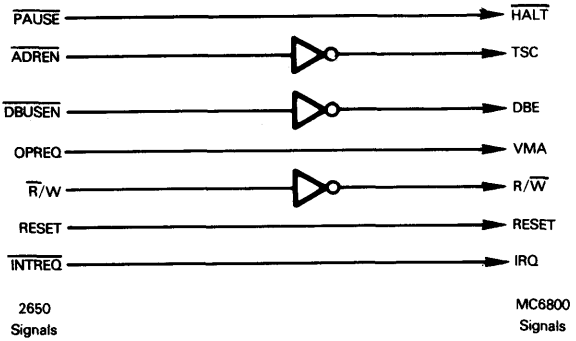


Figure 10-5. 2650 — MC6800 Signal Equivalents

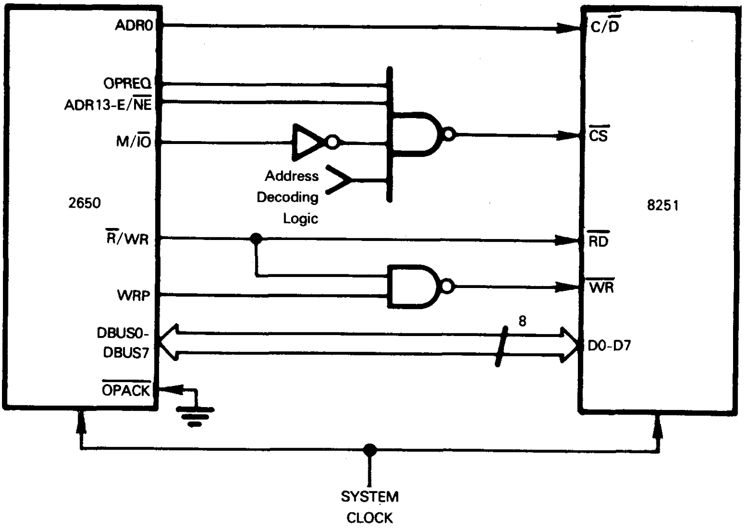


Figure 10-6. An 8251 USART Accessed By A 2650 As An I/O Device

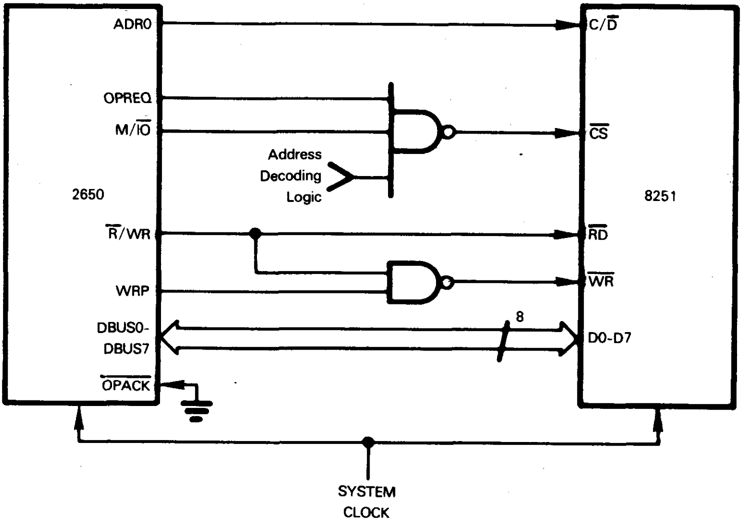


Figure 10-7. An 825l USART Accessed By A 2650 As A Memory Device

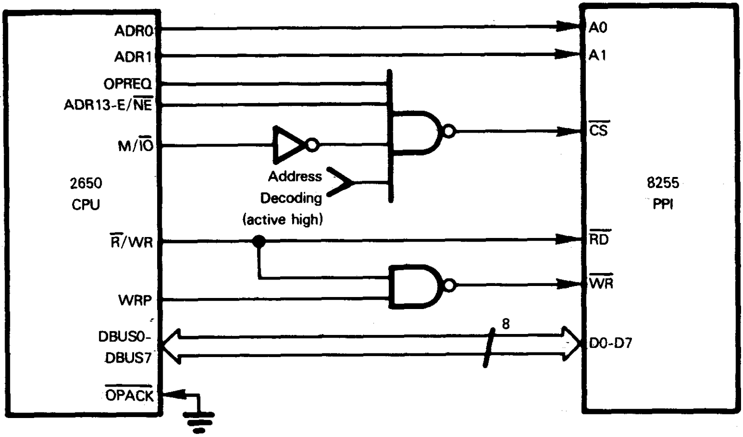


Figure 10-8. An 8255 PPI Accessed By A 2650 As An I/O Device

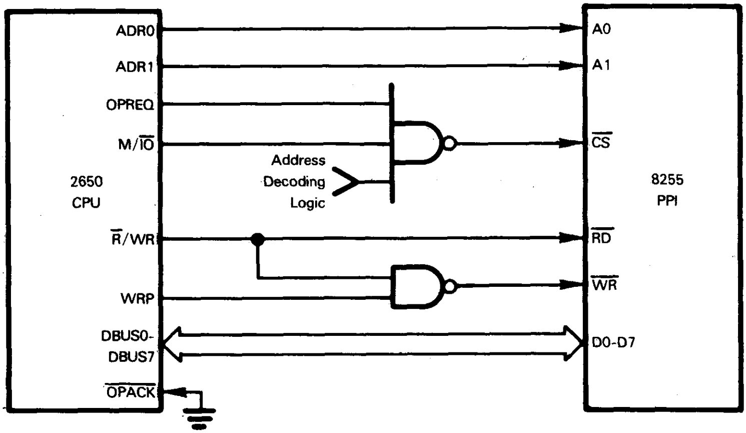


Figure 10-9. An 8255 PPI Accessed By A 2650 As A Memory Device

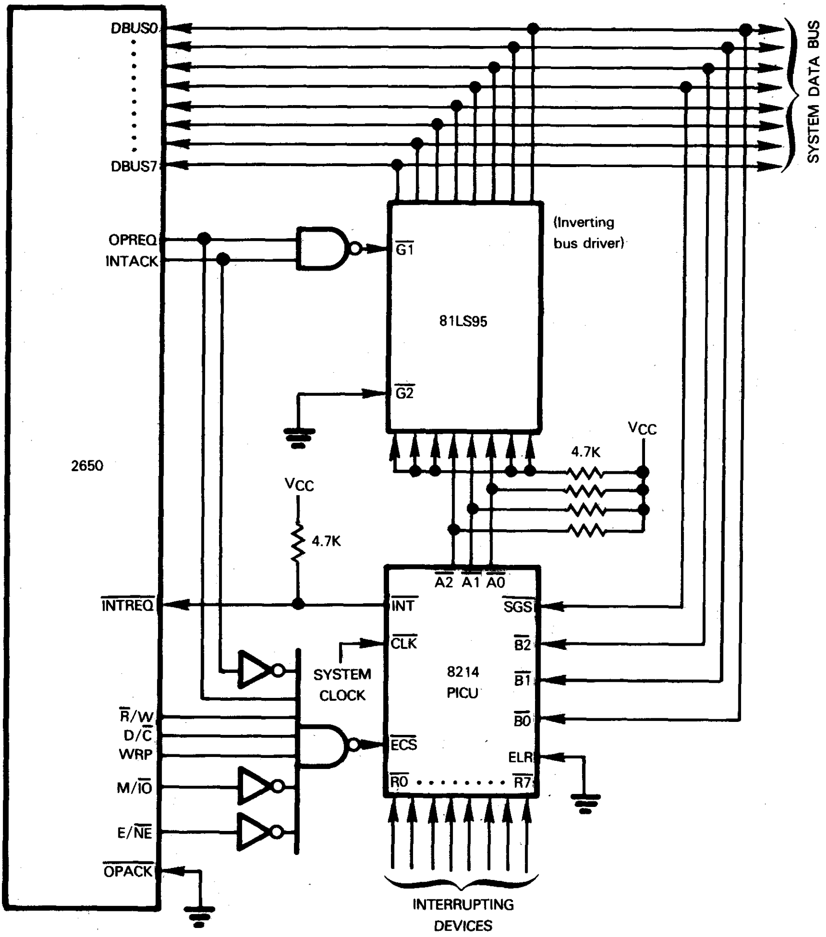


Figure 10-10. Vectored Interrupt Using The 8214 PICU With A 2650 CPU

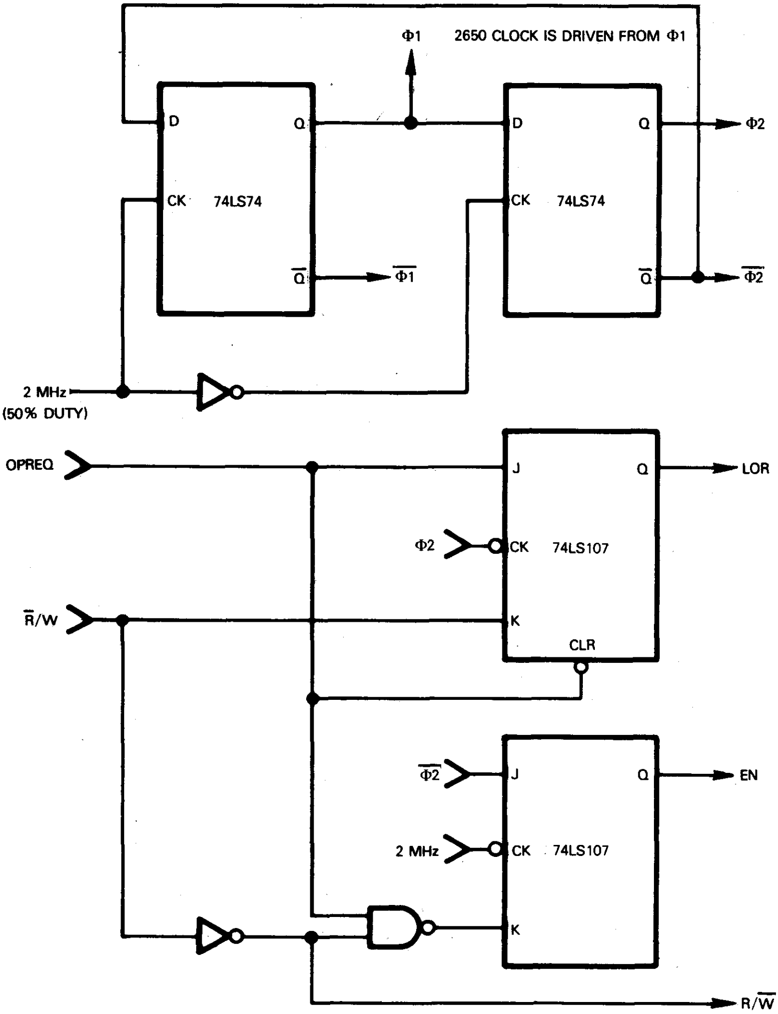


Figure 10-11. Synchronization Circuits In A 2650 — MC68XX Interface

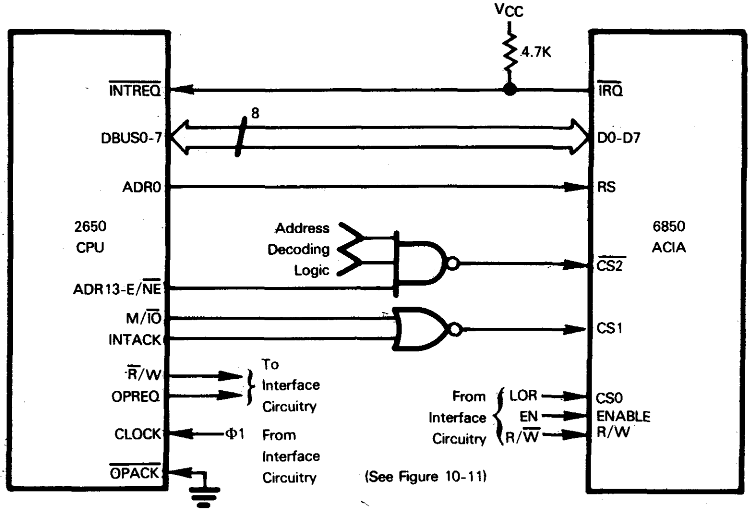


Figure 10-12. An MC6850 ACIA Connected To A 2650

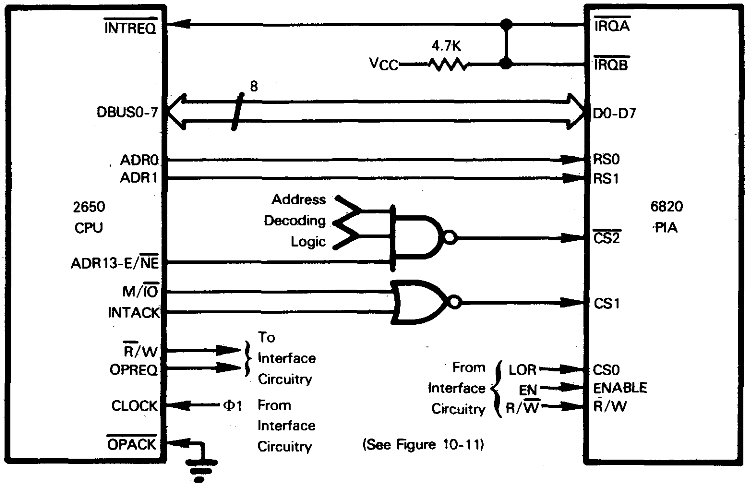
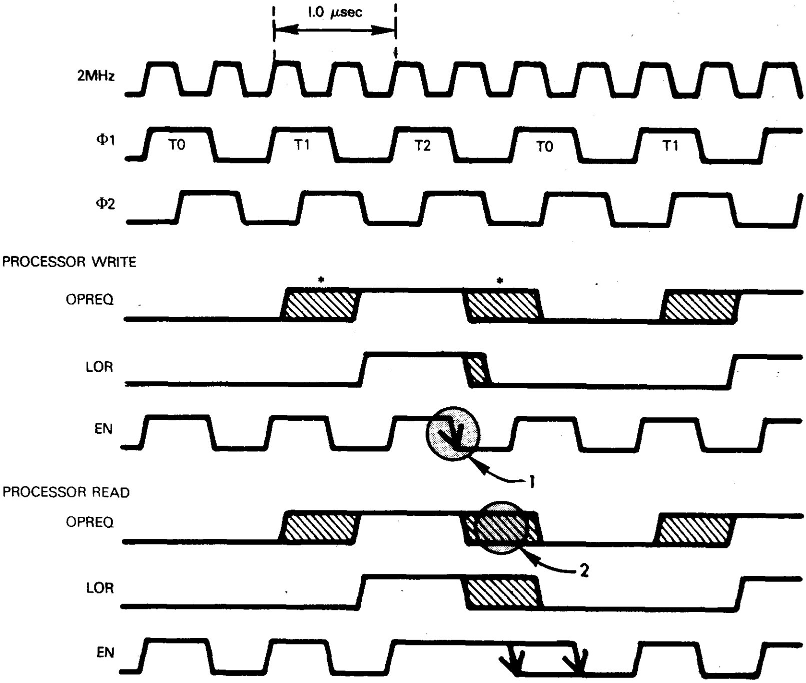


Figure 10-13. An MC6820 PIA Connected To A 2650



1) MC68XX latches data internally on negative transition.

2) Processor latches data on the negative transition of OPREQ; thereafter LOR and EN go to zero (but NOT before).

*OPREQ can make a transition any time within this 600 nsec. region.

Figure 10-14. Important Timing Considerations When Interfacing A 2650 CPU With MC68XX Series Devices

ELECTRICAL DATA

Here are specific electrical characteristics of the Signetics 2650.

ELECTRICAL CHARACTERISTICS

MAXIMUM GUARANTEED RATINGS[1]

Operating Ambient Temperature	0°C to +70°C
Storage Temperature	-65°C to + 150°C
All Input, Output, and Supply Voltages with respect to ground pin[3]	-0.5V to +6V
Package Power Dissipation[2]=IWPkg.	1.6W

PRELIMINARY 2650 DC ELECTRICAL CHARACTERISTICS

			LIMITS		
SYMBOL	PARAMETER	TEST CONDITIONS	MIN	MAX	UNIT
I_{LI}	Input Load Current	V_{IN} = 0 to 5.25V		10	μA
I_{LOH}	Output Leakage Currenr	ADREN, DBUSEN = 2.2V, V_{OUT} = 4V		10	μA
I_{LOL}	Output Leakage Current	ADREN, DBUSEN = 2.2V, V_{OUT} = 0.45V		10	μA
I_{CC}	Power Supply Current	V_{CC} = 5.25V, T_A = 0°C		100	mA
V_{IL}	Input Low		-0.6	0.8	V
V_{IH}	Input High		2.2	V_{CC}	V
V_{OL}	Output Low	I_{OL} = 1.6 mA	0.0	0.45	V
V_{OH}	Output High	I_{OH} = -100 μA	2.4	V_{CC}-0.5	V
C_{IN}	Input Capacitance	V_{IN} = 0V		10	pF
C_{OUT}	Output Capacitance	V_{OUT} = 0V		10	pF

Conditions: T_A = 0°C to 70°C, V_{CC} = 5V ±5%

NOTES:

1. Stresses above those listed under "Maximum Guaranteed Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operation sections of this specification is not implied.
2. For operating at elevated temperatures the device must be derated based on a +150°C maximum junction temperature and a thermal resistance of 50°C/W junction to ambient (40 pin IW package).
3. This product includes circuitry specifically designed for the protection of its internal devices from the damaging effects of excessive static charge. Nonetheless, it is suggested that conventional precautions be taken to avoid applying any voltages larger than the rated maxima.
4. Parameter valid over operating temperature range unless otherwise specified.
5. All voltage measurements are referenced to ground.
6. Manufacturer reserves the right to make design and process changes and improvements.
7. Typical values are at +25°C, nominal supply voltages, and nominal processing parameters.

PRELIMINARY AC CHARACTERISTICS

T_A=0°C to 70°C V_{CC}=5V±5% unless otherwise specified, see notes 1,2,3 & 4.

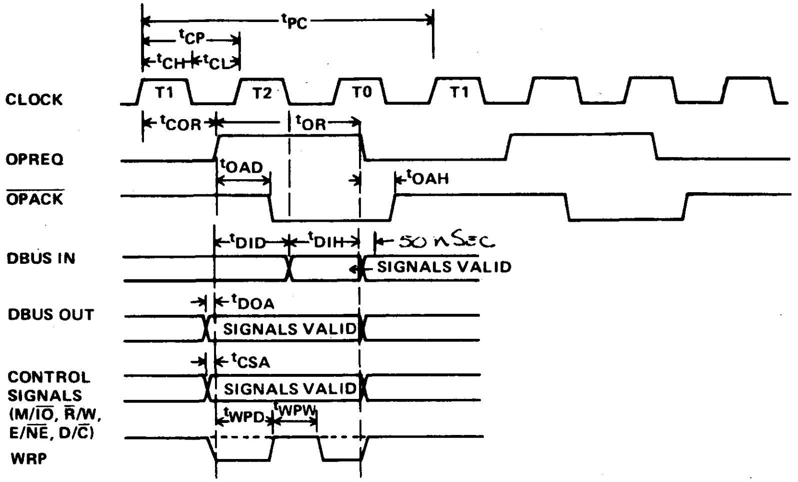
SYMBOL	PARAMETER	LIMITS		UNITS
		MIN	MAX	
t_{CH}	Clock High Phase	400	10,000	nsec
t_{CL}	Clock Low Phase	400	∞	nsec
t_{CP}	Clock Period	800	∞	nsec
t_{PC}[6]	Processor Cycle Time	2,400	∞	nsec
t_{OR}	OPREQ Pulse Width	$2t_{CH} + t_{CL} - 100$	∞	nsec
t_{COR}	Clock to OPREQ Time	100	700	nsec
t_{OAD}[7]	$\overline{\text{OPACK}}$ Delay Time	0	∞	nsec
t_{OAH}	$\overline{\text{OPACK}}$ Hold Time	0	∞	nsec
t_{CSA}	Control Signal Available	50		nsec
t_{DOA}	Data Out Available	50		nsec
t_{DID}[8]	Data in Delay	0	1000(8)	nsec
t_{DIH}[9]	Data in Hold	150		nsec
t_{WPD}	Write Pulse Delay	t_{CL}-100	t_{CL}-50	nsec
t_{WPW}	Write Pulse Width	t_{CL}	t_{CL}	nsec
t_{ABD}	Address Bus Delay		80	nsec
t_{DBD}	Data Bus Delay		120	nsec
t_{IRS}[10]	$\overline{\text{INTREQ}}$ Set up Time	0		nsec
t_{IRH}[10]	$\overline{\text{INTREQ}}$ Hold Time	0		nsec
t_{ORT}[5]	Output Buffer Rise Time		150	nsec

NOTES ON AC CHARACTERISTICS

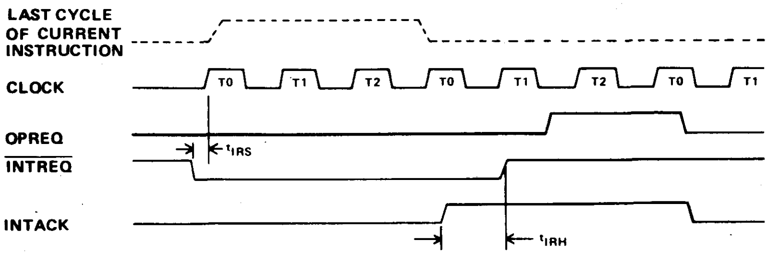
1. See preceding timing diagrams for definition of timing terms.
2. Input levels swing between 0.65 volt and 2.2 volts.
3. *Input signal transition times are 20ns.*
4. Timing reference level is 1.5 volts.
5. Load is -100μA at 20pF.
6. A Processor Cycle time consists of three clock periods.
7. In order to avoid slowing down the processor, $\overline{\text{OPACK}}$ must be lowered 100ns before the trailing edge of T2 clock, if $\overline{\text{OPACK}}$ is delayed past this point, the processor will wait in the T2 state and sample $\overline{\text{OPACK}}$ on each subsequent negative clock edge until $\overline{\text{OPACK}}$ is lowered.
8. In order to avoid slowing the processor down, input data must be returned to the processor in 1μs or less time from the OPREQ edge, at a cycle time of 2.4μs.
9. *Input data must be held until 50ns after OPREQ falls.*
10. In order to interrupt the current instruction, $\overline{\text{INTREQ}}$ must fall prior to the first clock of the last cycle of the current instruction. $\overline{\text{INTREQ}}$ must remain low until INTACK goes high.

CRITICAL TIMES

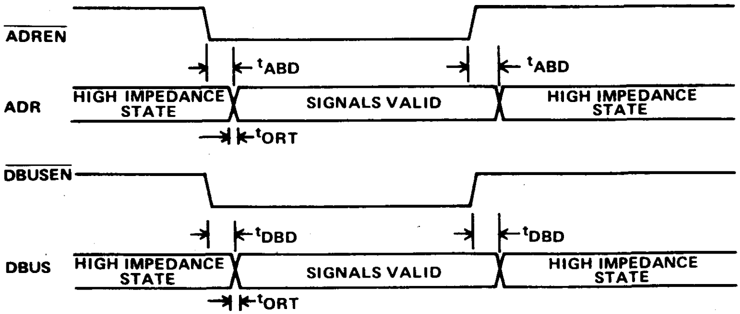
The following timing diagram describes the timing relationship between the various interface signals. The critical times are labeled and defined in the table of AC characteristics.



GENERAL TIMING



INTERRUPT TIMING



TRI-STATE BUS TIMING

2650 TIMING DIAGRAMS

Chapter 11
THE RCA COSMAC

We are going to describe the single chip CPU referred to as the CDP1802. This is a one-chip implementation of the previous two-chip CPU, consisting of the CDP1801 and CDP18101.

COSMAC and the F8 are the two most remarkable 8-bit microprocessors described in this book, when viewed as digital logic replacement devices. Both COSMAC and the F8 look very unappealing to the traditional minicomputer programmer, yet are powerful digital logic implementation devices.

Having classified COSMAC and the F8 together, how do the two devices compare against each other?

COSMAC and the F8 are both "low end" devices; that is, both are devices well suited to simple, high volume applications with limited programming needs. As compared to many other microprocessors described in this book, both COSMAC and the F8 are poor choices for low volume, program intensive applications; that is because both COSMAC and the F8 are relatively difficult to program.

But where does the transition from a simple application to a complex application occur?

For the F8, the transition tends to be slow. For COSMAC, it is sudden — an application is, or is not suited to COSMAC, with very little grey area.

The principal advantage of COSMAC is that it requires very little power, since it is fabricated using CMOS technology. If your application is going to be battery powered for any length of time, CMOS logic is strongly favored.

Both the power and the inflexibility of COSMAC are based on a subtly clever use of CPU logic, coupled with a somewhat primitive interface between CPU and external memory. Providing you can accommodate all program housekeeping using CPU registers for your read/write memory, COSMAC is a superb microprocessor. Program housekeeping in this case includes the program and data memory address maintenance associated with subroutines, interrupts and data accesses in general. There is a very large class of microprocessor applications that fit well within these restrictions and are well suited to COSMAC.

COSMAC is fabricated using CMOS technology. It operates with a single power supply and is very insensitive to noise. The power supply can vary between +3V and +12V.

CMOS technology also results in COSMAC having a very low power consumption and a broad operating temperature range. It is one of the few products described in this book that operates within the full military specification temperature range of -55°C to +125°C.

Using a +10V power supply, a 155 nanosecond clock results in instruction execution times of 2.5 or 3.75 microseconds.

The principal manufacturer for the COSMAC is:

RCA SOLID STATE DIVISION
P. O. Box 3200
Somerville, N.J. 08876

The second source is:

HUGHES AIRCRAFT INC.
Industrial Electronics Group
500 Superior Avenue
Newport Beach, CA 92663

THE COSMAC CPU

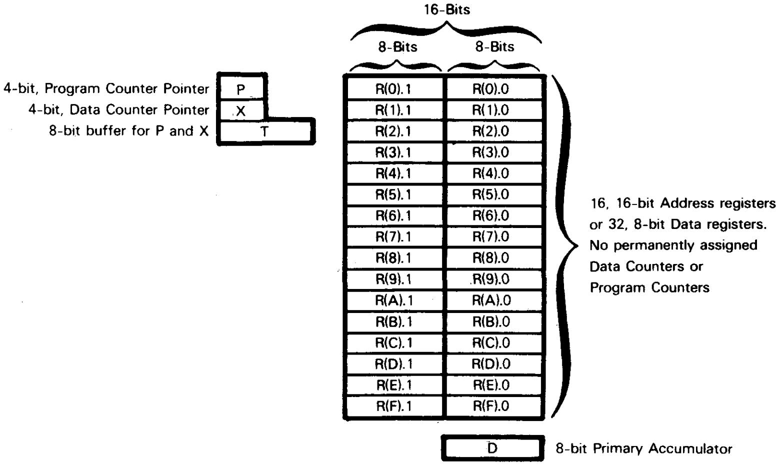
Functions implemented on the CDP1802 CPU are illustrated in Figure 11-1.

Logic to handle an external interrupt request is provided by the COSMAC CPU, along with an elementary ability to handle interrupt priority arbitration.

An unusual feature of COSMAC, as compared to other CPUs described in this book, is the fact that COSMAC provides an elementary DMA capability using CPU logic.

COSMAC PROGRAMMABLE REGISTERS

These are the programmable registers of the COSMAC CPU:



The D register functions as a primary Accumulator.

The sixteen, 16-bit registers may serve as Program Counters, Data Counters, or scratchpad memory.

As scratchpad memory, each 16-bit register consists of two 8-bit registers whose contents can be transferred to or from the primary Accumulator (D register).

The nomenclature RN is used to define a 16-bit General Purpose register. N may be any number in the range 0 - 15. When General Purpose registers are being treated as 8-bit data storage units, R(N).1 is used to identify the high order byte of General Purpose Register RN and R(N).0 is used to identify the low order byte of General Purpose

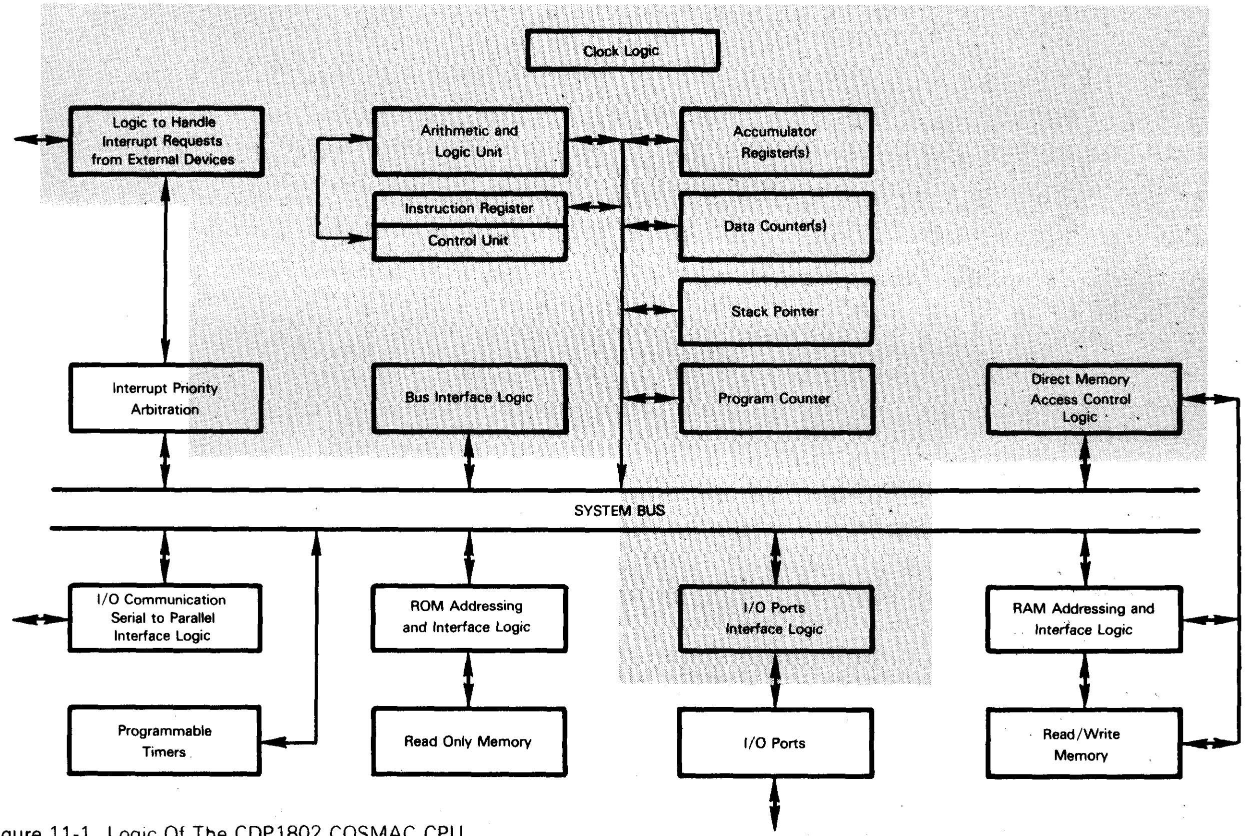

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 11-1. Logic Of The CDP1802 COSMAC CPU

Register RN. For example, R6 identifies the seventh 16-bit General Purpose register. This General Purpose register contains a high order byte, identified as R(6).1 and a low order byte identified as R(6).0.

The 4-bit P register identifies the 16-bit register which at any point in time is functioning as a Program Counter.

The 4-bit X register identifies the 16-bit register which at any point in time is functioning as a Data Counter.

COSMAC literature identifies a third 4-bit register, called the N register. On first reading, the N register may look like the X register, but in reality, the N register represents the low order four bits of the Instruction register. The N register is not a programmable register, as we define it.

The first three 16-bit registers also have dedicated functions. Register R0 is the Memory Address register used by the DMA logic of COSMAC.

Following an interrupt acknowledge, Register R1 is assumed to contain the beginning address for the interrupt service routine, and General Purpose Register R2 serves as a primitive Stack Pointer. A single instruction allows you to push the contents of the T register into the memory location addressed by General Purpose Register R2. Another single instruction loads P and X with the contents of the memory location addressed by General Purpose Register R2.

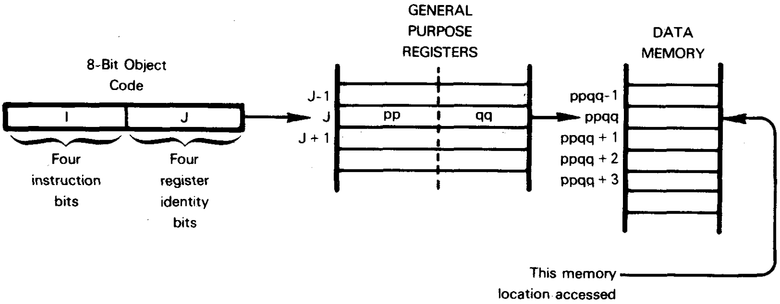
The T register is a simple, 8-bit buffer within which X and P register contents are stored following an interrupt.

COSMAC MEMORY ADDRESSING MODES

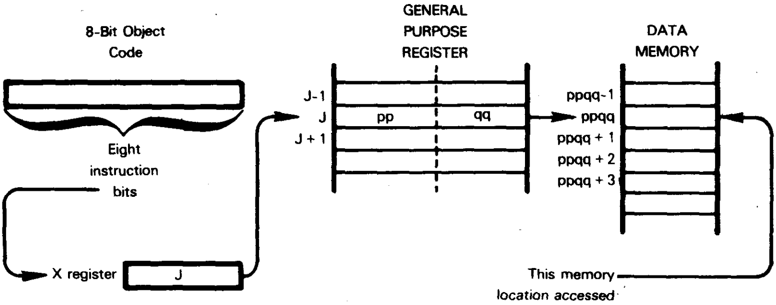
COSMAC offers implied addressing of data memory and direct addressing of program memory.

Any COSMAC instruction that accesses data memory indicates one of the sixteen General Purpose registers as providing the required memory address. Implied memory addressing with auto-increment or auto-decrement is also available in a limited number of cases.

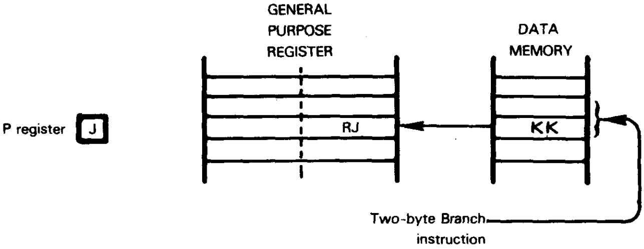
An instruction that accesses data memory may directly identify the General Purpose register wherein the implied data memory address will be found:



Alternatively, an instruction may specify that the X register points to the General Purpose register which is to be used as a Data Counter:



Branch instructions use direct memory addressing. COSMAC has two-byte and three-byte Branch instructions. A two-byte Branch instruction uses paged, direct addressing; the second byte of object code replaces the low order byte of the 16-bit General Purpose register currently serving as Program Counter:



In the illustration above, the P register contains a hexadecimal digit represented by J. General Purpose Register RJ is therefore currently serving as the Program Counter. A two-byte Branch instruction contains an 8-bit value, represented by KK, in the second object program byte. When a branch is executed, KK is loaded into R(J).0, the low order byte of General Purpose Register RJ. This represents straightforward, absolute paged direct addressing as described in Volume I, Chapter 6.

The second and third object code bytes of a three-byte Branch instruction provide a 16-bit address which replaces the entire contents of the General Purpose register currently serving as Program Counter. This is equivalent to simple non-paged direct addressing as described in Volume I, Chapter 6.

Program and data memory in a COSMAC microcomputer system may be common or separate. Because COSMAC has a wealth of control signals, it is almost as easy to implement program and data memory with duplicated memory addresses, and address spaces, as it is to implement program and data memory with separate addresses and address spaces. Thus, COSMAC can have separate program and data memories, as described for the SMS300, or it can have a shared address space as is the case for all other microcomputers described in this book.

COSMAC STATUS FLAGS

COSMAC has no Status register, but it does have seven flags which, in a rather unusual way, provide status information.

Two of the seven status flags are orthodox:

There is the Data Flag (DF), which is equivalent to the Carry status as we describe it.

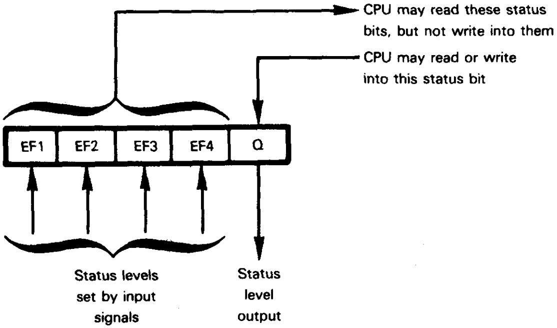
There is an Interrupt Enable flag which must be set to 1 if interrupts are enabled; this flag is set to 0 in order to disable interrupts.

Five of the seven status flags are direct logic control statuses.

There are four I/O flags (EF1 - EF4) which are connected directly to CPU pins. External logic can input high or low signals at these four pins. Subsequently, COSMAC Branch-on-Condition instructions can test any one of these four pins, then branch or not branch, depending on the status of the pin.

The fifth condition status is referred to as **the Q status.** This status **can be set or reset directly by appropriate COSMAC instructions.** Subsequent Branch-on-Condition instructions will test the Q status in order to determine whether or not the branch will occur. In addition, the Q status is connected to a pin which external logic can use in any way.

We may summarize the I/O and Q statuses as follows:


CPU may read these status bits, but not write into them
CPU may read or write into this status bit
EF1
EF2
EF3
EF4
Q
Status levels set by input signals
Status level output

In addition there are three control signals output by COSMAC (N0, N1 and N2). These three signals can be used as control/status outputs to external logic.

COSMAC CPU PINS AND SIGNALS

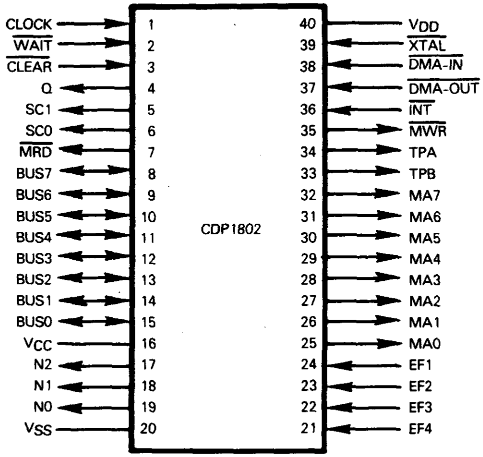
COSMAC CPU pins and signals are illustrated in Figure 11-2. A description of these signals is useful as a guide to the way in which the COSMAC microprocessor works. Signal names in Figure 11-2 conform with those used by COSMAC literature.

BUS0 - BUS7 is a standard bidirectional parallel Data Bus, usually called D0 - D7 for other microprocessors described in this book. All parallel data communications between the COSMAC CPU and external logic, memory or I/O occur via this Data Bus.

MA0 - MA7 represents an 8-bit Address Bus. Most other microprocessors described in this book use the symbols A0 - A7 for equivalent Address Bus lines. The fact that COSMAC only has eight address lines is very important. On the one hand, it frees up eight CPU DIP pins, which are used alternatively to provide a wealth of control signals. The disadvantage of having just eight Address Bus lines is that all addresses must be multiplexed; the high order address byte is output, followed by the low order address byte. Memory interface logic must now be more complex; it must first latch the high order address byte, then receive the low order address byte. In the type of very low cost, high volume system where package count is important, this could be a significant penalty. However, RCA does provide a ROM device which includes this address decode logic.

The remaining signals may be divided into timing, status and control signals.

The timing signals are CLOCK, $\overline{\text{XTAL}}$, TPA and TPB.


CDP1802
CLOCK 1
$\overline{\text{WAIT}}$ 2
$\overline{\text{CLEAR}}$ 3
Q 4
SC1 5
SC0 6
$\overline{\text{MRD}}$ 7
BUS7 8
BUS6 9
BUS5 10
BUS4 11
BUS3 12
BUS2 13
BUS1 14
BUS0 15
V_{CC} 16
N2 17
N1 18
N0 19
V_{SS} 20
40 V_{DD}
39 $\overline{\text{XTAL}}$
38 $\overline{\text{DMA-IN}}$
37 $\overline{\text{DMA-OUT}}$
36 $\overline{\text{INT}}$
35 $\overline{\text{MWR}}$
34 TPA
33 TPB
32 MA7
31 MA6
30 MA5
29 MA4
28 MA3
27 MA2
26 MA1
25 MA0
24 EF1
23 EF2
22 EF3
21 EF4

Pin Name	Description	Type
BUS0 - BUS7	Parallel Data Bus	Bidirectional
MA0 - MA7	Address Bus	Output
CLOCK	Externally generated clock	Input
$\overline{\text{XTAL}}$	External crystal connection	Input
TPA, TPB	Timing pulses	Output
EF1 - EF4	External flags	Input
Q	Q status	Output
SC0, SC1	State Code lines	Output
$\overline{\text{MWR}}$	Write pulse	Output
$\overline{\text{MRD}}$	Read level	Output
N0 - N2	I/O command	Output
$\overline{\text{WAIT}}$, $\overline{\text{CLEAR}}$	Control lines	Input
$\overline{\text{DMA-IN}}$, $\overline{\text{DMA-OUT}}$	Direct memory access control	Input
$\overline{\text{INT}}$	Interrupt request	Input
V_{DD}	Internal voltage supply	
V_{CC}	Input/Output voltage supply; logic 1	
V_{SS}	Ground; logic 0	

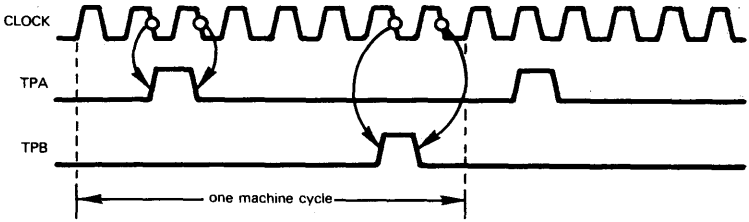
Figure 11-2. CDP1802 COSMAC CPU Signals And Pin Assignments

CLOCK is the principal timing signal input by external clock logic. Its frequency is up to 6.4 MHz when using a +10V power supply.

If you are using the on-chip clock logic, then you must connect an external crystal, with a parallel resistor, to the $\overline{\text{XTAL}}$ and CLOCK pins.

TPA and TPB are timing pulses output by the CPU to control external logic.

CLOCK, TPA, and TPB timing may be illustrated as follows:



The status signals are EF1 - EF4, Q and SC0 - SC1.

We have already encountered signals **EF1 - EF4.** These **are four signals which external logic can input high or low** and which can be tested by conditional Branch instructions.

Q is continuously output, reflecting the level of the Q status flag, which you can set or reset by executing appropriate COSMAC instructions. External logic can use the Q output signal in any way.

The two state signals SC0 and SC1 are output by the CPU to identify the type of machine cycle which is in progress. SC0 and SC1 are output as follows:

SC1	SC0	Machine Cycle Operation
0	0	Instruction Fetch
0	1	Instruction Execute
1	0	DMA Access
1	1	Interrupt Acknowledge

Typically, external logic will use the SC0 and SC1 signals as an integral part of device select logic in order to ensure that no device considers itself selected inappropriately.

Remaining signals may be classified generally as controls.

$\overline{\text{MWR}}$ is output as a low pulse after the second (low order) byte of any address has stabilized on the Address Bus. $\overline{\text{MWR}}$ indicates a memory access operation.

$\overline{\text{MRD}}$ subsequently indicates the direction of a data access. If $\overline{\text{MRD}}$ is low, then the CPU is reading data from memory or I/O devices. If $\overline{\text{MRD}}$ is high, then the CPU is writing to memory or I/O devices.

When an Input or Output instruction is executed, as against a Memory Reference instruction, a nonzero value is output via the three I/O command pins N0, N1 and N2. If all three pins are low, no I/O operation is in progress. How you use the three I/O command pins is up to you. They can, if you wish, identify an I/O port, in which case you can immediately address up to seven I/O ports. Any number of I/O ports can be addressed using a two-level system. Alternatively, you can use these pins to distinguish between command, status or data.

External logic can control the CPU via the $\overline{\text{WAIT}}$ and $\overline{\text{CLEAR}}$ inputs. These two inputs combine to force the CPU into the following states:

$\overline{\text{CLEAR}}$	$\overline{\text{WAIT}}$	CPU STATE
0	0	Load
0	1	Reset
1	0	Pause
1	1	Run

In the Load state, the CPU is idled and external logic can load memory directly, using the direct memory access logic provided by the CPU itself.

The Reset state is a typical reset. During a reset, the Instruction register, the X and P registers, R0 General Purpose register and the Q status are all reset to zero.

The Reset state should be terminated by entering the Run state. Thus you may look upon $\overline{\text{WAIT}}$ as a signal which is maintained high during a normal sequence of Run and Reset states; $\overline{\text{CLEAR}}$ then becomes equivalent to the single RESET signal provided by other microprocessors.

When you enter the Run state following a Reset, the P register will contain 0, therefore General Purpose Register 0 acts as a Program Counter. General Purpose Register R0 contains 0000, therefore the first instruction fetched following a Reset will have its object code stored in memory location 0000.

The Pause mode stops all internal CPU operations other than the CLOCK signal. Note that COSMAC is a static device. CPU operations can halt for any length of time with no loss of data.

The Run mode is the condition in which the CPU will normally operate.

$\overline{\text{DMA-IN}}$ and $\overline{\text{DMA-OUT}}$ are control signals input by external logic in order to perform direct memory access operations. $\overline{\text{DMA-IN}}$ specifies a data transfer from external logic to memory; $\overline{\text{DMA-OUT}}$ will cause a data transfer from memory to external logic. In each case, memory is addressed by General Purpose Register R0. External logic is implicitly identified — it is the source of the $\overline{\text{DMA-IN}}$ and $\overline{\text{DMA-OUT}}$ signals. Following a DMA transfer, General Purpose Register R0 contents are incremented.

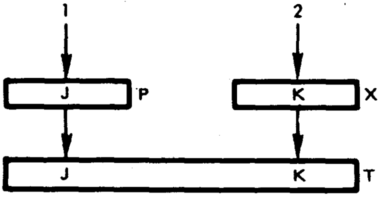
$\overline{\text{INT}}$ is a standard interrupt request input.

A SUMMARY OF COSMAC INTERRUPT PROCESSING

External logic can, at any time, request an interrupt by inputting a low signal at $\overline{\text{INT}}$. Providing interrupts are enabled, following execution of the current instruction, **the CPU will respond to the interrupt request with these three steps:**

1) The contents of the X and P registers are moved to the T register.
2) The P and X registers have the hexadecimal values 1 and 2 loaded into them, respectively.
3) Interrupts are disabled.

Steps 1 and 2 may be illustrated as follows:



The interrupt service routine now begins executing with the instruction addressed by General Purpose Register R1. Any data accessed by the interrupt service routine must be addressed by General Purpose Register R2.

In the event that an interrupt service routine may itself be interrupted, you can store the T register contents in memory, at the location addressed by General Purpose Register R2 (which is now pointed to by X). COSMAC does not handle nested interrupts easily. The same General Purpose registers, R1 and R2, address program and data memory following every single interrupt request; in order to handle nested interrupts, you must create an external memory Stack where you can store and restore the contents of General Purpose registers. In each case, program logic quickly gets out of hand.

As we have frequently stated, applications that rely upon extensive and complex interrupt nesting and priority arbitration are inherently unsuited to microprocessors. In most cases you can develop a less expensive implementation using multiple CPUs. Therefore do not look upon COSMAC interrupt handling as a product liability.

The four input signals, EF1 - EF4, are the only means directly available for external logic to identify itself when more than one external device can request an interrupt. Use of these external flag signals means that the interrupt service routine must begin with a number of Branch-on-Condition instructions that test the input flags to determine which is high.

More complex interrupt priority arbitration schemes must rely upon external logic, which will create some type of code for the CPU to read out of an identified I/O port or memory location in order to determine the interrupting source. Interrupt priority arbitration, as it is normally understood, becomes the complete responsibility of external logic.

COSMAC DIRECT MEMORY ACCESS LOGIC

Simple direct memory access is more easily handled by COSMAC than any other 8-bit microprocessor described in this book. All you have to do is load the appropriate memory address into General Purpose Register R0, then set this General Purpose register aside to service direct memory accesses. External logic subsequently causes data to be transferred to or from memory, via DMA, by inputting low pulses at $\overline{\text{DMA-IN}}$ or $\overline{\text{DMA-OUT}}$. A low pulse on either one of these signals will cause data to be transferred between the memory location addressed by General Purpose Register R0 and the external logic which requests the direct memory access. This transfer will occur as soon as the current instruction has completed execution.

THE COSMAC INSTRUCTION SET

Table 11-1 summarizes the COSMAC instruction set. The strength of the instruction set lies in the ability to communicate directly with external logic via the Q and $\overline{\text{EF}}$ signals. The ability to assign General Purpose registers in any way as Program and Data Counters also makes the instruction set very powerful — providing program modules are small and do not require more address storage than is provided by the General Purpose registers.

The weakness of COSMAC is that all register-to-register data transfers occur via the D register. Also all data transfers between the CPU and external logic occur via the D register. Providing you can change the contents of the X register instead of actually moving data between registers, you will have no problem; if not, the D register will become a bottleneck.

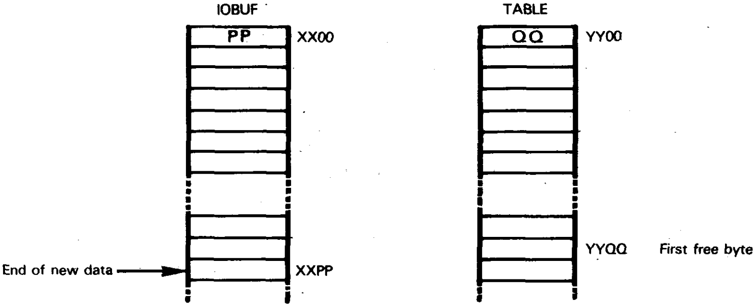
The COSMAC instruction set is not suited to mathematical manipulations; it is particularly unsuited to BCD operations. This is because COSMAC has no decimal mode of operation and the single status flag (DF) makes signed arithmetic very difficult to handle.

THE BENCHMARK PROGRAM

Now consider our benchmark program; for COSMAC it looks like this:

```
        LDI     TABHI     ..LOAD TABLE BASE ADDRESS HIGH ORDER BYTE
        PHI     R15       ..INTO R15 AND R13
        PHI     R13       ..R15 POINTS TO NEXT FREE TABLE BYTE
        LDI     00
        PLO     R13       ..R13 POINTS TO FIRST BYTE IN TABLE
        PLO     R14
        LDN     R13       ..ASSUME THAT DISPLACEMENT TO FIRST
        PLO     R15       ..FREE BYTE IS STORED IN FIRST TABLE BYTE
        LDI     IOBFHI    ..LOAD IOBUF START ADDRESS INTO R14
        PHI     R14
        LDN     R14       ..LOAD DISPLACEMENT TO END OF FILLED IOBUF
        PLO     R14
LOOP:   LDN     R14       ..LOAD NEXT BYTE FROM IOBUF
        STR     R15       ..STORE IN NEXT FREE TABLE BYTE
        INC     R15       ..INCREMENT R15
        DEC     R14       ..DECREMENT R14
        GLO     R14       ..TEST LOW ORDER BYTE OF R14
        BNZ     LOOP      ..IF NOT ZERO RETURN TO LOOP
        GLO     R15       ..AT END RESET FIRST BYTE OF
        STR     R13       ..TABLE TO NEW FIRST FREE BYTE ADDRESS
```

This is the memory map assumed by the benchmark program above:



Tables IOBUF and TABLE are both origined on page boundaries; that is to say, the low order eight bits of the origin address are zeros. Data in table IOBUF is stored backwards. The first byte of data to be moved from IOBUF to TABLE is stored at the highest memory address of IOBUF. This highest memory address, illustrated above by XXPP, is derived by adding the contents of the first IOBUF table byte to the origin address. Thus, the first byte of IOBUF stores that length of table IOBUF which is currently filled. COSMAC program logic can now decrement the initial IOBUF address from XXPP and, upon testing the low order byte equal to zero, logic knows that all data has been transferred.

The destination table stores the displacement to the first free table byte in the first byte of TABLE. Thus the address of the first free byte equals the origin plus the contents of the first TABLE byte.

Since the displacement to the first free byte of TABLE is stored in a single data byte, clearly TABLE cannot be more than 256 bytes long. Thus, IOBUF must contain less than 256 bytes at any time.

If you look at the COSMAC program, it appears rather long. The instruction loop itself only contains six instructions, which compares well with many other benchmark programs. What is deceptive about the benchmark program is the fact that we have taken a large number of instructions in order to load initial addresses into General Purpose registers. Remember, COSMAC has sixteen such General Purpose registers and the whole programming philosophy of this microcomputer is that you load addresses into General Purpose registers once, at the beginning of the program, and never again. In fact, the benchmark program points up both the strength and the weakness of the COSMAC instruction set. Its strength is that large numbers of addresses can be permanently stored within CPU registers, thence memory access becomes a trivial task. Its weakness is that it takes a lot of instructions to get memory addresses into General Purpose registers in the first place — and that becomes a liability if you have to reuse the same General Purpose register, in a number of different ways, within one program.

The following symbols are used in Table 11-1:

ADR8	8-bit address
ADR16	16-bit address
D	D register
DATA8	8-bit data unit
DEV	3-bit code: 1 through 7
DF	Data Flag or Carry
EFn	Pin status: EF1, EF2, EF3, or EF4
IE	Interrupt Enable bit
n	One of the numbers 1, 2, 3, 4
N	4-bit register select unit
N210	Three output pins, N2, N1, N0
P	4-bit Program Counter Pointer register
Q	Q status output flip-flop
R(z)	Specifies a register: if z is N the instruction operand specifies the register P the contents of the P register specify the register X the contents of the X register specify the register
T	T register
X	4-bit Data Counter Pointer register
x<y,z>	Bits y through z of a register or memory location. For example, T<7,4> represents the high order four bits of the T register.
[ ]	Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If an I/O port number is enclosed within the brackets, then the I/O port contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.
[[ ]]	Implied memory addressing; the contents of the memory location designated by the contents of a register.
Λ	Logical AND
V	Logical OR
∀	Logical Exclusive-OR
←	Data is transferred in the direction of the arrow.

Under the heading of STATUSES in Table 11-1, an X indicates statuses which are modified in the course of the instruction's execution. If there is no X, it means that the status maintains the value it had before the instruction was executed.

Table 11-1. COSMAC Instruction Set Summary

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				OF	IE	
I/O	INP	DEV	1			[[R(X)]]←[D]←BUS N210←[N<2,0>] Input data from Bus to Register D and memory. Output lower three bits of Register N to output pins N2, N1, N0.
	OUT	DEV	1			BUS←[[R(X)]] N210←[N<2,0>] [R(X)]←[R(X)] + 1 Output memory to Bus; output lower three bits of Register N to output pins; increment Data Counter.
PRIMARY MEMORY REFERENCE	LDN	N	1			[D]←[[R(N)]] Load D register via specified register. N may not be 0.
	LDA	N	1			[D]←[[R(N)]] [R(N)]←[R(N)] + 1 Load D register via specified register. Increment specified register.
	STR	N	1			[[R(N)]]←[D] Store D register via specified register.
	LDX		1			[D]←[[R(X)]] Load D register using implied addressing.
	LDXA		1			[D]←[[R(X)]] [R(X)]←[R(X)] + 1 Load D register using implied addressing. Increment Data Counter.
	STXD		1			[[R(X)]]←[D] [R(X)]←[R(X)]-1 Store D register using implied addressing. Decrement Data Counter.

Table 11-1. COSMAC Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				OF	IE	
SECONDARY MEMORY REFERENCE MEMORY OPERATE	OR		1			[D]←[[R(X)]] V [D] OR with D register using implied addressing.
	XOR		1			[D]←[[R(X)]]⊻[D] Exclusive-OR with D register using implied addressing.
	AND		1			[D]←[[R(X)]] Λ [D] AND with D register using implied addressing.
	ADD		1	X		[D]←[[R(X)]] + [D] Add to D register using implied addressing.
	ADC		1	X		[D]←[[R(X)]] + [D] + [DF] Add with Carry to D register using implied addressing.
	SD		1	X		[D]←[[R(X)]]-[D] Subtract D from memory using implied addressing.
	SDB		1	X		[D]←[[R(X)]]-[D]-[DF] Subtract with borrow from memory using implied addressing.
	SM		1	X		[D]←[D]-[[R(X)]] Subtract memory from D using implied addressing.
	SMB		1	X		[D]←[D]-[[R(X)]]-[DF] Subtract memory with borrow from D using implied addressing.
IMMEDIATE	LDI	DATA8	2			[D]←DATA8 Load immediate to D register.
IMMEDIATE OPERATE	ORI	DATA8	2			[D]←DATA8 V [D] OR immediate with D register.
	XRI	DATA8	2			[D]← DATA8⊻[D] Exclusive-OR immediate with D register.
	ANI	DATA8	2			[D]←DATA8 Λ [D] AND immediate with D register.

Table 11-1. COSMAC Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				OF	IE	
IMMEDIATE OPERATE (CONTINUED)	ADI	DATA8	2	X		[D]← DATA8 + [D] Add immediate to D register.
	ADCI	DATA8	2	X		[D]← DATA8 + [D] + [DF] Add immediate with Carry to D register.
	SDI	DATA8	2	X		[D]← DATA8-[D] Subtract D register from immediate data.
	SDBI	DATA8	2	X		[D]← DATA8-[D]-[DF] Subtract D register with borrow from immediate data.
	SMI	DATA8	2	X		[D]←[D]-DATA8 Subtract immediate from D register.
	SMBI	DATA8	2	X		[D]←[D]-DATA8-[DF] Subtract immediate with borrow from D register.
BRANCH AND SKIP	BR	ADR8	2			[R(P)<7,0>]← ADR8 Branch within same page to given address.
	LBR	ADR16	3			[R(P)]← ADR16 Branch to given address
	SKP		1			[R(P)]←[R(P)] + 1 Skip next byte.
	LSKP		1			[R(P)]←[R(P)] + 2 Skip next two bytes.
	NBR	ADR8	2			Same as SKIP
	NLBR	ADR16	3			Same as LSKP
BRANCH AND SKIP ON CONDITION	BZ	ADR8	2			If [D]=0; then [R(P)<7,0>]← ADR8 Branch within same page on D register zero.
	BNZ	ADR8	2			If [D]≠0; then [R(P)<7,0>]← ADR8 Branch within same page on D register nonzero.
	BDF	ADR8	2			If [DF]=1; then [R(P)<7,0>]← ADR8 Branch within same page on Carry set.

Table 11-1. COSMAC Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				OF	IE	
BRANCH AND SKIP ON CONDITION (CONTINUED)	BNF	ADR8	2			If [DF]=0; then [R(P)<7,0>]←ADR8 Branch within same page on Carry reset.
	BQ	ADR8	2			If Q=1; then [R(P)<7,0>]←ADR8 Branch within same page on output flip-flop set.
	BNQ	ADR8	2			If Q=0; then [R(P)<7,0>]←ADR8 Branch within same page on output flip-flop reset.
	Bn	ADR8	2			If EFn=1; then [R(P)<7,0>]←ADR8 Branch within same page on specified external flag set.
	BNn	ADR8	2			If EFn=0; then [R(P)<7,0>]←ADR8 Branch within same page on specified external flag reset.
	LBZ	ADR16	3			If [D]=0; then [R(P)]←ADR16 Branch absolute on D register zero.
	LBNZ	ADR16	3			If [D]≠0; then [R(P)]←ADR16 Branch absolute on D register nonzero.
	LBDF	ADR16	3			If [DF]=1; then [R(P)]←ADR16 Branch absolute on Carry set.
	LBNF	ADR16	3			If [DF]=0; then [R(P)]←ADR16 Branch absolute on Carry reset.
	LBQ	ADR16	3			If [Q]=1; then [R(P)]←ADR16 Branch absolute on output flip-flop set.
	LBNQ	ADR16	3			If [Q]=0; then [R(P)]←ADR16 Branch absolute on output flip-flop reset.
	LSZ		1			If [D]=0; then [R(P)]←[R(P)]+2 Skip two bytes if D register zero.
	LSNZ		1			If [D]≠0; then [R(P)]←[R(P)]+2 Skip two bytes if D register nonzero.
	LSDF		1			If [DF]=1; then [R(P)]←[R(P)]+2 Skip two bytes if Carry set.
	LSNF		1			If [DF]=0; then [R(P)]←[R(P)]+2 Skip two bytes if Carry reset.

Table 11-1. COSMAC Instruction Set Summary (Continued)

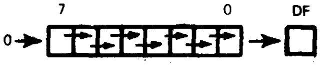
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				OF	IE	
BRANCH AND SKIP ON CONDITION (CONTINUED)	LSQ		1			If [Q]=1; then [R(P)]←[R(P)]+2 Skip two bytes if output flip-flop set.
	LSNQ		1			If [Q]=0; then [R(P)]←[R(P)]+2 Skip two bytes if output flip-flop reset.
	LSIE		1			If [IE]=1; then [R(P)]←[R(P)]+2 Skip two bytes if interrupts are enabled.
REGISTER-REGISTER MOVE	GLO	N	1			[D]←[R(N)<7,0>] Load D with low byte of specified register.
	GHI	N	1			[D]←[R(N)<15,8>] Load D with high byte of specified register.
	PLO	N	1			[R(N)<7,0>]←[D] Store D to low byte of specified register.
	PHI	N	1			[R(N)<15,8>]←[D] Store D to high byte of specified register.
REGISTER OPERATE	INC	N	1			[R(N)]←[R(N)]+1 Increment specified register.
	DEC	N	1			[R(N)]←[R(N)]-1 Decrement specified register.
	IRX					[R(X)]←[R(X)]+1 Increment Data Counter.
	SHR		1	X		 Shift D register right one bit. Shift bit 0 into Carry; reset bit 7.

Table 11-1. COSMAC Instruction Set Summary (Continued)

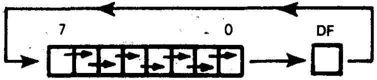
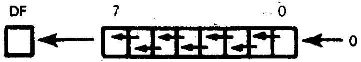
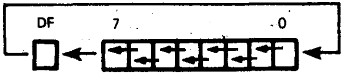
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				OF	IE	
REGISTER OPERATE (CONTINUED)	SHRC		1	X		 Shift D register right one bit through Carry.
	SHL		1	X		 Shift D register left one bit. Shift bit 7 into Carry; reset bit 0.
	SHLC		1	X		 Shift D register left one bit through Carry.
STACK	SAV		1			[[R(X)]]←[T] Save T register in memory.
	MARK		1			[T<7,4>]←[X] [T<3,0>]←[P] [[R(2)]]←[T] [R(2)]←[R(2)]-1 [X]←[P] Save X and P in T; then push onto Stack via Register 2. Decrement Register 2. Move P to X.

Table 11-1. COSMAC Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES		OPERATION PERFORMED
				OF	IE	
STACK (CONTINUED)	RET		1			[X]←[[R(X)] <7,4>] [P]←[[R(X)] <3,0>] [R(X)]←[R(X)] + 1 [IE]←1 Pop memory into X and P using implied addressing. Increment Data Counter. Enable interrupts.
	DIS		1			[X]←[[R(X)] <7,4>] [P]←[[R(X)] <3,0>] [R(X)]←[R(X(] + 1 [IE]←0 Pop memory into X and P using implied addressing. Increment Data Counter. Disable interrupts.
STATUS	SEP	N	1			[P]←N Set P register to N.
	SEX	N	1			[X]←N Set X register to N.
	SEQ		1			[Q]←1 Set output flip-flop.
	REQ		1			[Q]←0 Reset output flip-flop.
	IDL		1			Idle CPU. Wait for Interrupt/DMA-IN/DMA-OUT.
	NOP		1			No Operation

The following symbols are used in Table 11-2:

aaaa	4 bits selecting one of the 16 registers
bbb	3-bit data unit output to N2, N1, N0 lines
PP	8-bit address
QQ	Second 8 bits of a 16-bit address
XX	8-bit immediate data unit

Table 11-2. COSMAC Instruction Set Object Codes

INSTRUCTION	OBJECT CODE	BYTES	MACHINE CYCLES
ADC	74	1	2
ADCI DATA8	7C XX	2	2
ADD	F4	1	2
ADI DATA8	FC XX	2	2
AND	F2	1	2
ANI DATA8	FA XX	2	2
BDF ADR8	33 PP	2	2
BNF ADR8	3B PP	2	2
BNQ ADR8	39 PP	2	2
BNZ ADR8	3A PP	2	2
BNI ADR8	3C PP	2	2
BNZ ADR8	3D PP	2	2
BN3 ADR8	3E PP	2	2
BN4 ADR8	3F PP	2	2
BQ ADR8	31 PP	2	2
BR ADR8	30 PP	2	2
BZ ADR8	32 PP	2	2
B1 ADR8	34 PP	2	2
B2 ADR8	35 PP	2	2
B3 ADR8	36 PP	2	2
B4 ADR8	37 PP	2	2
DEC N	0010aaaa	1	2
DIS	71	1	2
GHI N	1001aaaa	1	2
GLO N	1000aaaa	1	2
IDL	00	1	2
INC N	0001aaaa	1	2
INP P	01101bbb	1	2
IRX	60	1	2

Table 11-2. COSMAC Instruction Set Object Codes (Continued)

INSTRUCTION		OBJECT CODE	BYTES	MACHINE CYCLES
LBDF	ADR16	C3	3	3
		PP		
		QQ		
LBNF	ADR16	C3	3	3
		PP		
		QQ		
LBNQ	ADR16	C9	3	3
		PP		
		QQ		
LBNZ	ADR16	CA	3	3
		PP		
		QQ		
LBQ	ADR16	C1	3	3
		PP		
		QQ		
LBR	ADR16	C0	3	3
		PP		
		QQ		
LBZ	ADR16	C2	3	3
		PP		
		QQ		
LDA	N	0100aaaa	1	2
LDI	DATA8	F8	2	2
		XX		
LDN	N	0000aaaa	1	2
LDX		F0	1	2
LDXA		72	1	2
LSDF		CF	1	3
LSIE		CC	1	3
LSNF		C7	1	3
LSNQ		C5	1	3
LSNZ		C6	1	3
LSQ		CD	1	3
LSKP		C8	1	3
LSZ		CF	1	3
MARK		79		2
NBR		38	2	2
NLBR		C8	3	3
NOP		C4	1	3
OR		F1	1	2
ORI	DATA8	F9	2	2
		XX		
OUT	P	01100bbb	1	2
PHI	N	1011aaaa	1	2
PLO	N	1010aaaa	1	2
REQ		7B	1	2
RET		70	1	2
SAV		78	1	2
SEQ		7A	1	2
SEP	N	1101aaaa	1	2
SEX	N	1110aaaa	1	2
SD		F5	1	2
SDB		75	1	2

Table 11-2. COSMAC Instruction Set Object Codes (Continued)

INSTRUCTION		OBJECT CODE	BYTES	MACHINE CYCLES
SDBI	DATA8	7D XX	2	2
SDI	DATA8	FD XX	2	2
SHL		FE	1	2
SHLC		7E	1	2
SHR		F6	1	2
SHRC		76	1	2
SKP		38	1	2
SM		F7	1	2
SMB		77	1	2
SMBI	DATA8	7F XX	2	2
SMI		FF XX	2	2
STR	N	0101aaaa	1	2
STXD		73	1	2
XOR		F3	1	2
XRI	DATA8	FB XX	2	2

ELECTRICAL DATA

The following pages contain specific electrical and timing characteristics of the COSMAC CDP1802.

Preliminary CDP1802D, CDP1802CD

ELECTRICAL CHARACTERISTICS at T_A = 25°C

CHARACTERISTIC			CONDITIONS		CDP1802D TYPICAL VALUES	CDP1802CD TYPICAL VALUES	UNITS
			V_O (V)	V_{CC}, V_{DD} (V)			
Quiescent Device Current, I_L			–	5, 5	100	500	μA
			–	10, 10	500	–	
			–	15, 15	1000	–	
Total Power Dissipation: OP CODE "00" (See Fig. 4)	f = MHz	3.2	–	5, 5	6	8	mW
		5.0	–	5, 10	30	–	
		6.4	–	10, 10	40	–	
Output Voltage: Low-Level, V_{OL}			–	5, 5	0.01	0.01	V
			–	10, 10	0.01	–	
High-Level, V_{OH}			–	5, 5	5	5	
			–	10, 10	10	–	
Noise Immunity: Inputs Low, V_{NL}			0.5	5, 5	2.25	2.25	V
			−1	10, 10	3.45	–	
Inputs High, V_{NH}			4.5	5, 5	2.25	2.25	
			9	10, 10	3.45	–	
Output Drive Current: N-Channel (Sink), I_DN			0.4	5, 5	1.5	1.5	mA
			0.5	10, 10	3.0	–	
P-Channel (Source), I_DP			2.5	5, 5	−1.6	−1.6	
			4.6	5, 5	−0.4	−0.4	
			9.5	10, 10	−0.9	–	
Input Leakage Current (Any Input), I_{IL}, I_{IH}			–	5, 5	±1	±1	μA
			–	15, 15	±1	–	

MAXIMUM RATINGS,

Absolute-Maximum Values

Storage-Temperature Range (T_{stg})
. . . . −65 to +150°C

Operating-Temperature Range (T_A)
. . . . −55 to +125°C

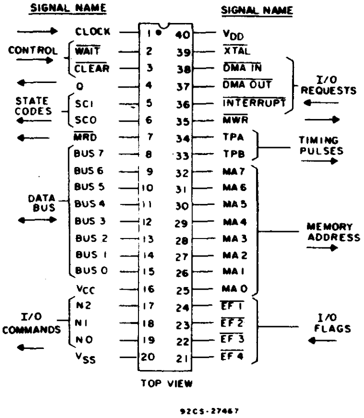
DC Supply-Voltage Range (V_{CC}, V_{DD})
(All voltage values referenced to V_{SS} terminal)
$V_{CC} \leqslant V_{DD}$:
CDP1802D . . . . −0.5 to +15 V
CDP1802CD . . . . −0.5 to +7 V

Power Dissipation Per Package (P_D):
For T_A = −55 to +100°C
. . . . 500 mW
For T_A = +100 to +125°C
. . . . Derate Linearly to 200 mW

Device Dissipation Per Output Transistor:
For T_A = −55°C to +125°C . . . . 100 mW

Input Voltage Range, All Inputs
. . . . −0.5 to V_{DD} +0.5 V

Lead Temperature (During Soldering):
At distance 1/16 ± 1/32 inch (1.59 ± 0.79 mm)
from case for 10 s max. . . . . +265°C



Terminal Assignment for CDP1802

OPERATING CONDITIONS at T_A = 25°C Unless Otherwise Specified

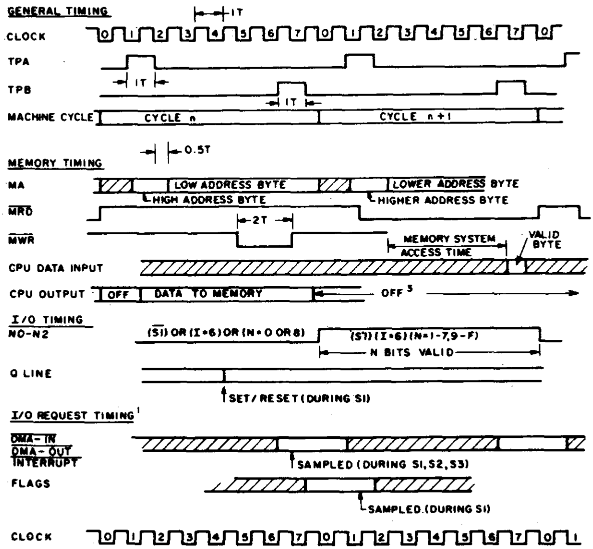
For maximum reliability, nominal operating conditions should be selected so that operation is always within the following ranges.

CHARACTERISTIC	CONDITIONS		TYPICAL VALUES		UNITS
	V_{CC}[1] (V)	V_{DD} (V)	CDP1802D	CDP1802CD	
Supply-Voltage Range (At T_A = Full Package-Temperature Range)	–	–	3 to 12	4 to 6	V
Recommended Input Voltage Range	–	–	V_{SS} to V_{CC}	V_{SS} to V_{CC}	V
Clock Input Rise or Fall Time, t_r or t_f	3-15	3-15	5	5	µs
Instruction Time[2] (See Fig. 3)	5	5	5	5	µs
	5	10	3.2	–	
	10	10	2.5	–	
DMA Transfer Rate	5	5	400	400	KBytes/sec
	5	10	625	–	
	10	10	800	–	
Clock Input Frequency, f_{CL}	5	5	DC - 3.2	DC - 3.2	MHz
	5	10	DC - 5.0	–	
	10	10	DC - 6.4	–	
Clock Pulse Width, t_{WL}, t_{WH}	5	5	160	160	ns
	5	10	100	–	
	10	10	80	–	
Clear Pulse Width	5	5	300	300	ns
	5	10	200	–	
	10	10	150	–	

Notes:

[1] $V_{CC} \leqslant V_{DD}$; for CDP1802CD $V_{DD} = V_{CC}$ = 5 volts.

[2] Equals 2 machine cycles—one Fetch and one Execute operation for all instructions except Long Branch and Long Skip, which require 3 machine cycles—one Fetch and two Execute operations.

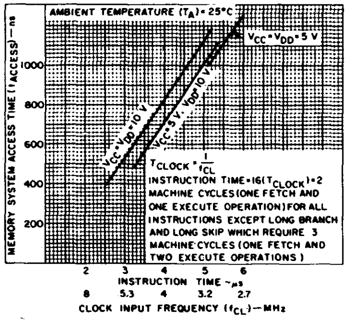


NOTES:
1. USER GENERATED SIGNALS
2. SHADING INDICATES "DONT CARE" OR INTERNAL DELAY
3. "OFF" INDICATES HIGH-IMPEDANCE STATE

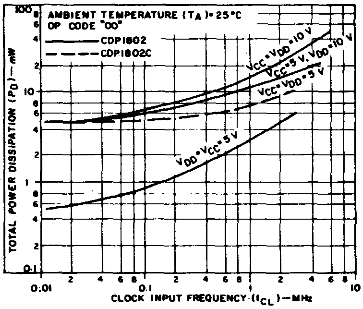
92CM-27440

Timing diagram.

Preliminary CDP1802D, CDP1802CD



Typical instruction time vs. memory system access time.



Typical total power dissipation vs. clock input frequency.

Chapter 12

IM6100 MICROCOMPUTER DEVICES

The IM6100 is an almost exact reproduction of the PDP-8E minicomputer.

The PDP-8 is a 12-bit minicomputer, therefore the IM6100 is a 12-bit microcomputer.

The very existence of the IM6100 is testimony to one of the less well understood aspects of minicomputers, versus microcomputers: people tend to place too much emphasis on "creeping featurism". The majority of applications that are going to use a microcomputer could be implemented with almost any microcomputer described in this book. The economics of exact chip counts and product development expense is worth exploring, but in most cases detailed comparative evaluations of instruction sets and addressing modes are a waste of time and money; enhancement of one product as compared to another will rarely have any significant economic impact. This is true of microcomputers today and it was also true of minicomputers yesterday. The PDP-8 was the first minicomputer; compared to nearly any other minicomputer on the market today, the PDP-8 is a very primitive device. Yet there are more PDP-8s in the world than any other minicomputer. Despite the large number of new, more powerful minicomputers that are available, the PDP-8 continues, from year to year, to rank among the leaders in minicomputer sales volume.

It is this popularity of the PDP-8, for all its shortcomings as a minicomputer, that has given birth to the IM6100. Many design features of the IM6100 are dubious, when looked upon from the microcomputer user's point of view. It is safe to say that no microcomputer designer would have seen fit to develop a product even remotely like the IM6100, but for the predecessor PDP-8. The IM6100 exists to participate in the continuing sales volume of PDP-8, and to take advantage of the huge library of PDP-8 software which is available — much of it at no cost.

You must look at the IM6100 (and the microNOVA) from a totally different perspective, as compared to any other microcomputer described in this book; do not look for justification of IM6100 design features in terms of a microcomputer application's needs, rather accept the IM6100 for what it is — a very low cost reproduction of something which already exists; a product whose existence is justified by a large established product market and a prior base of existing software.

In addition to the IM6100 CPU, we are going to describe the IM6101 Parallel Interface Element. The IM6402 UART is also available; however, it is not described in this chapter.

All IM6100 microcomputer devices use a single power supply which may range between + 4V and + 11V.

Using a 250 nanosecond clock, instruction execution times range from 5 to 11 microseconds.

All IM6100 microcomputer devices use CMOS technology, which means that they are highly immune to noise in the power supply and they consume very little power. Recall that COSMAC is the only other microprocessor described in this book that offers CMOS technology.

The principal manufacturer of the IM6100 is:

INTERSIL, INC.
10900 North Tantau Avenue
Cupertino, CA 95014

The second source is:

HARRIS SEMICONDUCTOR DIVISION
P.O. Box 883
Melbourne, FLA - 32901

THE IM6100 CPU

Functions implemented on the IM6100 CPU are illustrated in Figure 12-1. IM6101 Parallel Interface Element logic is also shown.

Bus interface logic is shown as implemented by the IM6101. This is because the bus control signals input to and output by the CPU do not conform with the standard PDP-8 bus, or with typical microcomputer busses. You are going to need additional logic either to create a PDP-8 bus equivalent, or to reduce IM6100 control signals to manageable microcomputer bus proportions. The IM6101 creates a microcomputer type of System Bus.

Direct memory access control logic is also shown as half present. The CPU has logic which will respond to a DMA request by floating the System Bus; however, the actual DMA transfer, including creation of memory addresses, is the responsibility of external logic.

Observe that clock logic is provided on the CPU chip.

IM6100 PROGRAMMABLE REGISTERS

The IM6100 has just three programmable registers as we define them: an Accumulator, a Program Counter and the MQ register. All three registers are twelve bits wide.

The Accumulator is a typical primary Accumulator. With one single exception, it is the only source or destination within the CPU for data being operated on.

The MQ register is a simple buffer for the Accumulator. The only operation you can perform on the MQ register contents is to OR it with the Accumulator contents; the result is returned to the Accumulator.

The Program Counter, being 12 bits wide, limits the IM6100 to an address space of 4096 memory words.

Intersil literature describes additional registers, but these are not programmable registers as we define them.

The IM6100 has no Data Counter. There is a Memory Address register within the CPU, but you have no direct access to this register. It is a very simple depository for addresses which are automatically computed by CPU logic during the execution of memory reference instructions.

IM6100 MEMORY SPACE

Since the IM6100 is frequently going to be used in minicomputer type applications, we will precede our discussion of memory addressing modes with a discussion of memory implementation.

The fact that the IM6100 is normally limited to an address space of 4096 memory words is not a particularly severe handicap if you look upon this device as a microcomputer. Other microcomputers have similar limitations, including the recent entry into the field, the EA9002. The fact that the IM6100 is a 12-bit microcomputer extends to its memory; that is to say, its address space consists of 4096 12-bit memory words, not 4096 memory bytes.

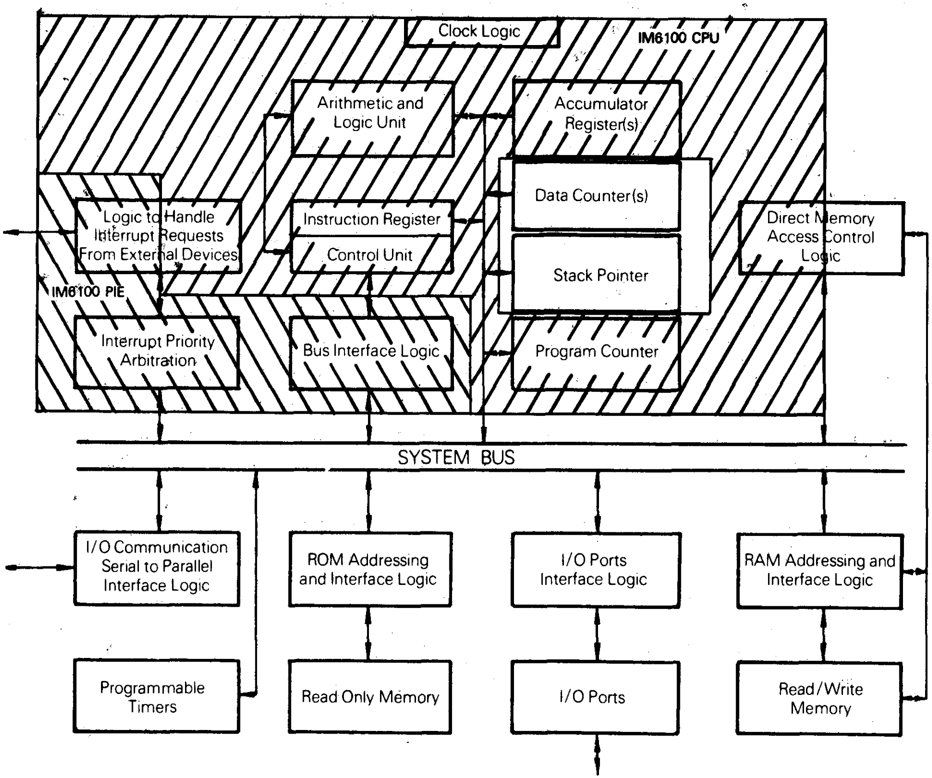

Clock Logic
IM6100 CPU
Arithmetic and Logic Unit
Accumulator Register(s)
Data Counter(s)
Logic to Handle Interrupt Requests From External Devices
Instruction Register
Control Unit
Stack Pointer
Direct Memory Access Control Logic
IM6100 PIE
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 12-1. Logic Of The IM6100 CPU And The IM6101 Parallel Interface Element

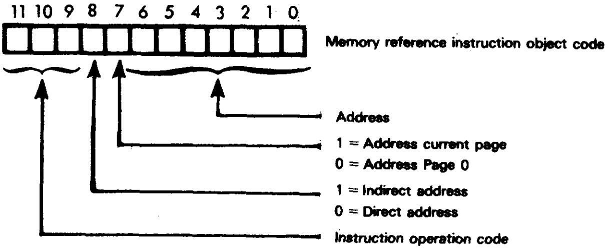
While 4096 memory bytes is more than sufficient external memory for a majority of microcomputer application, it is insufficient for most minicomputer applications — and that is where the IM6100 will be frequently used. Digital Equipment Corporation, the manufacturer of the PDP-8, recognized the need for more memory and developed an extended memory control hardware module as a stock item. This module is, in effect, a one-of-eight decoder which allows you to address eight 4096 memory word banks; at any time the extended memory control unit selects one memory bank and deselects the other seven. Intersil has an equivalent extended memory control unit available, but its cost will at least rival the cost of the CPU. This added cost can only be justified when the IM6100 is being used to provide the CPU for a low-cost minicomputer — and the fact that the IM6100 is a single-chip LSI device is purely coincidental.

Memory addressing modes that we are about to describe apply to a single 4096-word memory bank. If you have more than one such memory bank, then each one must be considered as a separate and distinct entity. This is important because the nature of the IM6100 demands that if program memory is in ROM, then ROM and RAM must be present in external memory. Thus, if you have more than one memory bank, each memory bank must include ROM and RAM.

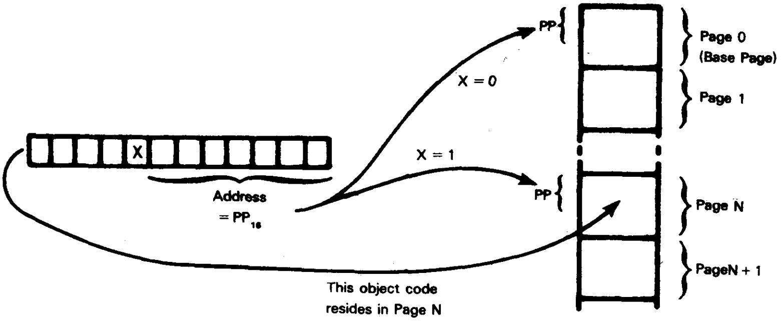
IM6100 MEMORY ADDRESSING MODES

IM6100 memory reference instructions use absolute, paged, direct addressing and indirect addressing.

All IM6100 instruction object codes occupy a single 12-bit word. There are no two-word or three-word object codes. All memory reference instructions have the following object code format:



A memory reference instruction that uses direct addressing has seven address bits; thus memory is divided into 128-word pages. The memory page bit gives you the option of directly addressing a memory word on Page 0, or within the instruction's page:

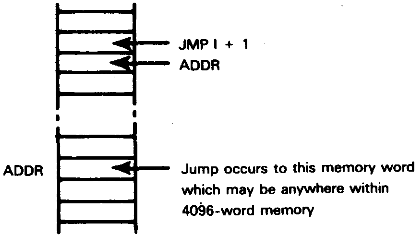


This is standard, absolute paged direct addressing, as described in Volume I, Chapter 6.

A memory reference instruction with indirect addressing simply takes the 12-bit word accessed by the direct memory address and interprets this 12-bit word's contents as the effective memory address. This is standard indirect addressing. In the case of the IM6100, **a memory reference instruction can access an indirect memory address either on the base page or on the instruction's current page.**

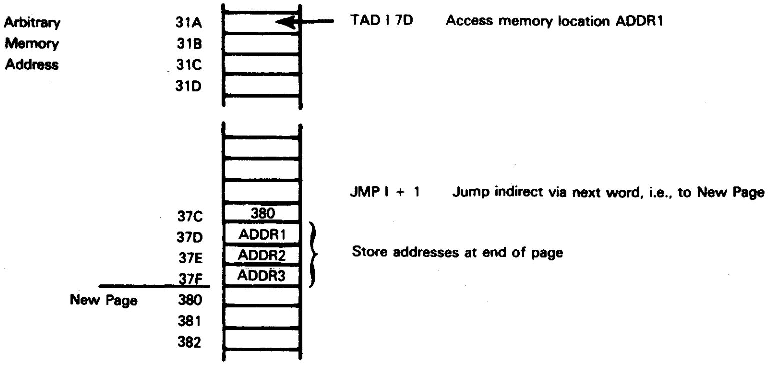
You can use indirect addressing to create the equivalent of a two-word, nonpaged direct addressing Jump instruction.

To do this, store the 12-bit absolute direct address directly following the Jump Indirect instruction. This may be illustrated as follows:


JMP I + 1
ADDR
ADDR
Jump occurs to this memory word which may be anywhere within 4096-word memory

You cannot use this technique with any memory reference instruction other than a Jump. That is because any other instruction would leave the Program Counter pointing to the indirect address as the next object code to be executed.

For memory reference instructions other than a Jump, reserve a few memory words at the end of the current page to store indirect addresses. This may be illustrated as follows:


Arbitrary Memory Address
31A
31B
31C
31D
TAD I 7D
Access memory location ADDR1
JMP I + 1
Jump indirect via next word, i.e., to New Page
37C
380
37D
ADDR1
37E
ADDR2
37F
ADDR3
Store addresses at end of page
New Page
380
381
382

The IM6100 also has auto-indexed indirect addressing. If you store an indirect address in any one of the eight memory words with addresses 008_{16} through $00F_{16}$ then, when the IM6100 CPU fetches this address, it will also increment and return it.

For example, you can store the beginning address of a table in memory location 008_{16}. You can subsequently read sequential table words by indirectly accessing the table. The IM6100 benchmark program illustrates this use of auto-indexing.

It is just as well that the IM6100 has indirect addressing with auto-increment, because it has no Data Counter or implied memory addressing. Volume I, Chapter 6 discusses the problems that result from using direct addressing to access sequential memory locations when programs are stored in read-only memory.

Note that the IM6100 makes no distinction between program and data memory. Thus Jump instructions use exactly the same memory addressing options as memory read or write instructions. The concept of separate program and data memory is a microcomputer phenomenon, because it was only with the advent of the microcomputer that programs started to be stored in read-only memory. Minicomputers use read/write memory for programs and data — and frequently a minicomputer will make no clear separation between the memory spaces that will be assigned to programs as against data.

The way in which the IM6100 handles subroutine calls represents an excellent illustration of the fact that minicomputer concepts can run into trouble in the world of microcomputers.

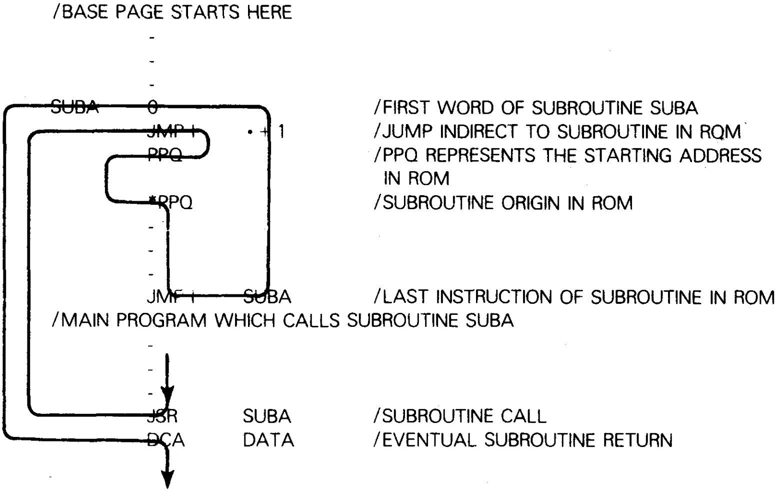
When a JSR instruction is executed, the return address is stored in the first word of the subroutine's object code.

The scheme certainly made sense to the PDP-8 designers; they visualized memory as a general read/write depository for programs and data. This scheme is nonviable when programs are stored in read-only memory, since you cannot write a return address in read-only memory. In order to use subroutines with an IM6100, you must origin all subroutines in read/write memory, then jump to a program sequence stored in read-only memory. This may be illustrated as follows:

```
/BASE PAGE STARTS HERE
            -
            -
            -
SUBA        0                       /FIRST WORD OF SUBROUTINE SUBA
            JMP I      • + 1        /JUMP INDIRECT TO SUBROUTINE IN ROM
            PPQ                     /PPQ REPRESENTS THE STARTING ADDRESS IN ROM
            *PPQ                    /SUBROUTINE ORIGIN IN ROM
            -
            -
            -
            JMP I      SUBA         /LAST INSTRUCTION OF SUBROUTINE IN ROM
/MAIN PROGRAM WHICH CALLS SUBROUTINE SUBA
            -
            -
            -
            JSR        SUBA         /SUBROUTINE CALL
            DCA        DATA         /EVENTUAL SUBROUTINE RETURN
```

Let us examine the path of instruction execution illustrated above.

Begin by looking at the JSR SUBA instruction in the main program which calls subroutine SUBA. SUBA is a label representing a location in the base page of memory. When the JSR SUBA instruction is executed, the address of the next instruction, arbitrarily illustrated above as a DCA instruction, will be stored in the memory word with label SUBA. The first instruction executed following the jump to subroutine is the instruction stored in the memory location following SUBA; this is the JMP I • + 1 instruction. This instruction jumps indirect via the address stored in the next memory location; we represent this memory location's contents with PPQ. PPQ is the address of the first instruction to be executed within the subroutine. This instruction, and all subsequent subroutine instructions are stored in read-only memory. The last instruction executed by the subroutine in read-only memory is the JMP I SUBA instruction. This instruction performs an indirect jump via the address stored at SUBA. This is the address of the DCA DATA instruction. This execution sequence may be illustrated as follows:



```
        /BASE PAGE STARTS HERE
                -
                -
                -
        SUBA    0                   /FIRST WORD OF SUBROUTINE SUBA
                JMP I     . + 1     /JUMP INDIRECT TO SUBROUTINE IN RQM
                PPQ                 /PPQ REPRESENTS THE STARTING ADDRESS
                                      IN ROM
                *PPQ                /SUBROUTINE ORIGIN IN ROM
                -
                -
                -
                JMP I     SUBA      /LAST INSTRUCTION OF SUBROUTINE IN ROM
        /MAIN PROGRAM WHICH CALLS SUBROUTINE SUBA
                -
                -
                -
                JSR       SUBA      /SUBROUTINE CALL
                DCA       DATA      /EVENTUAL SUBROUTINE RETURN
```

Handling subroutine calls through RAM has some non-obvious repercussions. First of all, at least the first page of every 4096-word memory bank must be read/write memory. In all probability, there will be more than one page of read/write memory.

Next, if you are going to initiate subroutines in Page 0 RAM, then when you power up the system, you must load this RAM from ROM. This is because RAM will lose its contents when powered down. Thus, every restart or reset procedure must include the execution of an instruction sequence which moves a block of data from ROM to Page 0 RAM.

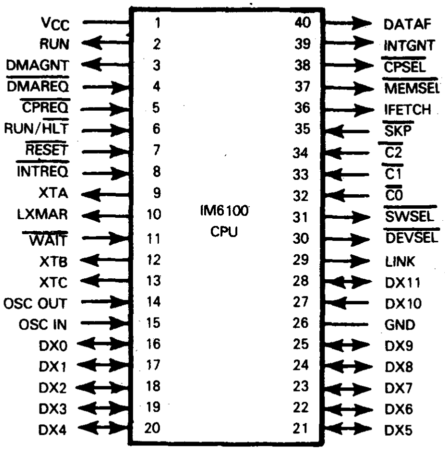
Possibly the most serious problem associated with calling subroutines through Page 0 RAM is the fact that existing PDP-8 software does not do that. Thus, if you are going to implement programs in read-only memory, the existing PDP-8 software base is not available to you — and that is one of the principal reasons for the IM6100's existence. Converting existing PDP-8 programs, so that subroutines are called through Page 0 RAM, is not a simple task. If you look again at the discussion of direct, paged addressing given in Volume I, Chapter 6, you will see that there are very significant problems associated with memory mapping. Programs cannot lie across page boundaries; therefore, the addition of a few instructions to any one program can have serious consequences. In some cases it may be possible to generate special assemblers and compilers that convert existing source programs into object programs which partition memory into ROM for programs and RAM for data, allowing subroutines to be called via the base page — but that assumes the base page has free space available for this purpose.

IM6100 STATUS FLAGS

The IM6100 has a single Carry status; it is called the Link or L status by PDP-8 and IM6100 literature.

IM6100 CPU PINS AND SIGNALS

IM6100 CPU pins and signals are illustrated in Figure 12-2. Once again, the minicomputer ancestry of the IM6100 is evident from the complex control signals input and output by the CPU. Minicomputer designers favor a rich variety of control signals on a System Bus because that makes the job of designing peripheral device controllers easier. Most microcomputers don't have



Pin Name	Description	Type
DX0 - DX11	Data and Address Bus	Bidirectional
OSC OUT	Crystal or external clock	Input
OSC IN	Crystal in or external clock ground	Input
XTA, XTB, XTC	External coded minor cycle timing	Output
LXMAR	Load external memory address strobe	Output
$\overline{\text{DEVSEL}}$	Device select for I/O transfers	Output
IFETCH	Instruction Fetch	Output
$\overline{\text{MEMSEL}}$	Memory select for memory references	Output
DATAF	Execution phase of indirect addressing instruction	Output
LINK	Link status	Output
RUN/$\overline{\text{HLT}}$	Run/Halt control	Input
RUN	CPU running status	Output
$\overline{\text{RESET}}$	Reset	Input
$\overline{\text{WAIT}}$	Wait state control	Input
$\overline{\text{C0}}$, $\overline{\text{C1}}$, $\overline{\text{C2}}$, SKP	CPU control during I/O operation	Input
$\overline{\text{DMAREQ}}$	DMA request	Input
DMAGNT	DMA grant	Output
$\overline{\text{INTREQ}}$	Interrupt request	Input
INTGNT	Interrupt grant	Output
$\overline{\text{CPREQ}}$	Control panel interrupt request	Input
$\overline{\text{CPSEL}}$	Control panel memory select	Output
$\overline{\text{SWSEL}}$	Switch register select	Output
VCC, GND	Power and Ground	

Figure 12-2. IM6100 CPU Signals And Pin Assignments

any peripheral devices, and complex System Busses simply increase the complexity and cost of surrounding the CPU with support logic. After examining the summary of IM6100 pins and signals which follows, compare it to the 8080A described in Chapter 4; then compare it with the MCS6500 described in Chapter 7. The MCS6500 represents the ultimate in simplicity.

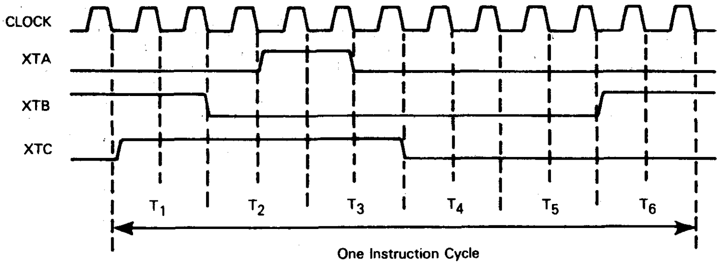
The IM6100 has a single 12-bit multiplexed Data and Address Bus represented by pins DX0 - DX11. Memory and I/O interface logic must use appropriate control signals in order to demultiplex data and addresses off this single bus.

The remaining signals can be divided into timing, bus control, CPU control, DMA and interrupt control.

Let us consider timing signals first.

OSC IN and OSC OUT are clock signal pins. If you are using the internal clock logic, then a crystal must be connected across these two pins. If you are using an externally generated clock signal, then it must be input via OSC OUT — OSC IN must be grounded.

XTA, XTB and XTC are three timing signals which are output for external logic to identify the state of an instruction's execution. Timing and states may be illustrated as follows:



Let us now look at the signals output by the CPU to define events on the System Bus.

LXMAR is output as a high pulse which external logic can use to strobe an address off the Data/Address Bus.

$\overline{\text{DEVSEL}}$, likewise, is output as a low pulse when information on the Address/Data Bus must be interpreted by I/O devices as device identification or I/O operation control.

IFETCH is output high for the duration of an instruction fetch. IFETCH may be used as a synchronization signal identifying the beginning of a new instruction cycle.

$\overline{\text{MEMSEL}}$ is output as a low pulse during a memory reference operation. Memory interface logic determines whether a memory read or a memory write is in progress via the condition of the XTA, XTB and XTC signals.

DATAF is a signal output high during the execute phase of an instruction that uses indirect addressing. This signal is required by extended memory control hardware if you have more than 4096 words of memory in your microcomputer system.

LINK is a signal output at all times to represent the level of the Link status. We include this signal among control outputs because you can use it as a direct external logic control signal. By executing instructions to set or reset the Link status you can modify the level of this control signal on a real time basis.

Let us now consider the control signals input by external logic to control CPU operations.

RUN/$\overline{\text{HLT}}$ is a control input which allows external logic to halt the CPU. This signal is similar to the Halt input switch some 8-bit microcomputers have, but its purpose in the IM6100 is to give control panel logic some means of executing program instructions one at a time. This helps in debugging. Whenever the CPU is running the **RUN** control signal is output high.

$\overline{\text{RESET}}$ is a typical reset input. When input low, it clears all CPU registers except for the Program Counter, which is loaded with FFF$_{16}$.

$\overline{\text{WAIT}}$ is a typical control input, used by slow external logic which needs to acquire more time to respond to a memory or I/O access. As long as $\overline{\text{WAIT}}$ is input low, the CPU will maintain register and signal levels but not advance the state of an instruction's execution.

C0, C1, C2 and SKP are very unusual input control signals. During an I/O operation, that is while an IOT instruction is being executed, **external logic can use these four control signals in order to determine CPU operations.**

Control signals C0, C1 and C2 are interpreted by the CPU as follows:

C2	C1	C0	
0	0	X	Transfer data from DX0 - DX11 to Program Counter (execute an absolute jump)
0	1	X	Add data on DX0 - DX11 to Program Counter (execute a program relative jump)
1	0	0	Load data from DX0 - DX11 to Accumulator
1	0	1	OR data from DX0 - DX11 with Accumulator
1	1	0	Transfer Accumulator contents to DX0 - DX11, then clear Accumulator
1	1	1	Transfer Accumulator contents to DX0 - DX11

X represents "don't care"; C0 may be 0 or 1.

If external logic inputs SKP low during an IOT instruction, then the CPU will skip the instruction which immediately follows the IOT. SKP logic is separate and distinct from C0, C1 and C2 logic.

Two signals support DMA operation. **External logic requests DMA access by inputting a low signal via $\overline{\text{DMAREQ}}$.** As soon as the current instruction has completed execution **the CPU responds by outputting DMAGNT high.** At this point the Data/Address Bus is floated. External logic must provide all DMA transfer signals; the only thing the CPU does in response to a DMA request, is float the Data/Address Bus for a single instruction cycle. The bus is floated for as long as $\overline{\text{DMAREQ}}$ is held low.

Interrupt logic reflects the IM6100 minicomputer heritage. **Normal interrupts are requested via $\overline{\text{INTREQ}}$ being input low. Upon acknowledging an interrupt the CPU will output INTGNT high.** Microcomputers are no different; but an IM6100 control panel interrupt request has its own dedicated $\overline{\text{CPREQ}}$ signal. Microcomputers do not allow so directly for the possible presence of a control panel.

Two additional control signals are provided to support the presence of a control panel. The IM6100 control panel will have its own memory in order to support logic required by switches and indicators of the control panel. **Following a control panel interrupt, $\overline{\text{CPSEL}}$ is output low instead of $\overline{\text{MEMSEL}}$, so that programs can be executed out of control panel memory, rather than out of main memory.**

There is also an instruction which reads the contents of control panel switches and ORs them with the contents of the Accumulator. **$\overline{\text{SWSEL}}$ is output low in order to inform control panel logic that switch levels must be returned as data on the Data/Address Bus.**

A SUMMARY OF IM6100 INTERRUPT PROCESSING

As we have just described, the IM6100 has two separate and distinct sets of interrupt logic: *one for a control panel, the other for general external logic.*

When general external logic requests an interrupt by inputting $\overline{\text{INTREQ}}$ low, the CPU completes execution of the current instruction; if interrupts are enabled it outputs INTGNT high, then executes a JSR instruction to memory location 0.

You must therefore leave memory location 0 free to store the interrupt service return address; and beginning at memory location 1 you must have the instruction sequence which will be executed to begin the interrupt service routine. Usually you will have a single instruction at memory location 1 which jumps indirect to an interrupt service routine beyond the base page.

Handling single levels of interrupt is quite straightforward with the IM6100. Handling nested interrupts is not so straightforward, unless you use an IM6101 Parallel Interface Element. In order to handle nested interrupts, you must create a programmed Stack in which you can store return addresses outside memory location 0.

The IM6101 Parallel Interface Element provides interrupt priority arbitration logic, plus a vectored interrupt acknowledge. The IM6101 is described next.

A control panel interrupt request differs from an external interrupt request in some important ways. First of all, the control panel interrupt request will be acknowledged even if the CPU is in a Halt state. Following a control panel interrupt request the CPU acknowledges the interrupt by storing the return address in memory location 0 — as it did for an external interrupt request. However, the control panel interrupt service routine is assumed to begin at memory location FFF_{16}, in control panel memory. Control panel memory is identified by the $\overline{\text{CPSEL}}$ signal being output low instead of $\overline{\text{MEMSEL}}$ during any memory reference operation.

THE IM6100 INSTRUCTION SET

The IM6100 instruction set is unusual because of limitations imposed by the fact that every single instruction generates a single 12-bit object code.

The IM6100 is very deficient in memory reference instructions; it has absolutely no immediate instructions but it has an incredible wealth of register operate instructions and I/O instructions. **Instructions are summarized in Table 12-1.**

Look first at the memory reference instructions. There is no simple memory read instruction or memory write instruction. The TAD instruction performs a binary add of memory with the Accumulator, leaving the result in the Accumulator. In order to read the contents of a memory word, you must clear the Accumulator, and then add memory to the Accumulator.

DCA is a deposit and clear instruction which is close to a memory write. When this instruction is executed the contents of the Accumulator are written to memory and the Accumulator is then cleared.

The only Boolean logic instructions provided AND the contents of memory with the Accumulator. You can also OR the MQ register and Accumulator contents. You must create an XOR from the AND, OR and complement.

There is a single Jump instruction which uses absolute, paged direct or indirect addressing. There are no conditional Jump instructions; however, there are a wealth of conditional Skip instructions. In order to perform conditional branches, you must use skip logic.

The total absence of immediate instructions results from the fact that no instructions have two words of object code. Where you would have used an immediate instruction, you must instead use the TAD instruction to add a constant to the zeroed Accumulator. It is important to note that given the architecture of the IM6100 CPU, immediate instructions are not very valuable — and the lack of them is not consequential. Since you only have one Accumulator and no Data Counters, you do not need immediate instructions in order to load initial addresses or data.

The IM6100 instruction set and CPU philosophy is the exact opposite of COSMAC or the EA9002.

The following symbols are used in Table 12-1.

A Accumulator

*ADDR Addressing operands. * indicates indirect mode specified. ADDR may be zero page or current page address as described in the text.

CMND Three-bit I/O command.

DEV Six-bit Device address

EA Effective Address generated by *ADDR operands.

IE Interrupt Enable flip-flop.

L Link status

MQ MQ register

PC Program Counter

SR Switch register — a 12-bit register external to the CPU.

x<y> The yth bit of the quantity x. For example, A<0> specifies the low bit of the Accumulator.

[ ] Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.

Λ Logical AND

V Logical OR

← Data is transferred in the direction of the arrow.

Under the heading of STATUS in Table 12-1, an X indicates that the Link is modified in the course of the instruction's execution. If there is no X, it means that the Link maintains the value it had before the instruction was executed.

Table 12-1. IM6100 Instruction Set Summary

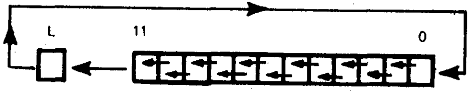
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	OPERATION PERFORMED
				C	
I/O	IOT	DEV,CMND	1		[DEV]←[CMND] Issue the command to the device.
PRIMARY MEMORY REFERENCE	DCA	*ADDR			[EA]←[A] [A]←0 Deposit the Accumulator in memory; then clear Accumulator.
MEMORY OPERATE	AND	*ADDR	1		[A]←[A] ∧ [EA] AND Accumulator with memory.
	TAD	*ADDR	1	X	[A]←[A] + [EA] Add memory to Accumulator.
	ISZ	*ADDR	1		[EA]←[EA] + 1 If [EA] = 0; skip Increment memory and skip if zero.
JUMP	JMP	*ADDR			[PC]←EA Branch unconditional.
	JMS	*ADDR			[EA]←[PC] + 1 [PC]←EA + 1 Jump to subroutine unconditional.
REGISTER OPERATE	IAC		1	X	[A]←[A] + 1 Increment Accumulator.
	RAL		1	X	 Rotate Accumulator left one bit through Link.

Table 12-1. IM6100 Instruction Set Summary (Continued)

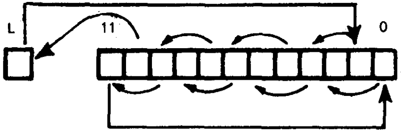
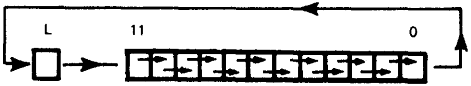
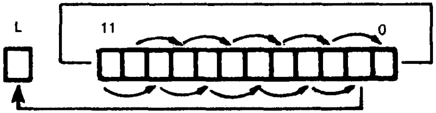
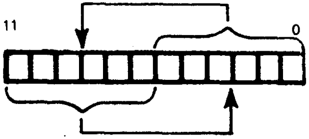
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	OPERATION PERFORMED
				C	
REGISTER OPERATE (CONTINUED)	RTL		1	X	 Rotate Accumulator left two bits through Link.
	RAR		1	X	 Rotate accumulator *right one* bit through Link.
	RTR			X	 Rotate Accumulator right two bits through Link.
	BSW				 [A] Swap the two halves of the Accumulator.

Table 12-1. IM6100 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	OPERATION PERFORMED
				C	
REGISTER OPERATE (CONTINUED)	CMA		1		$[A] \leftarrow \overline{[A]}$ Complement Accumulator contents.
	CNA		1		$[A] \leftarrow \overline{[A]} + 1$ Negate (twos complement) Accumulator contents.
	CLA		1		$[A] \leftarrow 0$ Clear Accumulator.
	CLA IAC		1		$[A] \leftarrow 1$ Clear, then increment Accumulator.
	STA		1		$[A] \leftarrow FFF_{16}$ Set Accumulator bits to all ones.
BRANCH ON CONDITION	SNL		1		If $[L]=1$; $[PC] \leftarrow [PC]+2$ Skip on Link set.
	SZL		1		If $[L]=0$; $[PC] \leftarrow [PC]+2$ Skip on Link reset.
	SZA		1		If $[A]=0$; $[PC] \leftarrow [PC]+2$ Skip on Accumulator zero.
	SNA		1		If $[A] \neq 0$; $[PC] \leftarrow [PC]+2$ Skip on Accumulator nonzero.
	SZA SNL		1		If $[A]=0$ or $[L]=1$; $[PC] \leftarrow [PC]+2$ Skip if either Accumulator zero or Link set.
	SNA SZL		1		If $[A] \neq 0$ and $[L]=0$; $[PC] \leftarrow [PC]+2$ Skip if Accumulator nonzero and Link reset.
	SMA		1		If $[A] < 0$; $[PC] \leftarrow [PC]+2$ Skip if Accumulator negative.
	SPA		1		If $[A] \geqslant 0$; $[PC] \leftarrow [PC]+2$ Skip if Accumulator positive or zero.
	SMA SNL		1		If $[A] < 0$ or $[L]=1$; then $[PC] \leftarrow [PC]+2$ Skip if Accumulator negative or Link set.

Table 12-1. IM6100 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	OPERATION PERFORMED
				C	
BRANCH ON CONDITION (CONTINUED)	SPA SZL		1		If [A] > 0 and [L]=0; then [PC]←[PC] + 2 Skip if Accumulator positive and Link reset.
	SMA SZA		1		If [A] ≤ 0; then [PC]←[PC] + 2 Skip if Accumulator zero or negative.
	SPA SNA		1		If [A] > 0; then [PC]←[PC] + 2 Skip if Accumulator positive.
	SMA SZA SNL		1		If [A] ≤ 0 or [L]=1 Skip if Accumulator less than or equal zero or if Link set.
	SPA SNA SZL		1		If [A] > 0 and L=0 Skip if Accumulator positive and Link reset.
BRANCH ON CONDITION AND OPERATE	SZA CLA		1		If [A]=0; [PC]←[PC] + 2. [A]←0 Skip on Accumulator zero. Clear Accumulator.
	SNA CLA		1		If [A]≠0; [PC]←[PC] + 2. [A]←0 Skip on Accumulator nonzero. Clear Accumulator.
	SMA CLA		1		If [A] < 0; [PC]←[PC] + 2. [A]←0 Skip on Accumulator negative. Clear Accumulator.
	SPA CLA		1		If [A] ≥ 0; [PC]←[PC] + 2. [A]←0 Skip on Accumulator greater than or equal zero. Clear Accumulator.
REGISTER-REGISTER MOVE	LAS		1		[A]←[SR] Load Accumulator from Switch register.
	MQL		1		[MQ]←[A] [A]←0 Load MQ register from Accumulator. Clear Accumulator.

Table 12-1. IM6100 Instruction Set Summary (Continued)

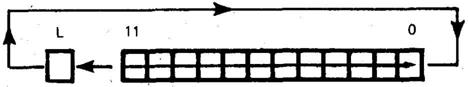
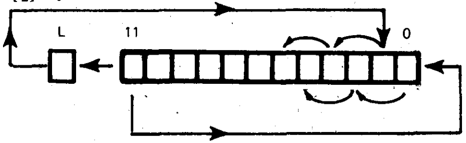
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	OPERATION PERFORMED
				C	
REGISTER-REGISTER MOVE (CONTINUED)	SWP		1		[A]⟷[MQ] Exchange Accumulator and MQ.
	CAM		1		[A]←0 [MQ]←0 Clear Accumulator and MQ.
	ACL		1		[A]←[MQ] Load MQ into Accumulator.
	CLA SWP		1		[A]←0 [A]⟷[MQ] Clear Accumulator; then swap Accumulator and MQ.
REGISTER-REGISTER OPERATE	OSR		1		[A]←[A] V [SR] OR Accumulator with Switch register.
	MQA		1		[A]←[A] V [MQ] OR Accumulator with MQ.
STATUS AND REGISTER OPERATE	CLL RAL		1	X	[L]←0  Clear Link, then rotate Accumulator left one bit through Link.
	CLL RTL		1	X	[L]←0  Clear Link, then rotate Accumulator left two bits through Link.

Table 12-1. IM6100 Instruction Set Summary (Continued)

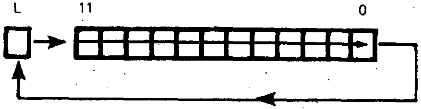
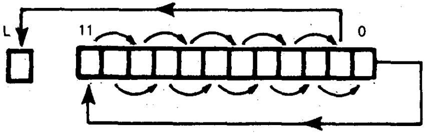
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	OPERATION PERFORMED
				C	
STATUS AND REGISTER OPERATE (CONTINUED)	CLL RAR		1	X	[L]←0  Clear Link, then rotate Accumulator right one bit through Link.
	CLL RTR		1	X	[L]←0  Clear Link, then rotate Accumulator right two bits through Link.
	CLA CLL		1		[A]←0 [L]←0 Clear Accumulator and Link.
	GTL		1		[A]←0 [A<0>]←[L] Clear Accumulator, then rotate Link into low bit.

Table 12-1. IM6100 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	OPERATION PERFORMED
				C	
BRANCH	SKP		1		[PC]←[PC]+2 Skip next instruction.
INTERRUPT	SKON		1		If [IE]=1; [PC]←[PC]+2 If interrupts enabled, skip next instruction.
	ION		1		[IE]←1 Enable interrupts.
	IOF		1		[IE]←0 Disable interrupts.
	SRQ		1		Skip next instruction if Interrupt Request bus is low.
STATUS	CML		1	X	[L]←[L] Complement Link.
	CLL		1	X	[L]←0 Reset Link.
	STL		1	X	[L]←1 Set Link.
	GTF		1		A<11> ← [L] A<9> ← $\overline{\text{INTREQ}}$ A<7> ← [IE] Get flags.
	RTF		1	X	[L]←A<11>; [IE]←1 Return Link and enable interrupts after the execution of the next sequential instruction.
	CAF		1	X	[L]←0 [A]←0 [IE]←0 Clear all flags.
	HLT		1		Halt
	NOP		1		No Operation

The following symbols are used in Table 12-2.

a	One bit which determines if indirect addressing is used.
b	One bit which determines if current or zero page is used.
ccccccc	Seven-bit page address.
dddddd	Six-bit device code.
eee	Three-bit I/O command.

Most instructions are described in this manner:

mnemonic xxxx
yyy

where xxxx is the octal object code associated with the mnemonic and yyy is the hexadecimal object code associated with the mnemonic. IM6100 literature uses octal notation.

Some instructions have this form in the machine cycles column:

a/b/c

a is the number of cycles required using normal addressing.
b is the number of cycles required using indirect addressing.
c is the number of cycles required using auto-indexed addressing.

Table 12-2. IM6100 Instruction Set Object Codes

INSTRUCTION	OBJECT CODE	BYTES	MACHINE CYCLES
ACL	7701 FC1	1	1
AND *ADDR	000abccccccc	1	1.0/1.5/1.6
BSW	7002 E02	1	1.5
CAF	6007 ·C07	1	1.7
CAM	7621 F91	1	1.0
CIA	7041 E21	1	1
CLA	7200 E80	1	1
CLA CLL	7300 EC0	1	1
CLA IAC	7201 E81	1	1
CLA SWP	7721 FD1	1	1
CLL	7100 E40	1	1
CLL RAL	7104 E44	1	1.5
CLL RAR	7100 E48	1	1.5
CLL RTL	7106 E46	1	1.5
CLL RTR	7112 E4A	1	1.5
CMA	7040 E20	1	1
CML	7020 E10	1	1
DCA *ADDR	011abccccccc	1	1.1/1.6/1.7
GTF	6004 C04	1	1.7
GTL	7204 E84	1	1
HLT	7402 F02	1	1
IAC	7001 E01	1	1
IOF	6002 C02	1	1.7
ION	6001 C01	1	1.7
IOT DEV,CMND	110ddddddeee	1	1
ISZ *ADDR	010abccccccc	1	1
JMP *ADDR	101abccccccc	1	1.0/1.5/1.6
JMS *ADDR	100abccccccc	1	1.1/1.6/1.7
LAS	7604 F84	1	1.5
MQA	7501 F41	1	1

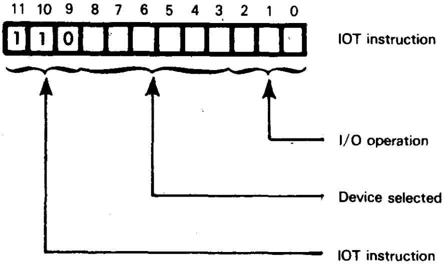
Table 12-2. IM6100 Instruction Set Object Codes (Continued)

INSTRUCTION	OBJECT CODE	BYTES	MACHINE CYCLES
MQL	7421 F11	1	1
NOP	7000 E00	1	1
OSR	7404 F04	1	1.5
RAL	7004 E04	1	1.5
RAR	7010 E08	1	1.5
RTF	6004 C04	1	1.7
RTL	7006 E06	1	1.5
RTR	7012 E0A	1	1.5
SGT	6006 C06	1	1.7
SKON	6000 C00	1	1.7
SKP	7410 F08	1	1
SMA	7500 F40	1	1
SMA CLA	7700 FC0	1	1
SMA SNL	7520 F50	1	1
SMA SZA	7540 F60	1	1
SMA SZA SNL	7560 F70	1	1
SNA	7450 F28	1	1
SNA CLA	7650 FA8	1	1
SNA SZL	7470 F38	1	1
SNL	7420 F10	1	1
SPA	7510 F48	1	1
SPA CLA	7710 FC8	1	1
SPA SNA	7550 F68	1	1
SPA SNA SZL	7570 F78	1	1
SPA SZL	7530 F58	1	1
SRQ	6003 C03	1	1.7
STA	7240 EA0	1	1

Table 12-2. IM6100 Instruction Set Object Codes (Continued)

INSTRUCTION	OBJECT CODE	BYTES	MACHINE CYCLES
STL	7120 E50	1	1
SWP	7521 F51	1	1
SZA	7440 F20	1	1
SZA CLA	7640 FA0	1	1
SZA SNL	7460 F30	1	1
SZL	7430 F18	1	1
TAD *ADDR	001abccccccc	1	1.0/1.5/1.6

IM6100 I/O instructions are also unusual. At one extreme, you could say that the IM6100 only has one I/O instruction, which outputs a 9-bit code on the Data/Address Bus, which external logic can interpret in any way. In practice, the PDP-8 minicomputer interprets this 9-bit code as follows:



If you are designing a product from scratch, there is no reason why you must use the 9-bit code output by an IOT instruction as illustrated above. If you are using existing PDP-8 software, you are forced to conform to the above IOT instruction interpretation.

The most unusual feature of IM6100 I/O instructions is the fact that external devices can talk back and control the CPU via the $\overline{C0}$, $\overline{C1}$, $\overline{C2}$ and $\overline{SKP}$ control inputs which we have already described.

THE IM6100 BENCHMARK PROGRAM

The IM6100 benchmark program may be illustrated as follows:

```
        CLA                CLEAR THE ACCUMULATOR
        TAD      IOBUF     LOAD IO BUF BASE ADDRESS INTO
        DCA      8         AUTO-INCREMENT LOCATION
        TAD      TABLE     LOAD TABLE FIRST FREE BYTE ADDRESS
        DCA      9         INTO AUTO-INCREMENT LOCATION
        TAD      CNT       LOAD BYTE COUNT
        IAC      CMA       COMPLEMENT AC AND INCREMENT
        DCA      INDEX     SAVE IN RAM
LOOP    TADI     8         LOAD NEXT WORD FROM IOBUF
        DCAI     9         STORE IN NEXT FREE TABLE WORD
        ISZ      INDEX     INCREMENT BYTE COUNT COMPLEMENT
        JMP      LOOP      RETURN FOR MORE
        TAD      9         AT END RESTORE NEW TABLE FIRST
        DCA      TABLE     FREE BYTE ADDRESS
```

The benchmark program illustrated above uses auto-increment memory locations 8 and 9 to indirectly address IOBUF and TABLE. These two tables can be of any length within the constraints of the available 4096-word address space. Three other words in the base page are reserved to store the IOBUF base address, the address of the first free byte in TABLE and the byte count. An additional memory word in the base page is used to store the complement of the byte count. This location is represented by the label INDEX.

THE IM6101 PARALLEL INTERFACE ELEMENT (PIE)

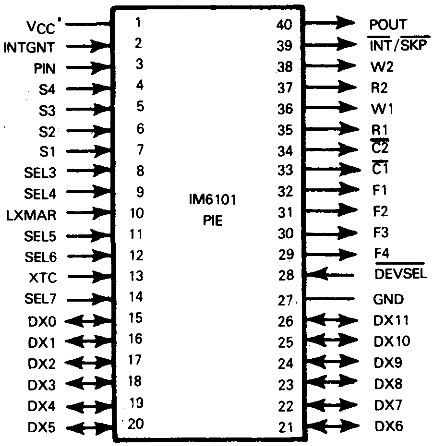
The IM6101 Parallel Interface Element should be viewed as a necessary adjunct to the IM6100 CPU — just as the 8228 System Bus Controller should always be used with the 8080A CPU.

The IM6100 CPU, being a copy of the PDP-8 minicomputer, has a number of features which are not well suited to the average microcomputer application; but that is no fault of the IM6100 chip designer — his product was specified for him. The IM6101, on the other hand, is a well thought-out part that goes a long way towards rectifying the problems that you are likely to encounter if you try to design logic around the IM6100 CPU.

Up to 31 IM6101 Parallel Interface Elements may be connected to a single IM6100 CPU. Each IM6101 provides four Flag signal outputs, four Sense signal inputs, two WRITE output strobes, and two READ output strobes. The IM6101 also enhances IM6100 interrupt processing capability by providing an interrupt request daisy chain along with vectored response to an interrupt acknowledge.

Figure 12-1 illustrates the logic provided by an IM6101 Parallel Interface Element.

The IM6101, like all members of the IM6100 family, is fabricated using CMOS technology; it requires a single power supply that may range between + 4 and + 10V and is packaged as a 40-pin DIP.



Pin Name	Description	Type
DX0 - DX11	Data and Address Bus	Bidirectional
LXMAR, $\overline{DEVSEL}$, XTC	Control signals from CPU	Input
$\overline{C1}$, $\overline{C2}$, $\overline{SKP}$	CPU control signals	Output
$\overline{INT}$	Interrupt Request	Output
SEL3 - SEL7	Individual IM6101 Select	Input
F1 - F4	Control Flags	Output
W1, W2	Write Pulse Lines	Output
R1, R2	Read Pulse Lines	Output
S1 - S3	Status Lines	Input
PIN	Priority In	Input
POUT	Priority Out	Output
VCC, GND	Power and Ground	

Figure 12-3. IM6101 Parallel Interface Element Signals And Pin Assignments

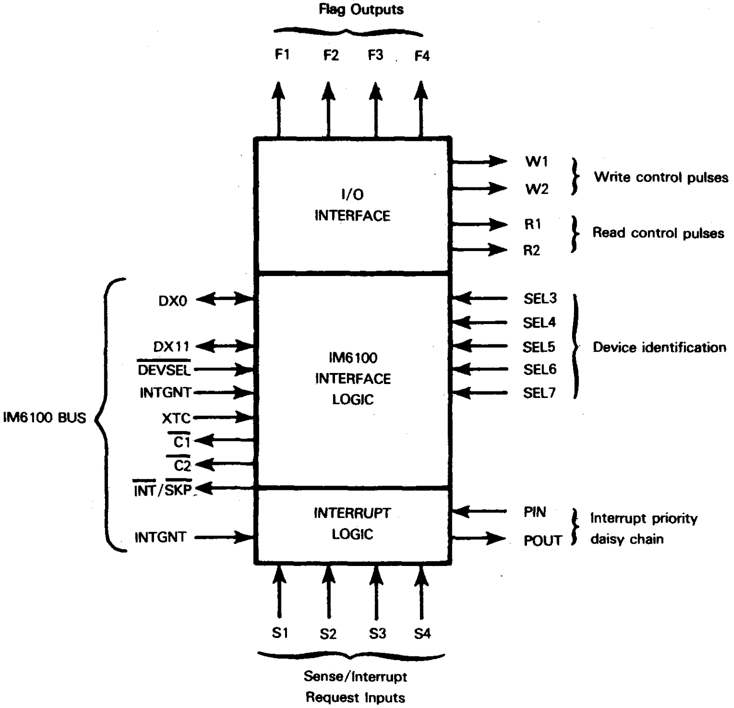


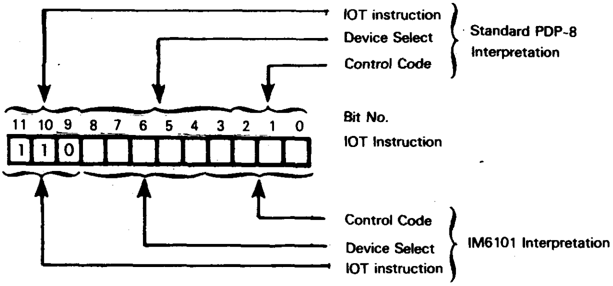
Figure 12-4. Logic Of The IM6101 PIE

IM6101 PARALLEL INTERFACE ELEMENT PINS AND SIGNALS

Figure 12-3 illustrates the pins and signals of the IM6101 Parallel Interface Element. We will combine a description of these pins and signals with a discussion of IM6101 logic and capabilities. Figure 12-4 illustrates the important logic features of the IM6101.

IM6101 PROGRAMMING

You access an IM6101 Parallel Interface Element using IOT instructions; this is how the IM6101 will interpret an IOT instruction code as it appears on the Data/Address Bus:



Note that the Im6101 and the PDP-8 differ in their interpretation of the IOT instruction code.

Table 12-3. IM6101 Interpretation Of IOT Instruction Control Bits 3 - 0

CONTROL Bit: 3 2 1 0	INTERPRETATION
0 0 0 0	Generate a low pulse output on READ1.
1 0 0 0	Generate a low pulse output on READ2.
0 0 0 1	Generate an active pulse output on WRITE1.
1 0 0 1	Generate an active pulse output on WRITE2.
0 0 1 0	Test the S1 status input. If it is active, output a low pulse via $\overline{\text{INT/SKP}}$, to be interpreted by the IM6100 CPU as an $\overline{\text{SKP}}$ pulse.
0 0 1 1	Test the S2 status input. If it is active, output a low pulse via $\overline{\text{INT/SKP}}$, to be interpreted by the IM6100 CPU as an $\overline{\text{SKP}}$ pulse.
1 0 1 0	Test the S3 status input. If it is active, output a low pulse via $\overline{\text{INT/SKP}}$, to be interpreted by the IM6100 CPU as an $\overline{\text{SKP}}$ pulse.
1 0 1 1	Test the S4 status input. If it is active, output a low pulse via $\overline{\text{INT/SKP}}$, to be interpreted by the IM6100 CPU as an $\overline{\text{SKP}}$ pulse.
0 1 0 0	Place the contents of Control Register A on the Data Bus as data. The IM6100 CPU will OR Control Register A contents with the Accumulator contents.
0 1 0 1	Write the contents of the Accumulator to Control Register A.
1 1 0 1	Write the contents of the Accumulator to Control Register B.
1 1 0 0	Write the contents of the Accumulator to the Interrupt Vector register.
0 1 1 0	Set Output Signal F1 high and set Control Register A bit 3 to one.
1 1 1 0	Set Output Signal F3 high and set Control Register A bit 1 to one.
0 1 1 1	Reset Output Signal F1 low and reset Control Register A bit 3 to zero.
1 1 1 1	Reset Output Signal F3 low and reset bit 10 of Control Register A to zero.

In order to communicate with the IM6100 CPU, the IM6101 PIE connects to the 12 Data/Address Bus lines DX0 - DX11; it also has the necessary additional control and timing signals LXMAR, $\overline{\text{DEVSEL}}$ and XTC. The IM6101 does not need or receive timing signals XTA and XTB.

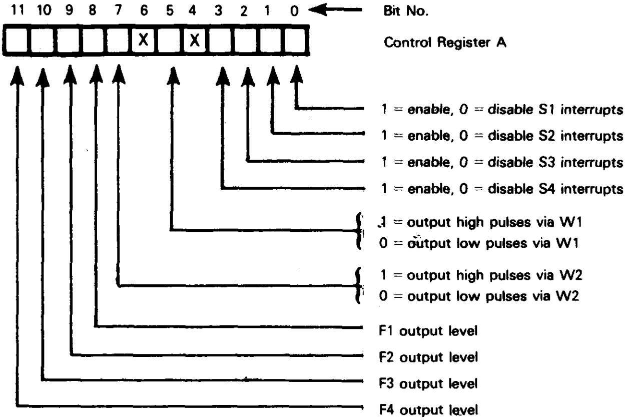
Three of the four CPU control signals are returned: $\overline{\text{C1}}$, $\overline{\text{C2}}$ and $\overline{\text{SKP}}$. The IM6101 PIE does not use $\overline{\text{C0}}$. Notice that **the IM6101 PIE outputs $\overline{\text{INT}}$ and $\overline{\text{SKP}}$ signals via a single pin.** This is possible because these two signals are active at different times during an instruction cycle. Thus a low INT pulse will never be mistaken for a skip input control and similarly an SKP will never be interpreted as an interrupt request.

The input signals SEL3 - SEL7 give each IM6101 its identity. IM6101 logic compares bits 4 through 8 of the IOT instruction with the signal levels being input at SEL3 through SEL7. Upon finding a match, the IM6101 considers itself selected. You will normally tie signals SEL3 through SEL7 individually to power or ground, thus permanently defining the IM6101 select code; but you can connect these signals to programmable external logic and thence change the IM6101 select code under program control.

Note that a zero device select code is interpreted by the IM6100 CPU as an internal instruction. This device select code cannot be used to select an IM6101; that is why only thirty-one IM6101 PIEs may be addressed by a single IM6100 CPU, even though a 5-bit select code is available.

The four low order bits of an IOT instruction object code are interpreted as control information. Table 12-3 explains how controls are interpreted.

There are two Control registers within the IM6101: Control Register A and Control Register B. Control Register A can be written into or its contents can be read. Control Register A contents are interpreted as follows:


11 10 9 8 7 6 5 4 3 2 1 0
Bit No.
X
X
Control Register A
1 = enable, 0 = disable S1 interrupts
1 = enable, 0 = disable S2 interrupts
1 = enable, 0 = disable S3 interrupts
1 = enable, 0 = disable S4 interrupts
1 = output high pulses via W1
0 = output low pulses via W1
1 = output high pulses via W2
0 = output low pulses via W2
F1 output level
F2 output level
F3 output level
F4 output level

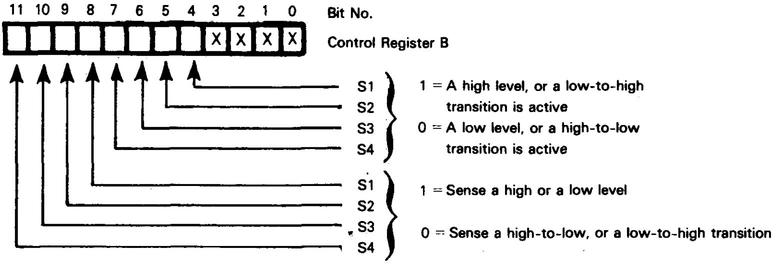
The status of the four Flag outputs, F1 - F4, is determined by the contents of the four high order Control Register A bits. In addition, specific control instructions shown in Table 12-3 allow F1 and F3 to be set or reset. You can therefore modify F1 and F3 in two ways — by executing specific IOT instructions, or by loading appropriate information into the flag bits of Control Register A.

Bits 5 and 7 of Control Register A determine whether the Write output signals W1 and W2 will pulse high or low. The actual high or low pulse, however, is triggered by the appropriate write control, as defined in Table 12-3. Note that you cannot select read pulse levels; when the appropriate IOT instruction pulses one of the read lines, the read pulse will always be low.

You use bits 0 through 3 of Control Register A to determine whether the status inputs S1 - S3 are to function as interrupt requests, or as statuses which will trigger IM6100 CPU skip control logic. You can define the function of each signal in any way and thus create any combination of interrupt requests and skip controls.

Control Register B determines what will constitute an "active" state for each of the four individual sense inputs. Each sense input has two control bits in Control Register B.

one of which determines whether signal level or signal transition will constitute the active state; the other control bit determines polarity. Here is Control Register B format:


11 10 9 8 7 6 5 4 3 2 1 0
Bit No.
X X X X
Control Register B
S1
S2
S3
S4
1 = A high level, or a low-to-high transition is active
0 = A low level, or a high-to-low transition is active
S1
S2
S3
S4
1 = Sense a high or a low level
0 = Sense a high-to-low, or a low-to-high transition

By appropriately setting the two bits of Control Register B which are assigned to any sense input, you can cause a high level, a low level, a high-to-low transition or a low-to-high transition to be the active sense signal state.

Note carefully that Control Register B determines what will constitute an active sense condition. Control Register B does not hold sense input information.

Excluding interrupt logic, let us now summarize the IM6101 in terms of the interface it presents to the CPU and the interface it presents to external logic.

To the CPU, the IM6101 is a simple I/O device; it does everything an I/O device is supposed to do and that is what is important — because the IM6100 demands much more of its I/O devices than any of the 8-bit microprocessors we have described. The IM6101 takes care of the complex IM6100 bus control signals, and external logic beyond the IM6101 now sees a simple interface consisting of four control outputs (FL1 - FL4), four sense inputs or interrupt request inputs (S1 - S4), plus simple read and write strobes. This is a microprocessor interface we can recognize.

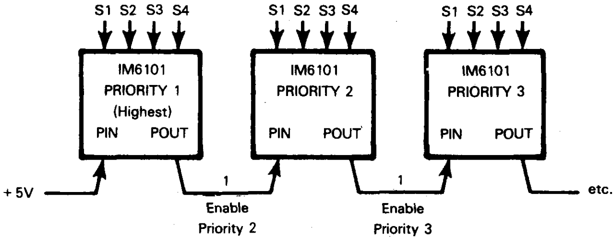
Observe that the IM6101 provides necessary control logic for parallel data to be transferred into or out of an IM6100 microcomputer system, but the IM6101 has no I/O ports. In order to implement I/O ports, you must include an appropriate 12-bit register or buffer, then use the read or write pulses as enable signals.

IM6101 INTERRUPT HANDLING LOGIC

The IM6101 has typical daisy chain priority interrupt logic, implemented via the PIN and POUT signals.

PIN must be a high input if an IM6101 is to acknowledge an interrupt request arriving via one of the four sense lines.

Therefore, the IM6101 electrically closest to the CPU must have its PIN input connected to power, equivalent to a high input so that its interrupt request logic will always be enabled. So long as no interrupt request is active at this highest priority IM6101, a high signal will be output via POUT; it becomes the PIN input for the next IM6101 in the daisy chain:

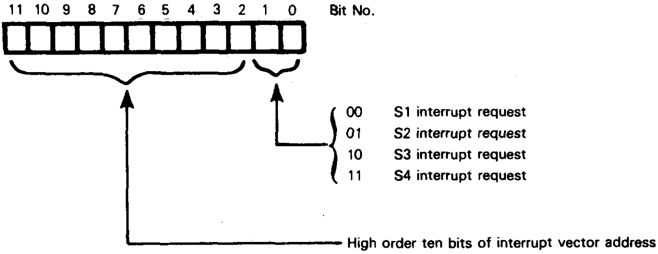


As soon as an interrupt request occurs via one of the sense lines at an IM6101, it immediately sends out interrupt request low pulses via $\overline{\text{INT}}/\overline{\text{SKP}}$; simultaneously the IM6101 outputs POUT low, thus disabling all interrupt request logic at lower priority PIEs in the daisy chain.

The IM6100 CPU acknowledges the interrupt request, providing interrupts are enabled at the CPU, by executing a "Jump-to-Subroutine at memory address 000" instruction. Thus the interrupt return address is stored in memory location 000 and the instruction object code stored in memory location 001 becomes the first instruction executed following the interrupt acknowledge. Upon acknowledging an interrupt, the IM6100 outputs INTGNT high. The first IOT instruction executed, of any type or to any device, resets INTGNT low.

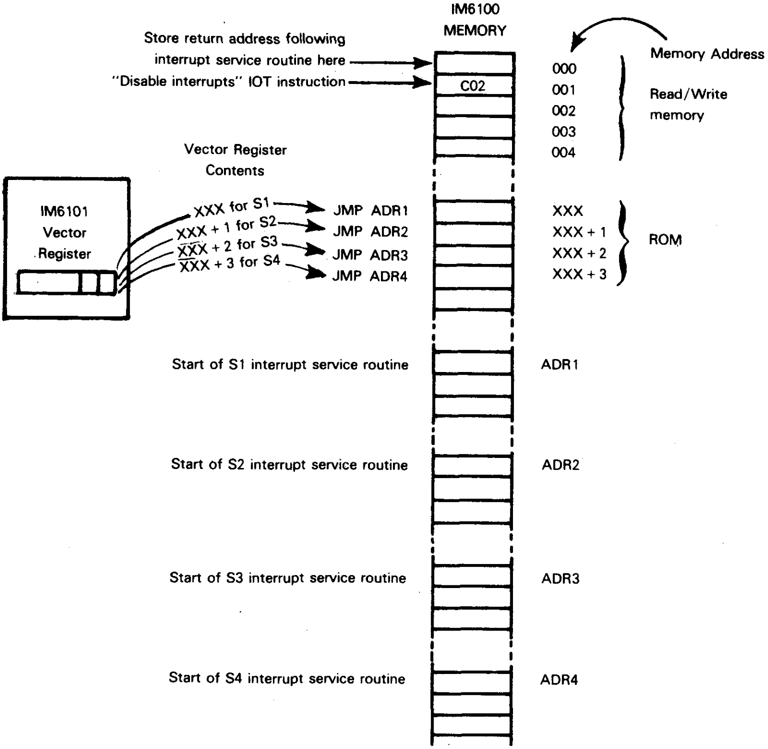
So far this is elementary daisy chain interrupt request/acknowledge logic.

The IM6101 has an Interrupt Vector register which you can write into with the appropriate IOT control code as defined in Table 12-3. The Vector register contents are interpreted as follows:



As we have already stated, the first IOT instruction executed by the IM6100 following an interrupt acknowledge will reset INTGNT low; this IOT instruction's execution will simultaneously cause the IM6101 which requested the interrupt to place its interrupt vector address contents on the Data Bus (DX0 - DX11). The IM6101 also outputs low signals at $\overline{\text{C1}}$ and $\overline{\text{C2}}$; when the IM6101 receives low inputs at $\overline{\text{C1}}$ and $\overline{\text{C2}}$, it interprets the Data Bus contents as an address, which will be loaded into the Program Counter. Thus program execution jumps to the vector address output by the IM6101.

Normally the instruction executed out of memory location 001 by the IM6100 will be a "Disable Interrupt" instruction; this is an IOT instruction with object code $C02_{16}$. Being an IOT instruction, it will cause the IM6101 which requested the interrupt to force a branch to the one specific memory location which has been assigned to the active interrupt request line. In this memory location you should store a JMP instruction, taking program execution to the interrupt service routine dedicated to the acknowledged interrupt request. This logic may be illustrated as follows:


IM6100
MEMORY
Store return address following
interrupt service routine here
"Disable interrupts" IOT instruction
C02
Memory Address
000
001
002
003
004
Read/Write
memory
Vector Register
Contents
IM6101
Vector
Register
XXX for S1
XXX + 1 for S2
XXX + 2 for S3
XXX + 3 for S4
JMP ADR1
JMP ADR2
JMP ADR3
JMP ADR4
XXX
XXX + 1
XXX + 2
XXX + 3
ROM
Start of S1 interrupt service routine
ADR1
Start of S2 interrupt service routine
ADR2
Start of S3 interrupt service routine
ADR3
Start of S4 interrupt service routine
ADR4

A word of caution when handling interrupts with an IM6100: you must save the return address which is in memory location 000 if you are going to use subroutines or nested interrupts.

DATA SHEETS

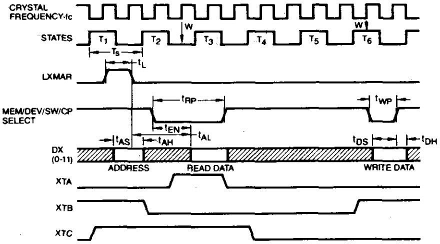
The following pages contain specific electrical characteristics for the IM6100 and IM6101 PIE.

ABSOLUTE MAXIMUM RATINGS

Supply Voltage	IM6100/C +4.0V to +7.0V	Operating Temperature Range	
	IM6100A +4.0V to 11.0V	Commercial	0°C to +75°C
Input or Output Voltage Applied	GND −0.3V to V_{CC} +0.3V	Industrial	−40°C to +85°C
Storage Temperature Range	−65°C to +125°C	Military	−55°C to +125°C

DC CHARACTERISTICS V_{cc} = 5.0V ± 10% (IM6100), 10.0V ± 10% (IM6100A), T_A = Commercial, Industrial or Military

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Logical "1" Input Voltage	V_{IH}		70% V_{cc}			V
Logical "0" Input Voltage	V_{IL}				20%V_{CC}	V
Input Leakage	I_{IL}	OV ≤ V_{IN} ≤ V_{CC}	−1.0		1.0	μA
Logical "1" Output Voltage	V_{OH2}	I_{OUT}=0	V_{cc}−0.01			V
Logical "1" Output Voltage	V_{OH1}	I_{OH}= −0.2mA	2.4			V
Logical "0" Output Voltage	V_{OL2}	I_{OUT} =0			GND +0.01	V
Logical "0" Output Voltage	V_{OL1}	I_{OL} =1.6 mA			0.45	V
Output Leakage	I_O	OV ≤ V_o ≤ V_{cc}	−1.0		1.0	μA
Supply Current	I_{CC}	V_{CC} = 5.0 volts			2.5	mA
		V_{CC} = 10.0 volts			10.0	mA
		C_L= 50 pF; TA = 25°C				
		F_{CLOCK} = Operating Frequency				
Input Capacitance	C_{IN}			5.0		pF
Output Capacitance	C_O			8.0		pF



IM6100 TIMING AND STATE SIGNALS

AC CHARACTERISTICS (T_A = 25° C), **Derate 0.390/°C**

PARAMETER	SYMBOL	IM6100 V_{CC} = 5.0 f_c = 4MHz	IM6100A V_{CC} = 10.0 f_c = 8 MHz	IM6100C V_{CC} = 5.0 f_c = 3.3MHz	UNITS
Major State Time	T_s	500	250	600	ns
LXMAR Pulse Width	t_L	240	120	280	ns
Address Setup Time	t_{AS}	50	30	80	ns
Address Hold Time	t_{AH}	250	125	280	ns
Access Time From LXMAR	t_{AL}	500	250	600	ns
Output Enable Time	t_{EN}	240	120	280	ns
Read Pulse Width	t_{RP}	700	350	800	ns
Write Pulse Width	t_{WP}	240	120	280	ns
Data Setup Time	t_{DS}	240	120	280	ns
Data Hold Time	t_{DH}	100	50	160	ns

ABSOLUTE MAXIMUM RATINGS

Supply Voltage	
IM6101	+8.0V
IM6101A	+12.0V
Applied Input or Output Voltage	GND −0.3V to V_{CC} +0.3V
Storage Temperature Range	−65°C to 150°C

Operating Temperature Range	
Industrial	−40°C to 85°C
Military	−55°C to 125°C
Operating Voltage Range	
IM6101	4V to 7V
IM6101A	4V to 11V

DC CHARACTERISTICS V_{CC} = Operating Voltage Range T_A = Temperature Range

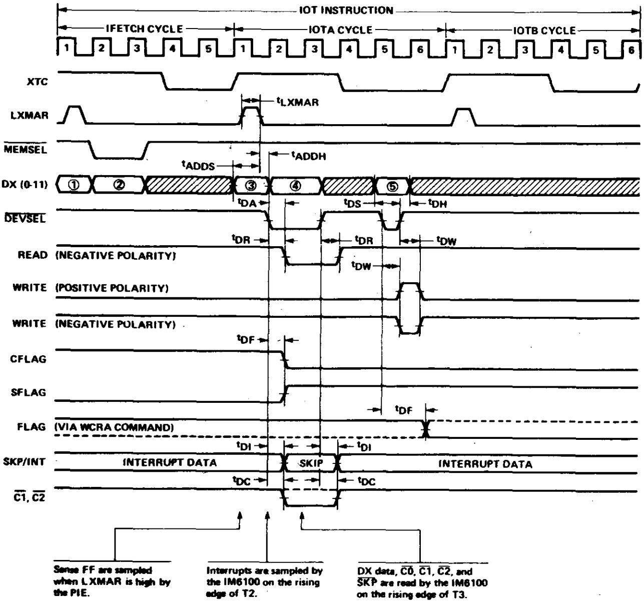
PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Logical "1" Input Voltage	V_{IH}		70% V_{CC}			V
Logical "0" Input Voltage	V_{IL}				20% V_{CC}	V
Input Leakage	I_{IL}	0V ≤ V_{IN} ≤ V_{CC}	−1.0		1.0	μA
Logical "1" Output Voltage	V_{OH2}	I_{OUT} - 0	V_{CC} - 0.01			V
Logical "1" Output Voltage	V_{OH1}	I_{OH} = −0.2 mA	2.4			V
Logical "0" Output Voltage	V_{OL2}	I_{OUT} = 0			GND + 0.01	V
Logical "0" Output Voltage	V_{OL1}	I_{OL} = 2.0 mA			0.45	V
Output Leakage	I_0	QV ≤ V_0 ≤ V_{CC}	−1.0		1.0	μA
Supply Current	I_{CC1}	V_{IN} = V_{CC}		1.0		μA
	I_{CC2}	V_{CC} = 5V f_{IM6100} = 4 MHz		1.0		mA
Input Capacitance	C_I			5	7	pf
Output Capacitance	C_O			8	10	pf
Input/Output Capacitance	C_{ID}			8	10	pf

AC CHARACTERISTICS T_A = 25°C C_L = 50pf

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Delay from DEVSEL to READ	t_{DR}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		150 75		ns ns
Delay from DEVSEL to WRITE	t_{DW}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		150 75		ns ns
Delay from DEVSEL to FLAG	t_{DF}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		200 100		ns ns
Delay from DEVSEL to C1, C2	t_{DC}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		200 100		ns ns
Delay from DEVSEL to SKP/INT	t_{DI}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		200 100		ns ns
Delay from DEVSEL to DX	t_{DA}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		200 100		ns ns
LXMAR pulse width	t_{LXMAR}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		200 100		ns ns
Address setup time	t_{ADDS}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		50 25		ns ns
Address hold time	t_{ADDH}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		100 50		ns ns
Data setup time	t_{DS}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		200 100		ns ns
Data hold time	t_{DH}	IM6101 V_{CC} = 5V IM6101A V_{CC} = 10V		50 25		ns ns

TIMING DIAGRAM

Timing for a typical IOT transfer is shown below. During IFETCH the processor obtains from memory an IOT instruction of the form 6XXX. During the IOTA the processor places that instruction back on the DX lines ③ and pulses LXMAR transferring address and control information for the IOT transfer to all peripheral devices. A low going pulse on DEVSEL while XTC is high ④ is used by the addressed PIE along with decoded control information to generate C1, C2, SKP and controls for data transfers to the processor. Control outputs READ1 and READ2 are used to gate peripheral data to the DX lines during this time. A low going pulse on DEVSEL while XTC is low ⑤ is used to generate WRITE1 and WRITE2 controls. These signals are used to clock processor accumulator instruction data into peripheral devices.



Sense FF are sampled when LXMAR is high by the PIE.

Interrupts are sampled by the IM6100 on the rising edge of T2.

DX data, $\overline{C0}$, $\overline{C1}$, $\overline{C2}$, and $\overline{SKP}$ are read by the IM6100 on the rising edge of T3.

All PIE timing is generated from IM6100 signals LXMAR, DEVSEL, and XTC. No additional timing signals, clocks, or one shots are required. Propagation delays, pulse width, data setup and hold times are specified for direct interfacing with the IM6100.

Chapter 13
THE SMS300

The SMS300 is described by its manufacturer as a "microcontroller" rather than a "microprocessor". This distinction draws attention to the very unique capabilities of the SMS300 which make it the most remarkable device described in this book.

The SMS300 is designed to serve as a signal processor, operating at very high speed. The SMS300 can handle applications of this type at more than 10 times the speed of any other device described in this book. On the other hand, the SMS300 has a very limited ability to access read/write memory, or to perform arithmetic operations — particularly when handling multibyte arithmetic.

If yours is a high speed, signal processing application, then give the SMS300 serious consideration; otherwise, the SMS300 is probably not for you.

We describe the SMS300 in Chapter 13 because it lies between the 8-bit microcomputers, which we have just described, and the 16-bit devices described in Chapters 14 through 17. The SMS300 accesses program memory as 16-bit words, while accessing data in 8-bit units.

Devices we are going to describe in this chapter include the SMS300 Interpreter and the SMS360 IV Byte. The unique nature of the SMS300 means that if you are going to use this device, in all probability you will do so using highly specialized configurations which Scientific Micro Systems provides as standard modules. These modules are packaged in what look like overgrown dual in-line packages; these modules are not described in this chapter, but they are summarized in Table 13-1.

Table 13-1. SMS300 Microcomputer System Modules

PACKAGE	PINS		FUNCTIONS		
		CPU	PROGRAM ROM	DATA RAM	IV BYTES
SMS330	64	SMS300	0, 512, 1024 or 2048 words		
SMS331	64		512, 1024 or 2048 words		
SMS332	64			256 bytes	One IV Byte used in address logic. 128 IV Byte addresses reserved for memory.

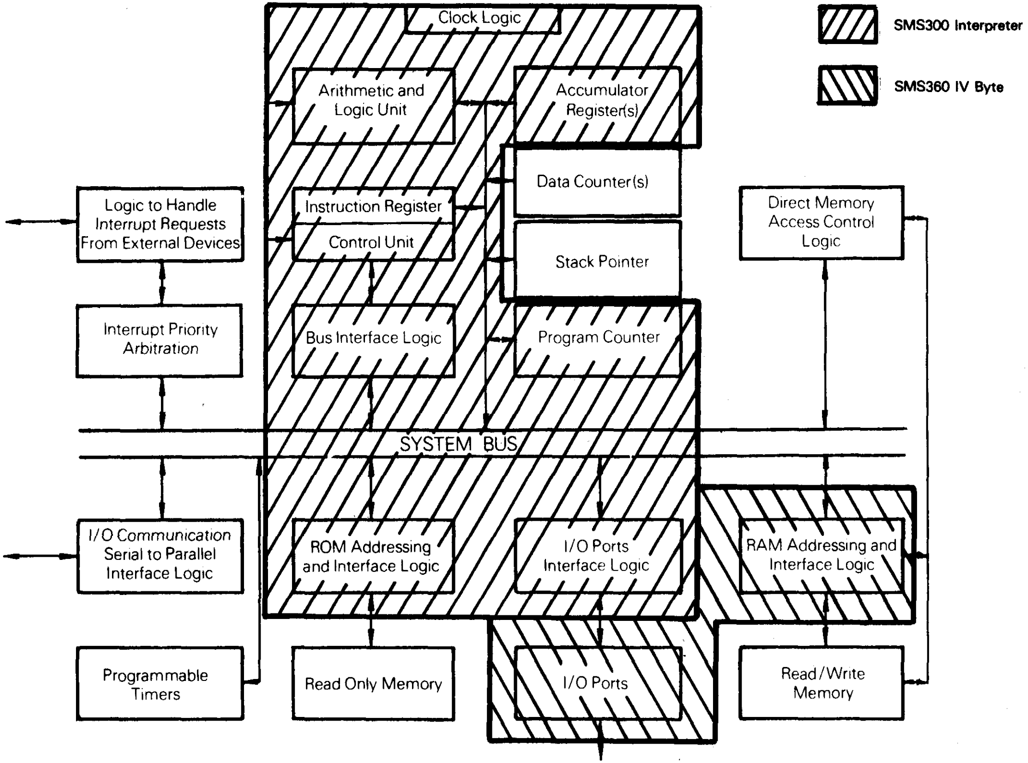

Clock Logic
Arithmetic and Logic Unit
Accumulator Register(s)
Data Counter(s)
Instruction Register
Control Unit
Stack Pointer
Bus Interface Logic
Program Counter
Logic to Handle Interrupt Requests From External Devices
Interrupt Priority Arbitration
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory
SMS300 Interpreter
SMS360 IV Byte

Figure 13-1. Logic Of The SMS300 Interpreter And SMS360 IV Byte

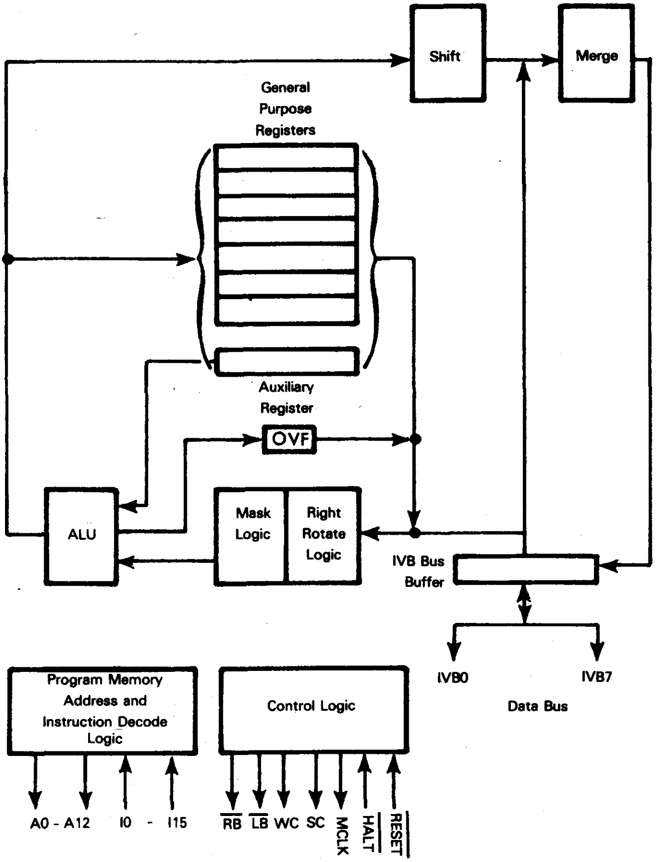


Figure 13-2. A Logic Overview Of The SMS300 Interpreter

All SMS300 devices are manufactured using bipolar technology. For this reason, devices have very fast logic; but conversely, they consume a great deal of power.

The principal source for the SMS300 is:

SCIENTIFIC MICRO SYSTEMS, INC.
520 Clyde Avenue
Mountain View, CA 94043

The second source is:

SIGNETICS
811 E. Arques Avenue
Sunnyvale, CA 94086

Note that even though Scientific Micro Systems is listed as the prime source for the SMS300, Signetics, the second source, is the sole manufacturer of this product line.

THE SMS300 INTERPRETER

Figure 13-1 illustrates that part of our general microcomputer logic which is implemented by the SMS300 Interpreter. Figure 13-2 provides a functional overview of this device.

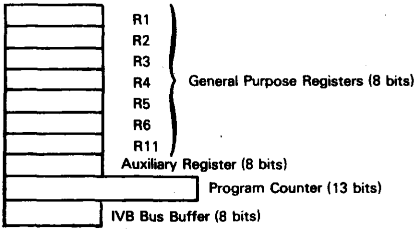
The SMS300 is manufactured using bipolar LSI technology; it is packaged as a 48-pin DIP. A single +5V power supply is required.

Using a 150 nanosecond clock, instructions execute in 300 or 600 nanoseconds. However, comparing SMS300 instruction execution times with other microcomputer instruction times can be misleading. A single SMS300 instruction, when simply manipulating data, can be the equivalent of five "typical" microcomputer instructions; on the other hand, it may take four or more SMS300 instructions to perform a memory access which could be accomplished using one "typical" microcomputer instruction.

It is important to note that the very fast SMS300 clock demands external logic with appropriately fast response times. You are therefore highly restricted in the size of memory, and the type of I/O device which you can include in an SMS300 microcomputer system.

SMS300 ADDRESSABLE REGISTERS

Addressable registers of the SMS300 may be illustrated as follows:


R1
R2
R3
R4
R5
R6
R11
General Purpose Registers (8 bits)
Auxiliary Register (8 bits)
Program Counter (13 bits)
IVB Bus Buffer (8 bits)

The seven General Purpose registers and the Auxiliary register constitute eight primary Accumulators. The result of any ALU operation may be stored in the Auxiliary register, or in any one of the seven General Purpose registers. ALU operations that require a single data input may receive this input from any General Purpose register, or from the Auxiliary register. ALU operations that require two data inputs will receive the second data input from the Auxiliary register only.

The IVB Bus buffer operates as a source or destination for data in the same way as a General Purpose register; it can be the destination of an ALU operation, or it can be the source for one ALU input. Strictly speaking, the IVB Bus buffer is not a programmable register, in that there are no instructions that will simply load data into the IVB Bus buffer or read data out of the IVB Bus buffer. However, **any instruction that outputs data on the IVB Bus or reads data off the IVB Bus will also write into the IVB Bus buffer.**

The strange General Purpose register numbering reflects instruction object code interpretations which we will describe later in this chapter. SMS300 assembly language uses register designations to identify a number of operations that have nothing to do with programmable registers; do not be confused.

The Program Counter is thirteen bits wide; thus, a total of 8192 program memory words may be addressed. The Program Counter is one feature of SMS300 logic which is not unusual; at all times, this register addresses the next program memory location from which an instruction code will be fetched.

Scientific Micro Systems' literature describes an Instruction Address register, but this is not a programmable register; it is simply a location within which effective program memory addresses are computed before being output to the program memory.

Observe that the SMS300 has no Data Counter, Stack Pointer or other logic via which external data memory can be addressed.

SMS300 STATUS FLAGS

The SMS300 has a single status flag, referred to in the manufacturer's literature as the Overflow (OVF) flag. This flag is, in fact, **a Carry status, as we would define it.**

In keeping with the generally unusual architecture of the SMS300, the Overflow status flag is addressed as though it were the low order bit of General Purpose Register 8 (10 octal).

SMS300 MEMORY ADDRESSING

The SMS300 can access program memory and I/O devices; the SMS300 has no logic capable of addressing data memory.

SMS300 PROGRAM MEMORY ADDRESSING

Program memory is addressed in 16-bit words; up to 8192 words of program memory can be addressed. You can address program memory in order to fetch instruction object codes, but that is all. You cannot store data tables in program memory, because there is absolutely no way of transferring the contents of a program memory word to any data register. Also, there is absolutely no way in which you can write into program memory.

SMS300 DATA AND I/O ADDRESSING

All data and external logic is addressed as 8-bit data units, via 512 I/O port addresses. If you want to have read/write memory present in an SMS300 system, you must set aside a block of contiguous I/O port addresses in order to select individual bytes of read/write memory; alternatively, you must access 8-bit buffers, via I/O port addresses, in order to create the memory address and Data Busses which are needed by external read/write memory. For example, you could address 65,536 bytes of external read/write memory by allocating two 8-bit I/O ports to hold 16 bits of data which will create a memory Address Bus; a third 8-bit I/O port must be set aside as a buffer, holding data being written out to external memory or being read from external memory.

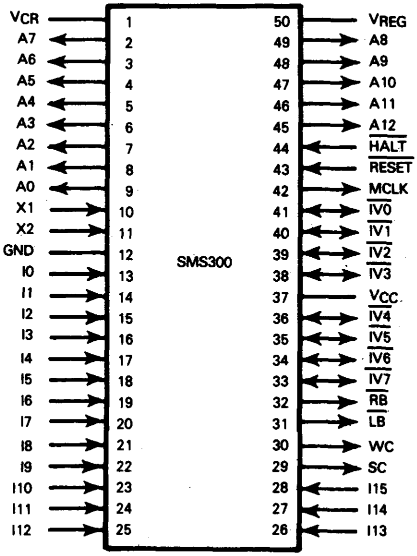
SMS360 IV BYTES

The SMS360 Interface Vector Bytes (IV Bytes), which are described later in this chapter, have been designed to operate as I/O ports, read/write memory and the SMS300 Interpreter external logic interface. Because of the unique architecture of the SMS300, and particularly because of its very high speed, you will probably find that the IV Bytes have no substitutes in any SMS300 microcomputer system.

SMS360 IV BYTE ADDRESSING

Looking at the SMS300 from the frame of reference of any other microcomputer described in this book, an IV Byte is a simple, 8-bit parallel I/O port. But unlike the I/O ports of other microprocessors, SMS300 instructions that access an I/O port do not identify the I/O port that is to be accessed. You must first execute an instruction which selects an I/O port; then any instruction which specifies an I/O port access will access the most recently selected I/O port. You can have two I/O ports simultaneously selected, since the SMS300 divides a total of 512 addressable I/O ports into a left bank and a right bank — within each bank a single IV Byte can be selected.

As we have already stated, if you want to have read/write memory present in an SMS300 microcomputer system, you must create the address and Data Bus required by the external read/write memory using IV bytes. This is no different than using I/O ports of any other microcomputer system described in this book in order to create Address and Data Busses. The reason the SMS300 can get away with such an apparently clumsy method of accessing read/write memory is because of the very high speed of instruction execution — and because of the fact that the SMS300 is simply not designed for data manipulations that use a lot of read/write memory. For the type of signal processing application that is well suited to an SMS300, 512 bytes of external read/write memory will be more than sufficient.



Pin Name	Description	Type
A0 - A12	Program Memory Address Bus	Output
$\overline{\text{IV0}}$ - $\overline{\text{IV7}}$	Interface Vector Byte Bus	Bidirectional
$\overline{\text{RB}}$, $\overline{\text{LB}}$, WC, SC	Control Signals	Output
MCLK	Synchronizing Clock	Output
$\overline{\text{HALT}}$	CPU Halt	Input
$\overline{\text{RESET}}$	Reset	Input
X1, X2	Crystal Connections	Input
I0 - I15	Instruction Bus	Input
VREG	Reference Voltage to Pass Transistor	
VCR	Regulated Supply Voltage from Pass Transistor	
VCC, GND	Power and Ground	

Figure 13-3. SMS300 Interpreter Signals And Pin Assignments

SMS300 PINS AND SIGNALS

SMS300 pins and signals are illustrated in Figure 13-3.

All addresses are output to program memory via the Address Bus lines A0 - A12. Note carefully that addresses cannot be output via A0 - A12 to data memory. The only time an address will be output via the Address Bus is during an instruction fetch operation. **The fetched instruction object code will be returned via the sixteen instruction pins, I0 - I15.**

$\overline{\text{IV0}}$ - $\overline{\text{IV7}}$ is a combined Address and Data Bus via which external logic is accessed by the SMS300. You will find it easiest to understand this bus if you visualize it as a multiplexed I/O port address and I/O Data Bus.

The two control signals, $\overline{\text{RB}}$ and $\overline{\text{LB}}$, may be looked upon as an extension to the IVB Bus when an I/O port address is being output via this bus. Whenever an address is being output on the IVB Bus, either $\overline{\text{RB}}$ or $\overline{\text{LB}}$ will be low, while the other signal is high. You can use these two signals in order to decode the address on the IVB Bus as selecting one or two 256 I/O port banks. We will describe how to output I/O port addresses, as against data, later in this chapter.

The WC and SC control outputs further define the contents of the IVB Bus as follows:

SC	WC	
0	0	Data is input to the SMS300 via the IVB Bus
0	1	Data is output on the IVB Bus by the SMS300
1	0	An I/O port address is output on the IVB Bus by the SMS300
1	1	Never output

MCLK is a synchronizing clock signal which is output as a high pulse during the last quarter of every instruction cycle.

The $\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ signals are absolutely standard.

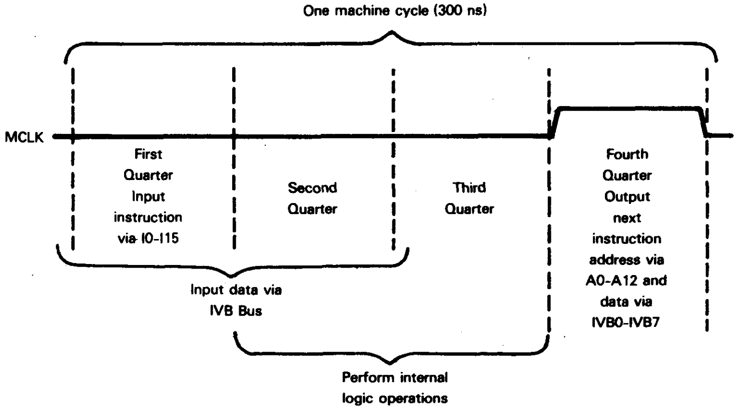
When $\overline{\text{HALT}}$ is input low, the SMS300 will cease executing instructions until $\overline{\text{HALT}}$ is input high again.

When $\overline{\text{RESET}}$ is input low and is held low for at least one machine cycle, the Program Counter contents are set to zero; subsequently program execution will begin again with execution of the instruction stored in memory location zero.

The two inputs X1 and X2 are used either to connect a crystal or a capacitor. If the SMS300 Interpreter is being used at maximum speed (150 nanosecond signal frequency) then you must connect a crystal across X1 and X2. If you are using a slower clock, then a capacitor connected across these two inputs will suffice.

SMS300 INSTRUCTION EXECUTION AND TIMING

SMS300 instructions are executed in either one or two machine cycles. Minimum instruction cycle time is 300 nanoseconds. Each instruction cycle is divided into 75 nanosecond quarters as follows:


One machine cycle (300 ns)
MCLK
First Quarter Input instruction via I0-I15
Second Quarter
Third Quarter
Fourth Quarter Output next instruction address via A0-A12 and data via IVB0-IVB7
Input data via IVB Bus
Perform internal logic operations

During the fourth quarter of a machine cycle, the address for the next machine cycle's instruction object code is output via the Address Bus, A0 - A12.

During the first quarter of the next machine cycle, the addressed instruction object code is input via the Instruction Bus, I0 - I15.

During the second and third quarters of a machine cycle, data is input off the IVB Bus by the SMS300, if necessary; then any internal operations on data are performed.

During the fourth quarter, in addition to the next address being output to program memory, data is output to the IVB Bus, if necessary.

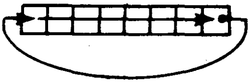
Within the rather simple looking instruction timing illustrated above, some very complex event sequences can occur as a result of the SMS300 Interpreter's unique internal logic organization. Timing and propagation delays are quite complex and must be examined with care using vendor literature as your guide.

The SMS300 Interpreter's internal logic is unique because **a good deal of it is distributed along various data paths. This is illustrated in Figure 13-2.**

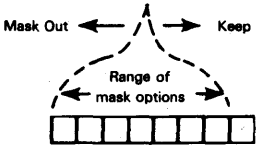
Consider the implications of the shift, merge, rotate and mask logic positions.

SMS300 ROTATE AND MASK LOGIC

Data entering the Arithmetic and Logic Unit, either from the IVB Bus Buffer, or from a General Purpose register, must pass through both the rotate and mask logic. The rotate logic optionally allows the entering eight data bits to be right-rotated by any number of bit positions:



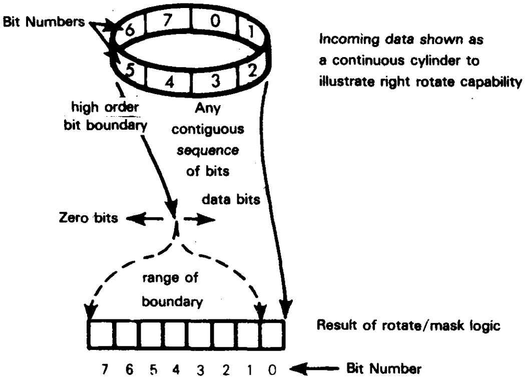
The mask logic optionally allows you to take the output from the rotate logic and mask off any number of bits, beginning with the high-order bit:



Masked out bit positions are replaced by 0.

Thus the data entering the ALU from either a General Purpose register or the IVB Bus register may be rotated and/or masked before being operated on.

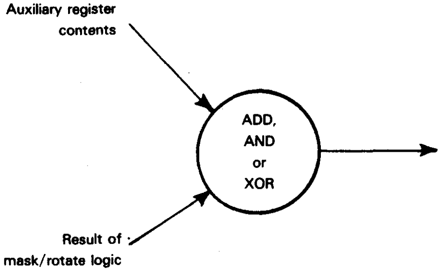
Combining the rotate and mask logic that we have just described, the input to the ALU may be illustrated as follows:



Suppose an input is right-rotated five bit positions, then the two high order bits are masked off; this would be the result:

	7	6	5	4	3	2	1	0	Bit No.
Initial value:	A7	A6	A5	A4	A3	A2	A1	A0	
After right rotate:	A2	A1	A0	A7	A6	A5	A4	A3	
After mask:	0	0	A0	A7	A6	A5	A4	A3	

The result of the rotate/mask logic illustrated above becomes an Arithmetic and Logic Unit (ALU) input; it may be the only ALU input, or it may be one of two ALU inputs. If it is the only ALU input, it will simply be passed through the ALU. If it is one of two ALU inputs, then the second input is the unmodified contents of an 8-bit Auxiliary register. You may Add, AND or XOR the two operands:

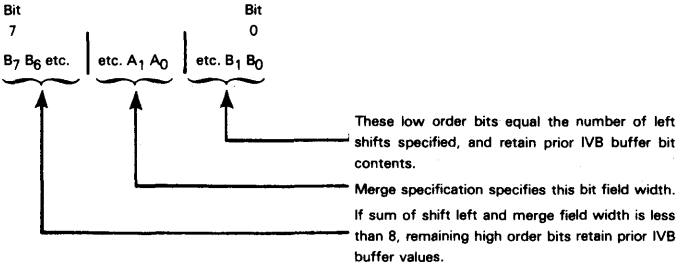


Thus, the ALU output may be the unmodified result of rotate and mask logic, or it may be the output from an arithmetic or logical operation, as illustrated above. In either case, the ALU output may be stored in the Auxiliary register, or in one of the General Purpose registers; or it may be output to the IVB Bus.

SMS300 SHIFT AND MERGE LOGIC

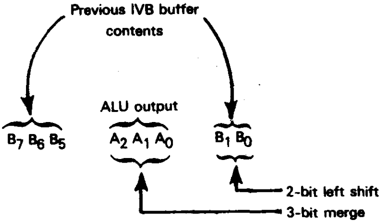
Data being transferred to the IVB Bus passes through shift and merge logic. This shift and merge logic combines in a very unusual way. ALU output, if shifted, may be shifted left from one to seven bits. However, zeroes are not shifted in to the low order bits; rather, any prior contents of the IVB Buffer are moved into the vacated bit positions.

In addition, you can specify the number of high order bits which will retain their IVB Buffer values. This may be illustrated as follows:


Bit 7
Bit 0
B7 B6 etc.
etc. A1 A0
etc. B1 B0
These low order bits equal the number of left shifts specified, and retain prior IVB buffer bit contents.
Merge specification specifies this bit field width.
If sum of shift left and merge field width is less than 8, remaining high order bits retain prior IVB buffer values.

Thus you create a new IVB Bus output by inserting from 1 to 8 new data bits anywhere in the old data bit field. In the illustration above, A_1 represents new data bits; B_1 represents old IVB Buffer bits.

Suppose you specify a 2-bit left shift and a 3-bit merge; this would be the result:



Figures 13-4 through 13-7 illustrate the four possible data paths that may be specified by SMS300 instructions. In all four figures, data entering the ALU from the Accumulator is optional, but if present, requires an Add, AND or XOR operation to be performed.

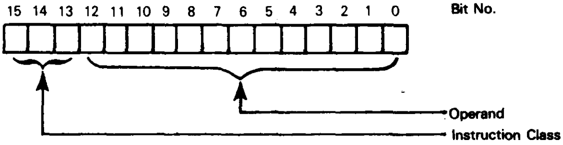
THE SMS300 INSTRUCTION SET

Manufacturer's literature refers to the SMS300 as having eight instructions in its instruction set. This statement does a gross injustice to the SMS300. The problem with the SMS300 instruction set is that we cannot neatly categorize instructions as we have done for any other product described in this book; one SMS300 instruction may perform a data move, plus five additional operations.

Therefore, in order to summarize the SMS300 instruction set in Table 13-3, we list individual instructions that perform many operations under each of the instruction classes that may apply.

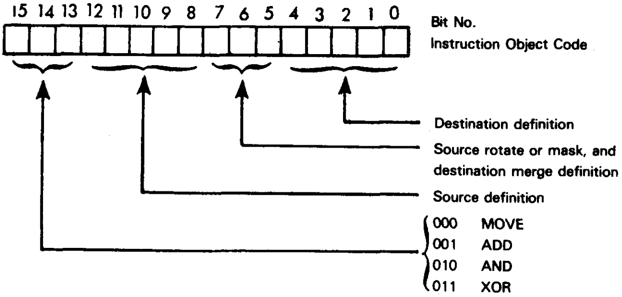
Table 13-3 will help you understand what the true comparison is between the SMS300 instruction set and other microcomputer instruction sets. However, **Table 13-3 will do nothing to help you understand SMS300 assembly language.** This is because of the strange assembly language mnemonics adopted by Scientific Micro Systems for the SMS300 Assembler. But without some understanding of SMS300 instruction code any further discussion of assembly language mnemonics will have little meaning; therefore let us take a look at these object codes, and simultaneously look at the assembly language syntax that goes with them.

The one general statement that can be made for all SMS300 instructions is that every instruction has a single, 16-bit object code; the 3 high order object code bits define the instruction class, while the next 13 bits provide additional operand or qualifying data. This may be illustrated as follows:



Now we are going to make the discussion which follows conform to the rest of this book by numbering instruction words and data byte bits from right to left; and we are going to use hexadecimal object code notation. Scientific Micro Systems' literature, by way of contrast, numbers data words from left to right, and uses a form of bastardized octal notation to describe instruction object codes.

The first four classes of SMS300 instructions have identical object code formats which may be illustrated as follows:



The "Source definition" and "Destination definition" are defined as register numbers; since each definition is five bits wide, a register number in the range 00_{16} through $1F_{16}$ (00_8 through 37_8) may be specified. But you get to specify a lot more than a source or destination register. Table 13-2 summarizes the possibilities.

SMS300 assembly language syntax closely follows the object code format; this may be illustrated as follows:

```
        LABEL     OP        S,N,D
```

LABEL represents any normal assembly language instruction label; as usual, LABEL will be optional.

OP represents the operation or instruction mnemonic. OP may be MOVE, ADD, AND, or XOR, depending on which of the four instructions is being executed. OP corresponds to bytes 15, 14 and 13 of the instruction code.

The assembly language operand field consists of three terms S, N and D.

With reference to the instruction object code we have illustrated above, S represents bits 8 through 12, the source definition.

N represents bits 5 through 7 which may provide rotation, mask or merge parameters, depending on the nature of S and D.

D represents bits 0 through 4 of the instruction object code and provides the destination definition.

The problem with the S, N and D terms of the operand field is that they are not really operands as one would normally define them in an assembly language instruction set. These three fields also help identify part of the instruction operation, or mnemonic. If you approach SMS300 assembly language realizing that its operand field is really an extension of the mnemonic field, you will have less trouble understanding individual instructions.

The various ways in which a MOVE, ADD and/or XOR instruction may be executed are illustrated in Figures 13-4 through 13-7. Let us look at these possibilities in more detail.

When a register is specified as both the source and destination of data, Figure 13-4 defines the operation. Referring to this figure, note that the second data is rotated, but it is not masked. The second ALU input will only occur if you are executing an Add, AND or XOR instruction; and in each case the second ALU input will be the unmodified contents of the Auxiliary register.

The classes of instruction illustrated in Figure 13-4 can be listed under the following categories:

1) A Register-Register Move. This involves specifying a Move instruction with different registers as the data source and destination, but no right rotate.

2) Register Operate. By specifying the same register as the source and destination for a MOVE, you can create a Register Operate instruction if you also specify some degree of right rotation. You can create additional Register Operate instructions by specifying the Auxiliary register as both source and destination for an Add, AND or XOR instruction.

3) Register-Register Operate. By specifying an Add, AND or XOR operation that does not use the Auxiliary register as both source and destination, you create Register-Register Operate instructions.

Table 13-2. SMS300 Source And Destination Object Code Interpretations

CODE			INTERPRETATION	
BINARY	OCTAL	HEX	SOURCE DEFINITION	DESTINATION DEFINITION
00000	00	00	Auxiliary register	
00001	01	01	General Purpose Register R1	
00010	02	02	General Purpose Register R2	
00011	03	03	General Purpose Register R3	
00100	04	04	General Purpose Register R4	
00101	05	05	General Purpose Register R5	
00110	06	06	General Purpose Register R6	
00111	07	07	All zero input	Output an 8-bit I/O port address to a left bank IV Byte
01000	10	08	OVF status (low order bit only)	Not allowed
01001	11	09	General Purpose Register R11	
01010 through 01110	12 16	0A 0E	No operation	
01111	17	0F	All zero input	Output an 8-bit I/O port address to a right bank IV Byte.
10XXX	2X	1X	Contents of left bank IV Byte selected by most recent 07 output is loaded into IVB buffer; this data is then right rotated X bit positions, on its way to the ALU. IVB buffer holds unrotated input.	ALU output is shifted left 7-X bit positions. After passing through merge logic, merge logic output will be stored in IVB buffer, and in left bank IV Byte most recently selected by an 07 output.
11XXX	3X	18 to 1F	Identical to 10XXX, except that right bank IV Byte most recently selected by a 0F (or 17) output is accessed.	

Consider some possibilities.

In order to complement any register's contents, load FF_{16} into the Auxiliary register (using an XMIT instruction), then XOR the General Purpose register contents with the Auxiliary register contents, returning the results to the General Purpose register. These two instructions can be executed in 600 nanoseconds.

You can AND or XOR Auxiliary register bits, within a data byte by specifying the Auxiliary register as the source and destination for an AND or XOR instruction with right rotate. The ability to perform logical operations on bits within a single 8-bit unit is very useful if you are treating the contents of a register as status, representing individual signal levels rather than treating the bits contiguously, as data items.

Apparently absent instructions, such as Register Increment, Register Decrement, OR and Compare, can be generated by using the Auxiliary register to hold appropriate intermediate data.

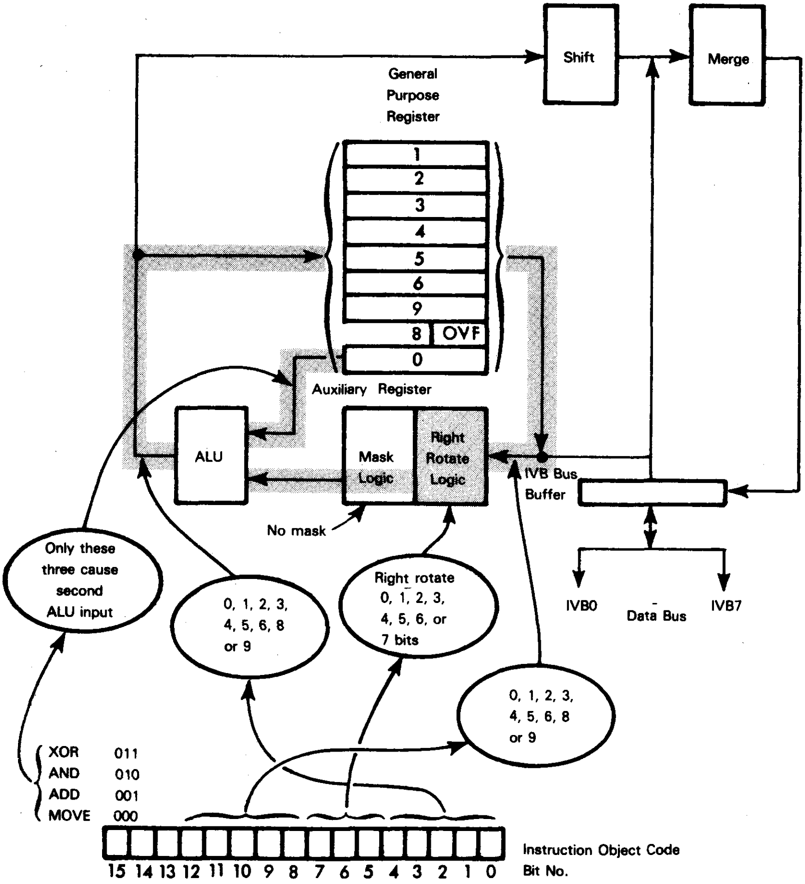


Figure 13-4. An SMS300 Register-To-Register Instruction's Execution

Figure 13-5 illustrates Move, Add, AND and XOR instructions where the IVB Bus is the data source and a General Purpose register is the data destination. Referring to Figure 13-5, observe that the mask and right rotate logic are both involved. Bits 5, 6 and 7 of the instruction object code, which in Figure 13-4 specify the amount of right rotation, in Figure 13-5 specify the degree of masking which will occur. Bits 8, 9 and 10 in Figure 13-5 specify the amount of right rotation which will occur.

SMS300 assembly language mnemonics do not discriminate between this new use of bits 5, 6 and 7. You will still write assembly language instructions with the format:

```
        LABEL     OP          S,N,D
```

S now defines the right rotate while N defines the masking operation.

Now consider instructions which specify an IV byte as the data destination. **Figure 13-6 illustrates instructions where a General Purpose register is the instruction source; Figure 13-7 illustrates IV byte-to-IV byte operations.**

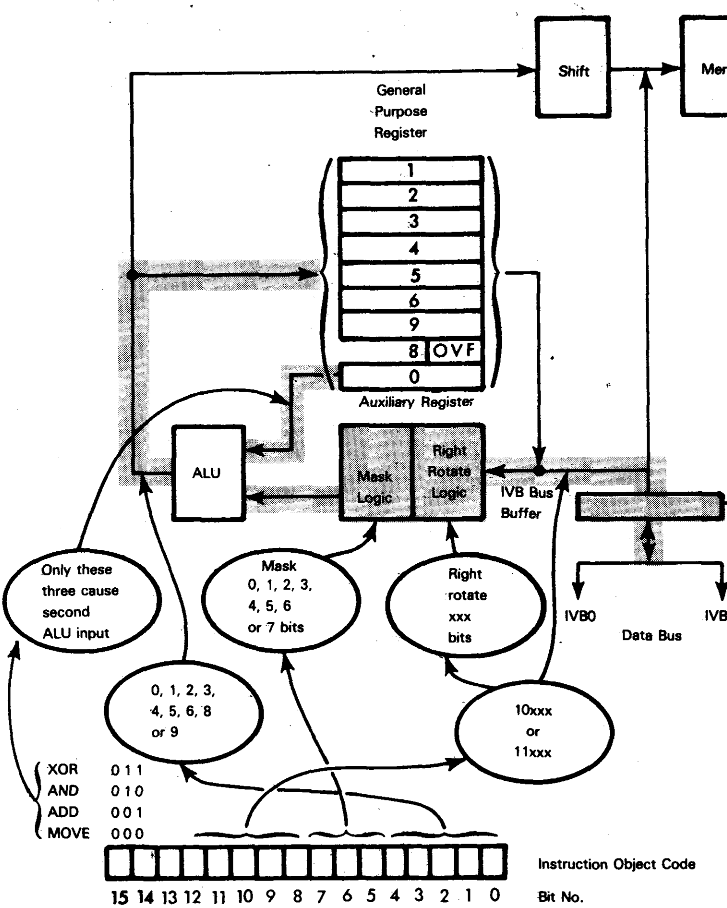


Figure 13-5. An SMS300 IV Byte-To-Register Instruction's Execution

There are three instruction classes which include immediate data.

The XEC instruction, identified by 100 in the three high order object code bits, uses the 13 operand bits to compute a temporary program memory address out of which the next instruction object code will be fetched. When an XEC instruction is executed the Program Counter contents are not incremented.

The NZT instruction, specified by 101 in the three high order object code bits, provides the SMS300 with its conditional logic.

The XMIT instruction, represented by 110 in the three high order object code bits, provides the SMS300 with its immediate instructions.

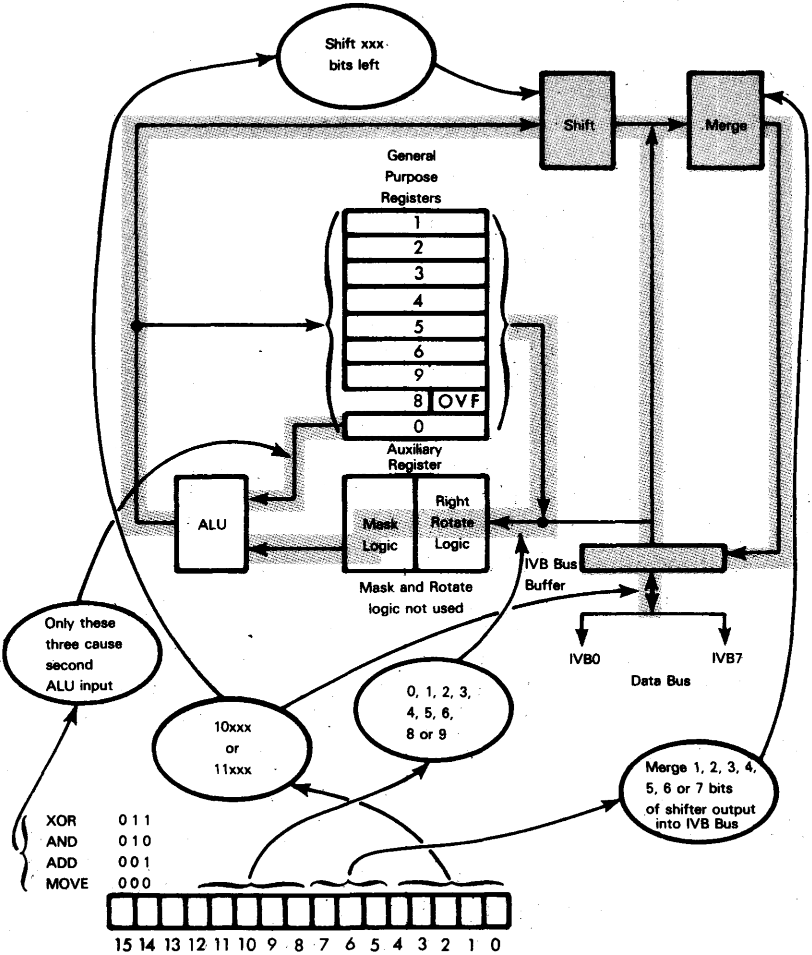


Figure 13-6. An SMS300 Register-To-IV Byte Instruction's Execution

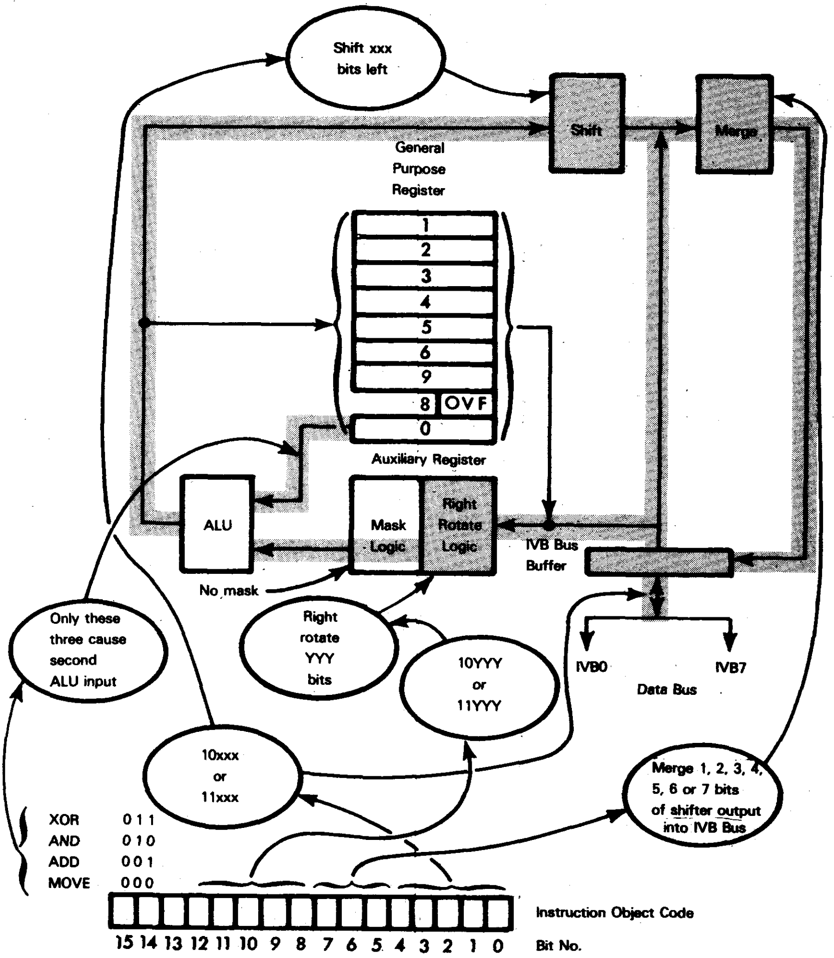
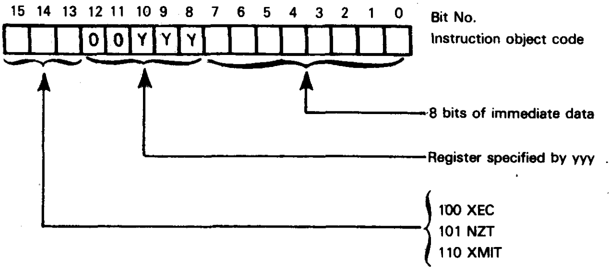


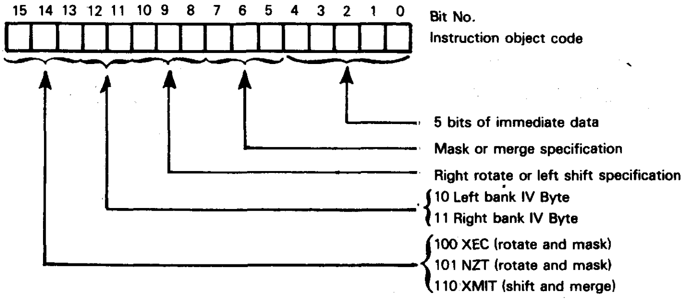
Figure 13-7. An SMS300 IV Byte-To-IV Byte Instruction's Execution

All three instructions, XEC, NZT and XMIT use one of the two following instruction object code formats:

Format A:



Format B:


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bit No.
Instruction object code
5 bits of immediate data
Mask or merge specification
Right rotate or left shift specification
10 Left bank IV Byte
11 Right bank IV Byte
100 XEC (rotate and mask)
101 NZT (rotate and mask)
110 XMIT (shift and merge)

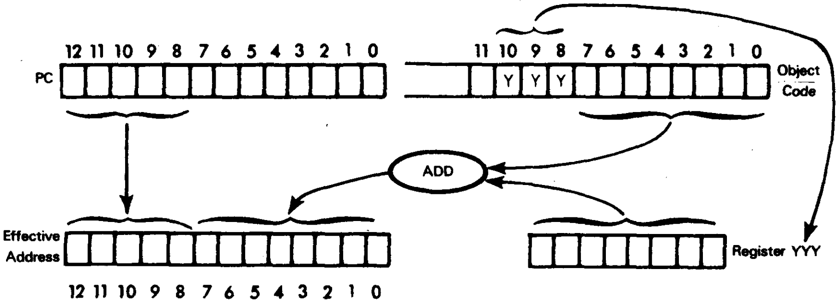
For all three instructions, XEC, NZT and XMIT, the Format A object code uses bits 8 through 12 to specify a General Purpose register, or the Auxiliary register.

The Format B instruction object code uses bits 5 through 12 to specify the currently selected left bank or right bank IV byte, where byte contents will be subject to a mask and a rotate, as illustrated in Figure 13-5.

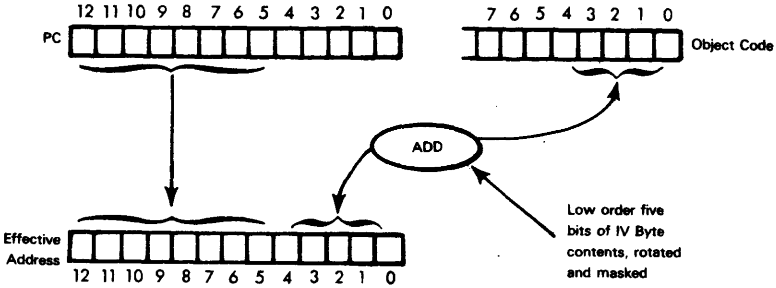
Let us take another look at how the XEC, NZT and XMIT instructions use the data generated by their operand bits.

The XEC instruction allows you to sit in one object code, continuously re-executing this single object code, while it points to another object code which actually gets executed. The address of the object code which actually gets executed is computed in one of two ways:

1) For the Format A object code, the current five high order Program Counter bits are concatenated with the 8-bit sum of the specified register contents, plus the immediate data:



2) With the second object code format, the 8 high order current Program Counter bits are concatenated with the 5-bit sum of the immediate data, plus the rotated and masked IV byte data:



You may use XEC instructions in one of two ways:

1) You may create a branch table of Jump instructions: based on the contents of any General Purpose register or IV byte, you may jump to one of 256 locations using Format A instruction object code, or one of 32 locations using Format B instruction object code.

2) External logic may directly control the sequence in which instructions are executed. The XEC instruction is equivalent to a single instruction which requires 600 nanoseconds to execute: 300 nanoseconds to process the XEC instruction's object code and another 300 nanoseconds to execute the object code fetched in response to the XEC instruction. If you are using the Format B instruction, external logic can use the second 300 nanosecond time interval to load new data into the selected IV byte. Thus, external logic can indefinitely control instruction execution sequence within an SMS300 microcomputer system.

The NZT instruction uses the 13 operand bits to provide a branch address, and to identify data that will be tested for a zero or a nonzero value; for a nonzero value the immediate data provides an absolute, paged branch address.

The Format A NZT instruction object code tests the contents of a General Purpose register; upon detecting a nonzero value, the eight immediate data bits are loaded into the eight low order Program Counter bits — thus causing an absolute paged branch to occur within a 256-word program memory page. For zero General Purpose register contents, the next sequential instruction is executed in the normal way.

The Format B NZT instruction tests the contents of a selected IV byte, subject to rotate and mask logic. Upon detecting a nonzero result, the five immediate data bits are loaded into the low order five Program Counter bits thus causing an absolute, paged branch to occur within a 32-word program memory page. If a zero result is detected, program execution continues with the next sequential instruction.

Thus the NZT instruction allows you to base branch logic on the contents of the Overflow status, or on any bit field, in any General Purpose register, Auxiliary register or external addressable location. We cannot classify such a wide-ranging instruction as a single instruction; it would not conform with the definition of a single assembly language instruction as used by any other microcomputer described in this book.

In the case of the XMIT instruction, the immediate data gets loaded into the General Purpose register specified by a Format A instruction, or the external IV byte specified by a Format B instruction. In the case of a Format B instruction, the immediate data is shifted and merged, as illustrated in Figure 13-7, before being output to the identified IV byte. Recall that the identified IV byte will be the byte most recently selected by a Move instruction that specifies Register 7 or F as the destination.

The Jump instruction is the only one which remains to be described; it is also the simplest to describe. When this instruction is executed, the 13 operand bits are loaded directly into the Program Counter; thus you perform a simple unconditional jump to any location in program memory.

Observe that the SMS300 has no subroutine or interrupt handling logic.

Subroutine logic can be created using the XEC instruction and an appropriate jump table, but this is rather clumsy. In most cases it will be simpler to do without subroutines.

The lack of interrupt logic is probably inconsequential. Given the fact that the SMS300 can execute instructions in 300 nanoseconds, polling on status will invariably be a satisfactory way of allowing external logic to control events within the SMS300 microcomputer system.

A very effective way of allowing external logic to control the SMS300 microcomputer system is to have the system continuously re-execute an ineffective instruction as the result of an XEC. For example, the XEC could point to an instruction which simply moves the contents of a General Purpose register back into itself. Using Format B for the XEC instruction, external logic could modify the contents of the selected external IV byte in order to force program execution to branch in one of 31 different directions.

THE SMS300 BENCHMARK PROGRAM

The benchmark program we have been using throughout this book is particularly ill suited to the SMS300; in fact, it could well illustrate a benchmark program that a competitor would select in order to make the SMS300 look bad. This is because the SMS300 is not good at memory addressing. The SMS300 would never be used in an application that principally reads blocks of data into read/write memory, then moves blocks of data around read/write memory.

The SMS300 has no ability to address read/write memory; as we have already described earlier in this chapter, should you require the presence of read/write memory in an SMS300 system, you are going to have to create a memory Address Bus and Data Bus for the external read/write memory; IV bytes must be used to create these busses.

We will therefore change the benchmark program so that a sequence of data bytes entering via the left bank IV byte must immediately be output via a right bank IV byte. The first byte read will be interpreted as identifying the number of data bytes to follow. Now the benchmark will appear as follows:

```
        XMIT    AUX,377       LOAD 377 OCTAL INTO THE AUXILIARY REGISTER TO
                              DECREMENT COUNTER
        XMIT    20,0,SRCE     SELECT SOURCE IV BYTE IN LEFT BANK
        XMIT    30,0,DST      SELECT DESTINATION IV BYTE IN RIGHT BANK
        MOVE    R1,0,SRCE     LOAD COUNTER INTO R1
LOOP    MOVE    SRCE,0,DST    MOVE NEXT DATA BYTE
        ADD     R1,0,R1       DECREMENT COUNTER
        NZT     R1,LOOP
```

The following symbols are used in Table 13-3:

A	Auxiliary register
ADDR	13-bit address value
DATA5	5-bit data unit
DATA8	8-bit data unit
DISP5	5-bit address value
DISP8	8-bit address value
IV1, IV2	IV Byte
(L)	Optional field length for IV Byte
PC	Program Counter
(R)	Optional rotate value for register
RX,RY	Any General Purpose registers
x<y,z>	Bits y through z of the specified value. For example, PC<7,0> is the low byte of the Program Counter.
[ ]	Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.
Λ	Logical AND
∀	Logical Exclusive-OR
←	Data is transferred in the direction of the arrow.

Under the heading of STATUS in Table 13-3, an X indicates OVF is modified in the course of the instructions execution. If there is no X, it means that the status maintains the value it had before the instruction was executed.

Table 13-3. SMS300 Instruction Set

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS OV	OPERATION PERFORMED
I/O	MOVE	IV1,(L),IV2	2		[IV2]←[IV1] Move data from IV Byte to IV Byte.
	MOVE	IV1,(L),RX	2		[RX]←[IV1] Move data from IV Byte to register.
	MOVE	RX,(L),IV1	2		[IV1]←[RX] Move data from register to IV Byte.
	ADD	IV1,(L),IV2	2	X	[IV2]←[IV1] + [A] Add IV Byte to Auxiliary register, store result in IV Byte.
	ADD	IV1,(L),RX	2	X	[RX]←[IV1] + [A] Add IV Byte to Auxiliary register, store result in register.
	ADD	RX,(L),IV1	2	X	[IV1]←[RX] + [A] Add register to Auxiliary register, store result in IV Byte.
	AND	IV1,(L),IV2	2		[IV2]←[IV1] $\wedge$ [A] AND IV Byte with Auxiliary register, store result in IV Byte.
	AND	IV1,(L),RX	2		[RX]←[IV1] $\wedge$ [A] AND IV Byte with Auxiliary register, store result in register.
	AND	RX,(L),IV1	2		[IV1]←[RX] $\wedge$ [A] AND register with Auxiliary register, store result in IV Byte.
	XOR	IV1,(L),IV2	2		[IV2]←[IV1] $\veebar$ [A] Exclusive-OR IV Byte with Auxiliary register, store result in IV Byte.
	XOR	IV1,(L),RX	2		[RX]←[IV1] $\veebar$ [A] Exclusive-OR IV Byte with Auxiliary register, store result in register.
	XOR	RX,(L),IV1	2		[IV1]←[RX] $\veebar$ [A] Exclusive-OR register with Auxiliary register, store result in IV Byte.
	XMIT	DATA5,(L),IV1	2		[IV1]←DATA5 Transmit immediate to IV Byte.
REGISTER-REGISTER MOVE	MOVE	RX,(R),RY	2		[RY]←[RX] Move contents of one General Purpose register to another.

Table 13-3. SMS300 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	OPERATION PERFORMED
				OV	
REGISTER OPERATE	MOVE	RX,(R),RX	2		Rotate contents of a general purpose register and store result in the same register.
REGISTER-REGISTER OPERATE	ADD	RX,(R),RY	2	X	[RY]←[RX] + [A] Add Register X to Auxiliary register, store result in Register Y.
	AND	RX,(R),RY	2	X	[RY]←[RX] ∧ [A] AND Register X with Auxiliary register, store result in Register Y.
	XOR	RX,(R),RY	2	X	[RY]←[RX] ⊻ [A] Exclusive-OR Register X with Auxiliary register, store result in Register Y.
IMMEDIATE	XMIT	DATA8,RX	2		[RX]←DATA8 Load immediate to General Purpose register.
BRANCH ON CONDITION	NZT	RX,DISP8	2		If [RX] ≠ 0; [PC<7,0>]←DISP8 Branch if register contents nonzero.
	NZT	IV1,(L),DISP5	2		If [IV1] ≠ 0; [PC<4,0>]←DISP5 Branch if IV Byte is nonzero.
JUMP	JMP	ADDR	2		[PC]←ADDR Unconditional jump.
	XEC	RX,DISP8	2		Execute instruction at the following address: [ADDR<12,8>]←[PC<12,8>] [ADDR<7,0>]←[RX] + DISP8. Do not increment PC.
	XEC	IV1,(L),DISP5	2		Execute instruction at the following address: [ADDR<12,5>]←[PC<12,5>] [ADDR<4,0>]←[IV1] + DISP5 Do not increment PC

The following symbols are used in Table 13-4:

a	one bit of immediate address.
ddddd	5 bits choosing destination register or IV Byte.
i	one bit of immediate data
lll	three bits specifying length of IV Byte field.
rrr	three bits specifying the number of rotates performed.
sssss	5 bits choosing source register or IV Byte.

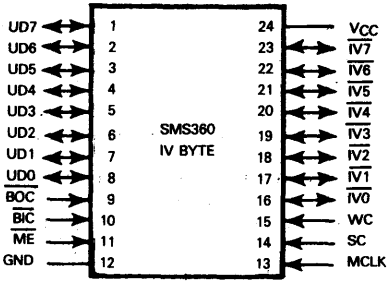
The sssss and ddddd fields are restricted in the following ways:

If sssss or ddddd represents a register, it must be in the range 00_8 — 17_8.

If sssss or ddddd represents an IV Byte, it must be in the range 20_8 — 37_8.

Table 13-4. SMS300 Instruction Set Object Codes

INSTRUCTION		OBJECT CODE	BYTES	MACHINE CYCLES
ADD	IV1,(L),IV2 IV1,(L),RX RX,(L),IV1	001ssssslllddddd	2	1
ADD	RX,(R),RY	001sssssrrrddddd	2	1
AND	IV1,(L),IV2 IV1,(L),RX RX,(L),IV1	010ssssslllddddd	2	1
AND	RX,(R),RY	010sssssrrrddddd	2	1
JMP	ADDR	111aaaaaaaaaaaaa	2	1
MOVE	IV1,(L),IV2 IV1,(L),RX RX,(L),IV1	000ssssslllddddd	2	1
MOVE	RX,(R),RX	000sssssrrrsssss	2	1
MOVE	RX,(R(,RY	000sssssrrrddddd	2	1
NZT	IV1,(L),DISP5	101ssssslllaaaaa	2	1
NZT	RX,DISP8	101sssssaaaaaaaa	2	1
XEC	IV1,(L),DISP	100ssssslllaaaaa	2	1
XEC	RX,DISP	100sssssaaaaaaaa	2	1
XMIT	DATA5,IV1	110dddddllliiiii	2	1
XMIT	DATA8	110dddddiiiiiiii	2	1
XOR	IV1,(L),IV2 IV1,(L),RX RX,(L),IV1	011ssssslllddddd	2	1
XOR	RX,(R),RY	011sssssrrrddddd	2	1



Pin Name	Description	Type
$\overline{\text{IV0}}$ - $\overline{\text{IV7}}$	IVB Bus	Tristate, Bidirectional
UD0 - UD7	External Logic Data Bus	Tristate, Bidirectional
$\overline{\text{ME}}$	Master Enable	Input
$\overline{\text{BIC}}$, $\overline{\text{BOC}}$	External IV Byte Control Lines	Input
MCLK	Master Clock	Input
SC, WC	IVB Bus Control	Input
V_{CC}, GND	Power and Ground	

Figure 13-8. SMS360 Interface Vector Byte Signals And Pin Assignments

THE SMS360 INTERFACE VECTOR BYTE (IV BYTE)

This device serves as an I/O port and IVB Bus interface for all external logic communicating with the SMS300 Interpreter.

The IV Byte is implemented using a bipolar LSI technology and is packaged as a 24-pin DIP. It requires a single +5V power supply.

SMS360 IV BYTE PINS AND SIGNALS

Figure 13-8 illustrates the pins and signals of the IV Byte. Figure 13-9 illustrates how an IV Byte will normally be used.

$\overline{\text{IV0}}$ - $\overline{\text{IV7}}$ represents the pins via which the IV Byte communicates with the IVB Bus. These pins represent the IV Byte interface with the SMS300 microcomputer system.

Pins UD0 - UD7 represent the 8-bit bus via which the IV Byte communicates with logic beyond the SMS300 microcomputer system.

$\overline{\text{ME}}$ is a master enable signal. This signal is connected to $\overline{\text{LB}}$ or $\overline{\text{RB}}$, output by the SMS300 Interpreter to distinguish between two 256 banks of I/O ports. $\overline{\text{ME}}$ is just one contributor to device select logic; we will describe the whole IV Byte select process later.

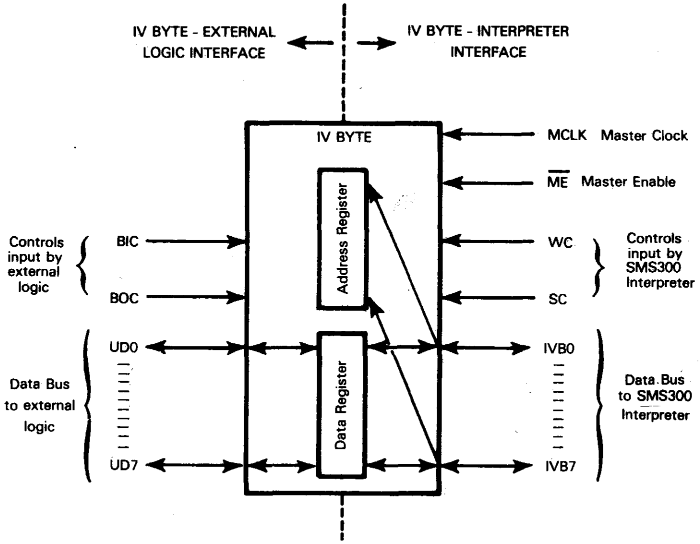

IV BYTE - EXTERNAL LOGIC INTERFACE
IV BYTE - INTERPRETER INTERFACE
IV BYTE
Address Register
Data Register
MCLK Master Clock
$\overline{ME}$ Master Enable
WC
SC
Controls input by SMS300 Interpreter
IVB0
IVB7
Data Bus to SMS300 Interpreter
Controls input by external logic
BIC
BOC
UD0
UD7
Data Bus to external logic

Figure 13-9. SMS360 IV Byte Control Sgnals And Interfaces

$\overline{BIC}$ and $\overline{BOC}$ are signals which control data flow between the IV Byte and external logic, via the UD Bus. $\overline{BIC}$ and $\overline{BOC}$ must be input to the IV Byte by external logic. MCLK, output by the SMS300 Interpreter, synchronizes actual data transfers. BIC, BOC and MCLK combine to control events on the UD Bus as follows:

$\overline{BIC}$	$\overline{BOC}$	MCLK	
1	0	X	IV Byte output data to external logic
0	X	1	External logic input data to IV Byte
0	X	0	Disable UD Bus
1	1	X	

X signifies "don't care"; the signal may be low or high.

SC and WC control the IVB Bus which connects all IVB bytes with the SMS300 Interpreter. Control signals SC and WC are automatically output by the SMS300 Interpreter. $\overline{BIC}$ contributes to IVB Bus logic in order to resolve access conflicts; external logic accessing the IV Byte via the UD Bus will have priority over an SMS300 Interpreter access occurring via the IVB Bus. As for UD Bus control logic, MCLK synchronizes data transfers occurring via the IVB Bus. IVB Bus control logic also requires $\overline{ME}$ to be low; observe that UD Bus logic ignores $\overline{ME}$. Combining SC, WC, $\overline{BIC}$, MCLK and $\overline{ME}$, this is how IVB Bus interface logic responds to control signals:

SC	WC	$\overline{BIC}$	MCLK	$\overline{ME}$	
X	X	X	X	1	IV Byte not selected; no operation.
0	0	X	X	0	IV Byte must place data contents on IVB Bus.
0	1	1	1	0	IV Byte reads IVB Bus as data.
1	0	X	1	0	IV Byte reads IVB Bus as a select address.
1	1	0	1	0	
1	1	1	1	0	IV Byte reads IVB Bus as its new select address.

SMS360 IV BYTE OPERATION

SMS360 IV BYTE SELECT LOGIC

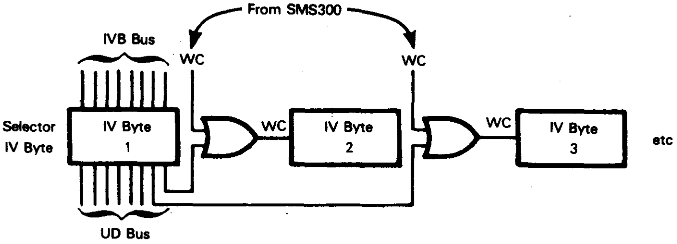
You must initialize every single IV Byte connected to an SMS300 Interpreter by outputting a select address to this IV Byte.

In order to provide an IV Byte with its select address, you must output data to the IV Byte via the IVB Bus, while SC, WC, $\overline{BIC}$ and MCLK are all high — and $\overline{ME}$, the master enable signal, is low. This signal combination overrides all other select logic on the IVB Bus interface of the IV Byte.

You will usually want to provide every IV Byte in an SMS300 microcomputer system with a unique select address. You can give two IV Bytes the same select address, but if you do, they will compete with each other during all IVB Bus accesses.

You will have to use some form of external logic in order to provide an IV Byte with its select address, since the SMS300 Interpreter never outputs SC and WC high simultaneously. If you are going to provide IV Bytes in an SMS300 microcomputer system with a single select address, which will apply forever, then you can have a switch or connector which forces the WC input to IV Bytes high, one IV Byte at a time, while a program is executed to write addresses to the appropriate bank of IV Bytes.

If you want to provide IV Bytes with new select addresses under program control, one method would be to set aside one IV Byte whose outputs will be ORed with the existing WC connections to other IV Bytes. You may write a mask into the first IV Byte in order to force another IV Byte's WC input high, thus allowing the second IV Byte to receive a new select address. This may be illustrated as follows:



SMS360 IV BYTE ACCESS LOGIC

Let us now look at dialogue occurring between an IV Byte and the SMS300 Interpreter via the IVB Bus. Note carefully that the following discussion applies only to the IV Byte-SMS300 interface. The IV Byte-external logic interface is controlled entirely by external logic manipulating the $\overline{BIC}$ and $\overline{BOC}$ control signals.

At any time, just one IV Byte will consider itself selected on the left bank of IV Bytes, and just one IV Byte will consider itself selected on the right bank of IV Bytes. In order to select an IV Byte, you execute a Move instruction which outputs data to Register 7 for the left bank, or F for the right bank. There is no Register 7 or F; in response to either of these Move instruction destination definitions, the SMS300 outputs data on the IVB Bus, just as it would for any normal data output operation, but control signals SC and WC are set to 1 and 0, respectively. A destination Register of 7 causes $\overline{LB}$ to be output low, while the destination address F causes $\overline{RB}$ to be output low. Thus, the net effect of executing a Move instruction specifying Register 7 or F as the destination is that the data moved is the address of the IV Byte which is going to consider itself selected; all other IV Bytes will at this time deselect themselves. If no IV Byte has a select address equal to the address output, then all IV Bytes will be deselected.

Once an IV Byte selects itself, it will remain selected until a subsequent Move to Register 7 or F causes a new Byte to select itself.

All SMS300 instructions that specify the IVB Bus as the source or destination of data will automatically access the single selected IV Byte — on the left or right bank of IV Bytes, whichever is being accessed by the Move instruction. Table 13-2 describes the way in which you specify whether the IV Byte selected on the left bank or right bank will be accessed.

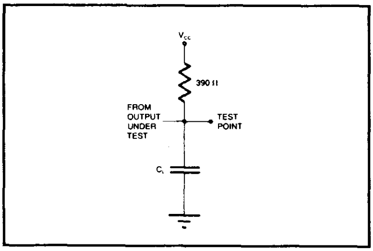
Observe that external logic will always have priority over the SMS300, should both simultaneously attempt to output data to an IV Byte. $\overline{\text{BIC}}$ will be input low by external logic whenever it is attempting to write to the IV Byte; but $\overline{\text{BIC}}$ low inhibits any attempt by the SMS300 Interpreter to write data into the IV Byte.

DATA SHEETS

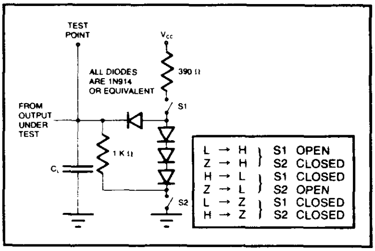
The following pages contain specific electrical characteristics of the SMS300 Interpreter and SMS360 Interface Vector Byte.

PARAMETER MEASUREMENT INFORMATION

Load Circuit for Open Collector Outputs

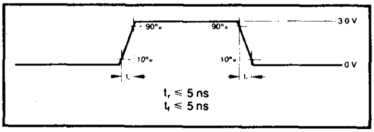


Load Circuit for Tri-state Outputs

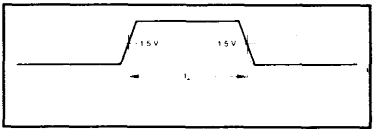


Note: C_L includes fixture capacitance.

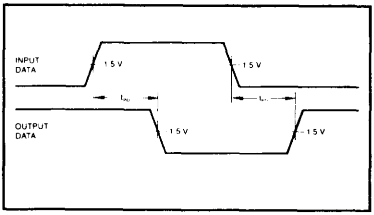
Input Waveform



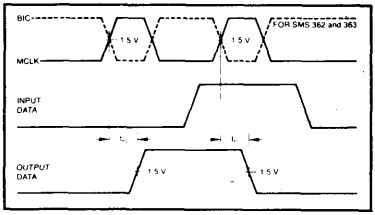
Clock Pulse Width



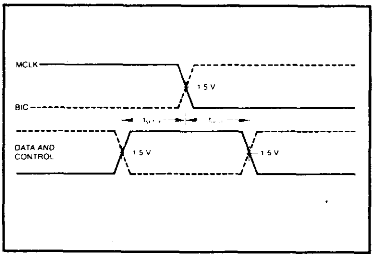
Data Delay Times
Input Data Referenced



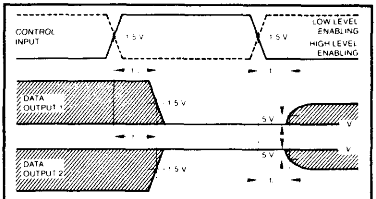
Data Delay Times
Clock Referenced



Setup and Hold Times



Output Enable and Disable Times (Tri-state Outputs)



Waveform #1 is for an output with internal conditions such that the output is low when the tri-state driver is enabled. Waveform #2 is for the opposite condition.

SPECIFICATIONS

Absolute Maximum Ratings:

Supply voltage (Note 1) .... 7 V   Input Voltage (Note 1) .... 5.5 V

TABLE 4. DC ELECTRICAL CHARACTERISTICS

(Limits apply for V_{CC} = 5 V ± 5% and 0°C ≤ T_A ≤ 70°C unless specified otherwise.)

Parameter	Symbol	Conditions	Limits			Units
			Minimum	Typical	Maximum	
High-Level Input Voltage	V_{IH}		2.0			V
Low-Level Input Voltage	V_{IL}				.8	V
Input Clamp Voltage	V_{CL}	I_I = −5 mA			−1	V
High-Level Input Current (Note 2)	I_{IH}	V_{CC} = 5.25 V V_{IH} = 5.25 V		<10	100	μA
Low-Level Input Current (Note 2)	I_{IL}	V_{CC} = 5.25 V V_{IL} = .5 V		−350	−550	μA
Low-Level Output Voltage	V_{OL}	V_{CC} = 4.75 V I_{OL} = 16 mA			.55	V
High-Level Output Voltage	V_{OH}	V_{CC} = 4.75 V I_{OH} = −3.2 mA	2.4			V
Short-Circuit Output Current (Note 3) UD Bus IV Bus	I_{OS}	V_{CC} = 4.75 V V_{CC} = 4.75 V	−10 −20			mA mA
Supply Current	I_{CC}	V_{CC} = 5.25 V		100	150	mA

Notes:

1. These limits do not apply during address programming.
2. The input current includes the Tri-state/Open Collector leakage current of the output driver on the data lines.
3. Only one output may be shorted at a time.

TABLE 5. AC ELECTRICAL CHARACTERISTICS

(Limits apply for V_{CC} = 5 V ± 5% and 0°C ≤ T_A ≤ 70°C.)

Parameter		Input	Output	Condition	Limits			Unit
					Minimum	Typical	Maximum	
t_{PD}	User Data Delay (Note 4)	UDX MCLK* BIC†	IVX	C_L = 50 pf		25 45 40	38 61 55	ns
t_{OE}	User Output Enable	BOC	UDX	C_L = 50 pf	18	26	47	ns
t_{OD}	User Output Disable	BIC BOC	UDX	C_L = 50 pf	18 16	28 23	35 33	ns
t_{PD}	IV Data Delay (Note 4)	IVBX MCLK	UDX	C_L = 50 pf		38 48	53 61	ns
t_{OE}	IV Output Enable	ME SC WC	IVX	C_L = 50 pf	14	19	25	ns
t_{OD}	IV Output Disable	ME SC WC	IVX	C_L = 50 pf	13	17	32	ns
t_W	Minimum Pulse Width	MCLK BIC†			40 35			ns
t_{SETUP}	Minimum Setup Time	UDX◇ BIC* IVX ME SC WC		(NOTE 5)	15 25 55 30 30 30			ns
t_{HOLD}	Minimum Hold Time	UDX◇ BIC* IVX ME SC WC		(NOTE 5)	25 10 10 5 5 5			ns

* Applies for SMS 360 and 361 only.

† Applies for SMS 362 and 363 only.

◇ Times are referenced to MCLK for SMS 360 and 361, and are referenced to BIC for SMS 362 and 363.

Notes:

4. Data delays referenced to the clock are valid only if the input data is stable at the arrival of the clock and the hold time requirement is met.
5. Setup and hold times given are for "normal" operation. BIC setup and hold times are for a user write operation. SC setup and hold times are for an IV Byte select operation. WC setup and hold times are for an IV Bus write operation. ME setup and hold times are for both IV write and select operations.

Address Programming

The IV Byte is manufactured such that an address of all high levels (> 2 V) on the IV Data Bus inputs matches the Byte's internal address. To program a bit so a low-level input (< 0.8 V) matches, the following procedure should be used:

1. Set all control inputs to their inactive state (BIC = BOC = ME = V_{CC}, SC = WC = MCLK = GND). Leave all IV Data Bus I/O pins open.
2. Raise V_{CC} to 7.75 V ± .25 V.
3. After V_{CC} has stabilized, apply a single programming pulse to the User Data Bus bit where a low-level match is desired. The voltage should be limited to 18 V; the current should be limited to 75 mA. Apply the pulse as shown in Diagram 1.

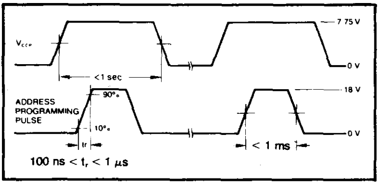


DIAGRAM 1. ADDRESS PROGRAMMING PULSE

4. Return V_{CC} to 0 V. (Note 6).
5. Repeat this procedure for each bit where a low-level match is desired.
6. Verify that the proper address is programmed by setting the Byte's status latch (IV0-IV7 = desired address, ME = WC = L, SC = MCLK = H). If the proper address has been programmed, data presented at the IV Bus will appear inverted on the User Bus outputs. (Use normal V_{CC} and input voltage for verification.)

After the desired address has been programmed, a second procedure must be followed to isolate the address circuitry. The procedure is:

1. Set V_{CC} and all control inputs to 0 V. (V_{CC} = BIC = BOC = ME = SC = WC = MCLK = 0 V). Leave all IV Data Bus I/O pins open.
2. Apply a protect programming pulse to every User Data Bus pin, one at a time. The voltage should be limited to 14 V; the current should be limited to 150 mA. Apply the pulse as shown in Diagram 2.

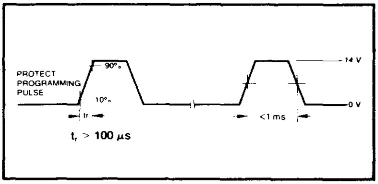


DIAGRAM 2. PROTECT PROGRAMMING PULSE

3. Verify that the address circuitry is isolated by applying 7 V to each User Data Bus pin and measuring less than 1 mA of input current. The conditons should be the same as in step 1 above. The rise time on the verification voltage must be slower than 100 μs.

TABLE 6. PROGRAMMING SPECIFICATIONS

Parameter	Symbol	Conditions	Limits			Units
			Minimum	Typical	Maximum	
Programming Supply Voltage Address Protect	V_{CCP}		7.5	 0	8.0	V V
Programming Supply Current	I_{CCP}	V_{CCP} = 8.0 V			250	mA
MAX TIME V_{CCP} > 5.25 V					1.0	s
Programming Voltage Address Protect			17.5 13.5		18.0 14.0	V V
Programming Current Address Protect					75 150	mA mA
Programming Pulse Rise Time Address Protect			.1 100		1	μs μs
Programming Pulse Width			.5		1	ms

Notes:

6. If all programming can be done in less than 1 second, V_{CC} may remain at 7.75 V for the entire programming cycle.

Chapter 14
THE NATIONAL SEMICONDUCTOR PACE

PACE was developed by National Semiconductor as a single chip implementation of the multi-chip IMP-16. Since it was the first 16-bit, single chip microprocessor, PACE is the first 16-bit microprocessor described in this book.

PACE is a 16-bit microprocessor because it handles data in 16-bit units. In many ways, however, the internal architecture of PACE has an 8-bit orientation; and this is something you should keep in mind while reading this chapter, because it does result in PACE having program execution speeds that are comparable to, rather than being significantly faster than, the 8-bit microprocessors we have described in earlier chapters.

Philosophically, PACE is a CPU that provides numerous control signals, and therefore assumes that you will use standard logic for the rest of the microcomputer system rather than relying upon a family of special devices. However, the majority of the PACE signals and pins are not TTL compatible; therefore the CPU must be surrounded by supporting devices that make appropriate signal conversions. Although these supporting devices can consist of standard logic, National Semiconductor provides several special support devices that simplify system design.

The only current manufacturer for PACE is:

NATIONAL SEMICONDUCTOR, INC.
2900 Semiconductor Drive
Santa Clara, CA 95050

Although there is an agreement between Rockwell International and National Semiconductor to exchange microcomputer technical information and produce each other's products, at the present time Rockwell International has not elected to second source PACE.

As shown in Figure 14-1, a typical PACE microcomputer will consist of a mixture of special-purpose PACE support devices and standard devices. **The PACE microcomputer devices described in this chapter consist of:**

- **The PACE CPU**
- **The System Timing Element (STE), which generates clock signals for PACE and the system.**
- **The Bidirectional Transceiver Element (BTE), which converts the MOS-level PACE signals to TTL-level signals for other devices. The BTE is 8 bits wide.**
- **The Microprocessor Interface Latch Element (MILE), which provides an 8-bit, bidirectional latched interface between the PACE System Bus and external devices.**

The remainder of the functional blocks shown in Figure 14-1 can be implemented with standard logic or with devices from other microcomputer families.

PACE POWER SUPPLY

The PACE CPU requires +5V, +8V and -12V power. The +8V is a substrate voltage requirement of the CPU and can be derived from the +5V power using a few discrete components. Therefore a PACE system can be implemented using only two primary power supplies: +5V and -12V.

PACE INSTRUCTION EXECUTION SPEED

PACE (IPC-16A/520D) uses a 750 nanosecond clock to provide typical instruction execution times in the range of 12 to 30 microseconds. Before making direct comparisons of these instruction execution times with those of other devices, however, note carefully that because of the 16-bit architecture of PACE it may take many instructions on another microcomputer to perform the same operations as a single PACE instruction.

PACE LOGIC LEVEL

MOS level signals are input and output by PACE.

P-channel silicon-gate, MOS/LSI technology is used with PACE.

A PACE MICROCOMPUTER SYSTEM OVERVIEW

Figure 14-1 conceptually illustrates a PACE system.

As with any mini- or microcomputer system, the CPU outputs data, address and control signals. In the case of PACE, the data and address signals use the same bus lines; therefore, they are said to be multiplexed.

THE SYSTEM TIMING ELEMENT (STE)

Timing signals needed by PACE are generated by the System Timing Element (STE). PACE signals are all MOS level; the STE therefore generates two sets of timing signals; one set are MOS level, for PACE, the other set are TTL level for external logic.

THE BIDIRECTIONAL TRANSCEIVER ELEMENT (BTE)

Since PACE signals are MOS level, Bidirectional Transceiver Elements (BTEs) must be present to translate out-going signals from MOS to TTL levels, and to translate incoming signals from TTL to MOS levels. BTEs are quite indiscriminating in the signals they translate; in either direction, any signal arriving at an input pin is faithfully reproduced at the corresponding output pin. Control signal options allow a BTE to operate bidirectionally, to drive output signals only, or to place both the MOS and TTL outputs in a high-impedance mode. Since the BTE is 8 bits wide, two BTEs operating bidirectionally provide buffering for the 16-bit Address/Data Bus. A third BTE, operating in the drive-only mode, provides buffering for the PACE control signals (NADS, ODS, IDS, and Flags).

As compared to microcomputers with TTL level logic, PACE may appear to be handicapped by the need for BTE devices; this is only true in very small systems. Typically, microcomputers that output TTL level signals have a very limited load; at most, eight devices may be driven by the TTL level signals before buffer amplifiers are required. The BTE, on the other hand, can handle up to 30 TTL level loads; therefore, for more complex microcomputer applications (and these are the applications for which PACE is intended) the presence of BTEs constitutes no penalty, since the BTEs serve as buffer amplifiers.

TTL LEVEL PACE BUS

A complete TTL level bus is created by combining BTE outputs with the TTL level timing signals output by the STE. Remember, though, that the 16 address/data lines are multiplexed. External logic that can demultiplex these lines and that can respond to the PACE timing and control signal logic can connect directly to the TTL level address/data lines. For example, National Semiconductor provides ROM and RAM devices with on-chip address latches; these devices can interface directly to the TTL level bus.

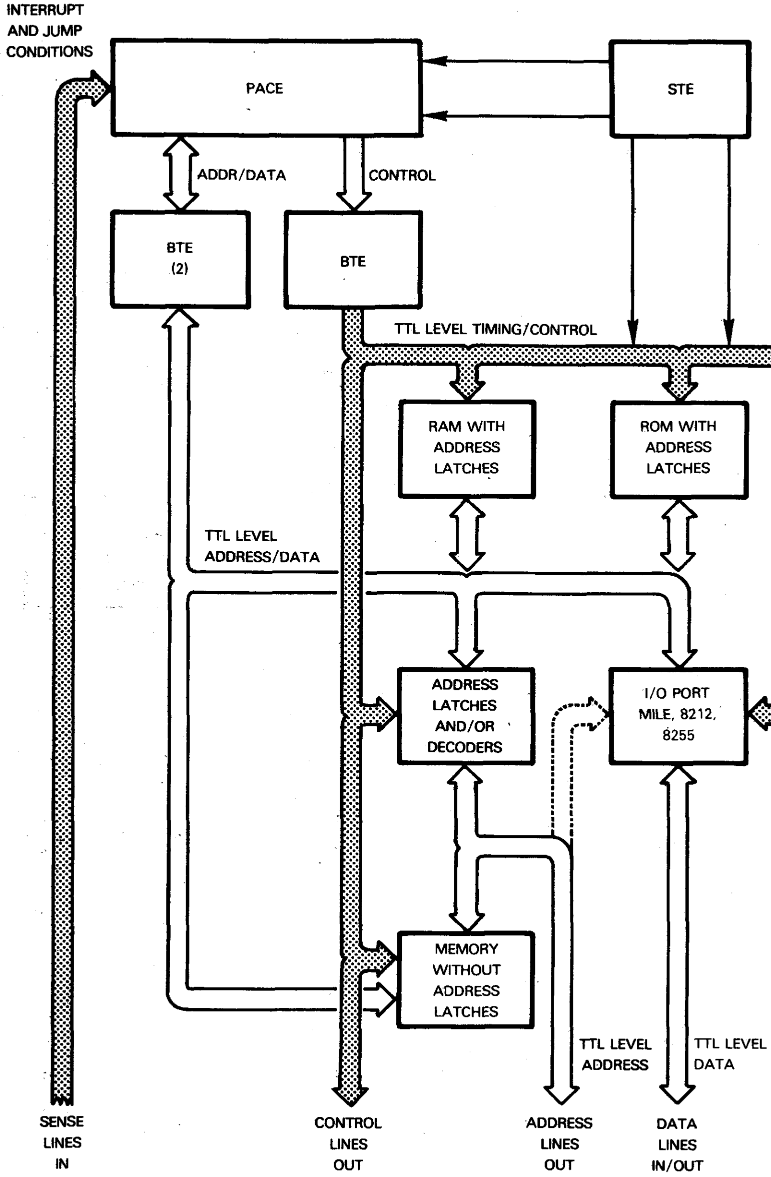


Figure 14-1. A National Semiconductor PACE Microcomputer System

ADDRESS LATCHES AND DECODERS

If memory devices or I/O ports are used that cannot demultiplex the address/data lines, you must provide separate logic to perform this function. No special PACE family devices are available for this purpose; however, standard logic devices can be used. For example, two hex flip-flop devices and a quad flip-flop device would provide a latched 16-bit Address Bus. The PACE Address Data Strobe (NADS) signal can be used as the CLK input to the flip-flops. In many systems this is the most effective approach since a latched Address Bus allows you to use simpler address decoding techniques to generate memory chip enable and I/O port select signals.

THE MICROPROCESSOR INTERFACE LATCH ELEMENT (MILE)

The Microprocessor Interface Latch Element (MILE) is a bidirectional I/O port with chip select logic and status signals. The MILE latches data from the TTL Address/Data Bus for retransmission to peripheral devices, and captures data from peripheral devices for input to PACE. The MILE is the only true general-purpose device in the PACE family of microcomputer support devices. In Figure 14-1, several other devices are listed in the I/O port section along with the MILE. We have done this to indicate that **the I/O port function can also be implemented using devices from other microcomputer families. Towards the end of this chapter, we will decribe the use of these other devices within a PACE microcomputer system.**

THE BIDIRECTIONAL TRANSCEIVER ELEMENT (BTE)

In Figure 14-2, the BTEs are shown as providing a thin slice through memory and I/O port interface logic; this is symbolic of the fact that BTEs are providing MOS-TTL signal level translations.

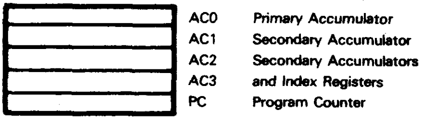
Memory interface and I/O port interface logic is shown in Figure 14-1 as being only partially present. You will need additional logic in most systems to demultiplex the Address/Data Bus.

PACE I/O PORTS

The bidirectional data pins of the Microprocessor Interface Latch Elements (MILEs) constitute the actual PACE I/O ports.

PACE PROGRAMMABLE REGISTERS

PACE has four 16-bit Accumulators and a 16-bit Program Counter; these registers may be illustrated as follows:



Accumulator AC0 may be likened to a primary Accumulator as described for our hypothetical microcomputer in Volume I.

Accumulator AC1 is a secondary Accumulator.

Accumulators AC2 and AC3 are equivalent to a combination of secondary Accumulators and Index registers.

Recall from Volume I, Chapter 6 that an Index register differs from a Data Counter in that the Index register contents are added to a displacement (which is provided by a memory reference instruction) in order to determine the effective memory address.

The Program Counter serves the same function in a PACE system as it does in our hypothetical microcomputer described in Volume I.

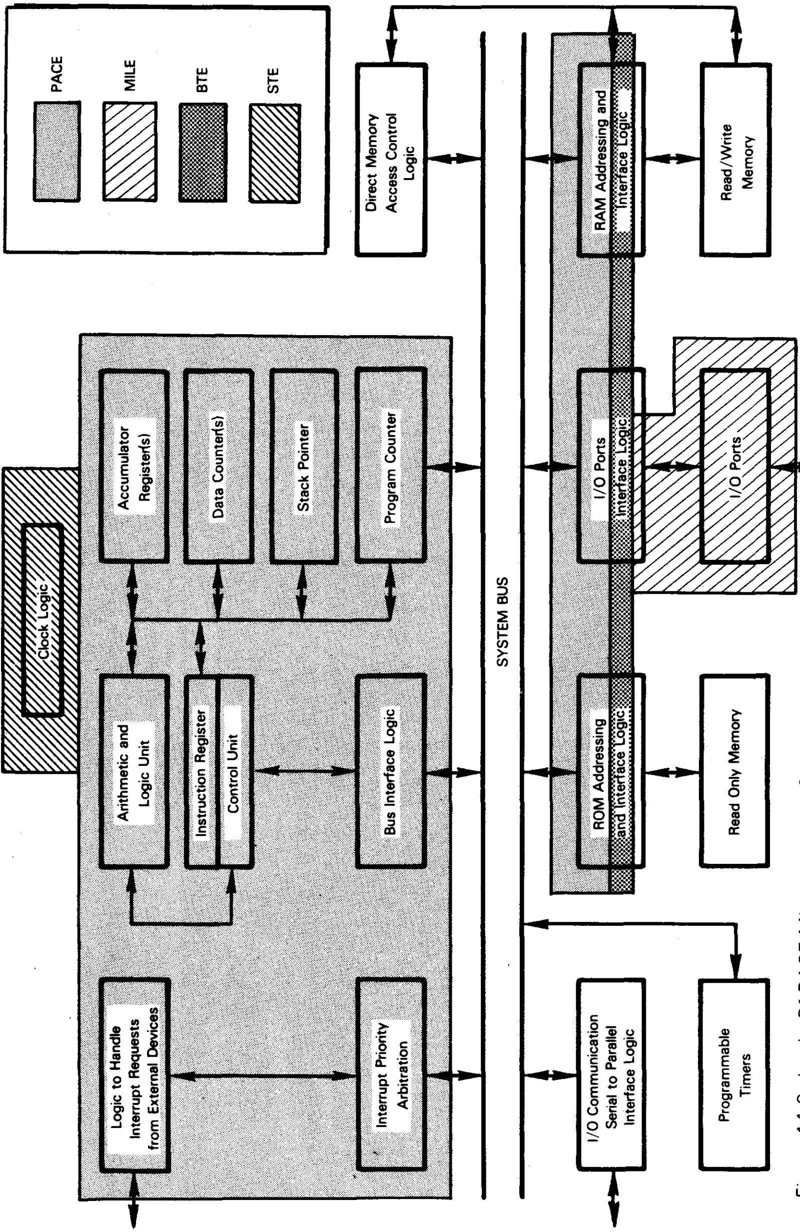

PACE
MILE
BTE
STE
Clock Logic
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Arithmetic and Logic Unit
Instruction Register
Control Unit
Bus Interface Logic
Logic to Handle Interrupt Requests from External Devices
Interrupt Priority Arbitration
Direct Memory Access Control Logic
SYSTEM BUS
RAM Addressing and Interface Logic
Read/Write Memory
I/O Ports
Interface Logic
I/O Ports
ROM Addressing and Interface Logic
Read Only Memory
I/O Communication Serial to Parallel Interface Logic
Programmable Timers

Figure 14-2. Logic Of PACE Microcomputer System Devices

PACE STACK

A Stack is provided on the PACE chip. The Stack is 16 bits wide and 10 words deep. The PACE Stack is not a cascade stack, as described in Volume I, Chapter 6; rather, chip logic maintains its own Stack Pointer to identify the next free Stack word. The Stack Pointer is automatically incremented and decremented in response to Push and Pull operations. Stack Push and Pull operations are initiated by PACE logic during execution of Jump-to-Subroutine (JSR) and Return-from-Subroutine (RTS) instructions, and during interrupt processing to automatically save and restore the Program Counter.

In addition, **the Stack can be used for temporary storage of data or status information.**There are PACE instructions which allow you to transfer words between the Stack and any Accumulator, or the Status and Control Flag register. This capability can significantly reduce the number of memory accesses required (thus increasing system speed) and can also reduce read/write memory requirements since intermediate values can be stored on the Stack.

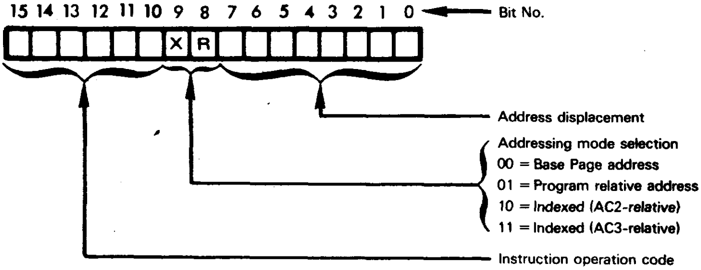
PACE STACK INTERRUPTS

Whenever the Stack becomes completely filled or emptied, an Interrupt Request is generated on the PACE chip. If you have enabled Stack Interrupts, program execution will be suspended allowing you to deal with the situation. This is an example of the minicomputer-like philosophy behind PACE. A microcomputer-oriented designer might conclude that a Stack Full or Empty condition represents a programming error, and as such, CPU real estate should not be wasted upon reducing front-end programming costs. However, the PACE philosophy assumes that you may be using the Stack as an easily accessible extension of read/write memory, instead of just for storage of the Program Counter during subroutine execution. Within this concept, a Stack Full condition will indicate that it is time to dump data accumulated on the Stack out to read/write memory.

PACE ADDRESSING MODES

Most PACE memory reference instructions use either direct or direct, indexed addressing. A few instructions also offer indirect addressing and pre-indexed, indirect addressing. Refer to Volume I, Chapter 6 for a description of these addressing modes.

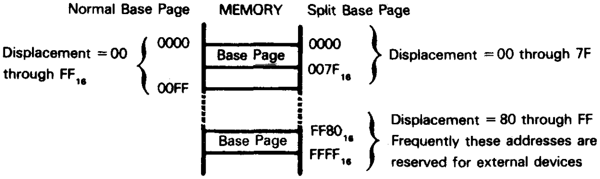
All PACE memory reference instructions have the following object code format:


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bit No.
X
R
Address displacement
Addressing mode selection
00 = Base Page address
01 = Program relative address
10 = Indexed (AC2-relative)
11 = Indexed (AC3-relative)
Instruction operation code

The 2-bit XR field lets you specify with each instruction the type of direct addressing you want used: base page, program relative or indexed (AC2- or AC3-relative). Since the address displacement is an 8-bit field in the instruction word, direct addresses are paged and each page consists of 256 words. Indexed and paged addressing variations have been described in Volume I, Chapter 6.

PACE SPLIT BASE PAGE

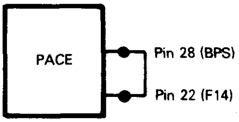
In addition, **PACE offers a variation of base page addressing, which is not described in Volume I, Chapter 6. There is a control signal (BPS) which is input to PACE, and allows the base page to be split between the top and bottom 128 words of memory,** as follows:



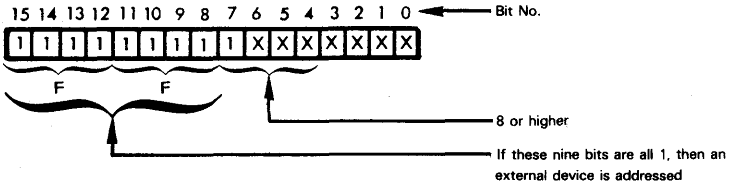
Normal Base Page
MEMORY
Split Base Page
Displacement = 00 through FF$_{16}$
0000
00FF
Base Page
Base Page
0000
007F$_{16}$
FF80$_{16}$
FFFF$_{16}$
Displacement = 00 through 7F
Displacement = 80 through FF
Frequently these addresses are reserved for external devices

BPS high splits the base page; BPS low keeps the base page as the bottom 256 words of memory.

Depending on how a PACE system has been configured, the base page may be permanently defined as split, or normal; or the base page may be varied between the two options under program control. There are a number of control flags (which are described next) that may be set or reset under program control; these control flags are output at PACE pins. If one of these flags is connected to the base page select pin, then setting or resetting this flag determines which base page option will be in effect:



Splitting the base page between the top and bottom of memory is useful in a PACE microcomputer system because it simplifies external device addressing.If we reserve all memory addresses in the range FF80$_{16}$ - FFFF$_{16}$ for external devices, then external logic merely has to AND the top nine bits of an address and thus determine if an external device (rather than a memory location) is being addressed:

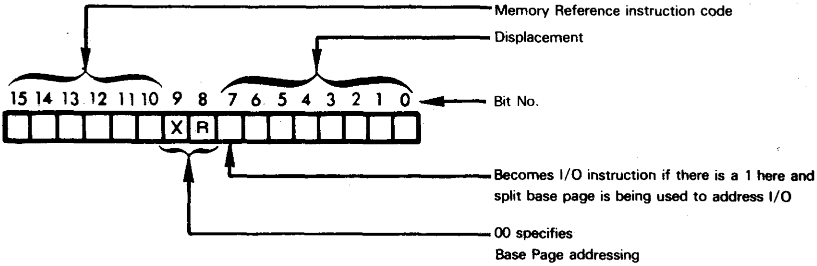


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bit No.
1 1 1 1 1 1 1 1 1 X X X X X X X
F
F
8 or higher
If these nine bits are all 1, then an external device is addressed

Splitting the base page also makes it easy to implement half of the base page in ROM, leaving the other half in RAM.

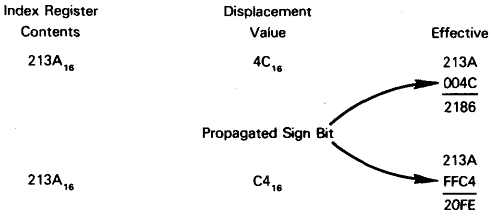
PACE SPLIT BASE PAGE TO ADDRESS I/O

To a programmer, this scheme provides an easy way of generating 128 external device addresses. If the split base page option is in effect, then base page, direct addressing can be interpreted as external device addressing, so long as the high order bit of the displacement is 1:


Memory Reference instruction code
Displacement
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X
R
Bit No.
Becomes I/O instruction if there is a 1 here and split base page is being used to address I/O
00 specifies Base Page addressing

PACE DIRECT INDEXED ADDRESSING

The base page and program relative options do not apply when the displacement is part of a direct, indexed address. **When indexed addressing is specified, PACE adds the contents of the displacement, as a signed binary number, to the contents of the identified Index register (AC2 or AC3). The sum becomes the effective address.** Here are some examples:

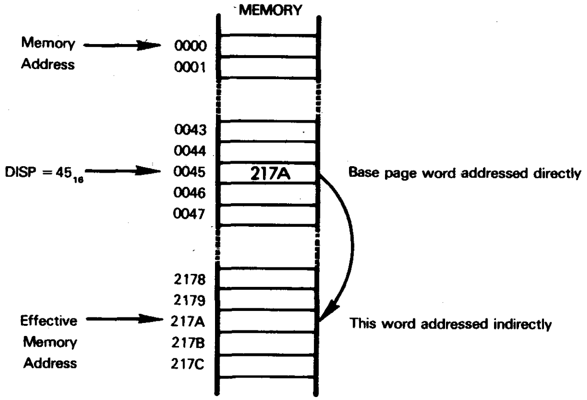


Index Register Contents	Displacement Value	Effective
$213A_{16}$	$4C_{16}$	213A 004C 2186
$213A_{16}$	$C4_{16}$	213A FFC4 20FE

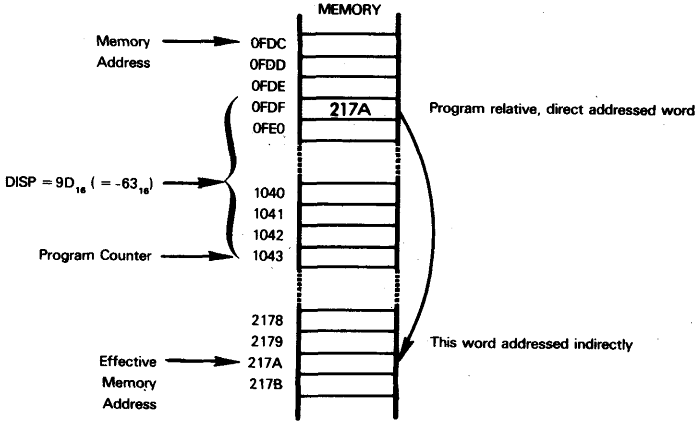
Observe that the high order bit of the displacement, being a sign bit, is propagated through the missing high order displacement byte.

Instructions that allow indirect addressing simply superimpose indirect addressing logic on the preceding direct address generation logic. For example, if indirect addressing without indexing is specified, then a base page or program relative direct address will be computed in the normal way, but the effective address is contained in the memory location identified by the direct address.

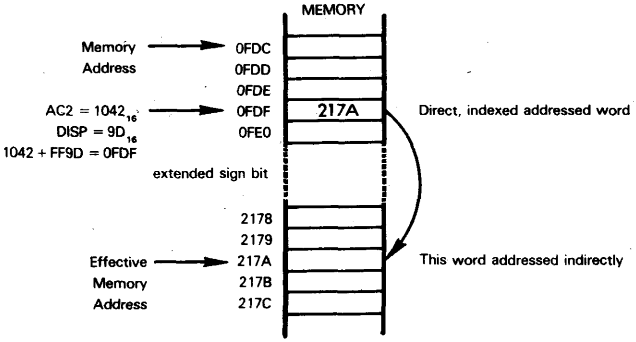
This illustration shows base page, indirect addressing; arbitrary memory addresses are used to make the illustration easier to understand:



This illustration shows program relative, indirect addressing; again using arbitrary memory addresses:

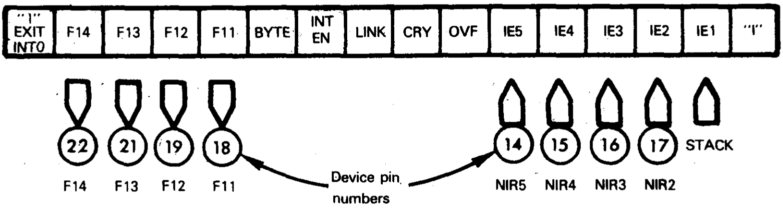


If indirect addressing with indexing is specified, then a direct address is first computed by adding the displacement, as a signed binary number, to the contents of the specified Index register; the direct indexed address thus computed provides the memory location where the indirect address will be found. This is illustrated as follows:



PACE STATUS AND CONTROL FLAGS

PACE has a 16-bit Status and Control Flag register. This register is on the CPU chip and is illustrated as follows:



Fourteen of the 16 register bits are used. Three of the 14 bits are status flags as we define a status flag. These three flags are:

Overflow (OVF), which is a typical Overflow status.

Carry (CRY), which is set and reset by arithmetic operations, as described for a typical Carry status.

Link (LINK), which is set and reset by Shift and Rotate instructions, as described for the hypothetical microcomputer's Carry status in Volume I, Chapter 7.

The separation of Carry into two statuses, one for shift and rotate operations, and the other for arithmetic operations, is a fairly common minicomputer feature; the advantage of separating these two statuses is that the results of arithmetic operations can be preserved across subsequent Shift and Rotate instructions.

BYTE causes data to be accessed in 8-bit lengths when this status is set to 1, or in 16-bit lengths when this status is set to 0.

Five bits (IE1 through IE5) are reserved for interrupt processing. These five bits selectively enable and disable PACE's five interrupt lines. One of these lines is reserved for the Stack Overflow interrupt, the other four lines are available for external device interrupt requests. There is also a single bit, master interrupt enable and disable (INT EN).

Bits F11, F12, F13 and F14 are control flags which are output directly to PACE device pins, and can be used in any way to control external devices. One use, to select normal or split base page addressing, has already been described.

Only the three status flags OVF, CRY and LINK are automatically set or reset in the course of instruction execution. The remaining 11 bits of the Status and Control Flags register are set and reset by instructions or instruction sequences that read data into, or write data out of, the Status and Control Flags register.

PACE CPU PINS AND SIGNALS

PACE CPU pins and signals are illustrated in Figure 14-3. A description of these signals is useful as a guide to the way in which a PACE microcomputer system works.

There are 16 data and address lines (D0 - D15), which are multiplexed for data input, data output and address output. Two control lines, ODS and NADS, identify output on the data and address lines as either data (ODS), or addresses (NADS). A further control line, IDS, is used to strobe data input.

The EXTEND control input is used by slow memories or external devices to lengthen an instruction's execution time by increasing the duration of a data input/output cycle; this extends the time available for memories or external devices to capture data output by PACE or to present input data to PACE.

The NINIT input control initializes PACE; the Program Counter is set to 0. The Stack Pointer, the Stack and the Status and Control Flags register are cleared.

BPS has already been described; it is used to select either normal or split base page, for base page direct addressing.

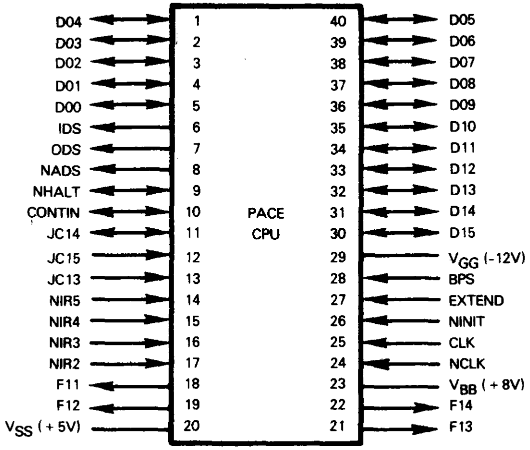
The NHALT control suspends instruction execution in between instructions, whereas EXTEND lengthens the execution time during an instruction. Suspending instruction execution is one way of performing direct memory access operations. **NHALT is also used in conjunction with CONTIN to initiate high priority interrupt processing.** Both of these uses of NHALT are described later.

The CONTIN signal is used to terminate a Halt condition and is also used as an output signal for interrupt acknowledgement. When CONTIN is properly sequenced with the NHALT signal, it initiates a high priority interrupt as we mentioned in the preceding paragraph. We'll provide a full discussion of the uses of NHALT and CONTIN later in this chapter. **CONTIN can also be used as a Jump condition input** in the same way as JC13, 14 and 15 which are described next.

JC13, 14 and 15 provide an interesting capability found in very few microcomputers discussed in this book; the condition of these three **inputs can be tested by a Branch-On-Condition (BOC) instruction,** thus allowing external control signals to directly manipulate PACE program instruction sequences.

F11, 12, 13 and 14 are the outputs for the corresponding flag bits in the Status and Control Flags register.

NIR2, 3, 4 and 5 are the external interrupt request lines. Interrupt priority arbitration logic is included on the PACE chip. NIR2 has the highest priority of the external interrupt lines and NIR5 is the lowest priority interrupt request.



PIN NAME	DESCRIPTION	TYPE
CLK, NCLK	Clock Lines	Input
*D00 - D15	Data/Address Lines	Tristate, Bidirectional
*IDS	Input Data Strobe	Output
*ODS	Output Data Strobe	Output
*NADS	Address Data Strobe	Output
*EXTEND	Clock Delay	Input
*NINIT	CPU Initialize	Input
*NHALT	†Stop CPU	Bidirectional
*CONTIN	†Continue Jump Condition	Bidirectional
*BPS	Base Page Select	Input
*JC13 - JC15	Control Flags	Output
*F11 - F14	Control Flags	Output
*NIR2 - NIR5	Interrupt Requests	Input
V_{BB}, V_{GG}, V_{SS}	Power Lines	Input
*JC13 - JC15	Jump Conditions	Input

*These signals connect to the System Bus.

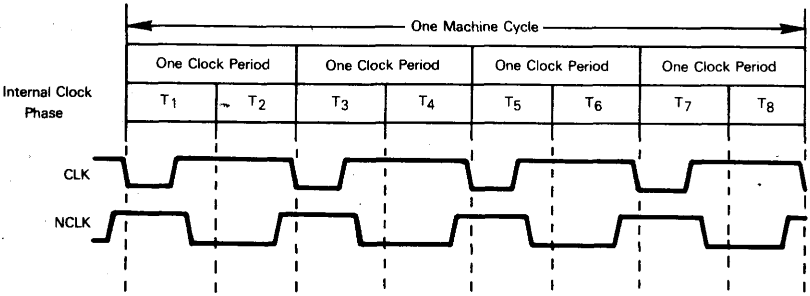
†These signals perform multiple functions. Refer to text.

Figure 14-3. PACE CPU Signals And Pin Assignments

PACE TIMING AND INSTRUCTION EXECUTION

PACE uses a combination of two clock signal inputs to time events internally within the microprocessor CPU. **The clock signals and the resultant internal clock phases can be illustrated as follows:**

PACE CLOCK SIGNALS



Several points should be noted regarding PACE timing. **The internal clock phases (T1 through T8) are meaningless to external logic since they are not accessible, nor are they needed for any external synchronization purposes.** We've shown them merely because they will simplify later discussions about data input/output operations. **Four external clock periods constitute a single PACE machine cycle.** Most PACE instructions require between four and seven machine cycles for execution.

PACE MACHINE CYCLE

So far as external logic is concerned, there are only three types of PACE machine cycles which can occur during execution of an instruction:

PACE MACHINE CYCLE TYPES

1) **A data input operation (read)** during which external logic must present a word of data to the CPU.
2) **A data output operation (write)** during which the CPU transmits a word of data to external logic.
3) **An internal operation** during which no CPU-initiated activity occurs on the System Bus.

All PACE instructions include one or more data input cycles, and two or more internal operation cycles. Only a few instructions include data output cycles. The first machine cycle of any instruction's execution must, of course, be an instruction fetch operation — which to external logic is simply a data input cycle. **Let us therefore begin by examining the data input cycle.**

Figure 14-4 illustrates timing for a standard data input machine cycle. Notice that the address data is only present on the data lines for the first portion of the cycle. The NADS signal is sent out approximately in the center of the time interval during which the address data is valid; therefore, either the leading edge or trailing edge of NADS can be used to clock the address data. The IDS signal is sent out at about the same time as the address information is taken off the data lines — well before the time when input data is expected by the CPU. This gives external logic time to prepare the input data. The input data needs to be valid only for a short time interval later in the machine cycle. Exact timing is given in the PACE data sheet at the end of this chapter.

PACE DATA INPUT CYCLE

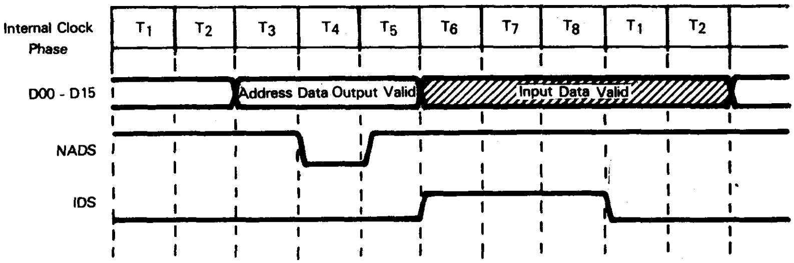


Figure 14-4 PACE Data Input Timing

PACE DATA OUTPUT CYCLE

Figure 14-5 illustrates timing for a standard data output cycle. The address-output portion of the cycle is identical to that of the data input cycle just described; the ODS signal is sent out at the same part of the cycle as IDS was. At approximately the same time that ODS is sent out, the output data word is placed on the data lines. The output data remains valid beyond the end of the ODS signal so that the trailing edge of ODS can be used as the clock for external data latches.

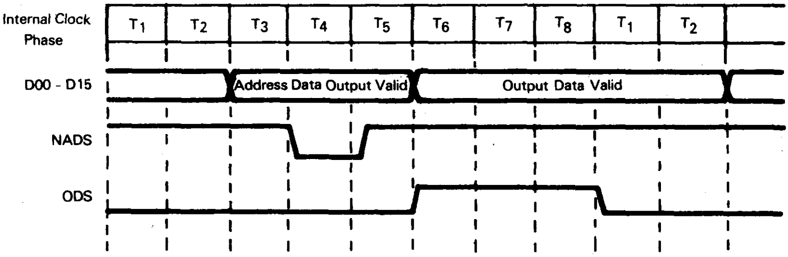


Figure 14-5 PACE Data Output Timing

PACE EXTEND SIGNAL FOR SLOW I/O OPERATIONS

The data input/output cycles just described allow approximately two clock periods for external logic to respond. If this time interval is too short, the EXTEND signal input to the CPU can be used to lengthen the I/O cycle by multiples of the clock period (one clock period equals two internal clock phases.) **The EXTEND signal** can be placed high during address time or immediately after the start of IDS or ODS, but it **must be high before the end of internal clock phase 6 as shown in Figure 14-6.**

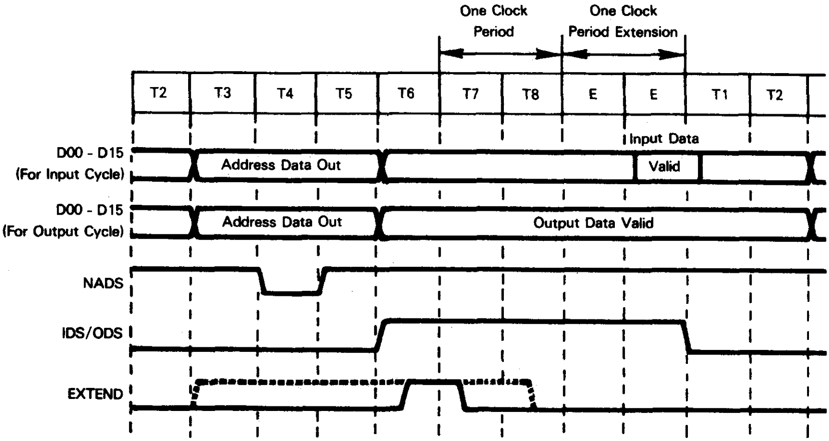


Figure 14-6. Using PACE EXTEND Signal To Lengthen I/O Cycles

The timing shown in Figure 14-6 provides the minimum I/O cycle extension of one clock period. The maximum extension permitted by PACE is 2 microseconds; so with a clock period of 750 nanoseconds, this means that only two clock period extensions can be added to an input/output cycle.The second clock period extension is achieved by holding the EXTEND signal high for one additional clock period beyond the timing shown in Figure 14-6.

Notice that the EXTEND signal does just what its name implies; it simply extends the duration of the data transfer portion of an I/O machine cycle. The trailing edge of the IDS or ODS signals is delayed and, for data input, the time until valid input data must be present is delayed. On data output cycles, the valid output data is simply maintained on the data lines by the CPU for an extended period of time.

The EXTEND signal can also be used to suspend CPU input activity. This use of EXTEND will be described later under the heading of Direct Memory Access.

THE INITIALIZATION OPERATION

A NINIT low signal input to the PACE CPU initializes the microprocessor. The NINIT signal is the equivalent of the Reset signal described for other microcomputers in this book. While NINIT is held low, PACE operation is suspended; IDS and ODS are reset low. NINIT must be held low for a minimum of eight clock periods to give the CPU time to respond. After NINIT goes high again, this is what happens:

1) The internal Stack Pointer is cleared.
2) All flags and interrupt enables are set low (except Level 0 Interrupt Enable which is set high).
3) The Accumulators contain arbitrary values.
4) The Program Counter is set to zero.
5) 16 clock periods after NINIT returns high, the NADS signal is output high. The first instruction is thus fetched from memory location zero (0000_{16}).

Figure 14-7 illustrates the timing for the initialization operation. Note that the NINIT signal is shown going low after power and clocks are both stable. The NINIT signal must be applied whenever the CPU is powered-up; if NINIT is held low before clocks and/or power have stabilized, the NADS and NHALT output signals may have undefined states for eight clock pulses after the trailing edge of NINIT.

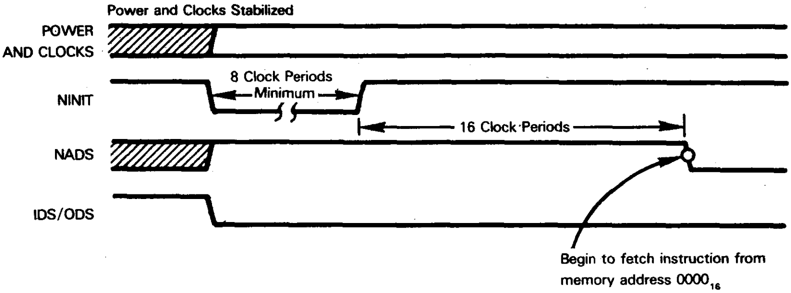

Power and Clocks Stabilized
POWER AND CLOCKS
NINIT
8 Clock Periods Minimum
16 Clock Periods
NADS
IDS/ODS
Begin to fetch instruction from memory address 0000_{16}

Figure 14-7. PACE Initialization Timing

THE HALT STATE AND PROCESSOR STALL OPERATIONS

Most microprocessors described in this book have a Hold state, which typically describes a CPU condition during which there is no CPU-initiated activity on the System Busses: external logic can then perform Direct Memory Access operations. The PACE CPU has an equivalent state that can be initiated under program control or by external logic. **When this state is initiated under program control (by executing a Halt instruction) PACE literature calls it the Halt state; when initiated by external logic, it is called a Processor Stall.**

During normal program execution, the CPU NHALT control line provides a high output. When a Halt instruction is executed, the NHALT output is driven low to indicate that CPU activity is suspended. While in the Halt state, the NHALT output has a 7/8 duty cycle; that is, every eighth clock phase, the NHALT output goes high. If the NHALT output is merely used to drive an indicator on a control panel, this 7/8 duty cycle is of little concern; but, if the NHALT signal is used as a logic signal, the 7/8 duty cycle must be accounted for. **The Halt state is terminated by setting the CONTIN input signal high for a minimum of 16 clock cycles, and then low for at least four clock cycles, as shown in Figure 14-8.** CPU operation then resumes by executing the next instruction, that is, the instruction that follows the Halt instruction.

THE PACE HALT STATE

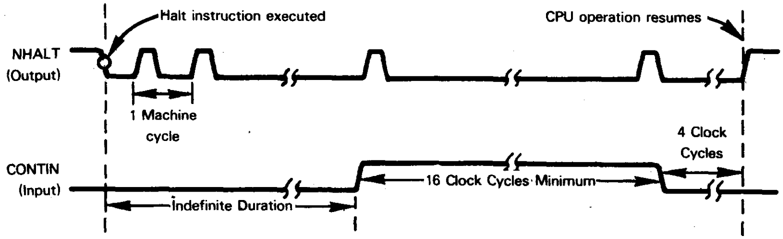


Figure 14-8. Terminating PACE Halt State

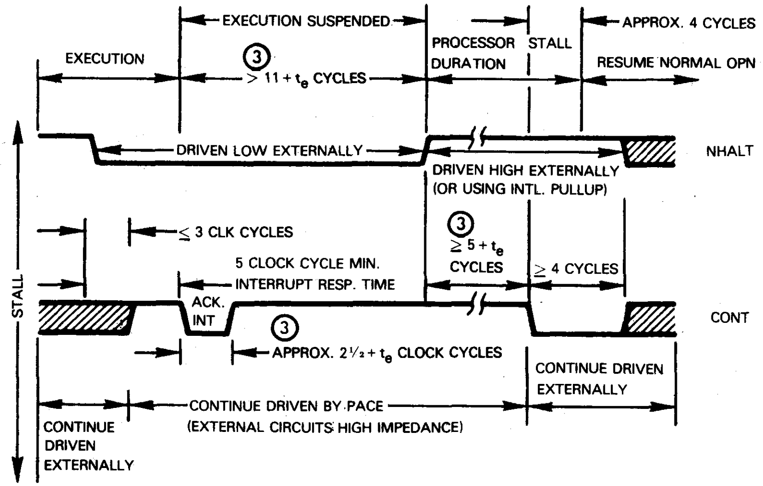
As we have just seen, **the PACE NHALT and CONTIN signals are interrelated.** We mentioned earlier that **these signals are also multifunctional.** We will describe separately each of the functions that can be implemented with NHALT and CONTIN. **Do not use these signals to implement more than one function unless your application absolutely requires the additional functions.** Critical and complicated timing relationships are required by the CPU

PACE NHALT AND CONTIN SIGNALS ARE MULTIFUNCTIONAL

to differentiate between various functions. Timing is further complicated by some circuit problems in the CPU's interrupt system which we will describe later.

PACE PROCESSOR STALL

The PACE CPU can be forced into the Halt state by external logic. PACE literature defines this operation as a Processor Stall. A Processor Stall uses both NHALT and CONTIN as control signal inputs. Figure 14-9 shows the timing sequence required. The NHALT input must be driven low by external logic to initiate the sequence. CPU operation is then suspended after execution of the current instruction is completed. The minimum response time is five clock cycles. The maximum response time is equal to the longest instruction execution time (refer to Table 14-2). There is no maximum time limit for a Processor Stall. The CPU simply remains in the Halt state until it is terminated by the CONTIN input signal, which must be properly sequenced with the removal of the NHALT input as shown in Figure 14-9.



NOTES:

1. EXTERNALLY GENERATED TTL INPUTS OVERRIDE PACE MOS OUTPUTS.
2. ▨ CROSSHATCH INDICATES "DON'T CARE" INPUT STATE.
3. t_e = DURATION OF EXTEND DURING PACE I/O CYCLES. TIMING ASSUMES NO OTHER EXTENDS AND NO SUSPENDS.

Figure 14-9. Timing Diagram For PACE Processor Stall Using NHALT And CONTIN Signals

PACE PROCESSOR STALL AND LEVEL 0 INTERRUPT SIMILARITIES

Let us take another look at the beginning of the Processor Stall timing sequence. **Notice that when the CPU has completed the current instruction and recognized the stall request, the CONTIN output signal is briefly driven low by the CPU.** This pulse is referred to as ACK INT (Acknowledge Interrupt) and can be used to let external logic know that the CPU is responding to the stall request. It may seem inappropriate for PACE to provide an

Acknowledge Interrupt response when we are initiating a Processor Stall. However, as we shall see later in this chapter, **a Level 0 Interrupt request begins with exactly the same timing sequence as a Processor Stall; in fact, the reaction of the CPU is the same for both operations until that point in the sequence where NHALT goes high.** Therefore, the initial response of ACK INT is always sent out after NHALT is driven low.

DIRECT MEMORY ACCESS OPERATIONS

At the beginning of our discussion about the PACE Halt state and Processor Stall, we mentioned that these were the equivalent of Hold states provided by other microprocessors. In actuality, **there are some significant differences between the PACE Halt state and the Hold state described for other microprocessors in this book. Because of these differences, Direct Memory Access operations with PACE are not so straightforward as with many other microprocessors.**

FLOATING PACE SYSTEM BUSSES

First, notice that the descriptions of the Halt state and Processor Stall do not mention floating the System Busses. This is because **the PACE CPU never floats the outputs of its data lines or control signals.** Remember, however, that the design of any realistic PACE system is going to require buffer/drivers for the PACE data lines and I/O control signals. At the beginning of this chapter we mentioned the BTE which is part of the PACE microcomputer family and performs this buffering function.

When we describe the BTE later in this chapter, you will see that it has control signals that place its outputs in the high impedance mode — that is, its outputs can be floated. Thus, ***it is the BTE control signals that must be manipulated in order to float the* System Busses for Direct Memory Access (DMA) operations. But first, we must have a way of determining whether the PACE CPU is going to be using the System Busses.** There are several methods of making this determination; we will conceptually examine each of them within the context of three different DMA schemes:

1) DMA block data transfers initiated by PACE CPU.
2) DMA block data transfers initiated by external logic.
3) Cycle stealing DMA transfers.

PACE CPU INITIATED DMA BLOCK DATA TRANSFERS

From a hardware point of view, the simplest method of implementing a DMA capability in a PACE system is to have the CPU initiate block transfers of data. Consider the following approach. PACE will treat an external DMA controller as a peripheral device and will establish initial conditions such as starting address, word count, and direction (memory read or write). This information can be passed to the controller by treating its registers as memory locations and using PACE Store instructions to write into the registers. When the required information has been passed, the CPU simply executes a Halt instruction. As we described earlier, **when a Halt instruction is executed, the NHALT control output line from the CPU is driven low. This signal could thus be used by the DMA controller as an indication that the CPU will not be using the System Bus and the DMA transfer can begin. When the transfer is completed, the DMA controller would use the CONTIN input to PACE, as shown in Figure 14-8, to terminate the Halt instruction and normal CPU operation would resume.**

DMA BLOCK DATA TRANSFERS INITIATED BY EXTERNAL LOGIC IN PACE SYSTEMS

Most microprocessors havé a Bus Request input signal that can be used by external logic to request access to the System Busses. **In a PACE system, the NHALT input signal can be used to force the CPU into a Processor Stall as described earlier and thus free the System Busses for DMA operations. The Acknowledge Interrupt (ACK INT) pulse on the CONTIN output line shown in Figure 14-9 is then equivalent to a Bus Grant signal**

and the DMA controller may begin the data transfer. When the transfer is complete, the CONTIN line is used as a control input line to the CPU to terminate the Processor Stall.

CYCLE-STEALING DMA IN PACE SYSTEMS

Cycle stealing DMA operations typically transfer a single word over the System Busses during a brief interval when the CPU is not using the busses. With this method, CPU operations need not be stopped; instead, they are only slowed down slightly or in some cases are not affected at all. **In order to implement cycle stealing DMA, external logic must have a way of detecting those time intervals when the CPU will not be using the System Busses.** We will describe two ways that this can be accomplished with the PACE CPU. The first method involves the use of the EXTEND input signal to the CPU to suppress or suspend input/output operations; the second method uses a special technique to sense when the CPU is beginning an internal (non-I/O) machine cycle.

EXTEND USED TO SUSPEND PACE I/O DURING DMA OPERATIONS

Earlier we described how to use the EXTEND input signal to lengthen the PACE CPU input/output cycles. The EXTEND signal can also be used to prevent the CPU from beginning an I/O cycle and thus ensure that the System Busses will be available to external devices for DMA operations.

Figure 14-10 illustrates both uses of the EXTEND signal. The CPU looks at the EXTEND input signal at internal clock phases T1 and T6. Notice that during I/O cycles, the IDS or ODS signal goes high at the beginning of T6 and low at the beginning of T1. If EXTEND is high during T6, then extra clock cycles are inserted after T8; this is the method that would be used to lengthen an I/O cycle. If EXTEND is high during T1, then the extra clock cycles are inserted between T3 and T4; this is the method we would use for DMA operations. The trailing edge of IDS/ODS indicates that the CPU has just completed an I/O cycle and is therefore not using the System Busses at this instant. By setting EXTEND high at this time, we suppress the beginning of another I/O cycle while we use the busses for a DMA transfer.

Notice that we are merely lengthening the beginning of the PACE machine cycle and thus delaying that part of the machine cycle where the CPU might begin I/O activity. We do not know whether the current machine cycle will be an internal machine cycle or an I/O cycle; and we do not care. We've just stolen the busses by slowing down the CPU.

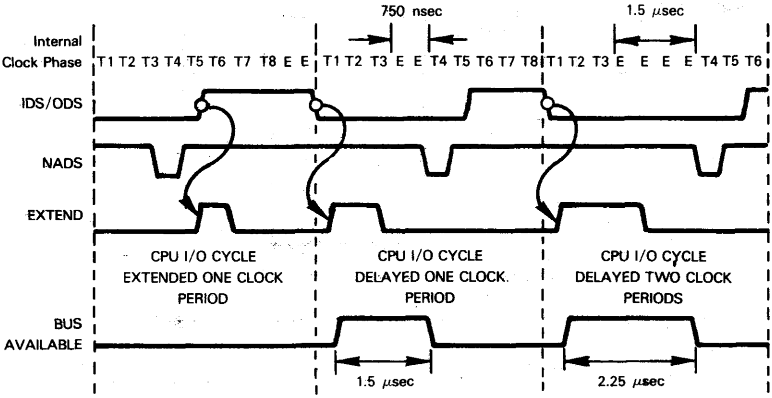

750 nsec
1.5 µsec
Internal Clock Phase
T1 T2 T3 T4 T5 T6 T7 T8 E E
T1 T2 T3 E E T4 T5 T6 T7 T8
T1 T2 T3 E E E E T4 T5 T6
IDS/ODS
NADS
EXTEND
CPU I/O CYCLE EXTENDED ONE CLOCK PERIOD
CPU I/O CYCLE DELAYED ONE CLOCK PERIOD
CPU I/O CYCLE DELAYED TWO CLOCK PERIODS
BUS AVAILABLE
1.5 µsec
2.25 µsec

Figure 14-10. Using PACE EXTEND Signal For Cycle Stealing DMA

This leads us to two drawbacks inherent in the EXTEND method of cycle stealing DMA. First, whenever we use the System Busses for a DMA transfer, we slow down the operation of the CPU. Secondly, we must wait until the PACE CPU has just completed an input/output cycle before we can perform the cycle steal. Since only about one-third of the CPU machine cycles are used for I/O, this means that bus access for DMA will be quite limited. Both of these drawbacks can be eliminated if we can find some technique for determining when the CPU is performing an internal (non-I/O) machine cycle. **We could then use the System Busses anytime that the CPU is not using them (which is more than 60% of the time) and we could perform the DMA transfer without slowing down CPU operations. We shall now describe just such a technique.**

CYCLE-STEALING DMA DURING PACE INTERNAL MACHINE CYCLES

We stated earlier in this chapter that the PACE internal clock phases (T1 through T8) are not available to external logic, and a cursory examination of PACE literature affirms this statement. However, the PACE data sheet includes a figure that shows equivalent circuits for PACE internal drivers and receivers. A detailed examination of this figure reveals a very interesting and useful fact: the JC13 (Jump Condition 13) pin on the CPU is intended as an input signal; but, because of the way in which the receiver for this signal is designed, it also produces an output pulse on the JC13 pin during every machine cycle. The output pulse occurs during T4 of each machine cycle and we can use this fact to design a very efficient cycle-stealing DMA arrangement.

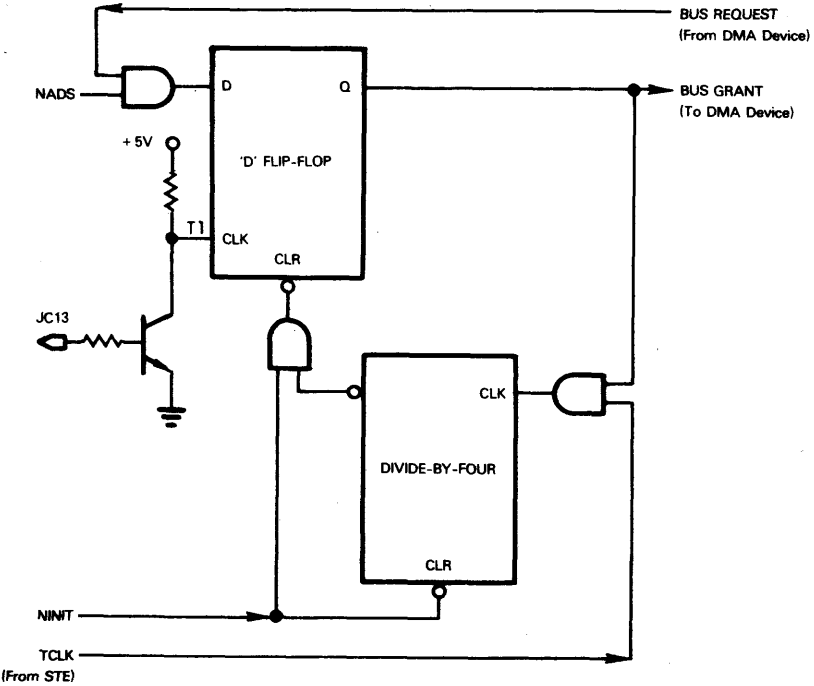


Figure 14-11. Idealized Circuit For Cycle-Stealing DMA During PACE Internal Machine Cycles

Figure 14-11 shows a circuit that uses the output pulse provided by JC13 to implement cycle-stealing DMA. Recall that the PACE CPU sends out a negative-going NADS pulse at T4 of every input/output cycle. This NADS signal is ANDed in our circuit with an external device's DMA Bus Request and applied to the D input of a flip-flop. The JC13 output pulse, which also occurs at T4, is inverted via a transistor and applied to the clock input of the flip-flop. Thus, if NADS is high at T4 (indicating that the current CPU machine cycle is not an I/O cycle) the flip-flop will be set if there is a Bus Request present. The output of this flip-flop is then used by external logic as a Bus Grant signal and the DMA transfer can be initiated. Since we do not know whether or not the next cycle will be a CPU I/O cycle, we must terminate DMA activity on the bus prior to the next T4 time. In Figure 14-11, this is accomplished using a divide-by-four counter.

The CLK input to the counter is a combination of the Bus Grant signal and the TCLK signal which is available from the PACE STE. This results in the timing shown in Figure 14-12. Notice that this scheme makes the bus available for about 7/8 of a machine cycle, or approximately 2.25 microseconds. If. you refer back to Figure 14-10 you will notice that this is about the same length of time as was obtained by using the maximum duration of EXTEND. So, we have not increased the maximum time available for a DMA transfer. But, we have made two significant gains: DMA transfers can occur more frequently, and these transfers do not slow down CPU operations.

We must add a final note of caution to the description of this otherwise straightforward DMA technique. There are several critical timing paths in the idealized circuit shown in Figure 14-11. Both the JC13 pulse and the NADS signal occur at T4, although the trailing edge of NADS does occur slightly after the trailing edge of JC13. Therefore, the components used to provide CLK and D inputs to the flip-flop must be selected carefully to ensure that there is not a race condition. Additionally, we have shown the Bus Grant signal being reset at the end of T3. Since the leading edge of NADS occurs at T4, this timing relationship can be critical. However, if external devices such as address latches and decoders use the trailing edge of NADS, this timing should present no problems.

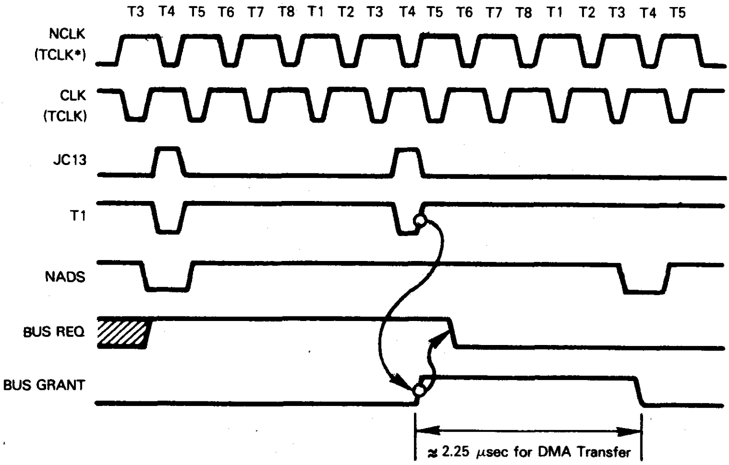

T3 T4 T5 T6 T7 T8 T1 T2 T3 T4 T5 T6 T7 T8 T1 T2 T3 T4 T5
NCLK
(TCLK*)
CLK
(TCLK)
JC13
T1
NADS
BUS REQ
BUS GRANT
≈ 2.25 μsec for DMA Transfer

Figure 14-12. Timing For Cycle-Stealing DMA During PACE Internal Machine Cycle

THE PACE INTERRUPT SYSTEM

The PACE CPU has one of the most complete on-chip interrupt systems of any of the microprocessors described in this book. Six separate levels of interrupts are provided: one internal, and five external interrupt request inputs, including a nonmaskable input. Priority logic is provided on the CPU, and all interrupts are vectored, thus eliminating any polling requirements. Because of the various ways in which interrupts can be initiated, and also because of a few problems that exist in the interrupt system, we will divide our description of the system into three parts:

1) **Low priority external interrupts**
2) **Internal (Stack) interrupts**
3) **Non-maskable (Level 0) interrupts**

But first, let us take an overview of the PACE interrupt system and look at those parts that are common to all interrupts.

PACE INTERRUPT PRIORITIES

Figure 14-13 depicts the interrupt logic that is contained on the PACE CPU. **The highest priority interrupt request is the nonmaskable Level 0 interrupt request which is initiated using the NHALT control input to the CPU. The lowest priority interrupt request is NIR5.**

ENABLING AND DISABLING PACE INTERRUPTS

The Stack Interrupt and each of the four lower-priority external interrupt requests can be individually enabled or disabled by setting or clearing associated bits (IE1 - IE5) in the Status and Control Flag register. Notice in Figure 14-13 that these bits are shown as providing the 'R' input to a latch. The 'S' input to each of these latches is the actual interrupt request line. The significance of this is rather subtle. It means that an interrupt request need not supply a continuous low level until it is acknowledged. Instead, any pulse exceeding one PACE clock period will set the associated interrupt request latch: this allows narrow timing or control pulses to be used as interrupt request inputs. Note, however, that the 'R' input to the latches overrides the 'S' input. Therefore, if the individual Interrupt Enable flag is reset, it not only prevents the latch from being set by interrupt requests, it will also clear a previously latched request that may or may not have been serviced.

A master interrupt enable (IEN) flag is also provided in the Status and Control Flag register. IEN must be set true to allow any of the latched interrupt requests to be recognized by the CPU.

PACE INTERRUPT RESPONSE

The PACE CPU checks for interrupts at the beginning of the instruction fetch routine that is performed after completion of each instruction. If an interrupt request is present (and enabled), the instruction fetch is aborted, the contents of the Program Counter are pushed onto the Stack, and the master interrupt enable (IEN) is set low. The CPU then loads the Program Counter with the address vector for your interrupt service routine and executes the instruction contained at that address. (We'll describe the address vectors in the next paragraph.) The interrupt request just described requires a total of 28 clock cycles from the time the interrupt is recognized by the CPU until the time when the first instruction of your interrupt service routine begins execution.

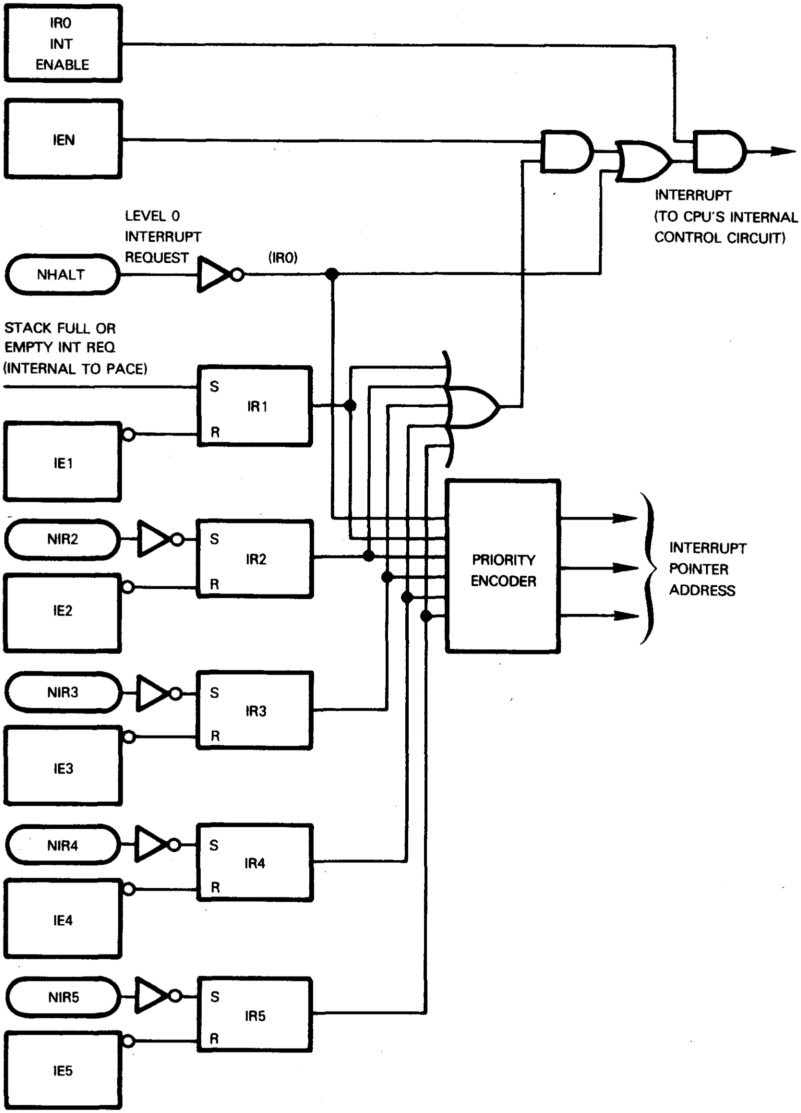


Figure 14-13. Internal View Of PACE Interrupt System

PACE INTERRUPT POINTERS

Memory locations 0002_{16} through 0008_{16} are used by the PACE interrupt system as pointer locations or address vectors. You load each of these locations with the starting address of your interrupt service routine for a particular level of interrupt. The interrupt level assignments for each location are as follows:

MEMORY LOCATION	INTERRUPT POINTER FOR	
2	Stack Interrupt	
3	NIR2	
4	NIR3	
5	NIR4	
6	NIR5	
7	Level 0	special case
8	Level 0	

When PACE responds to an interrupt, it loads the Program Counter with the contents of the memory location that is associated with the specific level of interrupt that is being acknowledged. Control is thus vectored to the proper service routine that you've designated for a given level of interrupt without performing any polling operations.

PACE INTERRUPT ACKNOWLEDGE AND RETURN FROM INTERRUPT

As part of the interrupt response we've just described, the PACE CPU sends out a low-going pulse on the CONTIN line. Refer back to Figure 14-9 and associated text for a description of the ACK INT pulse. The last instruction executed by your interrupt service routine must be a Return-from-Interrupt (RTI) instruction. This instruction sets IEN high to re-enable interrupts and then pulls the top of the Stack into the Program Counter to return program control to the point where it was interrupted. The RTI instruction does not clear the internal Interrupt Request latch; therefore your interrupt service routine must reset the latch (using a Pulse Flag instruction) or the same interrupt request would still be present after the RTI instruction. Once the latch has been cleared, it can then be re-enabled for subsequent interrupt requests.

SAVING PACE CPU REGISTERS DURING INTERRUPTS

In our description of interrupt response we made no mention of what happens to the CPU registers except for the Program Counter. The interrupt sequence does not save the contents of any registers except the Program Counter. If the program that was interrupted requires that the contents of the registers be saved and then restored, your interrupt service routine must perform these operations.

PACE STACK INTERRUPTS

The CPU's response to a stack interrupt is exactly the same as we've just described for external interrupts. However, as we've mentioned earlier in this chapter, the interrupt request is generated internally on the CPU chip and can be caused either by a Stack Full or Stack Empty condition. Remember that the 10-word Stack is part of the CPU chip. It consists of an internal RAM and a pointer that can address Stack words 0 through 9. A Stack Empty interrupt request is generated whenever the pointer is at 0 and a Pull instruction is executed. A Stack Full interrupt request occurs when the pointer is at 7 (eight entries on the Stack) and a Push instruction is executed to fill the ninth word. The tenth word of the Stack will then be used as part of the interrupt response to store the Program Counter contents. Unless you intend to extend the Stack out into main memory, your application program would probably not require an interrupt due to Stack Empty or Full since this would then be an error condition and can be avoided by careful programming. In this case, the Interrupt Enable flag for the Stack (IE1) can be turned off; then the full ten words of the Stack are available for use by your program.

PACE STACK INTERRUPT PROBLEMS

There is an additional reason for not using the Stack interrupt capability unless you really need it. **Current versions of the PACE CPU have an internal circuit problem that can cause improper interrupt response. If a Stack interrupt request occurs at the same time as an NIR3 or NIR5 interrupt request, the Stack interrupt address vector will be incorrectly accessed from location 0 instead of location 2.** The solution recommended in PACE literature is to load both of these locations with the Stack interrupt vector. This apparently straightforward solution is complicated by the fact that location 0 also happens to be the initialization address: whenever the CPU is initialized, the first instruction executed is the one that is contained in location 0. Thus, the word in location 0 must serve a dual purpose:

1) It serves as an instruction whenever the CPU is initialized.
2) It serves as an address vector if a Stack interrupt occurs at the same time as NIR3 or NIR4.

Here's an example. The object code for a Copy Flags to Register (CFR) instruction is 0400$_{16}$. So, if locations 0 and 2 both contained a value of 0400$_{16}$ the problem is solved. Your Stack interrupt service routine would have to begin at memory address 0400$_{16}$ but you would be correctly vectored to that address regardless of whether the interrupt error we've just described occurs. On initialization, the first instruction executed would be the CFR instruction: not a very useful initialization instruction, but at least no damage is done.

There are some instructions that should be avoided. **For a fuller discussion of this interrupt problem and the solution, refer to current PACE literature. Also keep in mind that the problem may be fixed in future versions of PACE.**

THE PACE NON-MASKABLE (LEVEL 0) INTERRUPT

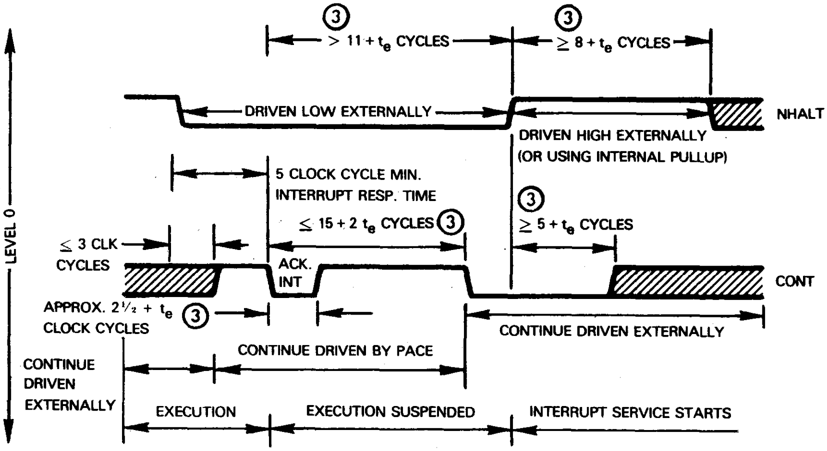
The PACE non-maskable (Level 0) interrupt cannot be disabled and differs from the other interrupt levels both in the way it is initiated and in the way the CPU responds to it.

INITIATING PACE LEVEL 0 INTERRUPTS

The Level 0 interrupt request is initiated using the NHALT control input signal in combination with the CONTIN input line. Figure 14-14 shows the timing relationships between NHALT and CONTIN that are required to initiate the non-maskable interrupt. If you compare this figure with Figure 14-9, you will notice that the Level 0 interrupt request and the Processor Stall begin in exactly the same way; NHALT is driven low by external logic and held low for some time after a low-going pulse (ACK INT) has been sent out on the CONTIN line. The only difference between the two operations is towards the end of the timing sequence. For a Processor Stall, NHALT is allowed to return high while CONTIN is still high; for a Level 0 interrupt, the CONTIN line must be driven low by external logic before the NHALT line is allowed to go high. This critical timing sequence is the only way that the CPU has to differentiate between a Processor Stall and a Level 0 interrupt. Notice that this Level 0 interrupt timing sequence never requires external logic to drive CONTIN high. Therefore, if you're not using the CONTIN line for any of its other multiple functions (including the ACK INT output pulse) you can merely tie CONTIN to ground and use NHALT to initiate the Level 0 interrupt.

PACE CPU LEVEL 0 INTERRUPT RESPONSE

The response of the PACE CPU to the Level 0 interrupt is subtly different from its response to other interrupts. These subtle differences are related to the slightly different purpose of a non-maskable interrupt versus a normal program interrupt request. A non-maskable interrupt is typically used only when there is a catastrophic error or failure (such as loss of power) or to implement a control panel for program development or debug purposes. Both of these uses require that an asynchronous, unplanned program termination have a minimum effect upon system status; that is, you want to leave behind a picture of the system as it looked immediately before the program termination occurred.



NOTES:

1. EXTERNALLY GENERATED TTL INPUTS OVERRIDE PACE MOS OUTPUTS
2. ▨ CROSSHATCH INDICATES "DON'T CARE" INPUT STATE.
3. t_e = DURATION OF EXTEND DURING PACE I/O CYCLES. TIMING ASSUMES NO OTHER EXTENDS AND NO SUSPENDS

Figure 14-14. Initiating PACE Level 0 Interrupt Using NHALT and CONTIN Signals

Remember that other levels of interrupts store the contents of the Program Counter or the Stack and reset the IEN flag in the Status and Control Flag register. This sequence obviously alters the 'picture' of the CPU since both Stack contents and Status and Control Flag register contents are changed. To avoid this, the Level 0 interrupt response by the CPU uses an external memory location to store the contents of the Program Counter. Memory location 0007$_{16}$ holds the address of the memory word where the Program Counter should be stored. Neither are the contents of the Status and Control Flag register altered; CPU internal circuitry resets an "IRO INT ENABLE" flag to prevent another interrupt from being recognized (refer to Figure 14-15) but this is not discernible to you. After the Program Counter has been saved in the designated memory location, the instruction contained in memory location 0008$_{16}$ is executed; this is the first instruction of your Level 0 interrupt service routine.

Notice that this sequence has not altered anything within the CPU that is discernible to you or to a program; the Stack, Accumulators, and Status and Control Flag register are all unchanged. Additionally, avoiding use of the Stack ensures that there will not be a Stack overflow — and in consequence a Stack interrupt generated by this interrupt response sequence.

RETURN FROM PACE LEVEL 0 INTERRUPT

The normal Return-from-Interrupt (RTI) instruction that must be executed at the end of your interrupt service routine causes the Program Counter to be restored from the Stack. **Since the Level 0 interrupt sequence does not utilize the Stack to store the Pro-**

gram Counter, a different technique must be used to return control to the interrupted program. First, you must execute a Set Flag (SFLG) or Pulse Flag (PFLG) instruction referencing bit 15 in the Status and Control Flag register. This bit always appears to be set to a'1', but must be referenced in this case to enable lower levels of interrupts. Next, you must execute a Jump Indirect (JMP@) through the location pointed to by the contents of memory location 0007_{16} to restore the original Program Counter contents.

PACE LEVEL 0 INTERRUPT PROBLEMS

As we mentioned earlier in our description of Stack interrupts, the PACE interrupt system is powerful, but current versions of PACE have some circuit problems. Another problem is associated with the Level 0 interrupt; it is more complicated to describe and more complicated to solve than the Stack interrupt problem. Here's the problem:

If a Level 0 interrupt occurs within the 12-clock-cycle period following the recognition of any other interrupt, PACE will either perform a Processor Stall (which we've described earlier) or PACE will execute the Level 0 interrupt — but using the wrong pointer address. In short, you don't know what might happen under these circumstances. There is a fix for this problem. It requires that external logic allow NHALT to be applied to the PACE CPU only while the NADS signal is present, and provided no Acknowledge Interrupt (ACK INT) has occurred since the last NADS pulse. ACK INT is accompanied by a negative-going pulse on the CONTIN line. Sound complicated? It is.

The circuit shown in Figure 14-15 is reproduced from current PACE literature and solves the problem we've just described. We won't attempt to describe here how this circuit solves the problem. Note that this circuit only takes care of Level 0 interrupt problems: if you also want to use NHALT and CONTIN to cause a Processor Stall, you'll have to design additional external logic.

Once again, we must advise that these interrupt system problems exist in current PACE CPU chips. You should refer to PACE literature for additional details and also to determine if these problems may have been resolved in later versions of PACE.

THE PACE INSTRUCTION SET

Table 14-1 summarizes the National Semiconductor PACE instruction set.

The primary memory reference instructions have typical minicomputer addressing modes. These instructions will also be used as I/O instructions, since external devices are identified via selected memory addresses.

PACE DIRECT ADDRESSING OPTIONS

In Table 14-1, "direct addressing options" means the instruction can reference memory using any of the direct, or direct indexed addressing options described for PACE.

"Indirect addressing options" similarly specifies any of the indirect addressing options described for PACE.

Both Branch and Skip instructions are provided, and each differs significantly from the philosophies described in Volume I, Chapter 6.

There are 16 conditions that can cause a Branch, as shown in Table 14-3. Notice that three of the conditions are determined by external inputs JC13, 14 and 15. If a Branch-on-Condition is true, then the displacement which is added to the Program Counter is an 8-bit, signed binary number as described in Volume I, Chapter 6.

There are three varieties of Skip-on-Condition instructions. SKNE, SKG and SKAZ compare the contents of an Accumulator to a memory location which is addressed using direct, or direct indexed addressing; based on the results of the comparison, the instruction following the Skip may or may not be executed. These three instructions are therefore combined Skip and Memory Reference instructions.

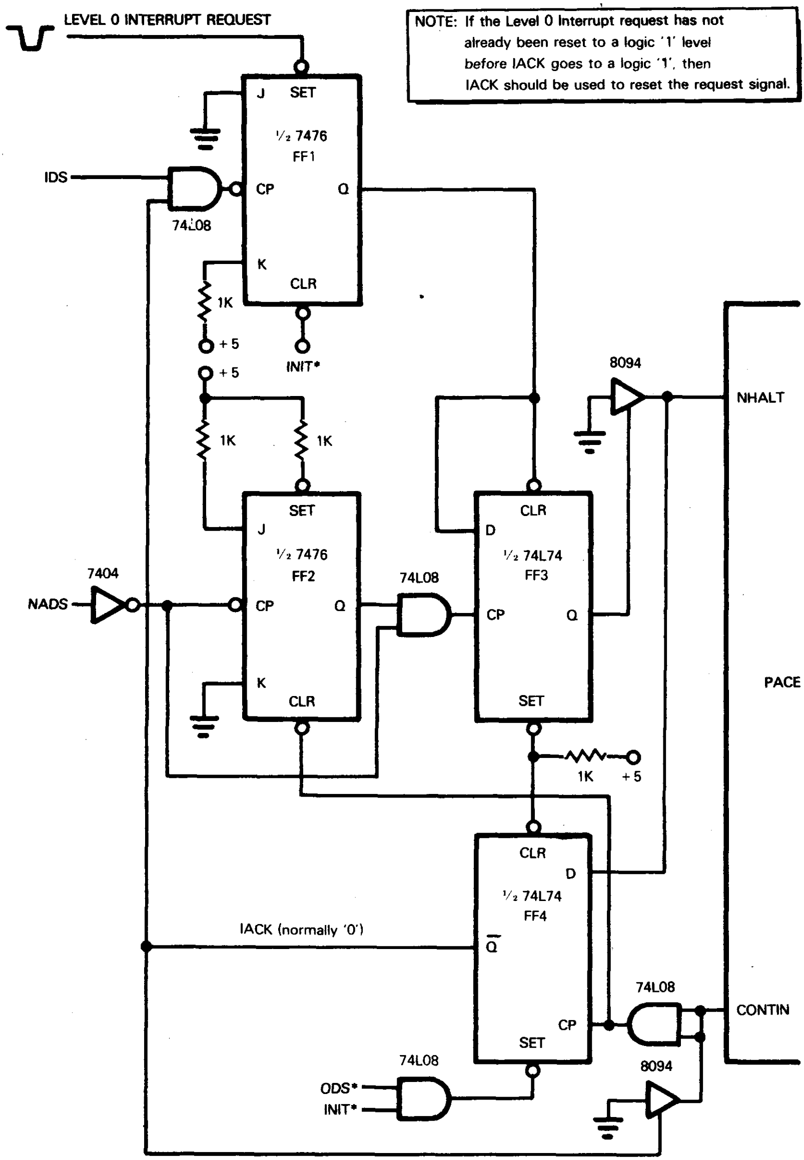

LEVEL 0 INTERRUPT REQUEST
NOTE: If the Level 0 Interrupt request has not already been reset to a logic '1' level before IACK goes to a logic '1', then IACK should be used to reset the request signal.
J
SET
½ 7476
FF1
IDS
CP
Q
74L08
K
CLR
1K
+5
INIT*
+5
1K
1K
8094
NHALT
SET
J
½ 7476
FF2
CLR
D
½ 74L74
FF3
7404
NADS
CP
Q
74L08
CP
Q
PACE
K
CLR
SET
1K
+5
CLR
D
½ 74L74
FF4
IACK (normally '0')
Q̄
74L08
CONTIN
CP
SET
74L08
8094
ODS*
INIT*

Figure 14-15. Circuit To Prevent Conflicts Between PACE Level 0 Interrupts And Lower Priority interrupts

ISZ and DSZ identify a memory location using direct, or direct indexed addressing; the contents of the addressed memory location are incremented (ISZ) or decremented (for DSZ); if after the increment or decrement operation the memory location contains a 0 value, then the Skip is performed.

The AISZ instruction adds an 8-bit, signed binary number to the contents of an Accumulator; if the result is 0, a Skip is performed.

These Skip instructions will be very familiar to minicomputer programmers, and on most microcomputers, are equivalent to a secondary Memory Reference or Immediate Operate instruction, followed by a Branch-on-Condition instruction.

The following symbols are used in Table 14-1:

AC0	Accumulator 0
C	Carry status
CC	4-bit Condition Code described in Table 14-3
D	Any Destination register
DATA8	8-bit binary data unit
DISP(X)	Direct or indexed addressing operands as explained in the text.
@ DISP(X)	Indirect addressing operands as explained in the text.
EA	The effective address generated by the specified operands.
f	4-bit quantity selecting a bit in the Flag Word.
FW	Flag Word described in the text.
IEN	Interrupt Enable status
l	A 1-bit unit determining whether LINK is included in the shift/rotate.
L	Link status
n	Seven bits determining how many single bit shift/rotates are performed.
O	Overflow status
PC	Program Counter
r	Any register of the Accumulator: AC0, AC1, AC2 or AC3
S	Any Source register
ST	Top word of on-chip Stack.
x<y,z>	Bits y through z of the quantity x. For example, r<7,0> is the low order byte of the specified register.
[ ]	Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified.If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.
[[ ]]	Implied memory addressing; the contents of the memory location designated by the contents of a register.
∧	Logical AND
∨	Logical OR
⊻	Logical Exclusive-OR
←	Data is transferred in the direction of the arrow.
⟷	Data is exchanged between the two locations designated on either side of the arrow.

Under the heading of STATUSES in Table 14-1, an X indicates statuses which are modified in the course of the instruction's execution. If there is no X, it means that the status maintains the value it had before the instruction was executed.

Table 14-1. PACE Instruction Set Summary

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES			OPERATION PERFORMED
				C	O	L	
PRIMARY MEMORY REFERENCE AND I/O	LD	r,DISP(X)	2				[r]←[EA] Load any Accumulator, direct addressing options.
	LD	0, ″DISP(X)	2				[AC0]←[EA] Load Primary Accumulator, indirect addressing options.
	ST	r,DISP(X)	2				[EA]←[r] Store any Accumulator, direct addressing options.
	ST	0, ″DISP(X)	2				[EA]←[AC0] Store Primary Accumulator, indirect addressing options.
	LSEX	0,DISP(X)	2				[AC0]←[EA](sign extended) Load a signed byte into Primary Accumulator; extend sign bit into high order byte. Direct addressing options.
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	ADD	r,DISP(X)	2	X	X		[r]←[r]+[EA] Add to any Accumulator, direct addressing options.
	DECA	0,DISP(X)	2	X	X		[AC0]←[AC0]+[EA]+[C] Add decimal with Carry to any Accumulator, direct addressing options.
	SUBB	0,DISP(X)	2	X	X		[AC0]←[AC0] - [EA]+[C] Subtract from Primary Accumulator with borrow, direct addressing options.
	AND	0,DISP(X)	2				[AC0]←[AC0] ∧ [EA] AND with Primary Accumulator, direct addressing options.
	OR	0,DISP(X)	2				[AC0]←[AC0] ∨ [EA] OR with Primary Accumulator, direct addressing options.
IMMEDIATE	LI	r,DATA8	2				[r<7,0>]← DATA8 (sign extended) Load immediate into any Accumulator. DATA8 is an 8-bit signed binary value. The sign bit is propagated through 8 high order bits.
	JMP	DISP(X)	2				[PC]← EA Jump by loading the effective direct address into the Program Counter.
	JMP	″DISP(X)	2				[PC]← EA Jump by loading the effective indirect address into the Program Counter.

Table 14-1. PACE Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES			OPERATION PERFORMED
				C	O	L	
IMMEDIATE (CONTINUED)	JSR	DISP(X)	2				[ST]←[PC] [PC]← EA Jump to subroutine direct. As JMP direct, but push old Program Counter contents onto Stack.
	JSR	@DISP(X)	2				[ST]←[PC] [PC]← EA Jump to subroutine indirect. As JMP indirect, but push old Program Counter contents onto Stack.
IMMEDIATE OPERATE	CAI	r,DATA8	2				[r]←[r]+DATA8 (sign extended) Complement contents of any register, then add immediate data.
BRANCH ON CONDITION	BOC	CC,DISP	2				If CC true; then [PC]← EA Branch on CC true, as defined in Table 14-3.
MEMORY REFERENCE AND SKIP (SEE TEXT)	SKNE	r,DISP(X)	2				If [r] ≠ [EA]; then [PC]←[PC]+1 Skip if any Accumulator not equal.
	SKG	0,DISP(X)	2				If [AC0] > [EA]; then [PC]←[PC]+1 **Skip if Primary Accumulator greater.**
	SKAZ	0,DISP(X)	2				If ([AC0] ∧ [EA]) = 0; then [PC]←[PC]+1 Skip if AND with Primary Accumulator is zero.

Table 14-1.. PACE Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES			OPERATION PERFORMED
				C	O	L	
MEMORY REFERENCE OPERATE AND SKIP	ISZ	DISP(X)	2				[EA]←[EA]+1 If [EA] = 0; then [PC]←[PC]+1 Increment memory, skip if zero.
	DSZ	DISP(X)	2				[EA]←[EA] - 1 If [EA] = 0; then [PC]←[PC]+1 Decrement memory, skip if zero.
IMMEDIATE OPERATE AND SKIP	AISZ	r,DATA8	2				[r]←[r]+DATA8 If [r] = 0; then [PC]←[PC]+1 Add immediate to any Accumulator. Skip if zero. DATA8 is an 8-bit signed binary immediate data value.
REGISTER-REGISTER MOVE	RCPY	S,D	2				[D]←[S] Move contents of any Accumulator (S) to any Accumulator (D).
	RXCH	S,D	2				[D]←→[S] Exchange contents of any Accumulators.
REGISTER-REGISTER OPERATE	RADD	S,D	2	X	X		[D]←[S]+[D] Binary add any Accumulator to any Accumulator.
	RADC	S,D	2	X	X		[D]←[S]+[D]+[C] Binary add with Carry any Accumulator to any Accumulator.
	RAND	S,D	2				[D]←[S] ∧ [D] AND any Accumulator with any Accumulator.
	RXOR	S,D	2				[D]←[S] ⊻ [D] Exclusive-OR any Accumulator with any Accumulator.
REGISTER OPERATE	SHL	r,n,1	2			X	Shift any Accumulator left n bits. Simple if 1 = 0; through Link if 1 = 1.
	SHR	r,n,1	2			X	Shift any Accumulator left n bits. Simple if 1 = 0, through Link if 1 = 1.
	ROL	r,n,1	2			X	As SHL, but rotate.
	ROR	r,n,1	2			X	As SHR, but rotate.

Table 14-1. PACE Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES			OPERATION PERFORMED
				C	O	L	
STACK	PUSH	r	2				[ST]←[r] Push any Accumulator contents onto Stack.
	PUSHF		2				[ST]←[FW] Push flags onto Stack.
	PULL	r	2				[r]←[ST] Pull top of Stack into any Accumulator.
	PULLF		2	X	X	X	[FW]←[ST] Pull top of Stack into flags.
	XCHRS	r	2				[ST]←→[r] Exchange contents of any Accumulator with top of Stack.
	RTS	DISP	2				[PC]←[ST] + DISP Return from subroutine. Move sum of DISP and top of Stack to PC. DISP is an 8-bit signed binary number.
INTERRUPT	RTI	DISP	2				[PC]←[ST]+ DISP [IEN]← 1 Return from interrupt. Like RTS, but enable interrupts.
STATUS	CFR	r	2				[r]←[FW] Copy flags to any Accumulator.
	CRF	r	2	X	X	X	[FW]←[r] Move any Accumulator contents to flags.
	SFLG	f	2				[FW<f>]← 1 Set flag f to 1. (f = 0 to 15).
	PFLG	f	2				[FW<f>]← 1 for four clock periods Pulse flag f (invert flag status for four clock periods). (f = 0 to 15).
	HALT		2				Halt

The following symbols are used in Table 14-2:

aa	Two bits choosing the destination register.
bb	Two bits choosing the Index register.
cccc	Four bits choosing the Condition Code. See Table 14-3.
ee	Two bits choosing the source register.
ffff	Four bits selecting a bit in the Flag Word.
l	One bit determining whether Link is included in a shift or rotate.
nnnnnnn	Seven bits determining how many single bit shifts or rotates are performed.
PP	8-bit signed displacement
QQ	Eight bits of immediate data.
x	A "don't care" bit.
XX	A "don't care" byte.

Table 14-2. PACE Instruction Set Object Codes

INSTRUCTION		OBJECT CODE	BYTES	MACHINE CYCLES			
				TOTAL	INTERNAL	INPUT	OUTPUT
ADD	r,DISP (X)	1110aabb PP	2	4	2	2	
AISZ	r,DATA8	011110aa QQ	2	5/6	4/5	1	
AND	0,DISP (X)	101010bb PP	2	4	2	2	
BOC	CC,DISP	0100cccc PP	2	5/6	4/5	1	
CAI	r,DATA8	011100aa QQ	2	5	4	1	
CFR	f	000001aa XX	2	4	3	1	
CRF	f	000010aa XX	2	4	3	1	
DECA	0,DISP (X)	100010bb PP	2	7	5	2	
DSZ	DISP (X)	101011bb PP	2	7/8	4/5	2	1
HALT		000000xx XX	2	-	-	1	
ISZ	DISP (X)	100011bb PP	2	7/8	4/5	2	1
JMP	DISP (X)	000110bb PP	2	4	3	1	
JMP	@DISP (X)	100110bb PP	2	4	2	2	
JSR	DISP (X)	000101bb PP	2	5	4	1	
JSR	@DISP (X)	100101bb PP	2	5	3	2	
LD	r,DISP (X)	1100aabb PP	2	4	2	2	
LD	0,@DISP (X)	101000bb PP	2	5	2	3	
LI	r,DATA8	010100aa QQ	2	4	3	1	
LSEX	0,DISP (X)	101111bb PP	2	4	2	2	
OR	0,DISP (X)	101001bb PP	2	4	2	2	

Table 14-2. PACE Instruction Set Object Codes (Continued)

INSTRUCTION		OBJECT CODE	BYTES	MACHINE CYCLES			
				TOTAL	INTERNAL	INPUT	OUTPUT
PFLG	f	0011ffff 0xxxxxxx	2	6	5	1	
PULL	r	011001aa XX	2	4	3	1	
PULLF		000100xx XX	2	4	3	1	
PUSH	r	011000aa XX	2	4	3	1	
PUSHF		000011xx XX	2	4	3	1	
RADC	S,D	0011101aa eexxxxxx	2	4	3	1	
RADD	S,D	011010aa eexxxxxx	2	4	3	1	
RAND	S,D	010101aa eexxxxxx	2	4	3	1	
RCPY	S,D	010111aa eexxxxxx	2	4	3	1	
ROL	r,n,l	001000aa nnnnnnnl	2	5 + 3n	4 + 3n	1	
ROR	r,n,l	001001aa nnnnnnnl	2	5 + 3n	4 + 3n	1	
RTI		011111xx PP	2	6	5	1	
RTS		100000xx PP	2	5	4	1	
RXCH	S,D	011011aa eexxxxxx	2	6	5	1	
RXOR	S,D	010110aa eexxxxxx	2	4	3	1	
SFLG	f	0011ffff 1xxxxxxx	2	5	4	1	
SHL	r,n,l	001010aa nnnnnnnl	2	5 + 3n	4 + 3n	1	
SHR	r,n,l	001011aa nnnnnnnl	2	5 + 3n	4 + 3n	1	
SKAZ	0,DISP (X)	101110bb PP	2	5/6	3/4	2	
SKG	0,DISP (X)	100111bb PP	2	7/8	5/6	2	
SKNE	r,DISP (X)	1111aabb PP	2	5/6	3/4	2	
ST	r,DISP (X)	1101aabb PP	2	4	2	1	1
ST	0,@DISP (X)	101100bb PP	2	4	1	2	1
SUBB	0,DISP (X)	100100bb PP	2	4	2	2	
XCHRS	r	000111aa XX	2	6	5	1	

*All instructions may take additional cycles if Extend Read and Extend Write are implemented.

Table 14-3. Branch Conditions For PACE BOC Instruction

Condition Code (CC)	Mnemonic	Condition
0000	STFL	Stack Full (contains nine or more words).
0001	REQ0	(AC0) equal to zero (see Note 1).
0010	PSIGN	(AC0) has positive sign (see Note 2).
0011	BIT0	Bit 0 of AC0 true.
0100	BIT1	Bit 1 of AC0 true.
0101	NREQ0	(AC0) is nonzero (see Note 1).
0110	BIT2	Bit 2 of AC0 is true.
0111	CONTIN	CONTIN (continue) input is true.
1000	LINK	LINK is true.
1001	IEN	IEN is true.
1010	CARRY	CARRY is true.
1011	NSIGN	(AC0) has negative sign (see Note 2).
1100	OVF	OVF is true.
1101	JC13	JC13 input is true (see Note 3).
1110	JC14	JC14 input is true.
1111	JC15	JC15 input is true.

NOTES:

1. If selected data length is 8 bits, only bits 0 through 7 of AC0 are tested.
2. Bit 7 is sign bit (instead of bit 15) if selected data length is 8 bits.
3. JC13 is used by PACE Microprocessor Development System and is not accessible during prototyping.

THE BENCHMARK PROGRAM

For PACE, our standard benchmark program adopts this modified form:

```
        LD      2,IOBUF    LOAD I/O BUFFER ADDRESS INTO AC2
        LD      0,@TABLE   LOAD ADDRESS OF FIRST FREE TABLE BYTE
        RCPY    0,3        MOVE TO AC3
LOOP    LD      0,0 (2)    LOAD NEXT BYTE FROM I/O BUFFER
        ST      0,0 (3)    STORE IN NEXT TABLE BYTE
        AISZ    2,1        INCREMENT AC2
        AISZ    3,1        INCREMENT AC3
        DSZ     IOCNT      DECREMENT I/O BUFFER LENGTH. SKIP IF ZERO
        JMP     LOOP       RETURN FOR MORE BYTES
        RCPY    3,0        MOVE AC3 CONTENTS TO AC0
        ST      0,@TABLE   RESTORE ADDRESS OF FIRST FREE TABLE BYTE
```

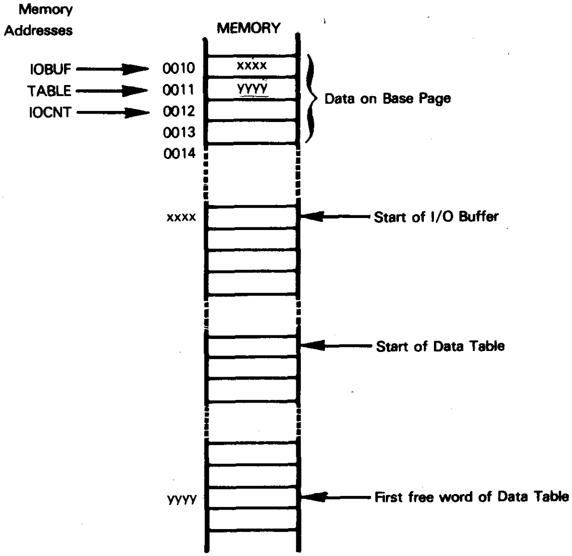
In order to take advantage of PACE's indirect addressing, three memory locations are reserved on page 0 as follows:

IOBUF   holds the beginning address of the I/O buffer.

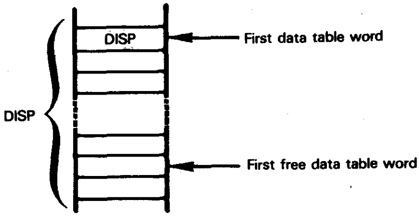
TABLE   holds the address of the first free byte in the permanent data table.

IOCNT   holds the number of data words in the I/O buffer.

Memory, as organized for the PACE benchmark program will look like this:



Suppose the benchmark program rules arbitrarily force the address of the first free data table byte to be computed as described for the Fairchild F8. A displacement must be stored in the first word of the data table, and this displacement must be added to the address of the first word of the data table, in order to compute the address of the first free data table word:



Now the PACE instructions:

```
        LD       0,@TABLE    LOAD ADDRESS OF FIRST FREE TABLE BYTE
        RCPY     0,3         MOVE TO AC3
```

must be replaced by these instructions:

```
        LD       3,TABLE     LOAD BEGINNING ADDRESS OF DATA TABLE
        LD       0,0 (3)     LOAD DISPLACEMENT TO FIRST FREE TABLE WORD
        RADD     0,3         ADD DISPLACEMENT TO AC3
```

The new displacement must be restored to the first data table word. The PACE instructions:

```
RCPY    3,0         MOVE AC3 CONTENTS TO AC0
ST      0,@TABLE    RESTORE ADDRESS OF FIRST FREE TABLE BYTE
```

must be replaced by these instructions:

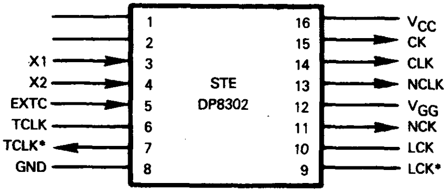
```
LD      0,TABLE     LOAD BEGINNING ADDRESS OF DATA TABLE IN AC0
CAI     0,1         FORM TWOS COMPLEMENT
RADD    0,3         SUBTRACT AC0 FROM AC3 TO FORM DISPLACEMENT
RCPY    3,0         MOVE DISPLACEMENT TO AC0
LD      3,TABLE     LOAD BEGINNING ADDRESS OF DATA TABLE IN AC3
ST      0,0 (3)     SAVE DISPLACEMENT IN FIRST FREE TABLE WORD
```

Forcing a PACE programmer to conform to programming logic suited to some other microcomputer's instruction set only proves that the two microcomputers have different instruction sets.

Observe that changes demanded of PACE in order to conform to the F8 ground rules will make the PACE benchmark program look quite poor. On the other hand, were the PACE ground rules imposed on the F8, the situation would be totally reversed. For example, PACE ground rules could specify that all tables may be more than 256 bytes long. This specification would have no impact whatsoever on the PACE benchmark program, but it would probably double the size of the F8 microcomputer program.

THE PACE DP8302 SYSTEM TIMING ELEMENT (STE)

The STE is a very elementary clock device; it accepts inputs from an external crystal and generates the MOS clock signals for PACE, plus a pair of TTL level clock outputs that can be used for synchronizing system operations. Figure 14-16 illustrates the pin assignments of the STE.



PIN NAME	DESCRIPTION	TYPE
X1, X2	External crystal connections	Input
CLK, NCLK	Damped MOS clocks to PACE	Output
CK, NCK	Undamped MOS clocks to PACE	Output
TCLK, TCLK*	TTL clocks to microcomputer system	Output
EXTC	External oscillator option	Input
LCK, LCK*	Non-overlap capacitor connection	
V_{CC}, V_{GG}	Power and Ground	

Figure 14-16. DP8302 System Timing Element (STE) Pins and Signals

STE CLOCK FREQUENCY

The frequency of the MOS clocks output by the STE is one-half the input crystal frequency. The STE is designed to operate with a 2.6667 MHz crystal. The MOS clock frequency is thus 1.3333 MHz which results in a clock period (tp) of 750 nanoseconds (tp = 1/f); this is the optimal clock period for the PACE CPU.

Two pairs of MOS clock outputs are generated by the STE; NCLK/NCLK* and NCK/NCK*. The first pair of outputs contain a 25Ω series damping resistor; typically, these outputs will be used in circuit board layouts where the STE-to-PACE interconnect lines are less than two inches. The other MOS outputs, NCK and NCK*, are undamped and you can select some other value of series damping resistors that might be better suited for your particular board layout.

GENERATING THE PACE SUBSTRATE BIAS VOLTAGE

In addition to the +5V and -12V power supplies typically needed with MOS devices, the PACE CPU has a third power supply requirement; a substrate bias voltage (V_{BB}) of +8V must be applied to the CPU chip. Since it is unlikely that any other devices in your microcomputer system would require this voltage level, the need for a third external system power source can be eliminated by providing a voltage converter circuit. **Figure 14-17 shows a circuit that generates the required V_{BB} voltage level;** the circuit requires only a few components and uses one of the STE's TTL clock outputs as a 'charge pump' for the circuit.

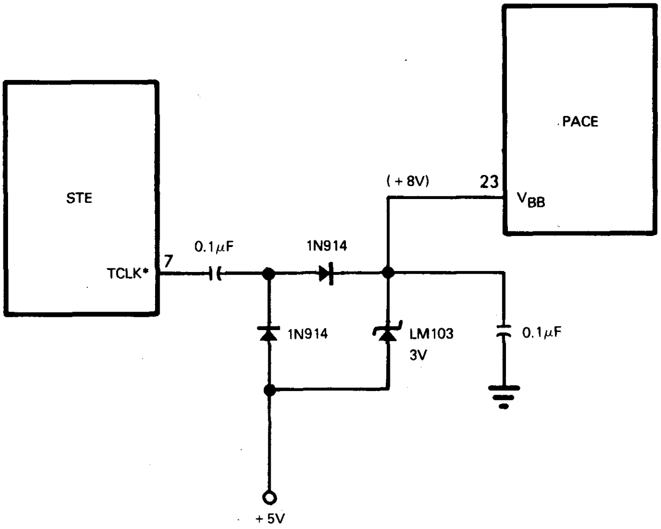
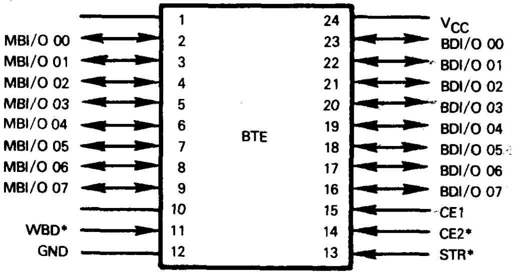


Figure 14-17. Circuit To Generate Substrate Bias Voltage (V_{BB}) For PACE CPU

THE PACE BIDIRECTIONAL TRANSCEIVER ELEMENT (BTE)

The DP8300 BTE is an 8-bit device that provides an interface between the PACE MOS-level signals and the TTL-level signals required by other devices in a microcomputer system. If you refer back to Figure 14-1 at the beginning of this chapter, you will see that a typical PACE microcomputer system requires three BTEs: two are used to buffer the CPU's 16 address/data lines, and the third is used as a TTL driver for the CPU's control signal outputs (NADS, ODS, IDS, F11 - F14).

Figure 14-18 shows the pin assignments for the BTE.



PIN NAME	DESCRIPTION	TYPE
MBI/O 00 - 07	MOS Bus Data Lines	Input/Output
BDI/O 00 - 07	TTL Bus Data Lines	Input/Output
CE1, CE2*, STR*, WBD*	Mode Control Signals	Input
V_{CC}, GND	+ 5V Power and Ground	

Figure 14-18. BTE Signals And Pin Assignments

Table 14-4 summarizes the operating modes of the BTE.

BTE MODE CONTROL SIGNALS

WBD* is the main mode control signal; when this signal is low, the other control signals are ignored and the BTE simply converts the MOS signals from the CPU into TTL-level output signals. The TTL outputs have a high fan-out capability and can service up to thirty 50 milliampere loads. **The BTE used to buffer the PACE control signals normally operates continuously in this 'drive-only' mode (Mode 1) and is kept in this mode by simply connecting the WBD* signal to ground.**

The BTEs used to buffer bidirectional (address/data) lines must be switched back and forth between Modes 1 and 2; Mode 1 is used for CPU data output and Mode 2 for CPU data input. The simplest way of accomplishing this is to continuously enable the CE1, CE2*, and STR* controls by connecting them to appropriate logic levels (+5V or ground) and then use the WBD* signal for directional control. For example, in a PACE system, the IDS signal from the CPU could be used as the input to WBD*. During a PACE data input cycle, IDS will go high at the appropriate portion of the cycle and place the BTE in Mode 2; IDS is low at all other times and the BTE will operate in Mode 1.

Table 14-4. PACE BTE Truth Table

MODE #	CONTROL INPUTS				MODE DESCRIPTION
	CE1	CE2*	STR*	WBD*	
1	X	X	X	0	Receive MOS signals and drive TTL signals
2	1	0	0	1	Receive TTL signals and drive MOS signals
3	0	0	0	1	Outputs in high-impedance state
	0	1	0	1	
	1	1	0	1	
4	X	X	1	1	On positive-edge transition of STR*, latch into Mode 2 or 3 as determined by state of CE1 and CE2*

X = don't care

In a DMA or multiprocessor we will need to use BTE Mode 3 to place the BTE outputs in a high-impedance state and thus free the System Busses for use by other devices. In such a system an externally generated Bus Grant signal could be used to place the BTE in Mode 3. Figure 14-19 illustrates one method of doing this: whenever the BUS GRANT signal is high, the BTE is in Mode 3. At other times the IDS signal operates as we've just described to switch the BTE back and forth between Modes 1 and 2.

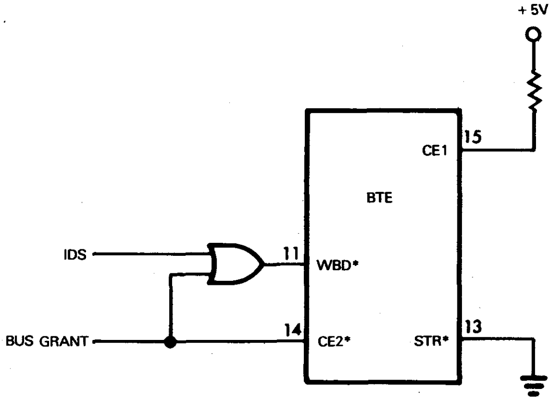


Figure 14-19. Signal Connections To Control BTE In A DMA System

The fourth BTE mode uses a negative-to-positive transition on the STR* input to latch the state of CE1 and CE2*, and then places the BTE in either Mode 2 or Mode 3. This latch mode function might be useful when the BTE is used as a simple input buffer. For example, in a system with multiplexed address/data lines (such as PACE), address outputs could be applied to CE1 and CE2* and an address strobe signal (such as NADS) connected to STR*. Then, when the BTE is selected by the appropriate address bits, the trailing edge of the strobe signal would gate TTL data through the BTE and apply the data to the MOS lines of the CPU. When the BTE is not selected (addressed), its outputs would be in the high impedance state (Mode 3).

THE DP8301 MICROPROCESSOR INTERFACE LATCH ELEMENT (MILE)

The DP8301 MILE is the most complicated of the limited family of PACE support devices, although it is still quite simple when compared to support devices available in other microcomputer families. Nonetheless, **the MILE is quite versatile; it is useful and cost-effective in any microcomputer system where a bidirectional I/O port is required.**

The MILE device consists of a data latch, bidirectional tristate buffers, device and mode select logic, and status flags for handshaking or interrupt logic. The data latch consists of eight D-type flip-flops. The inputs to the flip-flops can be either from the CPU (Bus) side or the peripheral side of the I/O port. The Q outputs of the flip-flops are connected to two tristate, noninverting buffers; one buffer can be enabled to drive the CPU (Bus) data lines, and the other buffer can drive the peripheral data lines. The device and mode select logic determines the source of data to be used as inputs to the flip-flops and also enables the desired output buffer(s). The status flags indicate to external logic what the contents of the flip-flops consist of. **Figure 14-20 is a block diagram representation of the MILE.**

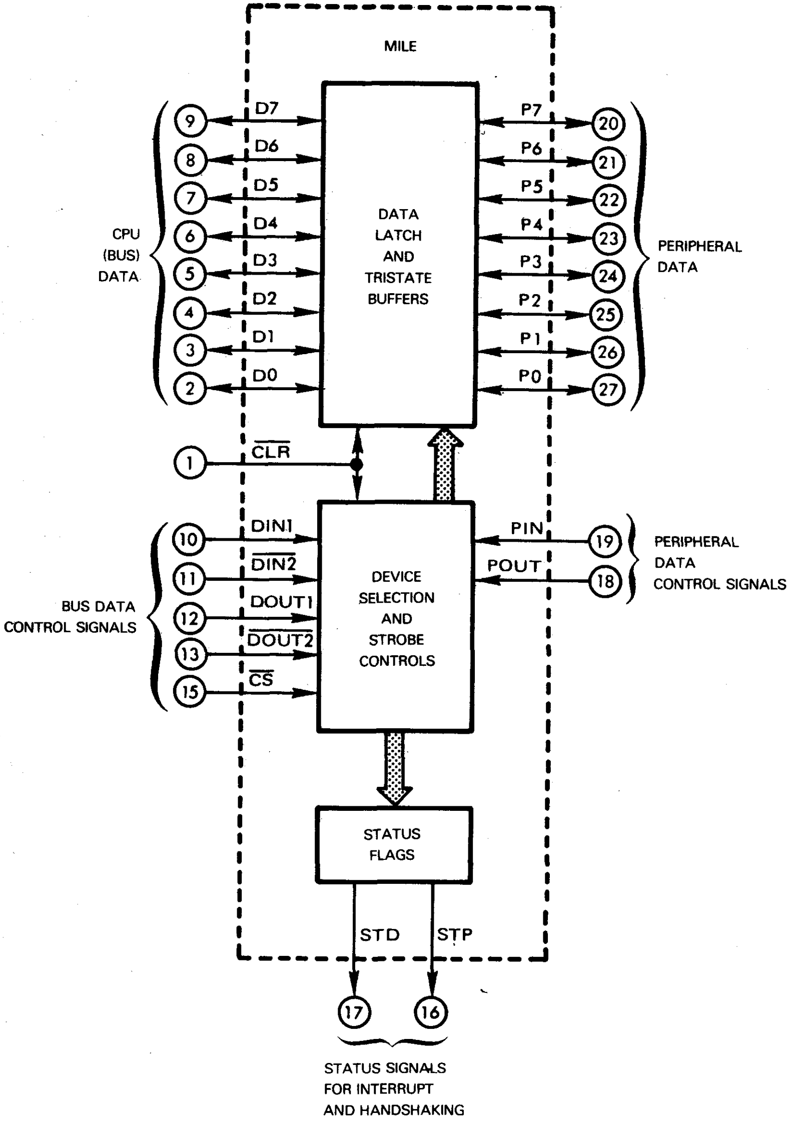
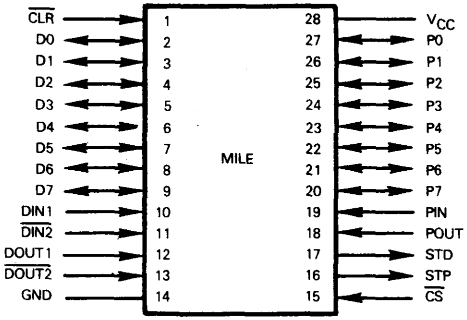


Figure 14-20. Block Diagram Of PACE DP8301 MILE

MILE PINS AND SIGNALS

DP8301 MILE pins and signals are illustrated in Figure 14-21. We will describe these pins and signals with reference to Figure 14-20, and to Table 14-5 which defines the available modes of operation for the MILE.



PIN NAME	DESCRIPTION	TYPE
D0 - D7	CPU (Bus) Data	Input/Output
P0 - P7	Peripheral Data	Input/Otput
DIN1, $\overline{\text{DIN2}}$	CPU (Bus) Data Input Mode Controls	Input
DOUT1, $\overline{\text{DOUT2}}$	CPU (Bus) Data Output Mode Controls	Input
PIN	Peripheral Data Input Mode Control	Input
POUT	Peripheral Data Output Mode Control	Input
STD	CPU (Bus) Data Status Signal	Output
STP	Peripheral Data Status Signal	Output
$\overline{\text{CS}}$	Device Select	Input
$\overline{\text{CLR}}$	Device Clear	Input
V_{CC}	Power	
GND	Ground	

Figure 14-21. MILE Signals And Pin Assignments

D0 - D7 are data input/output pins that are used by the CPU to write data into and read data out of the MILE. On a write (Mode 1) operation, these lines are applied to the eight data inputs of the D-type flip-flops that comprise the Data Latch within the MILE. On a read (Mode 2) operation, the Q outputs of the flip-flop are gated through tristate buffers and output via pins D0 - D7.

P0 - P7 are data input/output pins used by a peripheral device to write data into the MILE (Mode 3) or read data out of the MILE (Mode 4). It is important to note that the MILE provides only one Data Latch, essentially a set of eight D-type flip-flops. There are no separate latches for CPU data and peripheral data. Instead, the MILE control signals select which data lines (D0 - D7 or P0 - P7) will be applied to the D inputs of the Data Latch and/or Q outputs of the Data Latch. Because of this, some rules must be established to prevent conflicts. As we shall see shortly, the MILE Mode Control and Device Select logic establishes these rules and resolves potential conflicts with a built-in priority scheme. We'll discuss these rules in detail after we've finished describing the remaining MILE signals.

$\overline{\text{CS}}$ is the chip select signal and must be low for any data transfers to occur on the CPU side of the MILE. Notice that data transfers can occur on the peripheral side of the device regardless of the state of the $\overline{\text{CS}}$ signal. This may seem a bit strange at first, but it is actually quite logical. Recall that a CPU may have many I/O ports or may even treat the MILE simply as a memory location.Thus the input to the $\overline{\text{CS}}$ pin would typically consist of an address bit or a decoded device select signal. A CPU data transfer

Table 14-5. MILE Mode Control Truth Table

MODE #	DIN1	$\overline{\text{DIN2}}$	$\overline{\text{CS}}$	DOUT1	$\overline{\text{DOUT2}}$	PIN	POUT	MODE DESCRIPTION
1	1	0	0	X	X	X	X	Load CPU (Bus) Data into Data Latch
2	0	X	0	1	0	X	X	Gate contents of Data Latch onto CPU (Bus) lines
	X	1	0	1	0	X	X	
3	0	X	X	X	X	1	X	Load Peripheral Data into Data Latch
	X	1	X	X	X	1	X	
	X	X	1	X	X	1	X	
4	X	X	X	X	X	0	1	Gate contents of Data Latch onto Peripheral Data Lines
	1	0	0	X	X	X	1	

input output in out

CPU (Bus) Data Controls

Peripheral Data Controls

would occur only when the CPU specifically addresses the MILE. In contrast to this, the peripheral side of the MILE will typically be dedicated to servicing a simple peripheral device and therefore no addressing or select function would be necessary.

DIN1 and $\overline{\text{DIN2}}$ must be high and low, respectively, to enable the contents of the D0 - D7 lines to be loaded into the Data Latch.

DOUT1 and $\overline{\text{DOUT2}}$ must be high and low, respectively, to enable the contents of the Data Latch to be output on the D0 - D7 lines.

PIN must be high to enable the contents of the P0 - P7 lines to be loaded into the Data Latch.

POUT must be high to enable the contents of the Data Latch to be output on the P0 - P7 lines.

STD and STP are status flag output signals that can be used for handshaking or interrupt logic. STD goes high when the Data Latch is loaded from the D0 - D7 lines and goes low when the contents of the Data Latch are gated out onto the P0 - P7 lines. STP works in just the opposite way: it goes high when the Data Latch is loaded from the peripheral side (P0 - P7) and goes low when that data is output to the opposite side (D0 - D7).

CLR is the reset input signal. When it goes low it clears the Data Latch and sets STD and STP low.

MILE DEVICE SELECT AND MODE CONTROL LOGIC

Table 14-5 summarizes the functions of the seven control signal inputs to the MILE and the resultant operational modes that are produced. There are a couple of non-obvious aspects of this table. First, there is a definite priority structure inherent in the mode control logic; second, the MILE can operate in two modes simultaneously. To make this more clear, we will simplify the table. Let us combine the DIN1, $\overline{\text{DIN2}}$ and $\overline{\text{CS}}$ signals into a single control which we'll call DIN. We can do this since all three of these signals must be true at the same time in order to effect the desired mode. Let us do the same thing with DOUT1, $\overline{\text{DOUT2}}$ and $\overline{\text{CS}}$ and call the resultant control signal DOUT. **So now, DIN=DIN1·$\overline{\text{DIN2}}$·$\overline{\text{CS}}$ and DOUT=DOUT1·$\overline{\text{DOUT2}}$·$\overline{\text{CS}}$, and our truth table looks like this:**

MODE #	DIN	DOUT	PIN	POUT	FUNCTION
1	1	X	X	OK	Input to Latch from D0 - D7
2	0	1	OK	OK	Output from Latch to D0 - D7
3	0	OK	1	X	Input to Latch from P0 - P7
4	OK	OK	X	1	Output from Latch to P0 - P7

In this simplified table we've used the following conventions:

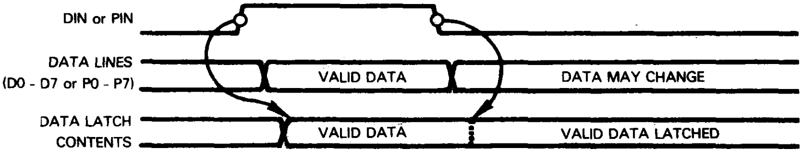
- A '1' means that the indicated signal must be true
- A '0' means that the signal must be false
- An 'X' means the signal is simply ignored
- An 'OK' means that the signal (and its associated function) can occur without interfering with the primary mode. However, only one secondary mode can be implemented at a time.

Now we can see that when DIN is true, the MILE is in Mode 1; Modes 2 (DOUT) and 3 (PIN) are disallowed since they would interfere with Mode 1 operation. Mode 4 (POUT) is permitted to occur at the same time as Mode 1 since there is no conflict. Here is what would happen: DIN would load the contents of D0 - D7 into the Data Latch, and POUT would gate the contents of the Data Latch out onto the P0 - P7 lines. Thus, the MILE would effectively be operating as a flow-through latch. Similar non-conflicting simultaneous functions are allowed in all four modes.

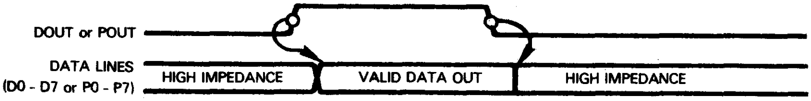
MILE TIMING CONSIDERATIONS

In the preceding paragraphs, we combined some of the MILE control signals and ended up with just four controlling inputs: DIN, DOUT, PIN and POUT. We will use these same four signals for a bit longer as we describe timing considerations for MILE input and output operations.

The DIN and PIN signals are used as the clock input to the Data Latch. When one of these signals is high, the contents of the Data Latch follow the selected data input lines (D0 - D7 or P0 - P7). When DIN or PIN returns low, the Data Latch will retain the input information that was present at the time of the clock transition. Thus, timing for a data input or write operation looks like this:



A data output or read operation is controlled by the DOUT or POUT signal. The timing is extremely simple since it only involves enabling the outputs of tristate buffers.

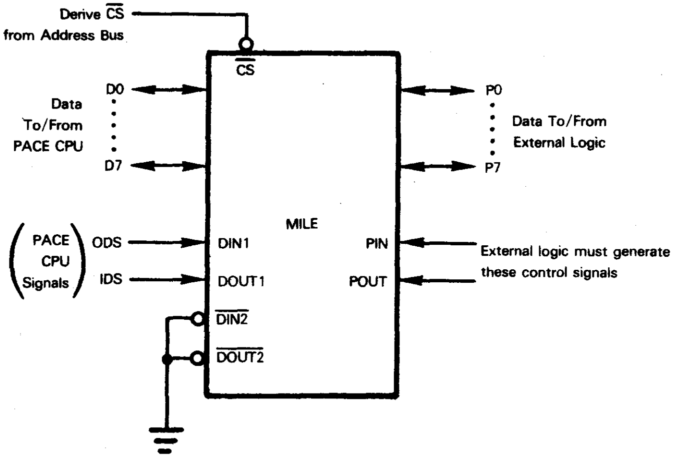


The contents of the Data Latch are simply gated out through tristate buffers onto the D0 - D7 or P0 - P7 (or both) as long as the DOUT or POUT signal is high. The outputs of the data lines are kept in the tristate high impedance mode at all other times.

After these discussions where we've treated several control signals as though they were a single control input, you may be beginning to wonder why so many control signals are provided: the same control obviously could be obtained with just a few signals. The answer is that multiple control signals make the CPU-I/O port interface easier to implement; therefore, **the MILE can be used easily with many different microcomputers. Let us present a few examples to demonstrate this flexibility.**

First, let us see how the MILE might be interfaced to a PACE CPU.

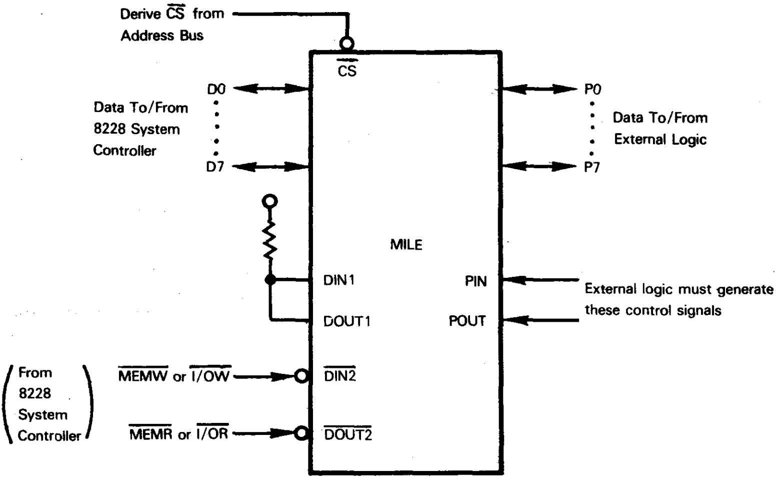
MILE USED WITH PACE CPU



Tie $\overline{DIN2}$ and $\overline{DOUT2}$ to Ground. Now ODS and IDS from PACE CPU provide mode control whenever $\overline{CS}$ is low.

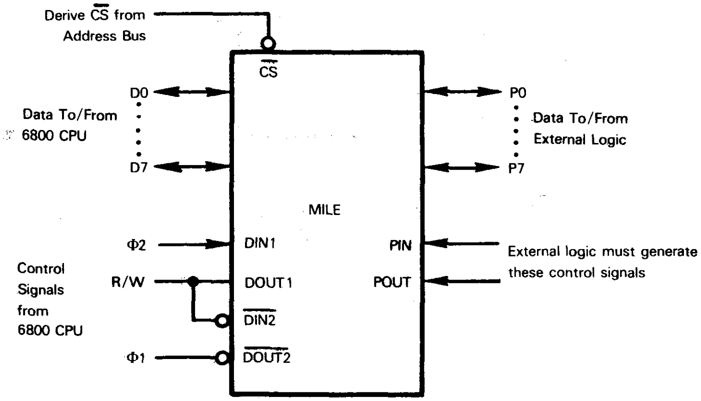
In this example two control inputs, $\overline{DIN2}$ and $\overline{DOUT2}$, are not used at all. In some PACE systems, however, we might connect these inputs to selected address bits and thus reduce the amount of address decoding logic required in the system.

Now let us look at how the MILE might be used in an 8080 system.


Derive $\overline{\text{CS}}$ from Address Bus
$\overline{\text{CS}}$
D0
D7
Data To/From 8228 System Controller
P0
P7
Data To/From External Logic
MILE
DIN1
DOUT1
PIN
POUT
External logic must generate these control signals
From 8228 System Controller
$\overline{\text{MEMW}}$ or $\overline{\text{I/OW}}$
$\overline{\text{MEMR}}$ or $\overline{\text{I/OR}}$
$\overline{\text{DIN2}}$
$\overline{\text{DOUT2}}$

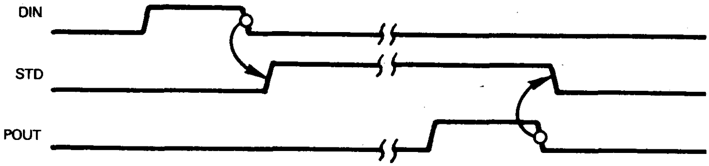
We see that the only difference between the 8080 and PACE usage of the MILE is that the polarity of the I/O control signals is reversed. Because the MILE provides both positive- and negative-true control inputs we don't have to use any inverters to obtain input signals of the required polarity.

For our final example we'll show how the MILE can be used in a 6800 system. In this example all of the control inputs to the MILE are needed.

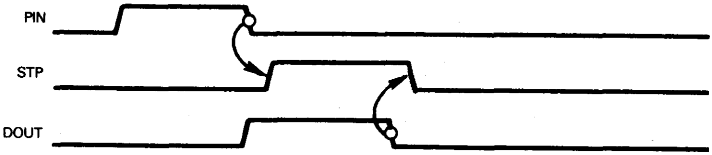

Derive $\overline{\text{CS}}$ from Address Bus
$\overline{\text{CS}}$
D0
D7
Data To/From 6800 CPU
P0
P7
Data To/From External Logic
MILE
Control Signals from 6800 CPU
Φ2
R/W
Φ1
DIN1
DOUT1
$\overline{\text{DIN2}}$
$\overline{\text{DOUT2}}$
PIN
POUT
External logic must generate these control signals

MILE STATUS SIGNALS

The MILE outputs two status signals that can be used to implement very simple I/O handshaking or interrupt schemes. The two signals (STD and STP) operate in exactly the same way: they are set high by a write operation, that is, when the MILE data latch is loaded with new data. The status signal output remains high until the data is read out of the MILE from the opposite side. To illustrate the timing for the status signals, let us once again combine the seven MILE control signals into the following four signals: DIN, DOUT, PIN, and POUT. Now the timing for the STD looks like this:



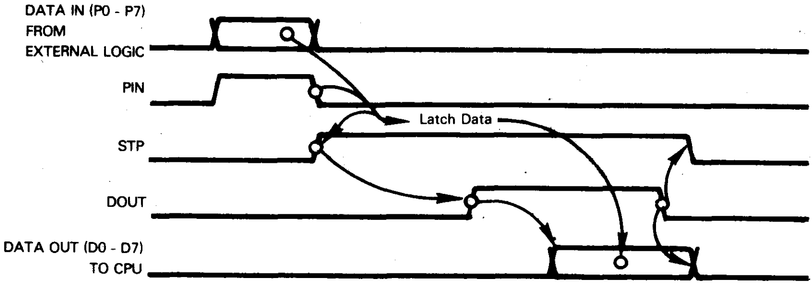
The timing for the STP is the same:



We've drawn these two timing diagrams so that they differ slightly from each other to illustrate a point: the write and read operations can overlap as we've shown for the STP timing. Notice that the DOUT signal is shown occurring while PIN is still high. You will recall from our earlier discussion that the MILE permits this since PIN and DOUT are non-conflicting operations. Remember also that we stated that in most applications the MILE will operate in only one mode at a time: a closer look at the timing for STP lends credence to that statement. The timing we've shown is not a very realistic presentation of the use of the status signals since the read operation (in this case DOUT) is shown being initiated before the status signal has been set by the trailing edge of the write operation (PIN). This implies that the CPU had some method, other than monitoring the status signal, of detecting when external logic had data ready for input to the CPU. Let us now look at some realistic ways in which the MILE status signals can be used.

MILE STATUS SIGNALS USED FOR INPUT WITH HANDSHAKING

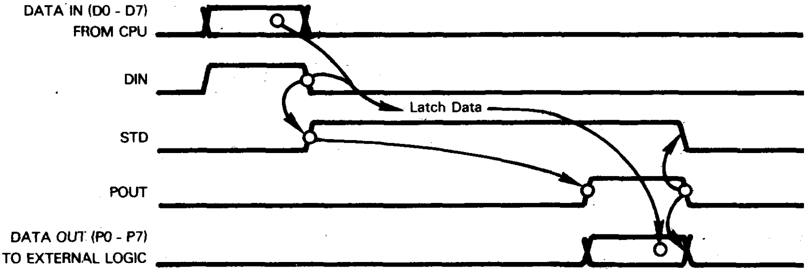
When the MILE is being used as an input port, we are primarily concerned with the STP status signal. Input timing may be illustrated as follows:



Notice that the STP signal does not go high until external logic has completed loading the data into the MILE: the trailing edge of PIN causes STP to be set high. Therefore, STP is a handshaking signal that can be used to signal the CPU that new data is in the input port and ready to be read. DOUT identifies an instruction that selects the MILE and reads the contents of the data latch. Thus, while STP is high, there is unprocessed data in the MILE. External logic must not input new data while STP is high; if it does, prior unprocessed data will be destroyed.

MILE STATUS SIGNALS USED FOR OUTPUT WITH HANDSHAKING

When using the MILE as an output port, we will concern ourselves primarily with the state of the STD signal. The timing is simply the mirror image of the input timing and may be illustrated as follows:



As illustrated above, when the STD signal goes high, it indicates to external logic that the CPU has loaded new data into the MILE and it is ready to be read. As long as STD remains high, the CPU should not output more data to the MILE since it would destroy the unprocessed data stored in the latch. When external logic has read the data out of the MILE, the STD signal goes low. This high-to-low transition could therefore be used as an interrupt input to the CPU to indicate that additional data can be transmitted to the output port.

One final note on the MILE and its status signals. As we have discussed earlier, the MILE logic prevents conflicting operations such as DIN and PIN from occurring simultaneously. Data can only be loaded into the MILE from one side at a time since the device provides just one data latch. If the MILE is being used as both an input port and an output port, however, there is nothing within the logic of the device that would prevent a DIN operation from being followed immediately by a PIN operation. This, of course, would normally be an error condition since the data loaded into the MILE by the CPU (DIN) would have been destroyed by the data from external logic (PIN).

Typically, in a system where the MILE is being used as a bidirectional port, the CPU and external logic would each be monitoring both the STP and STD signals. The data overrun condition we've just mentioned would then be prevented by the system input/output protocols you have established for the system. If the overrun condition should occur, both STP and STD would be high simultaneously and this condition could be used to generate a system error signal.

USING OTHER MICROCOMPUTER SUPPORT DEVICES WITH THE PACE CPU

As we have seen, there are only three support devices (STE, BTE, and MILE) for the PACE CPU, and two of these devices (the STE and BTE) are so vital to the implementation of a PACE system that they might well be considered an integral part of the CPU itself. Indeed, the functions provided by these two devices are provided on the CPU chip for some of the microcomputers described in this book.

But the PACE CPU has numerous control signals which allow general-purpose microcomputer support devices to be included in a PACE system.

MILE COMPARED TO 8080 FAMILY I/O PORTS

Since the MILE is the only true general-purpose device provided as a PACE support device, we will begin our discussion by comparing the capabilities of the MILE to those of other I/O ports.

In complexity and capabilities, the MILE falls somewhere between the 8212 I/O port and the 8255 Programmable Peripheral Interface (PPI) which were described in Chapter 4 of this book.

Like the 8212, the MILE provides a latched 8-bit port: but, while an 8212 can be used only as an input port or an output port, the MILE can operate bidirectionally. So, it is nearly the equivalent of two 8212s. We say 'nearly' because the MILE contains a single 8-bit data latch which may be loaded from either the CPU side or the peripheral side: two 8212s wired back-to-back as we described in Chapter 4, would provide two 8-bit latches so that the input port and the output port could be loaded simultaneously. This apparent advantage, however, might be of little value in most applications; so, it is quite reasonable to equate the capabilities of a single MILE to those provided by two 8212 devices. Note, however, that when all you require is a simple input port or output port, the 8212 would be the more logical choice (disregarding other extenuating factors) since the MILE would then be providing more logic than is required.

The 8255 PPI, you will recall, can provide three 8-bit ports that can be utilized in a variety of combinations or modes, and a programmable control register that is used to select the desired mode. Obviously the 8255 provides more capabilities and greater flexibility than the MILE. However, if we specify that the 8255 must provide an 8-bit, bidirectional input/output port, the gap between the two devices is greatly reduced. In Mode 2, Port A of the 8255 operates bidirectionally. Port B operates either as a simple input port or output port, and Port C provides handshaking control signals. **Since the MILE provides approximately equivalent handshaking signals, the 8255 has a net advantage of one 8-bit input or output port over the MILE. Thus, with the restrictions we've imposed, one 8212 and one MILE would provide about the same capabilities as a single 8255.**

Now that we've compared capabilities, let us see how these and other 8080 devices might be used with the PACE CPU. First, we'll take an overview of the general CPU-to-device interface that all the 8080 family of devices expect.

All of the 8080 family devices require that address information (or enabling/select signals derived from the address lines) be valid during the data transfer (read/write) portion of an input/output cycle. Recall that the PACE data lines are multiplexed: at the beginning of an input/output cycle, the data lines are used to output address information; the address information is then removed and the data lines are used for the actual input or output of data during the latter portion of the I/O cycle.

DEMULTIPLEXING THE PACE ADDRESS/DATA LINES

Thus, the first thing we must do to interface PACE to an 8080 family device is to demultiplex the PACE address/data lines. This must also be done even with the MILE device which was specifically designed to operate with the PACE CPU. In fact, it will be required with almost any device, including most memory devices, that are interfaced to PACE. **There are several different approaches that we can use to accomplish the required demultiplexing.**

The most obvious way is to use D-type flip-flops or data registers with the PACE NADS signal as the clock pulse. Here are some of the standard 7400 family devices that might be used:

- 7475 Double 2-Bit Gated Latches with Q and Q Outputs
- 7477 Double 2-Bit Gated Latches with Q Output Only
- 74100 Double 4-Bit Gated Latches
- 74166 Dual 4-Bit Gated Latches with Clear
- 74174 Hex D-Type Flip-Flops with Common Clock and Clear
- 74175 Quad D-Type Flip-Flops with Common Clock and Clear

Some of these devices require that the NADS signal be inverted to provide the necessary clocking signal. Remember, though, that PACE address information is valid during both the leading edge (high-to-low transition) and trailing edge (low-to-high transition) of NADS: this generally simplifies the demultiplexing operation.

In many systems you will not need to latch all 16 bits of address information since it would be an unusual application that required all of the 64K of address space that this provides. There will usually be some tradeoff between system address requirements (how many system devices require a latched Address Bus) and the type and amount of address decoding required. When a fully latched Address Bus is provided, then simpler nonlatched address decoders can be used. In fact, often address bits can then be used directly as device select signals, or simple AND/OR gate combinations can perform the decoding.

The alternative method of demultiplexing the address/data lines is to use address decoding devices that are clocked by the NADS signal and provide latched outputs. These latched outputs can then be used as the device/chip select signals during I/O cycles.

Many systems will use some combination of a fully latched Address Bus and simple or latched address decoders. In the discussions that follow, we will not generally describe in detail the method used to obtain the required addressing or select/enabling signals, since the method used is so dependent on the particular system that you are designing.

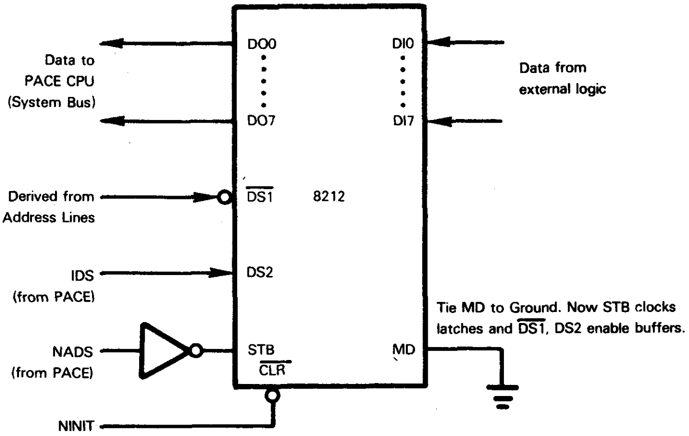
PACE CONTROL SIGNAL POLARITY CONSIDERATIONS

Once the PACE address/data lines have been demultiplexed, the only major considerations we are left with are to ensure that the input/output control signals are of the proper polarity, and to verify that there are no timing problems. We will see that generally the PACE I/O control signals must be inverted to operate with the 8080 family of devices, although the 8212 offers us a choice of using the PACE IDS and ODS signals in either their original or inverted form.

Now we will provide a few specific examples of how devices from the 8080 family can be used with the PACE CPU. In addition to assuming that required select/enabling signals are derived using some combination of address latches and decoders, we make one other assumption. The PACE CPU that we will be depicting consists not only of the CPU chip itself, but also includes an STE and three BTEs since, as we've described earlier in this chapter, these parts might well be *considered an integral part of the PACE CPU. Finally, our discussion of these* devices will be limited to conceptual treatments of how they might be interfaced to the PACE CPU. For detailed descriptions of the capabilities of these devices, and how they are interfaced to external logic, refer to Chapter 4 in this book.

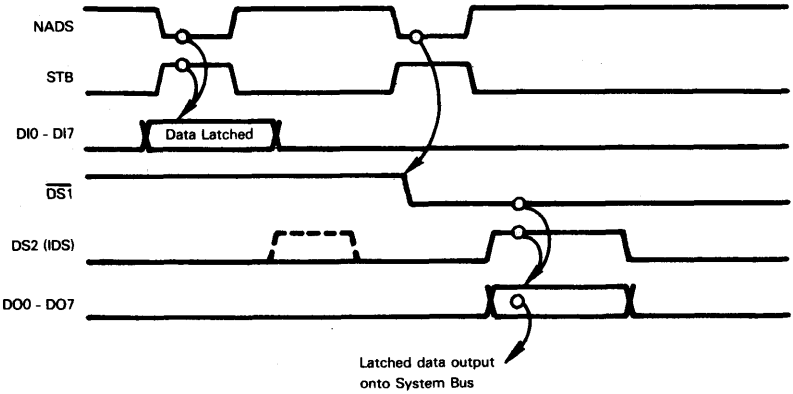
In our first example the 8212 I/O Port is used as a simple input port by the PACE CPU. The interconnections required are shown in the following figure:

THE 8212 USED AS A SIMPLE INPUT PORT IN A PACE SYSTEM



Data to PACE CPU (System Bus)
DO0
DO7
DI0
DI7
Data from external logic
Derived from Address Lines
DS1
8212
IDS (from PACE)
DS2
Tie MD to Ground. Now STB clocks latches and DS1, DS2 enable buffers.
NADS (from PACE)
STB
CLR
MD
NINIT

Here, the PACE Address Strobe signal (NADS) is inverted and used as the STB input to the 8212. Since MD is tied to ground, the STB signal clocks the data into the 8212: this will occur every time that PACE performs an input/output cycle, but the latched data will only be placed on the System Bus when the 8212 is selected. We accomplish device selection by applying a negative-true decoded address signal to the $\overline{\text{DS1}}$ input and then using the PACE IDS strobe signal as the DS2 input. Now, whenever the proper address is decoded, the IDS signal will cause the data that was previously latched by NADS to be placed on the System Bus for input to PACE. The timing would look like this:



Notice that the data from external logic will be latched whenever NADS occurs. The actual selection of the 8212 and input of the latched data to PACE might not occur for quite some time. Frequently, this arrangement will be completely acceptable. If not, then an input-with-handshaking arrangement which we will describe next, might provide a better solution.

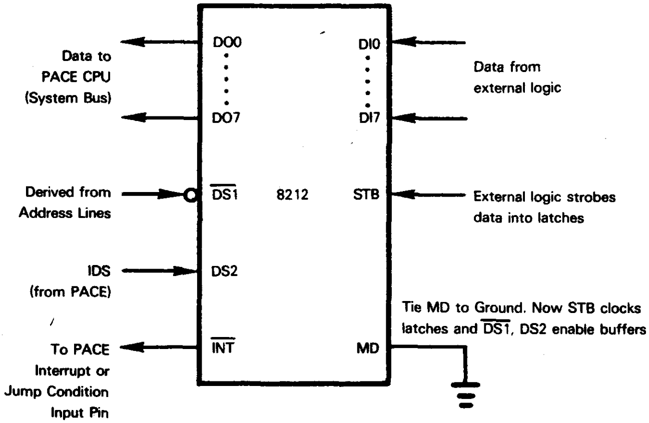
Before we proceed to our next example, let us make one more general comment about interfacing devices to the PACE CPU.

PACE is a 16-bit microcomputer: it can transfer 16 bits of parallel data in a single input or output cycle. All of the other devices that we will be discussing are 8-bit devices. Frequently, you may not need the full width of the 16-bit Data Bus when transferring data between the CPU and external logic. In these cases, you can simply connect the data lines to/from the support device to the less significant data lines (D0 - D7) of the PACE System Bus as we have shown in our first example. Masking of the unused, more significant data bits would then be handled under program control.

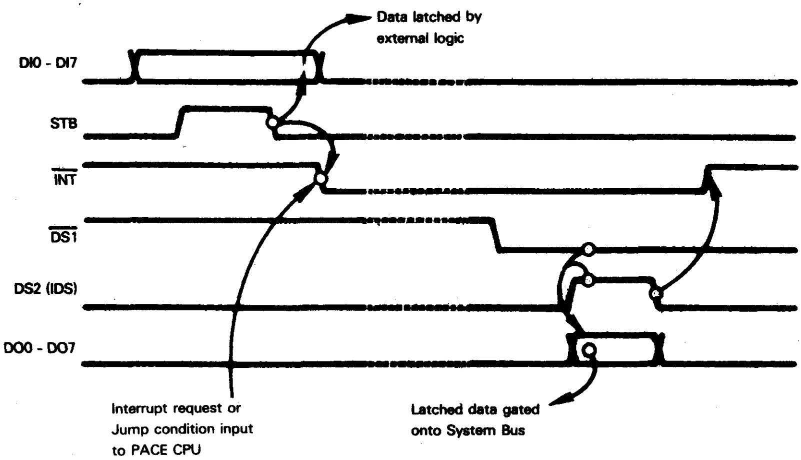
When you are going to utilize the full 16 bits of the Data Bus, you merely connect two 8-bit devices in parallel. One device would be connected as we've already described; the data lines of the other device would then be connected to the more significant bits (D8 - D15) of the System Bus. All other connections to the two devices (device select signals, strobe signals, etc.) would be identical.

THE 8212 USED IN A PACE SYSTEM FOR INPUT WITH HANDSHAKING

In this example, we will use the 8212 interrupt request signal $\overline{\text{INT}}$ to establish an input port with handshaking. The connection diagram is very similar to our first example:

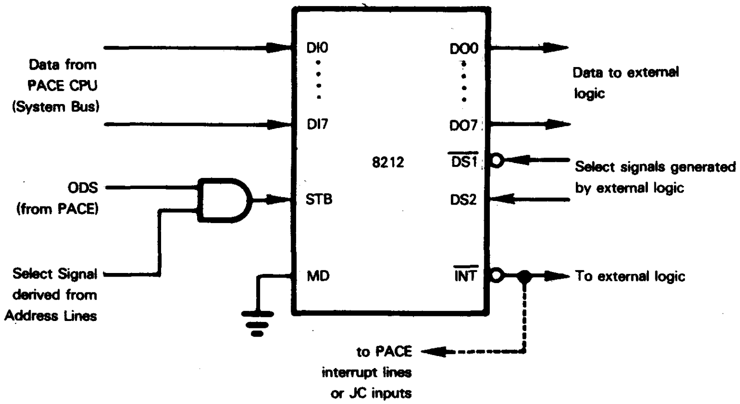


Here, the device select signals are the same as in our first example. However, instead of using the PACE NADS signal to clock data into the latches, we will require external logic to input the STB signal when it has data ready. When the data has been latched, the 8212 will output the $\overline{\text{INT}}$ signal which would be used as the input to one of the PACE CPU interrupt request lines (NIR2 - NIR5) or Jump Condition inputs (JC13 - JC15). The CPU would then execute a service routine program that would include an instruction to read the data from the input port. This instruction would send out the input port's address, thus generating the $\overline{\text{DS1}}$ signal, and then gate the latched data onto the System Bus when the IDS signal is generated. When the latched data is read out of the 8212, the INT signal returns high to complete the transaction. This sequence is summarized by the following timing diagram:


Data latched by external logic
DI0 - DI7
STB
$\overline{\text{INT}}$
$\overline{\text{DS1}}$
DS2 (IDS)
DO0 - DO7
Interrupt request or Jump condition input to PACE CPU
Latched data gated onto System Bus

Using the 8212 as an output port in a PACE system requires a simple reversal of the connections we have described in the two preceding examples, and we will now use the ODS (Output Data Strobe) signal from PACE instead of the IDS signal.

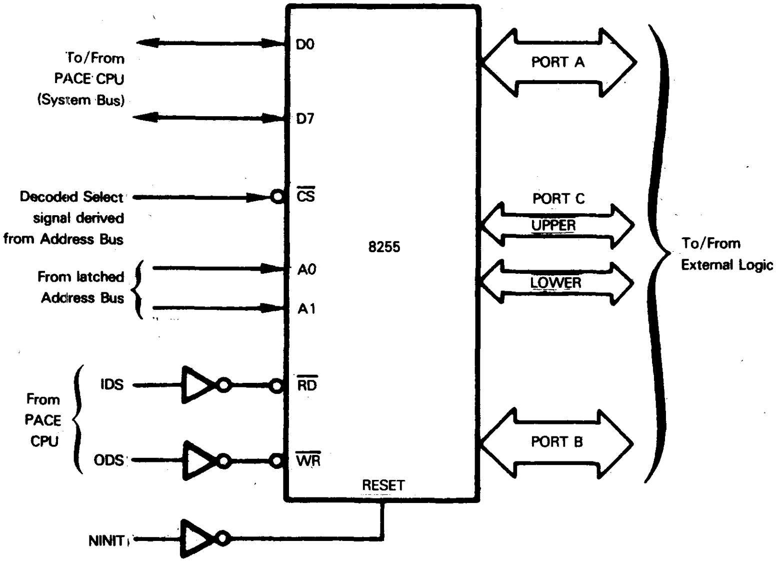
THE 8212 USED AS AN OUTPUT PORT IN A PACE SYSTEM


Data from PACE CPU (System Bus)
DI0
DI7
8212
DO0
DO7
Data to external logic
$\overline{\text{DS1}}$
DS2
Select signals generated by external logic
ODS (from PACE)
STB
Select Signal derived from Address Lines
MD
$\overline{\text{INT}}$
To external logic
to PACE interrupt lines or JC inputs

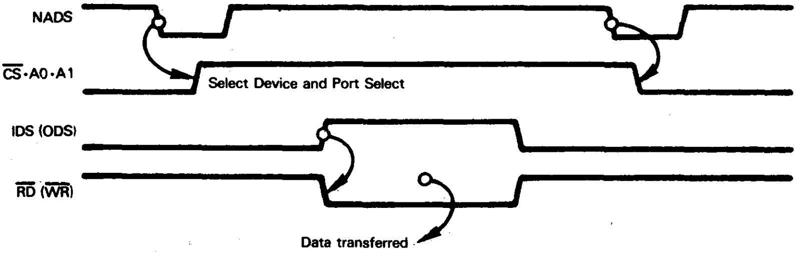
When the output port's address is sent out and decoded from the Address Bus, one input to the AND gate is enabled. The ODS signal then goes high to generate the STB signal and latch the contents of the system Data Bus into the 8212. This will cause the $\overline{\text{INT}}$ signal to go low and inform external logic that data has been loaded into the output port. The external logic would then generate the $\overline{\text{DS1}}$ and DS2 signals to gate the data out of the latches. When the data has been gated out, the $\overline{\text{INT}}$ signal will return high. This low-to-high transition could be used as an interrupt request or jump condition input to PACE to enable output of new data. **Notice that if we continuously enable the 8212 outputs by tying $\overline{\text{DS1}}$ to ground and DS2 to +5V, then whenever PACE loads a new data word into the latch, it would be immediately output to external logic.** This approach may be more advantageous in some applications.

8255 PPI DEVICES USED IN A PACE SYSTEM

Although the 8255 Programmable Peripheral Interface (PPI) is a more complicated device than the 8212, interfacing the 8255 to a PACE CPU is no more complicated (from a hardware point of view) than the PACE-to-8212 interfaces we've described. This is due to the programmability of the 8255; mode control is performed by your program instead of by hardwired signals. Let us look at an example to illustrate this point:



The $\overline{CS}$ signal selects the 8255 and this signal would typically be the output of an address decoder. The A0 and A1 inputs select one of the three I/O ports (A, B or C) or the 8255 Control registers. The $\overline{RD}$ and $\overline{WR}$ control signals are obtained by simply inverting the IDS and ODS signals from PACE. A generalized timing diagram for input/output operations would look like this:



If you refer back to the detailed description of the 8255 in Chapter 4, you will see that Port C can be used to provide handshaking signals for I/O control. Since these signals are fully described in Chapter 4, we will not discuss the various possibilities here. Generally, these signals would be used with the PACE CPU in the same ways that we earlier described for the 8212 $\overline{INT}$ signal.

TWO 8255 DEVICES USED FOR 16-BIT I/O PORTS WITH PACE

If two 8255s are used in parallel to provide 16-bit I/O ports, there is one special consideration beyond the general rules that we discussed earlier. Recall that mode control of the 8255 is accomplished by writing data into one 8-bit Control register within the device. When wired in parallel, one 8255 would be connected to bits 0 - 7 of the system Data Bus, and the other 8255 would be connected to bits 8 - 15. Therefore, when we send out a 16-bit control word from the PACE CPU to establish the desired mode of operation, the upper and lower bytes of the word must be identical.

THE 8251 USART AND 8253 PROGRAMMABLE COUNTER/TIMER USED IN PACE SYSTEMS

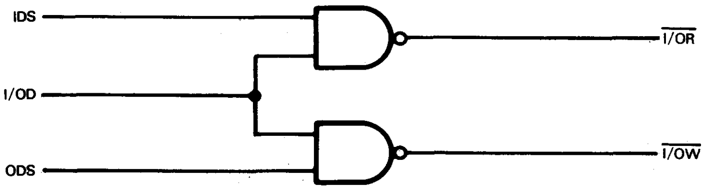
From a hardware point of view, interfacing either of these devices to a PACE CPU is no different than interfacing an 8255 PPI to PACE. All we need to do is invert the IDS and ODS signals from the CPU to obtain $\overline{RD}$ and $\overline{WR}$ (or $\overline{IOR}$ and $\overline{IOW}$) signals, and provide chip select and latched address bits for input to the devices. All other interfacing and usage considerations are software functions and are described in Chapter 4. We will not describe them here since those portions of the device descriptions apply regardless of the CPU being used.

PACE AND 8080 SYSTEM BUSSES COMPARED

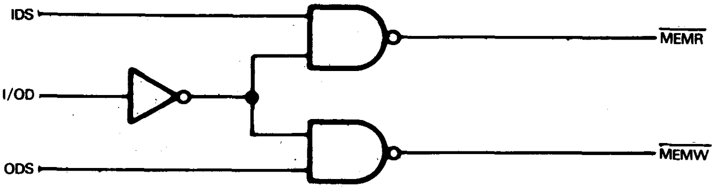
We will conclude our discussion of the use of 8080 devices in PACE systems by comparing PACE System Bus signals with those of 8080 systems. This comparison will be a useful guide for interfacing any 8080 device to a PACE system. Table 14-6 is a summary of PACE System Bus signals and the corresponding signals available in 8080 systems. Two separate columns are provided for 8080 signals: the first applies strictly to the 8080 CPU; the right-hand column refers to the signals present in a typical three-chip 8080 system consisting of the CPU, an 8228 System Controller, and an 8224 Clock Generator and Driver.

Since we have already discussed these signals in preceding paragraphs, we won't perform an item-by-item analysis of the table. Nonetheless, **there are a few signals in this table that do need additional explanation.**

We have included the PACE BPS signal in the I/O Control Signal group although it is not the type of signal you would normally classify within this group. However, you will recall that **when the BPS input is high, PACE operates in a Base-Page-Split mode:** base page then consists of the top 128 words of memory and the bottom 128 words of memory. In our earlier discussion of the BPS signal, we described how **this mode can be used to simplify addressing of I/O devices.** If you refer back to that discussion, you will see that **by doing a little address decoding we can come up with a signal that will tell us when PACE is addressing an I/O device** (as opposed to memory). Let us call this decoded signal 'I/O Device' (I/OD). Now, **we can combine this decoded signal with IDS and ODS as shown below to generate signals equivalent to the 8080 $\overline{I/OR}$ and $\overline{I/OW}$ signals.**



And if we invert the I/OD signal we can generate the 8080 $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ signals.



One other portion of Table 14-6 requires some explanation. **Notice that we have not drawn a line to separate the I/O control signals from the DMA-Related Signals. We've done this intentionally because there is some overlapping of functions with some of these signals.** For example, the PACE EXTEND signal can be used either to extend I/O cycles or to suspend I/O to allow DMA operations. We've also compared the PACE NHALT output signal to the 8080 WAIT signal. This comparison is valid if limited to the CPU Halt state initiated in either system by a Halt instruction. However, in 8080 systems the WAIT signal is also an acknowledgement to the READY or RDYIN input signals. There is no comparable EXTEND acknowledgement signal in PACE systems.

6800 SUPPORT DEVICES NOT COMPATIBLE WITH PACE

The 6800 family includes many devices that might be useful in PACE systems. Unfortunately, all of these devices have one common requirement which effectively makes them incompatible for use in a PACE system. That requirement is the enabling input signal E which, as we mentioned in Chapter 8, should more accurately be described as a synchronizing signal. In 6800 systems, E is usually generated by ANDing one of the primary system clock signals (Φ2) with the Valid Memory Address signal (VMA) from the 6800 CPU. The clock period of the resulting E signal can be no less than one microsecond. The clock signals (CLK and NCLK) used in PACE systems, however, cannot have a clock period greater than 850 nanoseconds and therefore cannot be used to simulate the 6800 Φ2 signal. Therefore, we cannot recommend using 6800 family devices in a PACE system.

Table 14-6. Comparing PACE System Busses To 8080 System Busses

SYSTEM BUS	PACE SYSTEM (CPU, STE, BTE) SIGNALS	8080 CPU SIGNALS	8080 SYSTEM (CPU, 8228, 8224) SIGNALS
Bidirectional Data Bus	D00 - D15 (16 Bits)	D0 - D7 (8 Bits)	DB0 - DB7 (8 Bits)
Address Bus	D00 - D15 Address information must be demultiplexed from Data Bus	A0 - A15	A0 - A15
Control Bus			
I/O Control Signals	NADS Strobe signal used by external logic to demultiplex address from Data Bus	–	–
	IDS	DBIN	$\overline{\text{MEMR}}$ and $\overline{\text{I/OR}}$
	ODS	$\overline{\text{WR}}$	$\overline{\text{MEMW}}$ and $\overline{\text{I/OW}}$
	BPS	–	–
	EXTEND	READY	RDYIN
	NHALT (output)	WAIT	WAIT
DMA-Related Signals	NHALT and CONTIN inputs	HOLD	HOLD
	CONTIN (ACK INT output)	HLDA	HLDA
	–	–	BUSEN
Interrupt Signals	NIR2 - NIR5	INT	INT
	CONTIN (ACK INT output)	D0 and SYNC	$\overline{\text{INTA}}$
	–	INTE	INTE
	Non-maskable Interrupt (CONTIN and NHALT inputs)	–	–
Initialize	NINIT	RESET	RESIN
Jump Condition Inputs	JC13 - JC15	–	–
Control Flag Outputs	F11 - F14	–	–

absolute maximum ratings

All Input or Output Voltages with Respect to Most Positive Supply Voltage (V_{BB})	+0.3V to −21.5V	Storage Temperature Range	−65°C to +150°C
Operating Temperature Range	0°C to +70°C	Lead Temperature (Soldering, 10 seconds)	300°C

electrical characteristics (T_A = 0°C to +70°C, V_{SS} = +5V ±5%, V_{GG} = −12V ±5%, V_{BB} = V_{SS} + 3V ±0.5V)

PARAMETER	CONDITIONS	MIN	MAX	UNITS
OUTPUT SPECIFICATIONS				
D00–D15, F11–F14, ODS, IDS, NADS (These are open drain outputs which may be used to drive DS3608 sense amplifiers, or may be used with pull-down resistors to provide a voltage output.)				
Logic "1" Output Current (Except F11–F14)	V_{OUT} = 2.4V	−1.0	−5.0	mA
Logic "1" Output Current, F11–F14 (Note 7)	V_{OUT} = 2.4V	−0.7	−5.0	mA
Logic "0" Output Current	$V_{GG} \leqslant V_{OUT} \leqslant V_{SS}$		±10	µA
NHALT, CONTIN (Low Power TTL Output.)				
Logic "1" Output Voltage	I_{OUT} = −650µA	2.4		V
Logic "0" Output Voltage	I_{OUT} = 300µA		0.4	V
INPUT SPECIFICATIONS				
D00–D15, NIR2–NIR5, EXTEND, JC13–JC15, CONTIN, NINIT, NHALT (These are TTL compatible inputs.) (Note 2)				
Logic "1" Input Voltage		$V_{SS}-1$	$V_{SS}+0.3$	V
Logic "0" Input Voltage		$V_{SS}-7$	$V_{SS}-4$	V
Pullup Transistor "ON" Resistance (D00–D15) (Note 3)	$V_{IN} = V_{SS}-1V$		7	kΩ
Pullup Transistor "ON" Resistance (all others)	$V_{IN} = V_{SS}-1V$		5	kΩ
Logic "0" Input Current (D00–D15)	V_{IN} = 0.4		−1.8	mA
Logic "0" Input Current (NHALT, CONTIN)	V_{IN} = 0.4		−12	mA
Logic "0" Input Current (all others)	V_{IN} = 0.4		−3.6	mA
Capacitance, Input and Output (except clocks)	$V_{IN} = V_{SS}$, f_T = 500 kHz		20	pF
BPS (This is a MOS Level Input.) (Note 4)				
Logic "1" Input Voltage		$V_{SS}-1$	$V_{SS}+0.3$	V
Logic "0" Input Voltage		V_{GG}	$V_{SS}-7$	V
Logic "1" Input Current	$V_{IN} = V_{SS}-1V$		100	µA
CLK, NCLK (These are MOS Clock Inputs)				
Clock "1" Voltage (Note 5)		$V_{SS}-1$	$V_{SS}+0.3$	V
Clock "0" Voltage		V_{GG}	$V_{GG}+1$	V
Input Capacitance (Note 6)		30	150	pF
Bias Supply Current	$V_{BB} = V_{SS}$ +3.0V		100	µA
V_{GG} Supply Current	t_p = .65µs, T_A = 25°C		40	mA
V_{SS} Supply Current	t_p = .65µs, T_A = 25°C		85	mA

TIMING SPECIFICATIONS (See *Figures 5 to 10* for additional timing information.)

CLK, NCLK (See *Figure 5*) (Referenced to 10% and 90% Amplitude)				
Rise and Fall Time (t_r, t_f)		10	50	ns
Clock Width ($t_{W\,CLK}$, $t_{W\,NCLK}$)		300	375	ns
Clock Non-Overlap (t_{NOVA}, t_{NOVB})		5		ns
Clock Period (t_p)		.65	.8	µs
EXTEND				
Individual Extend Duration			2	µs
Extend Setup Time (t_{ES}) (Note 10)		100		ns
Extend Hold Time (t_{EH}) (Note 13)		20		ns
Propagation Delay (t_{DD})				
NHALT, CONTIN (Note 9)	C_L = 20 pF		200	ns
NADS, IDS, ODS, D00–D15 (Note 8)	V_{OUT} = 2.4V		100	ns
D00–D15				
Input Setup Time (t_{DS}) (Note 11)		200		ns
Hold Time (t_{DH}) (Note 12)		0		ns
Turn-on or Turn-off Time of Pullup Transistor (t_{DC}) (Note 13)		150		ns
F11–F14 Pulse Flag (PFLG) Pulse Width		4t_p −300	4t_p +300	ns
NINIT Initialization Pulse Width		8		clock periods
NIR2–NIR5 Input Pulse Width to Set Latch		1		clock periods

Note 1: Maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended and should be limited to those conditions specified under dc electrical characteristics.
Note 2: Pullup transistor provided on chip (See *Figure 4*.)
Note 3: Pullup transistors on JC13, JC14, JC15 are turned on one out of 8 clock intervals. Pullup transistors on D00–D15 are turned on during last clock period of Input Data Strobe (IDS). Other pullup transistors are on continuously when in data input mode.
Note 4: Pulldown transistor provided on chip.
Note 5: Clamp diodes and series damping resistors may be required to prevent clock overshoot.
Note 6: Capacitance is not constant and varies with clock voltage and internal state of processor.
Note 7: For $V_{SS} \geq V_{OUT} \geq 2.0V$ output current is a linear function of V_{OUT}.
Note 8: Delay measured from valid logic level on clock edge initiating change to valid current output level
Note 9: Delay measured from valid logic level on clock edge initiating change to valid voltage output level.
Note 10: With respect to rising edge of NCLK. (See *Figure 9* and *10*.)
Note 11: With respect to falling edge of CLK. (See *Figure 7*.)
Note 12: With respect to the valid "0" level on the falling edge of Input Data Strobe (IDS). (See *Figure 7*.)
Note 13: With respect to valid logic level of appropriate clock.

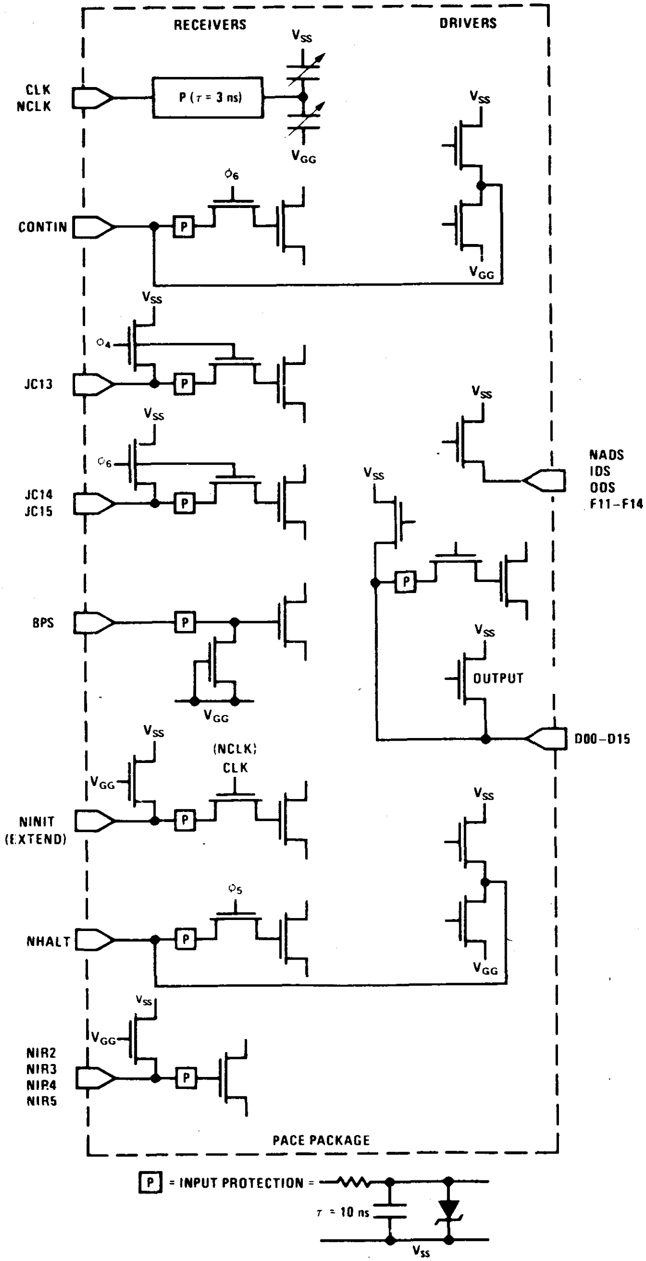
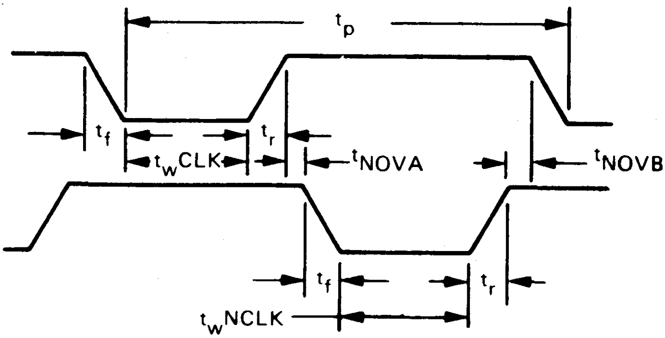


FIGURE 4. PACE Driver and Receiver Equivalent Circuits



where:

t_p = CLOCK PERIOD

t_{NOVA} = t_{NOVB} = CLOCK NONOVERLAP

t_{WCLK} = t_{WNCLK} = CLOCK WIDTH

FIGURE 5. External Clock Timing

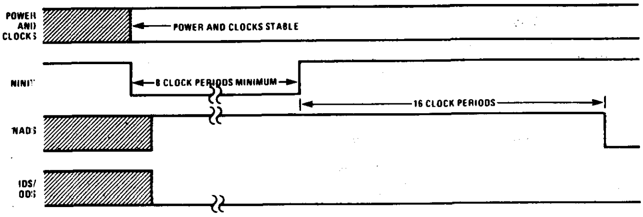
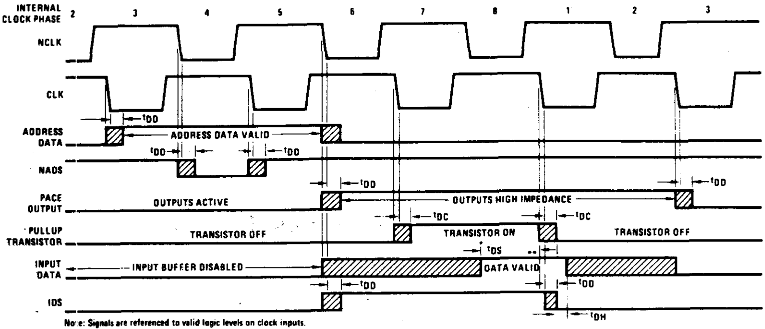


FIGURE 6. Initialization Timing



Note: Signals are referenced to valid logic levels on clock inputs.
Internal clock phases are shown for reference only, they are not available externally.

*V_{IN} must be > V_{SS} – 2.35V at this time if logic "1" input.

**V_{IN} must be valid level (i.e., V_{SS} – 1) at this time (this timing allows for pull-up transistor time constant).

Figure 7. Address Output and Data Input Timing

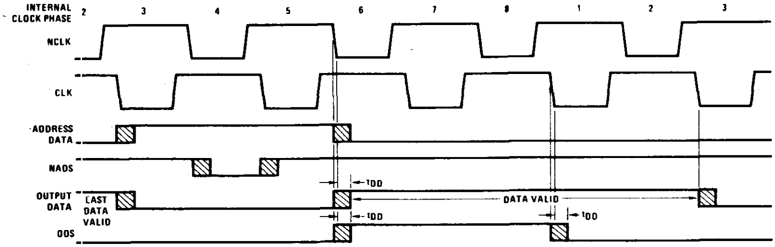


FIGURE 8. Data Output Timing

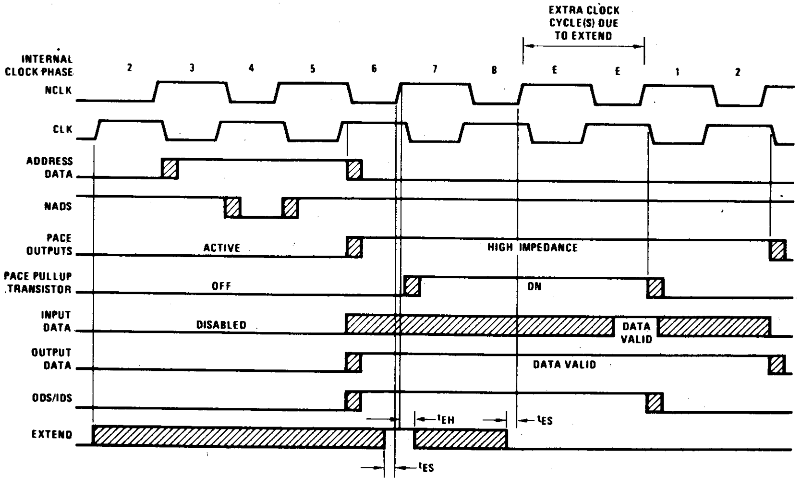


FIGURE 9. Extend I/O Signal Timing

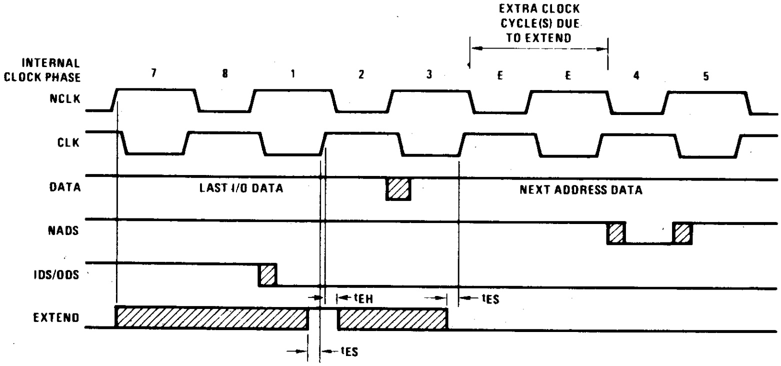


FIGURE 10. Suspend I/O Signal Timing

absolute maximum ratings [1]

Supply Voltage (V_{CC})	7.0 V
(V_{GG})	−15.0 V
Input Voltage	5.5 V
Storage Temperature	−65°C to +150°C
Lead Temperature (soldering, 10 seconds)	300°C

operating conditions

	Min.	Max.	Units
Supply Voltage (V_{CC})	4.75	5.25	V
(V_{GG})	−11.40	−12.6	V
Temperature	0	+70	°C

dc electrical characteristics (Notes 2 and 3)

Parameter	Conditions		Min.	Typ.	Max.	Units
OUTPUT SPECIFICATIONS:						
T CLK, T CLK* (TTL Clocks)						
V_{OH} Logic "1" Output Voltage	$V_{CC} = 4.75$ V	$I_{OH} = -1$ mA	3.65	4.25		V
V_{OL} Logic "0" Output Voltage	$V_{CC} = 4.75$ V	$I_{OL} = 32$ mA		0.25	0.4	V
I_{OS} Output Short Circuit Current	(Note 4), $V_{CC} = 5.25$ V, $V_O = 0$		−10	−33	−55	mA
CK, NCK, CLK, NCLK						
V_{OH} Logic "1" Output Voltage	$I_{OH} = -100\mu A$		$V_{CC} - 0.9$	4.5		V
V_{OL} Logic "0" Output Voltage	$V_{CC} = 4.75$ V	$I_{OL} = 100\mu A$		$V_{GG} + 0.1$	$V_{GG} + 0.25$	V
	$V_{GG} = -11.4$ V	$I_{OL} = 5$ mA		$V_{GG} + 0.2$	$V_{GG} + 0.5$	V
INPUT SPECIFICATIONS:						
EXTC						
V_{IH} Logic "1" Input Voltage			2.0			V
I_{IH} Logic "1" Input Current	$V_{CC} = 5.25$ V	$V_{IN} = 2.4$ V			40	µA
		$V_{IN} = 5.5$ V			1.0	mA
V_{IL} Logic "0" Input Voltage					0.8	V
I_{IL} Logic "0" Input Current	$V_{CC} = 5.25$ V	$V_{IL} = 0.4$ V		−0.9	−1.6	mA
V_{CLAMP} Input Clamp Diode	$V_{CC} = 4.75$ V	$I_{IL} = -12$ mA		−0.8	−1.5	V
POWER SUPPLY CURRENT						
I_{CC} Supply Current from V_{CC}	$V_{CC} = 5.25$ V			20	30	mA
I_{GG} Supply Current from V_{GG}	$V_{GG} = -12.6$ V			−40	−55	mA

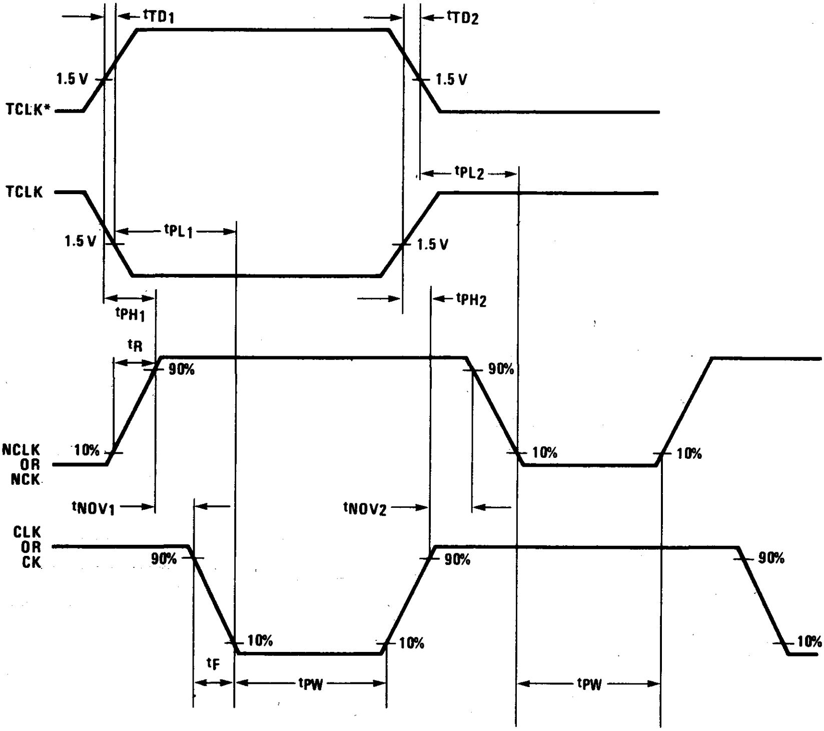
ac electrical characteristics Crystal Freauency at 2.6667 MHz $T_A = 0°C$ to $+70°C$, $V_{CC} - V_{GG} = +17$ V ± 5%

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
t_{NOV1}, t_{NOV2}	Non-Overlap Time	5	12		ns	See Note 5
t_{PW}	MOS Clocks Pulse Width (NCLK, CLK, NCK, CK)	300	320		ns	See Note 5
t_R	MOS Clocks Rise Time (NCLK, CLK, NCK, CK)			40	ns	See Note 5
t_F	MOS Clocks Fall Time (NCLK, CLK, NCK, CK)			40	ns	See Note 5
t_{PH1}, t_{PH2}	TTL Clocks to MOS Clocks High Level Delay	−40		40	ns	See Note 5
t_{PL1}, t_{PL2}	TTL Clocks to MOS Clocks Low Level Delay			80	ns	See Note 5
t_{TD1}, t_{TD2}	TTL Clock to TTL Clock Delay	−25		25	ns	See Note 5
t_{START}	Time Delay from Last Power Applied to MOS Clocks Stabilized			100	ms	See Figure 7

Notes:

1. "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. They are not meant to imply that the devices should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.
2. Unless otherwise specified, min/max limits apply across the 0°C to +70°C temperature range and $V_{CC} = 4.75$ V to 5.25 V, $V_{GG} = -11.4$ V to −12.6 V power supply range. All typicals are given for $V_{CC} = 5.0$ V, $V_{GG} = -12$ V, and $T_A = +25°C$.
3. All currents into device pins are shown as positive; currents out of device pins are shown as negative. All voltages are references to ground unless otherwise noted.
4. Only one output at a time should be shorted.
5. The test conditions for measuring AC parameters are shown in Figures 2 and 3, with $C_1 = C_2 = 60$ pF, $C_3 = 80$ pF, $C_{NOV} = 60$ pF. Load conditions for MOS clocks and TTL clocks are shown in Figures 4 and 5. Including probe and jig capacitance, $C_{L1} = 20$ to 80 pF, and $C_{L2} = 40$ pF.

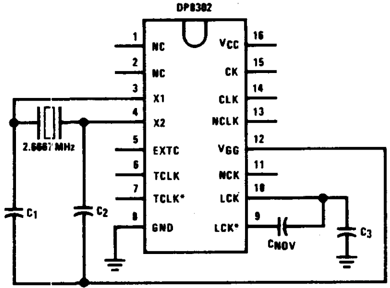
timing diagram



TIMES FOR NCLK, NCK, CLK, AND CK MEASURED AT 10% AND 90%

Figure 2.

test conditions



$C_1 = C_2 = 60$ pF, $C_3 = 80$ pF, $C_{NOV} = 60$ pF*

*ALL CAPACITORS ARE ±5%

Figure 3.

NCLK, NCK, CLK, CK LOAD

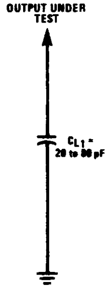


Figure 4.

TCLK*, TCLK LOAD

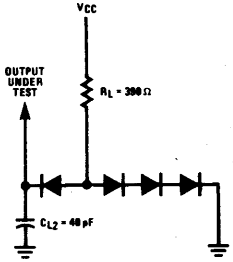


Figure 5.

typical characteristics

TYPICAL NON-OVERLAP TIME VS. NON-OVERLAP CAPACITOR

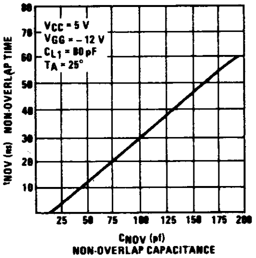


Figure 6.

t_{START} – TIME DELAY FROM LAST POWER APPLIED TO MOS CLOCKS STABILIZED.

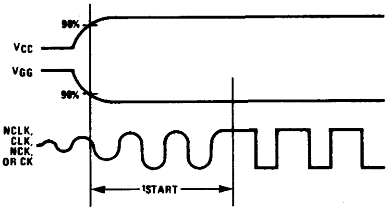


Figure 7.

absolute maximum ratings (Note 1)

Supply Voltage	7V
Input Voltage (All Inputs Except MBI/O Input Active)	5.5V
Output Voltage	5.5V
MOS Bus Input Current	±10 mA
Storage Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C

recommended operating conditions

	MIN	MAX	UNITS
Supply Voltage (V_{CC})	4.75	5.25	V
Temperature (T_A)	0	+70	°C

dc electrical characteristics (Notes 2 and 3)

PARAMETER		CONDITIONS		MIN	TYP	MAX	UNITS
TTL BUS PORT (BDI/O 00–07)							
V_{IH}	Logical "1" Input Voltage			2.0			V
V_{IL}	Logical "0" Input Voltage					0.8	V
V_{OH}	Logical "1" Output Voltage	WBD* = 0.8V,	I_{OH} = −1 mA	V_{CC}−1.1	V_{CC}−0.8		V
		MBI/O = 0.5 mA	I_{OH} = −5.2 mA	2.4	3.7		V
V_{OL}	Logical "0" Output Voltage	WBD* = 0.8V,	I_{OL} = 20 mA		0.25	0.4	V
		MBI/O = 100µA	I_{OL} = 50 mA		0.4	0.5	V
I_{OS}	Output Short Circuit Current	WBD* = 0.8V, MBI/O = 0.5 mA, V_{OUT} = 0V, V_{CC} = 5.25V, (Note 4)		−10	−35	−75	mA
I_{IH}	Logical "1" Input Current	WBD* = 2V, V_{IH} = 2.4V				80	µA
I_I	Input Current at Maximum Input Voltage	WBD* = 2V, V_{IH} = 5.5V, V_{CC} = 5.25V				1	mA
I_{IL}	Logical "0" Input Current	WBD* = 2V, V_{IL} = 0.4V			−10	−250	µA
V_{CLAMP}	Input Clamp Voltage	WBD* = 2V, I_{IN} = −12 mA			−0.2	−1.5	V
I_{OD}	Output/Input Bus Disable Current	WBD* = STR* = 2V, BDI/O = 0.4V to 4V, V_{CC} = 5.25V		−80		80	µA
MOS BUS PORT (MBI/O 00–07)							
I_0	Logical "0" Input Current	WBD* = 0.8V, $I_{OL(TTL)}$ = 50 mA, $V_{OL} \leq 0.5V$, (Note 5)		−5.0		0.10	mA
I_1	Logical "1" Input Current	WBD* = 0.8V, $I_{OH(TTL)}$ = −1 mA, $V_{OH} \geq V_{CC} - 1.1V$, (Notes 5 and 6)		0.50		5.0	mA
V_0	Logical "0" Input Voltage	WBD* = 0.8V, $I_{OL(TTL)}$ = 50 mA, $V_{OL} \leq 0.5V$				0.8	V
V_1	Logical "1" Input Voltage	WBD* = 0.8V, $I_{OH(TTL)}$ = −1 mA, $V_{OH} \geq V_{CC} - 1.1V$		2.0	1.5		V
V_{OH}	Logical "1" Output Voltage	WBD* = CE1 = BDI/O = 2V, $I_{OH(MOS)}$ = −1 mA, CE2* = STR* = 0.8V		2.4	3.3		V
V_{OL}	Logical "0" Output Voltage	WBD* = CE1 = 2V, $I_{OL(MOS)}$ = 5 mA, CE2* = STR* = BDI/O = 0.8V			0.28	0.5	V
I_{OS}	Output Short Circuit Current	WBD* = CE1 = BDI/O = 2V, V_{CC} = 5.25V, V_{OUT} = 0V, STR* = CE2* = 0.8V, (Note 4)		−7	−15	−45	mA
V_{CLAMP}	Input Clamp Voltage	I_{IN} = −12 mA				−1.5	V
I_{OD}	Output/Input Bus Disable Current	MBI/O = 0.4V to 4V, V_{CC} = 5.25V		−80		80	µA
CONTROL INPUTS (WBD*, CE1, CE2*, STR*)							
V_{IH}	Logical "1" Input Voltage			2.0			V
V_{IL}	Logical "0" Input Voltage					0.8	V
I_{IH}	Logical "1" Input Current	V_{IN} = 2.4V				20	µA
I_1	Input Current at Maximum Input Voltage	V_{IN} = 5.5V				1.0	mA

dc electrical characteristics (Continued) (Notes 2 and 3)

PARAMETER		CONDITIONS	MIN	TYP	MAX	UNITS
CONTROL INPUTS (WBD*, CE1, CE2*, STR*) (continued)						
I_{IL}	Logical "0" Input Current	V_{IN} = 0.4V		−250	−400	µA
V_{CLAMP}	Input Clamp Voltage	I_{IN} = −12 mA		−0.85	−1.5	V
POWER SUPPLY CURRENT						
I_{CC}	Power Supply Current	V_{CC} = 5.25V		70	110	mA

Note 1: "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. They are not meant to imply that the devices should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

Note 2: Unless otherwise specified, min/max limits apply across the 0°C to +70°C temperature range and the 4.75V to 5.25V power supply range. All typicals are given for V_{CC} = 5V and T_A = 25°C.

Note 3: All currents into device pins are shown as positive, out of device pins are negative. All voltages are referenced to ground unless otherwise noted.

Note 4: Only one output at a time should be shorted.

Note 5: The MBI/O Input Characteristic Graph illustrates this parameter and defines the regions of guaranteed logical "0" and logical "1" outputs. See equivalent input structure for clarification. When the MBI/O input is loaded with a high impedance source (open), the TTL output will be in the logic "0" state.

Note 6: The maximum MOS bus positive input current specification is intended to define the upper limit on guaranteed input clamp operation. At higher input currents (up to the absolute maximum rating) clamp operation is not guaranteed but TTL bus logic state is valid and no device damage will occur.

Note 7: In most applications the MOS bus data lines are higher impedance and more sensitive to noise coupling than TTL bus lines. Conservative design practice would dictate routing MOS bus lines away from high speed, low impedance TTL lines and MOS clock lines or providing a ground shield when they are adjacent.

ac electrical characteristics V_{CC} = 5V ±5%, T_A = 0°C to +70°C

PARAMETER		CONDITIONS		MIN	TYP	MAX	UNITS
DATA TRANSFER SPECIFICATIONS							
Receiving Mode (BDI/O Bus to MBI/O Bus)		WBD* = 3V, C_L = 15 pF,	t_{pd0}		17	40	ns
		R_L = 1 kΩ, *(Figures 4 and 6)*	t_{pd1}		20	40	ns
	Driving Mode (MBI/O Bus to	WBD* = CE1 = 0V,	t_{pd0}		40	60	ns
	BDI/O Bus)	STR* = CE2* = 3V,	t_{pd1}		40	60	ns
		C_L = 50 pF, R_L = 100 Ω,					
		(Figures 3 and 5)					
TRANSCEIVER MODE SPECIFICATIONS							
Select Bus							
t_{DS}	Chip Enable Data Set-Up	*(Figure 1)*		45	23		ns
t_{DH}	Chip Enable Data Hold	*(Figure 1)*		0			ns
t_{ES}	Set-Up	*(Figure 1)*		0			ns
TTL Data Bus (BDI/O 00–07)							
$t_{BD\ OD}$	Bus Data Output Disable	C_L = 5 pF, R_L = 100 Ω, *(Figure 1)*		5	20	50	ns
$t_{BD\ OE}$	Bus Data Output Enable	C_L = 50 pF, R_L = 100 Ω, *(Figure 1)*			25	80	ns
$t_{BD\ IE}$	Bus Data Input Enable	*(Figure 1)*			30		ns
$t_{BD\ ID}$	Bus Data Input Disable	*(Figure 1)*			30		ns
MOS Data Bus (MBI/O 00–07)							
$t_{MB\ OD}$	MOS Bus Output Disable	C_L = 15 pF, R_L = 1 kΩ, *(Figure 1)*		15	50	100	ns
$t_{MB\ OE}$	MOS Bus Output Enable	C_L = 15 pF, R_L = 1 kΩ, *(Figure 1)*			50	100	ns
$t_{MB\ ID}$	MOS Bus Input Disable	*(Figure 1)*			55		ns
$t_{MB\ IE}$	MOS Bus Input Enable	*(Figure 1)*			20		ns
Select Bus							
t_{CLR}	Clear Previous Chip Enable	*(Figure 2)*			25	50	ns

switching time waveforms and ac test circuits

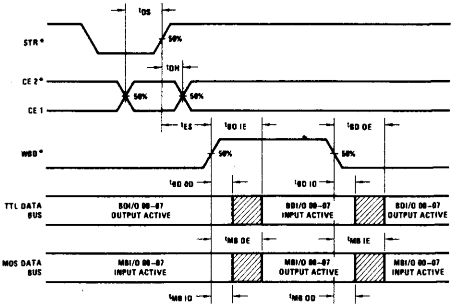


FIGURE 1

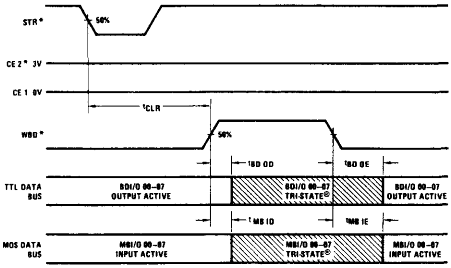


FIGURE 2

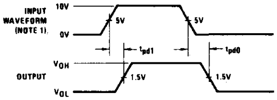
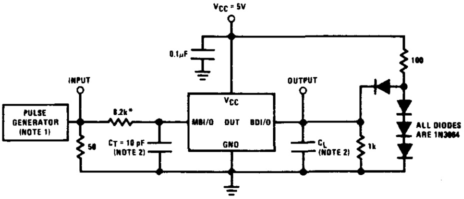


FIGURE 3. BDI/O Bus



*This input network simulates the actual drive characteristic of the PACE outputs

FIGURE 5. MBI/O to BDI/O ac Loads

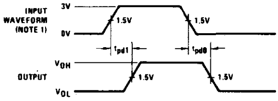


FIGURE 4. MBI/O Bus

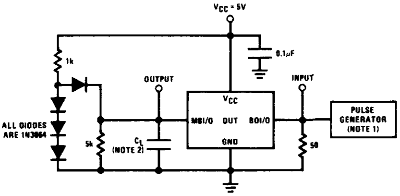


FIGURE 6. BDI/O to MBI/O ac Loads

Note 1: Freq = 1 MHz, duty cycle = 50%, $t_R = t_F \leq 10$ ns (refer to *Figures 5 and 6)*.

Note 2: All capacitance values include probe and jig capacitance (refer to *Figures 5 and 6)*.

Chapter 15
THE GENERAL INSTRUMENTS CP1600

Until the advent of the single chip MicroNova central processing unit (described in Chapter 17) the General Instruments CP1600 represented the most minicomputer-like of the 16-bit microcomputers.

The CP1600 has a very powerful instruction set that will surely please anyone planning to do a lot of programming; it also has excellent execution speeds. Instructions which are comparable to the National Semiconductor PACE have execution speeds that range from 1.6 to 4.8 microseconds. Unless your application has to make heavy use of indirect addressing, a feature which PACE has and that CP1600 does not have, the CP1600 is likely to execute programs more than twice as fast as PACE.

Philosophically, PACE and CP1600 are very similar.

The most important difference between PACE and the CP1600 is that PACE designers have been influenced by the Data General Nova architecture. CP1600 architects have been more influenced by the Digital Equipment Corporation PDP-11: this product is offered currently as the LSI 11, however it is not described in this book, since at the present time there is no single chip LSI 11 CPU.

Another important single difference is the fact that CP1600 signals are TTL compatible; therefore bus driver devices that convert signals from MOS to TTL levels are not required. However, equivalent buffer amplifier devices will be required, since you are not going to use a microprocessor as sophisticated as the CP1600 in a very simple configuration — and that is the only type of configuration which the CP1600 would be able to drive without buffer amplifiers.

The CP1600 is fabricated using NMOS ion implant LSI technology; the device is packaged as a 40-pin DIP.

Three power supplies are required: + 12V, + 5V and -3V.

Using a 200 nanosecond clock, instruction execution times range between 1.6 and 4.8 microseconds.

THE CP1600 MICROCOMPUTER SYSTEM OVERVIEW

Logic of our general microcomputer system which has been implemented by the CP1600 CPU is illustrated in Figure 15-1.

Observe that the CP1600 requires external logic to create its various timing and clock signals.

Some bus interface logic is shown as absent because a number of devices must surround the CP1600; these include:

1) An address buffer, since data and addresses are multiplexed on a single 16-bit bus.
2) Buffer amplifiers to provide the power required by the type of memory and I/O devices that will normally be connected to a CP1600 CPU.
3) A one-of-eight decoder chip to create eight individual control signals out of three controls output by the CP1600.

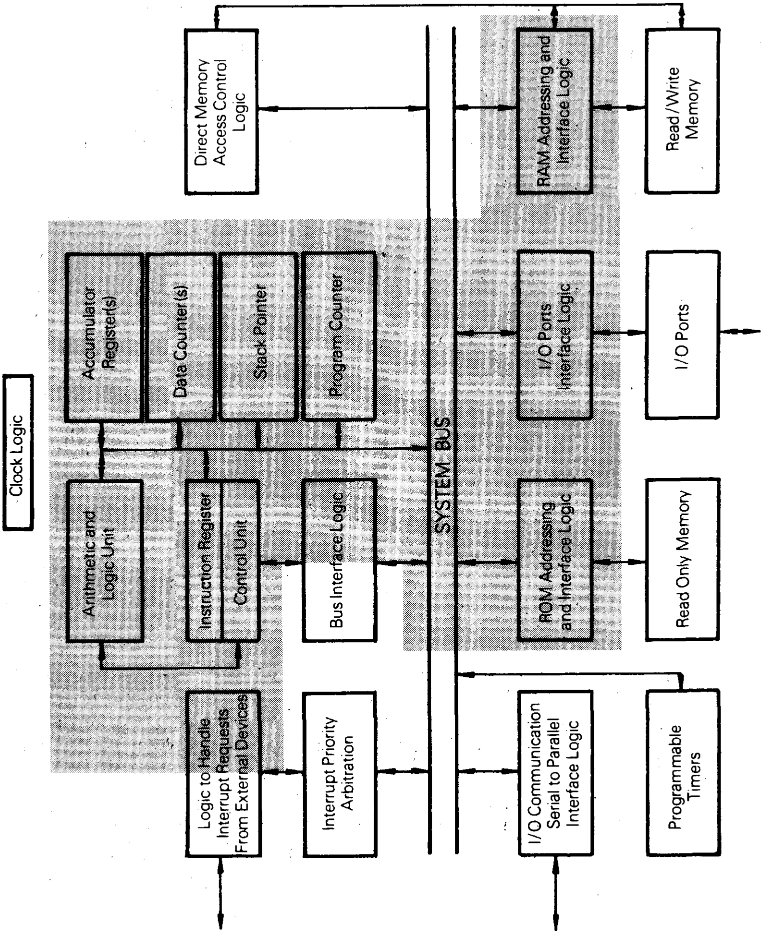

Clock Logic
Accumulator Register(s)
Data Counter(s)
Stack Pointer
Program Counter
Arithmetic and Logic Unit
Instruction Register
Control Unit
Bus Interface Logic
Logic to Handle Interrupt Requests From External Devices
Interrupt Priority Arbitration
Direct Memory Access Control Logic
SYSTEM BUS
RAM Addressing and Interface Logic
Read/Write Memory
I/O Ports Interface Logic
I/O Ports
ROM Addressing and Interface Logic
Read Only Memory
I/O Communication Serial to Parallel Interface Logic
Programmable Timers

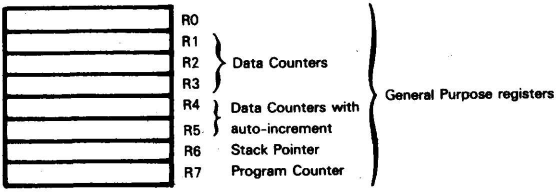
Figure 15-1. Logic Of The CP1600 CPU

4) A one-of-sixteen multiplex chip to funnel sixteen external status signals into the CP1600.

Were you to compare Figure 15-1 with an equivalent figure for a low-end microprocessor, such as the F8 (which is described in Chapter 2) the CP1600 may appear to offer fewer logic functions; but within the functions it does provide, the CP1600 provides considerably more logic and program execution capabilities. Where low-end microprocessors choose to condense, onto a single chip, simple implementations of different logic functions, high-end products such as PACE and the CP1600 choose to provide more devices — with greater capabilities on each device.

CP1600 PROGRAMMABLE REGISTERS

The CP1600 has eight 16-bit programmable registers which may be illustrated as follows:



The way in which the registers illustrated above are used is unusual, when compared to other microcomputers described in this book. All eight 16-bit registers can be addressed as though they were General Purpose registers; however, only Register R0 has no other assigned function. We may therefore look upon Register R0 as the Primary Accumulator for this CPU.

Registers R1, R2 and R3 serve as General Purpose registers, but may also be used as Data Counters.

In addition to serving as General Purpose registers, R4 and R5 may be used as auto-incrementing Data Counters. Memory reference instructions that identify Register R4 or R5 as holding the implied memory address will cause the contents of Register R4 or R5 to be incremented — after the memory reference instructions have completed execution.

Registers R6 and R7, in addition to being accessible as General Purpose registers, also serve as a Stack Pointer and a Program Counter, respectively.

Having the Stack Pointer accessible as a General Purpose register makes it quite simple to maintain more than one Stack in external memory; also, you can easily address the Stack by indexing backwards from the Stack Pointer.

Having the Program Counter accessible as a General Purpose register can be useful when executing various types of conditional branch logic.

While having the Stack Pointer and the Program Counter accessible as though they were General Purpose registers may appear strange, this is a feature of the PDP-11 minicomputer — and is a very powerful programming tool.

CP1600 MEMORY ADDRESSING MODE

The CP1600 addresses memory and I/O devices within a single address space.

When referencing external memory, you can use direct addressing, implied addressing, or implied addressing with auto-increment.

Direct addressing instructions are all two or more words long, where the second or last word of the instruction object code provides a 16-bit direct address.

Instructions that reference memory using implied addressing identify General Purpose Register R1, R2 or R3 as containing the implied address.

A memory reference instruction which identifies Register R4 or R5 as providing the external memory address will always cause Register R4 or R5 contents to be incremented — following the memory access; thus you have implied memory addressing with auto-increment.

The CP1600 has no stack reference instructions such as a Push or Pull; rather a variety of memory reference instructions can identify Register R6 as providing the implied address. When Register R6 provides the implied address, it is treated as an upward migrating Stack Pointer. When a memory write operation specifies Register R6 as providing the implied memory address, Register R6 contents will be incremented following the memory write. A memory read instruction that specifies Register R6 as providing the implied memory address will cause the contents of Register R6 to be decremented before the read operation occurs.

An unusual feature of the CP1600 is the fact that a variety of secondary memory reference instructions can also reference memory via the Stack Pointer. When these instructions are executed, Register R6 contents are decremented before the memory access occurs — as though a pull operation from the Stack was being executed.

Logically, Register R6, the Stack Pointer, is being handled as though it were a Data Counter with post-increment and pre-decrement.

CP1600 STATUS AND CONTROL FLAGS

The CP1600 CPU has four of the standard status flags; in addition it has some unusual control signals.

These are the four standard status flags:

Sign (S). This status is set equal to the high order bit of any arithmetic operation result.

Zero (Z). This status is set to 1 when any instruction's execution creates a zero result. The status is set to 0 for a nonzero result.

The Carry (C) and Overflow (O) statuses are standard carry and overflow, as described in Volume I, Chapter 2.

Four control signals are output (EBCA0 - EBCA3) during a Branch-on-External (BEXT) instruction. These four signals are output to reflect the low order four bits of the BEXT instruction's object code. External logic receives these four signals, and depending on their state, may or may not return a high input via EBCI. If EBCI is returned high, then the BEXT instruction will perform a branch; if EBCI is returned low, then the BEXT instruction will cause the next sequential instruction to be executed. The four control signals EBCA0 - EBCA3 therefore provide the CP1600 with a means of testing 16 external conditions.

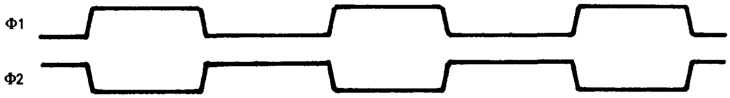
CP1600 CPU PINS AND SIGNALS

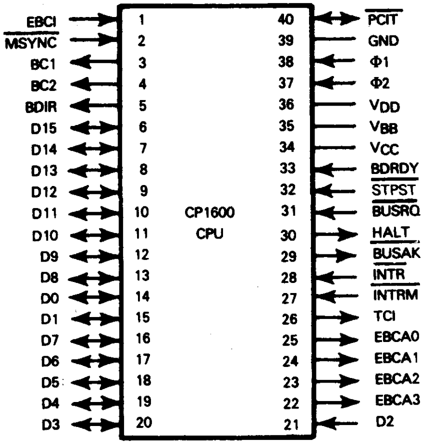
CP1600 CPU pins and signals are illustrated in Figure 15-2.

D0 - D15 is a multiplexed Address and Data Bus. Given a total of 40 pins in a package, CP1600 designers have been forced to share 16 pins between addresses and data. **Three control signals, BDIR, BC1 and BC2, identify the traffic on the Address/Data Bus. External logic (one MSI chip) must decode these three signals to create eight control signals, as summarized in Table 15-1.**

Remaining signals may be divided into four groups: timing, status/control, interrupt and DMA.

Two timing clock signals are required: Φ1 and Φ2. These are complementary clock signals which may be illustrated as follows:





Pin Name	Description	Type
D0 - D15	Data and Address Bus	Tristate, Bidirectional
BDIR, BC1, BC2	Bus control signals	Output
Φ1, Φ2	Clock signals	Input
MSYNC	Master Synchronization	Input
EBCA0 - EBCA3	External branch condition address lines	Output
EBCI	External branch condition input	Input
PCIT	Program Counter inhibit/software interrupt signal	Input
BDRDY	WAIT	Input
STPST	CPU stop or start on high-to-low transition	Input
HALT	Halt state signal	Output
INTR, INTRM	Interrupt request lines	Input
TCI	Terminate current interrupt	Output
BUSRQ	Bus request	Input
BUSAK	External bus control acknowledge	Output
VBB, VCC, VDD, GND	Power and Ground	

Figure 15-2. CP1600 CPU Signals And Pin Assignments

MSYNC is a somewhat unusual signal, as compared to other microcomputer clock signals in this book. Following power up, MSYNC must be held low for at least 10 milliseconds. On the subsequent rising edge of MSYNC, logic internal to the CP1600 CPU will synchronize the Φ1 and Φ2 clock signals to start a new machine cycle. Most of the CPU devices we have described in this book use a reset signal, or have internal power up logic which performs this clock synchronization.

CP1600 SYSTEM RESET

Another unusual feature of the MSYNC signal is the fact that it does not zero any internal CPU registers — not even the Program Counter. Instead, when MSYNC is raised high after having been input low, the CPU immediately reads the contents of the Data/Address Bus and interprets this 16-bit value as the address of the first instruction to be executed following the Reset. Thus, external logic must also receive the MSYNC signal and must provide the address of the first instruction to be executed following a system Reset.

Table 15-1. CP1600 Bus Control Signals

BC1	BC2	BDIR	SIGNAL	FUNCTION
0	0	0	NACT	The CPU is inactive and the Data/Address Bus is in a high impedance state.
0	0	1	BAR	A memory address must be input to the CPU via the Data/Address Bus.
0	1	0	IAB	Acknowledged external interrupt requesting logic must place the starting address for the interrupt service routine on the Address Bus.
0	1	1	DWS	Data write strobe for external memory.
1	0	0	ADAR	This signal identifies a time interval during which the Data/Address Bus is floated, while the contents of a Data Counter register is being interpreted as the effective memory address during an implied addressing operation.
1	0	1	DW	The CPU is writing data to external memory. DW will precede DWS by one machine cycle.
1	1	0	DTB	This is a read strobe which external memory or I/O logic can use in order to place data on the Data/Address Bus.
1	1	1	INTAK	This is an interrupt acknowledge signal. It is followed by IAD which is a strobe telling the external logic which is being acknowledged to identify itself by placing an address vector on the Data/Address Bus.

Now consider the status and control signals.

First of all, there **are the four control outputs** which we have already described: **EBCA0 - EBCA3. There is one conditional Branch instruction (BEXT) which will only branch if a high signal is input via EBCI.** When the BEXT instruction is executed, the low order four BEXT instruction object code bits are output via EBCA0 - EBCA3. External logic is supposed to decode these four signals by whatever means are appropriate — and thence determine whether EBCI should be input high or low. A high input, as we have just stated, will result in a branch; a low input will cause the next sequential instruction to be executed.

Now, in reality, there is no connection within CP1600 CPU logic between the EBCI input and the four EBCA0 - EBCA3 outputs. So far as external logic is concerned, the execution of a BEXT instruction is identified by signal levels output and maintained on the EBCA0 - EBCA3 outputs, while the EBCI input determines whether a branch will or will not occur. How external logic chooses to determine whether EBCI will be set high or low is entirely up to external logic. The only vital function served by EBCA0 - EBCA3 is to identify the instant at which a BEXT instruction is executed.

Another unusual control signal provided by the CP1600 is $\overline{\mathrm{PCIT}}$; this is a bidirectional signal. When input low, this signal prevents the Program Counter from being incremented following an instruction fetch. This signal is also output as a low pulse following execution of a software interrupt instruction. Instruction timing separates the active input and active output of this signal; providing external logic adheres to timing requirements, a conflict between input and output logic will never arise.

$\overline{\mathrm{BDRDY}}$ is equivalent to the WAIT signal we have described for a number of other microcomputers. BDRDY is input low by any external logic which requires more time in order to respond to an I/O access. Recall that the CP1600 uses a single address space to reference memory or I/O devices. The $\overline{\mathrm{BDRDY}}$ signal causes the CPU to enter a Wait state for as long as BDRDY is being input low; however, during the Wait state CPU logic is not refreshed. Thus a Wait state cannot last for more than 40 microseconds, or the contents of internal CPU locations will be lost.

STPST is a typical Halt/Reset input, except that it is an edge triggered signal. When external logic inputs a high-to-low transition via STPST, the CPU will complete execution of any interruptible instruction, then will enter a Halt state and output HALT high. If a non-interruptible instruction is being executed, then the Halt state will not begin until completion of the next interruptible instruction's execution. The Halt state will last until external logic inputs another high-to-low STPST transition — at which time the HALT output will be returned low and normal programming execution will continue. Execution of the HLT instruction also causes the CP1600 to enter a Halt state, as described above.

Let us now look at interrupt signals.

The CP1600 has two interrupt request inputs — INTR and INTRM. INTR has higher priority than INTRM. INTR cannot be disabled. Typically INTR will be used to trigger an interrupt on power failure or other catastrophe.

The interrupt acknowledge signal is created by external logic which must decode the BC1, BC2 and BDIR signals, as shown in Table 15-1. Observe that there are, in fact, two interrupt acknowledge signals; the first acknowledges the interrupt itself, while the second may be used as a strobe for external logic to return an interrupt address vector. This interrupt sequence is described later in this chapter.

The CP1600 has two additional interrupt-related signals which are unusual when compared to other microcomputers described in this book.

TCI is output high when an End-of-Interrupt instruction is executed. This signal makes it easy for external logic to generate interrupt priorities which extend across the execution of an interrupt service routine. We have discussed this subject in some detail while describing the 8259 Priority Interrupt Control Unit in Chapter 4.

THE CP1600 DMA LOGIC

DMA logic of the CP1600 is quite straightforward. **When external logic wishes to transfer data under DMA control, it inputs BUSRQ low. At the conclusion of the next interruptible instruction's execution, the CPU floats the System Bus and outputs BUSAK low.** External logic can keep control of the System Bus for as long as BUSRQ is being input low.

The System Data/Address Bus lines — D0 to D15 — are floated during a DMA operation.

THE CP1600 INTERRUPT LOGIC

The CP1600 uses a vectored interrupt processing system.

External logic requests an interrupt by inputting a low signal at either the INTR or INTRM pins.

Following the execution of the next interruptible instruction, the CP1600 acknowledges the interrupt by pushing Register R7 contents (the Program Counter) onto the Stack; then the CP1600 outputs 111, followed by 010 at BC1, BC2 and BDIR. External logic must respond by placing 16 bits of data on the Data/Address Bus. These 16 bits of data will be loaded into Register R7, the Program Counter, thus causing program execution to branch to the memory location where an interrupt service routine dedicated to the single acknowledge interrupt begins.

Recall that the PCIT signal is output low following execution of a software interrupt instruction. This is the only microcomputer described in this book which allows external logic to respond to a software interrupt in this fashion. Allowing external logic to respond to a software interrupt only makes sense when you anticipate your product being used in a minicomputer-like environment. Typically, the software interrupt will interface to logic of a front panel or console.

You may, if you wish, end an interrupt service routine by executing a Terminate Current Interrupt (TCI) instruction, in which case the TCI signal will be output high.

Following an interrupt acknowledge, the interrupt service routine must execute instructions in order to disable interrupts and save the contents of registers on the Stack. The exception is

Register R7, the Program Counter, which is automatically pushed onto the Stack following an interrupt acknowledge.

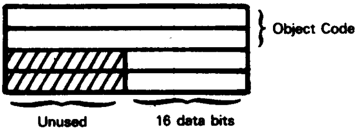
External logic is entirely responsible for any type of interrupt priority arbitration which may occur, and for the generation of the interrupt vector address which must be input following an interrupt acknowledge.

THE CP1600 INSTRUCTION SET

The CP1600 instruction set is relatively straightforward. Addressing modes are simple and instructions are typical of those we have seen and described for other microcomputers. Unusual features relating to addressing modes available with **individual instructions are summarized in Table 15-2,** which describes the CP1600 instruction set.

There are a few features of the CP1600 instruction set worth noting.

First of all, individual instructions allow you to access 8-bit or 16-bit data. If you execute an SDBD instruction, then subsequent data will be fetched from memory in 8-bit increments. For example, an immediate instruction that would normally input 16 bits of data from the second instruction object code word, will now require two trailing object code words. This may be illustrated as follows:



Another interesting feature of the CP1600 is the fact that all instruction codes occupy just 10 of the 16 available bits. The high order six instruction code bits are ignored at the present time but are planned for future expansion of the product family.

If you have never programmed a PDP-11 minicomputer, then you should pay particular attention to programming techniques that result from the Stack Pointer and Program Counter being accessed as general purpose registers.

A wide variety of Register Operate instructions allow you to compute data and load the result directly into Register R7, the Program Counter. In effect, these become computed Jump instructions.

The ability to manipulate Register R6, the Stack Pointer, as though it were a General Purpose register means that it is easy to maintain a number of different Stacks in external read/write memory.

The Jump-to-Subroutine instruction has a minicomputer flavor to it. Rather than saving the return address on the Stack, Register R7 contents are moved to General Purpose Register R4 or R5. A number of minicomputers will save a subroutine return address in a General Purpose register in this fashion. The problem with this logic is that you must execute an additional instruction within the subroutine to save the return address on the Stack if you are going to use nesting subroutines. If you are passing subroutine parameters, however, this is an excellent arrangement, for the Jump-to-Subroutine instruction places the address of the parameter list directly in a Data Counter with auto-increment. We have described the concept of parameter passing in Volume I, Chapter 7.

Note that the CP1600 instruction set lacks a logical OR.

In Tables 15-2 and 15-4, instruction length is given in terms of "words" rather than "bytes", as we have done in previous chapters. Since only the lower 10 bits of the CP1600 object code are presently used, system configurations need not have the full 16-bit word size. Hence, a word may be 10 to 16 bits wide, depending on the implementation.

The following notation is used in Table 15-2:

ADDR One word of direct address.

cond Condition on which a branch may be taken. Table 15-3 lists all 14 branch conditions.

DATA One word of immediate data.

DISP One word displacement. See Table 15-4 for location of sign bit.

E External branch condition.

EBCA0-3 The external branch condition address lines: EBCA0, EBCA1, EBCA2 and EBCA3.

EBCI The external branch condition input line.

LABEL A 16-bit direct address, target of a Jump instruction. See Table 15-4 for the bit format.

$\overline{\text{PCIT}}$ The software interrupt output line.

RB General Purpose Register R4, R5 or R6.

RD One of the General Purpose registers, used as a destination for operation results.

RM One of the General Purpose registers used as a Data Counter. R4 or R5, if specified, is auto-incremented after the memory access. R6 is incremented after a write, and decremented before a read.

RR General Purpose Register R0, R1, R2 or R3.

RS One of the General Purpose registers, used as the source of an operand.

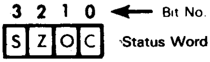
STATUSES

S the Sign status
C the Carry status
Z the Zero status
O the Overflow status

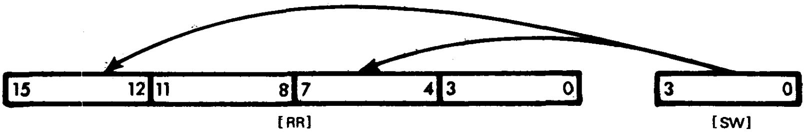
The following symbols are used in the STATUSES column:

X the status flag is affected by the operation.
A blank means the status flag is not affected.
0 the operation clears the status flag
1 the operation sets the flag
2 the Overflow flag is affected only on 2-bit shifts or rotates.

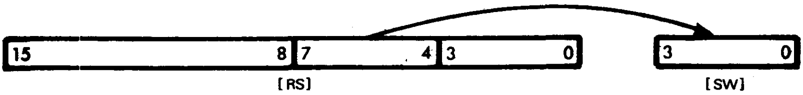
SW The Status Word, whose bits correspond to the condition of the status flags in the following way:



When the status word is copied into a register, it goes to the upper half of each byte:



When the status word is loaded from a register, it comes from the upper half of the lower byte:



x<y,z> Bits y through z of the Register x. For example, R7 <15,8> represents the upper byte of the Program Counter.

(,2) Indicates that the operand ",2" is optional.

⎺⊔⎺ A low pulse

[ ] Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.

[[ ]] Implied memory addressing; the contents of the memory location designated by the contents of a register.

Λ Logical And

∀ Logical Exclusive-OR

± Addition or subtraction of a displacement, depending on the sign bit in the object code.

← Data is transferred in the direction of the arrow.

Table 15-2. CP1600 Instruction Set Summary

TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
PRIMARY I/O AND MEMORY REFERENCE	MVI	ADR,RD	2					[RD]←[ADDR] Load register from memory, using direct addressing.
	MVI@	RM,RD	1					[RD]←[[RM]] Load register from memory, using implied addressing.
	MVO	RS,ADDR	2					[ADDR]←[RS] Store register to memory, using direct addressing.
	MVO@	RS,RM	1					[[RM]]←[RS] Store register to memory, using implied addressing. If RS=R4, R5, R6 or R7, then RS=RM is not supported.
SECONDARY I/O AND MEMORY REFERENCE	ADD	ADDR,RD	2	X	X	X	X	[RD]←[RD] + [ADDR] Add memory contents to register, using direct addressing.
	ADD@	RM,RD	1	X	X	X	X	[RD]←[RD] + [[RM]] Add memory contents to register, using implied addressing.
	SUB	ADDR,RD	2	X	X	X	X	[RD]←[RD] - [ADDR] Subtract memory contents from register, using direct addressing.
	SUB@	RM,RD	1	X	X	X	X	[RD]←[RD] - [[RM]] Subtract memory contents from register, using implied addressing.
	CMP	ADDR,RS	2	X	X	X	X	[RS] - [ADDR] Compare memory contents with registers, using direct addressing. Only the status flags are affected.
	CMP@	RM,RS	1	X	X	X	X	[RS] - [[RM]] Compare memory contents with register's, using implied addressing. Only the status flags are affected.
	AND	ADDR,RD	2	X	X			[RD]←[RD] Λ [ADDR] AND memory contents with those of register, using direct addressing.
	AND@	RM,RD	1	X	X			[RD]←[RD] Λ [[RM]] AND memory contents with those of register, using implied addressing.
	XOR	ADDR,RD	2	X	X			[RD]←[RD] ⊻ [ADDR] Exclusive-OR memory contents with those of register, using direct addressing.
	XOR@	RM,RD	1	X	X			[RD]←[RD] ⊻ [[RM]] Exclusive-OR memory contents with those of register, using implied addressing.

Table 15-2. CP1600 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
IMMEDIATE	MVII	DATA,RD	2					[RD]←DATA Load immediate to specified register.
	MVOI	RS,DATA	2					[[R7] + 1]←[RS] Store contents of specified register in immediate field of MVOI instruction. This is only possible if program memory is read/write memory (rather than ROM).
IMMEDIATE OPERATE	ADDI	DATA,RD	2	X	X	X	X	[RD]←[RD] + DATA Add immediate to specified register.
	SUBI	DATA,RD	2	X	X	X	X	[RD]←[RD] - DATA Subtract immediate data from specified register.
	CMPI	DATA,RS	2	X	X	X	X	[RD] - DATA Compare immediate data with contents of specified register. Only the status flags are affected.
	ANDI	DATA,RD	2	X	X			[RD]←[RD] ∧ DATA AND immediate data with contents of specified register.
	XORI	DATA,RD	2	X	X			[RD]←[RD]⊻DATA Exclusive-OR immediate data with contents of specified register.
JUMP	J	LABEL	3					[R7]←LABEL Jump to given address.
	JR	RS	1	X	X			[R7]←[RS] Jump to address contained in specified register.
	JSR	RB,LABEL	3					[RB]←[R7]; [R7]←LABEL Jump to given address, saving Program Counter in R4, R5, or R6.
	B	DISP	2					[R7]←[R7] + 2±DISP Branch relative to Program Counter contents.
BRANCH ON CONDITION	Bcond	DISP	2					If cond is true, [R7]←[R7] + 2±DISP Branch relative on given condition; otherwise, execute next sequential instruction.
	BEXT	DISP,E	2					EBCA0-3 ← E; If EBCI=1, [R7]←[R7] + 2±DISP Branch relative if external condition is true.

Table 15-2. CP1600 Instruction Set Summary (Continued)

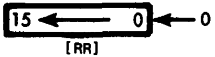
TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
REGISTER-REGISTER MOVE AND OPERATE	MOVR	RS,RD	1	X	X			[RD]←[RS] Move contents of source register to destination register.
	ADDR	RS,RD	1	X	X	X	X	[RD]←[RS] + [RD] Add contents of specified registers.
	SUBR	RS,RD	1	X	X	X	X	[RD]←[RD] - [RS] Subtract contents of source register from those of destination register.
	CMPR	RS,RD	1	X	X	X	X	[RD] - [RS] Compare registers' contents. Only the status flags are affected.
	ANDR	RS,RD	1	X	X			[RD]←[RD] Λ [RS] AND contents of specified registers.
	XORR	RS,RD	1	X	X			[RD]←[RD]⊻[RS] Exclusive-OR contents of specified registers.
REGISTER OPERATE	CLRR	RD	1	0	1			[RD]←[RD] V [RD] Clear specified register.
	TSTR	RS	1	X	X			[RS]←[RS] Test contents of specified register.
	INCR	RD	1	X	X			[RD]←[RD] + 1 Increment contents of specified register.
	DECR	RD	1	X	X			[RD]←[RD] - 1 Decrement contents of specified register.
	COMR	RD	1	X	X			[RD]←[RD] Complement contents of specified register (ones complement).
	NEGR	RD	1	X	X	X	X	[RD]←[RD] + 1 Negate contents of specified register (twos complement).
	ADCR	RD	1	X	X	X	X	[RD]←[RD] + [C] Add Carry bit to specified register contents.
	SLL	RR(,2)	1	X	X			 Shift logical left one or two bits, clearing bit 0 (and bit 1 if shifting twice).

Table 15-2. CP1600 Instruction Set Summary (Continued)

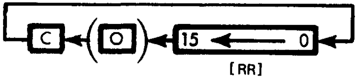
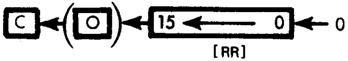
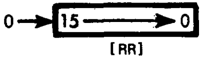
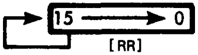
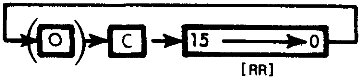
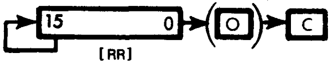
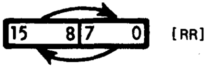
TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
REGISTER OPERATE (CONTINUED)	RLC	RR(,2)	1	X	X	X	2	 Rotate left one bit through Carry, or rotate 2 bits left through Overflow and Carry.
	SLLC	RR(,2)	1	X	X	X	2	 Shift logical left one bit into Carry, clearing bit 0, or shift left two bits into Overflow and Carry, clearing bits 0 and 1.
	SLR	RR(,2)	1	X	X			 Shift logical right one or two bits, clearing bit 15 (and bit 14 if shifting twice).
	SAR	RR(,2)	1	X	X			 Shift arithmetic right one or two bits, copying high order bit.
	RRC	RR(,2)	1	X	X	X	2	 Rotate right one bit through Carry, or rotate two bits right through Overflow and Carry.
	SARC	RR(,2)	1	X	X	X	2	 Shift arithmetic right one bit into Carry, or two bits into Overflow and Carry.
	SWAP	RR(,2)	1	X	X			 Swap bytes of register once, or twice.

Table 15-2. CP1600 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES S	Z	C	O	OPERATION PERFORMED
STACK	PSHR	RS	1					Separate mnemonics for MVO@ RS,R6.
	PULR	RD	1					Separate mnemonic for MVI@ R6,RD.
INTERRUPT	SIN	(2)	1					$\overline{PCIT}$ ← ⊔ Software interrupt.
	EIS		1					Enable interrupt syetem.
	DIS		1					Disable interrupt system.
	TCI		1					Terminate current interrupt.
	JE	LABEL	3					Jump to given address and enable interrupt system.
	JD	LABEL	3					Jump to given address and disable interrupt system.
	JSRE	RB,LABEL	3					Jump to given address, saving Program Counter in R4, R5 or R6, and enable interrupt system.
	JSRD	RB,LABEL	3					Jump to given address, saving Program Counter in R4, R5 or R6, and disable interrupt system.
STATUS	GSWD	RD	1					[RD<15,12>]←[SW]; [RD<7,4>]←[SW] Place Status Word in upper half of each byte of the specified register. RD may be R0, R1, R2 or R3.
	RSWD	RS	1	X	X	X	X	[SW]←[RS<7,4>] Load Status Word from bits 7 through 4 of the specified register.
	CLRC		1			0		[C]←0 Clear Carry.
	SETC		1			1		[C]←1 Set Carry.
	NOPP		2					No Operation.
	NOP	(2)	1					
	HLT		1					Halt after executing next instruction.
	SDBD		1					Set double byte data mode for next instruction, which must be of one of the following types: Primary or secondary I/O or memory reference Immediate or immediate operate If implied addressing through R1, R2, or R3 is used, the same byte will be accessed twice; addressing through R4, R5, or R7 will give bytes from the addressed location and that addressed after auto-increment. Direct addressing and Stack addressing are not allowed in double byte mode.

Table 15-3. CP1600 Branch Conditions And Corresponding Codes

MNEMONIC	BRANCH CONDITION	OBJECT CODE DESIGNATION
C LGT	$C = 1$ Carry (logical greater than)	0001
NC LLT	$C = 0$ No Carry (logical less than)	1001
OV	$O = 1$ Overflow	0010
NOV	$O = 0$ No overflow	1010
PL	$S = 0$ Plus	0011
MI	$S = 1$ Minus	1011
ZE EQ	$Z = 1$ Zero (equal)	0100
NZE NEQ	$Z = 0$ Nonzero (not equal)	1100
LT	$S \veebar O = 1$ Less than	0101
GE	$S \veebar O = 0$ Greater than or equal	1101
LE	$Z \vee (S \veebar O) = 1$ Less than or equal	0110
GT	$Z \vee (S \veebar O) = 0$ Greater than	1110
USC	$C \veebar S = 1$ Unequal sign and carry	0111
ESC	$C \veebar S = 0$ Equal sign and carry	1111

The following notation is used in Table 15-4:

Where 10 digits are shown, they are the 10 low order bits of a 10 to 16 bit word. (Word size depends on the system implementation.) Where 4 digits are shown, they represent the hexadecimal notation for an entire word (10 to 16 bits).

bb Two bits indicating one of the first three General Purpose registers.

cccc Four bits giving the branch condition, as shown in Table 15-3.

ddd Three bits indicating a destination register, RD.

eeee Four bits giving the external branch condition, E.

IIII One word of immediate data.

mmm Three bits indicating a Data Counter Register RM.

m One bit indicating the number of rotates or shifts:
0 one bit position
1 two bit positions

p One bit of immediate address

P One hexadecimal digit (4 bits) of immediate address.

rr Two bits indicating one of the first four General Purpose registers.

sss Three bits indicating a source register, RS.

z Sign of the displacement:
0 add the displacement to PC contents
1 subtract the displacement from PC contents

In the "Machine Cycles" column, when two numbers are given with one slash between (e.g., 7/9), execution time depends on whether or not a branch is taken. When two numbers are given, separated by two slashes (such as 8//11), execution time depends on which register contains the implied address.

Table 15-4. CP1600 Instruction Set Object Codes

INSTRUCTION	OBJECT CODE	WORDS	MACHINE CYCLES
ADCR RD	0000101ddd	1	6
ADD ADDR,RD	1011000ddd PPPP	2	10
ADD@ RM,RD	1011mmmddd	1	8//11
ADDI DATA,RD	1011111ddd IIII	2	8
ADDR RS,RD	0011sssddd	1	6
AND ADDR,RD	1110000ddd PPPP	2	10
AND@ RM,RD	1110mmmddd	1	8//11
ANDI DATA,RD	1110111ddd IIII	2	8
ANDR RS,RD	0110sssddd	1	6
B DISP	1000z00000 PPPP	2	7/9
Bcond DISP	1000z0cccc PPPP	2	7/9
BEXT DISP,E	1000z1eeee PPPP	2	7/9
CLRC	0006	1	4
CLRR RD	0111dddddd	1	6
CMP ADDR,RS	1101000sss PPPP	2	10
CMP@ RM,RS	1101mmmsss	1	8//11
CMPI DATA,RS	1101111sss IIII	2	8
CMPR RS,RD	0101sssddd	1	6
COMR RD	0000011ddd	1	6
DECR RD	0000010ddd	1	6
DIS	0003	1	4
EIS	0002	1	4
GSWD RR	00001100rr	1	6
HLT	0000	1	4
INCR RD	0000001ddd	1	6
J LABEL	0004 11pppppp00 PPPP	3	12
JD LABEL	0004 11pppppp10 PPPP	3	12
JE LABEL	0004 11pppppp01 PPPP	3	12
JR RS	0010sss111	1	7
JSR RB,LABEL	0004 bbpppppp00 PPPP	3	12
JSRD RB,LABEL	0004 bbpppppp10 PPPP	3	12
JSRE RB,LABEL	0004 bbpppppp01 PPPP	3	12
MOVR RS,RD	0010sssddd	1	6//7
MVI ADDR,RD	1010000ddd PPPP	2	10
MVI@ RM,RD	1010mmmddd	1	8//11

Table 15-4. CP1600 Instruction Set Object Codes (Continued)

INSTRUCTION	OBJECT CODE	WORDS	MACHINE CYCLES
MVII DATA,RD	1010111ddd	2	8
	IIII		
MVO RS,ADDR	1001000sss	2	11
	PPPP		
MVO@ RS,RM	1001mmmsss	1	9
MVOI RS,DATA	1001111sss	2	9
	IIII		
NEGR RD	0000100ddd	1	6
NOP (2)	000011010m	1	6
NOPP	1000z01000	2	7
	PPPP		
PSHR RS	1001110sss	1	9
PULR RD	1010110ddd	1	11
RLC RR(,2)	0001010mrr	1	6/8
RRC RR(,2)	0001110mrr	1	6/8
RSWD RS	0000111sss	1	6
SAR RR(,2)	0001101mrr	1	6/8
SARC RR(,2)	0001111mrr	1	6/8
SDBD	0001	1	4
SETC	0007	1	4
SIN (2)	000011011m	1	6
SLL RR(,2)	0001001mrr	1	6/8
SLLC RR(,2)	0001011mrr	1	6/8
SLR RR(,2)	0001100mrr	1	6/8
SUB ADDR,RD	1100000ddd	2	10
	PPPP		
SUB@ RM,RD	1100mmmddd	1	8//11
SUBT DATA,RD	1100111ddd	2	8
	IIII		
SUBR RS,RD	0100sssddd	1	6
SWAP RR(,2)	0001000nrr	1	6/8
TCI	0005	1	4
TSTR RS	0010ssssss	1	6//7
XOR ADDR,RD	1111000ddd	2	10
	PPPP		
XOR@ RM,RD	1111mmmddd	1	8//11
XORI DATA,RD	1111111ddd	2	8
	IIII		
XORR RS,RD	0111sssddd	1	6

THE BENCHMARK PROGRAM

For the CP1600 our benchmark program may be illustrated as follows:

```
      MVII   IOBUF,R4   LOAD THE I/O BUFFER STARTING ADDRESS INTO R4
      MVII   TABLE,R1   LOAD THE TABLE STARTING ADDRESS INTO R1
      MVI@   R1,R5      LOAD ADDRESS OF FIRST FREE TABLE WORD INTO R5
      MVII   CNT,R2     LOAD WORD COUNT INTO R2
LOOP  MVI@   R4,R0      LOAD NEXT DATA WORD FROM IOBUF
      MVO@   R0,R5      STORE IN NEXT TABLE WORD
      DECR   R2         DECREMENT WORD COUNT
      BNZE   LOOP       RETURN IF NOT END
      MVO@   R5,R1      RETURN ADDRESS OF NEXT FREE TABLE BYTE
```

This benchmark program makes very few assumptions. The input table IOBUF and the data table TABLE can have any length, and can reside anywhere in memory. The address of the first free word in TABLE is stored in the first word of the TABLE.

ELECTRICAL DATA

The following pages contain specific electrical and timing data for the General Instruments CP1600 CPU.

MAXIMUM RATINGS*

V_{DD}, V_{CC}, GND and all other input/output voltages with respect to V_{BB}	−0.3V to +18.0V
Storage Temperature	−55°C to +150°C
Operating Temperature	0°C to +70°C

*Exceeding these ratings could cause permanent damage to this device. Functional operation at this condition is not implied—operating conditions are specified below.

ELECTRICAL CHARACTERISTICS

Standard Operating Conditions (unless otherwise noted):
All voltages referenced to GND

V_{DD}=+12 V ±5%, 70mA(typ)
V_{CC}=+5 V ±5% , 12mA(typ)
V_{BB}=−3 V ±10% , 0.2mA (typ)
Temperature (T_A) = 0°C to +70°C

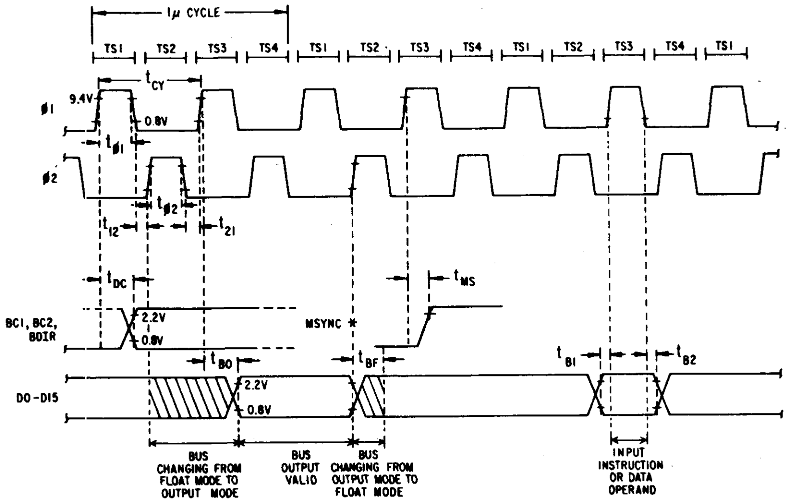
DC CHARACTERISTICS

CHARACTERISTIC		SYMBOL	MIN	TYP	MAX	UNITS	CONDITIONS
Clock Inputs:	High	V_{IHC}	10.4		V_{DD}	V	
	Low	V_{ILC}	0		0.5	V	
Logic Inputs:	Low	V_{IL}	0		0.65	V	
All Lines except BDRDY	High	V_{IH}	2.4		V_{CC}	V	
Bus Data Ready Line (See Note 1)	High	V_{IHB}	3.0		V_{CC}	V	
Logic Outputs:	High	V_{OH}	2.4	V_{CC}		V	I_{OH} = 100μA
Data Bus Lines D0-D15	Low	V_{OL}			0.5	V	I_{OL} = 1.6mA
Bus Control Lines, BC1, BC2, BDIR					0.45	V	I_{OL} = 2.0mA
All Other					0.45	V	I_{OL} = 1.6mA
AC CHARACTERISTICS (all notations refer to Bus Timing Diagram, unless otherwise noted)							
Clock Pulse Inputs, $\phi1$ or $\phi2$:							
Pulse Width		$t\phi1$, $t\phi2$	120 (CP1600) 70 (CP1600A)			ns	1 TTL
Skew ($\phi1$-$\phi2$ delay)		t12, t21	0			ns	Load &
Clock Period		t_{cy}	0.3 (CP1600) 0.2 (CP1600A)		2.0	μs	25 pF
Rise & Fall Times		tr, tf			15	ns	↓
Master SYNC: Delay from $\phi1$		tms			30	ns	
D0-D15 Bus Signals: output delay from $\phi1$ (float to output)		t_{BO}			120 (CP1600) 70 (CP1600A)	ns	
output delay from $\phi2$ (output to float)		t_{BF}		50		ns	↓
input setup time before $\phi1$		tB1	0			ns	
input hold time after $\phi1$		tB2	10			ns	
Bus Control Signals: BC1, BC2, BDIR							
Output delay from $\phi1$		t_{DC}			120 (CP1600) 70 (CP1600A)	ns	1 TTL
BUSAK Output delay from $\phi1$		t_{BU}		150		ns	Load &
TCI Output delay from $\phi1$		t_{TO}		200		ns	25 pF
TCI Pulse Width		t_{TW}		300		ns	↓
EBCA output delay from BEXT input		t_{DE}			150	ns	
EBCA wait time for EBCI input		t_{AI}			400	ns	
CAPACITANCE (TA = +25°C; V_{DD} = + 12V; V_{CC} = +5V; V_{BB} = −3V; $t\phi1$ = $t\phi2$ = 120ns)							
$\phi1$, $\phi2$ Clock Input Capacitance		$C\phi1$, $C\phi2$		20	30	pF	
Input Capacitance:							
D0-D15		CIN		6	12	pF	
All Other				5	10	pF	
Output Capacitance:							
D0-D15 in high impedance state				8	15	pF	

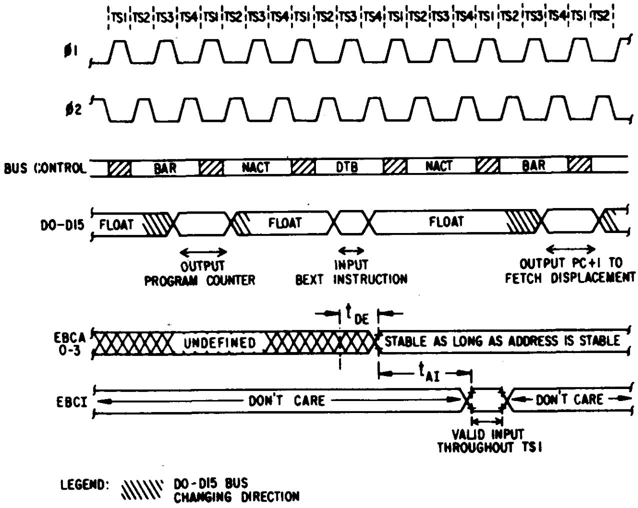
NOTES:

1. The Bus Data ReaDY (BDRDY) line is sampled during time period TSI after a BAR or ADAR bus control signal. BDRDY must go low requesting a wait state 50 ns before the end of TS1 and remain low for 50 ns minimum. BDRDY may go high asynchronously. In response to BDRDY, the CPU will extend bus cycles by adding additional microcycles up to a maximum of 40 μsec duration.

BUS TIMING DIAGRAM



TYPICAL INSTRUCTION SEQUENCE



BRANCH ON EXTERNAL CONDITION INSTRUCTION

Chapter 16
THE TMS9900

The TMS9900, manufactured by Texas Instruments, is a single chip central processing unit, which can compete effectively with minicomputers in interrupt-driven, signal processing applications.

Texas Instruments advertises the TMS9900 as a produrt which can be used at these three ascending levels:

1) As a single chip central processing unit, surrounded by custom-designed logic; that is, in a typical microprocessor digital logic application.
2) As a single board, minicomputer-type central processing unit.
3) As a total minicomputer replacement, complete with memory, device controllers and the normal logic found within a typical minicomputer system.

Programs written for a TMS9900 at any one of the three levels can, in theory, be used at any higher level. Texas Instruments advertises this upward program compatibility as an important feature of the various product levels. There is merit to Texas Instruments' program compatibility claims. The TMS9900 does indeed have a very powerful instruction set that is more independent of external hardware configurations than most; but that is a two-edged sword. The TMS9900 instruction set gains its upward compatibility by making a number of specific demands on external logic. But that makes the TMS9900 an impractical device to use in simple digital logic applications; package counts needed to implement the demands made on external logic result in TMS9900 configurations which are not price competitive. Therefore, when reading this chapter, look upon the TMS9900 as a product well suited to complex, low volume applications that require a significant amount of program generation.

The TMS9900 CPU is the only device described in this chapter. Additional support devices are advertised by Texas Instruments, although many of them are not yet available in commercial quantities.

The TMS9900 is manufactured using N-channel silicon gate MOS technology. It is packaged as a 64-pin DIP. Three power supplies are required: -5V, +5V and +12V.

Using a 333 nanosecond clock, instruction execution times range between 3 and 10 microseconds.

A TMS9900 FUNCTIONAL OVERVIEW

Figure 16-1 illustrates that part of our general microcomputer system logic which is implemented by the TMS9900 CPU.

The most important features of Figure 16-1 are:

1) **The absence of programmable registers, and**
2) **The presence of significant interrupt handling logic.**
3) **Serial-to-parallel data conversion logic.**

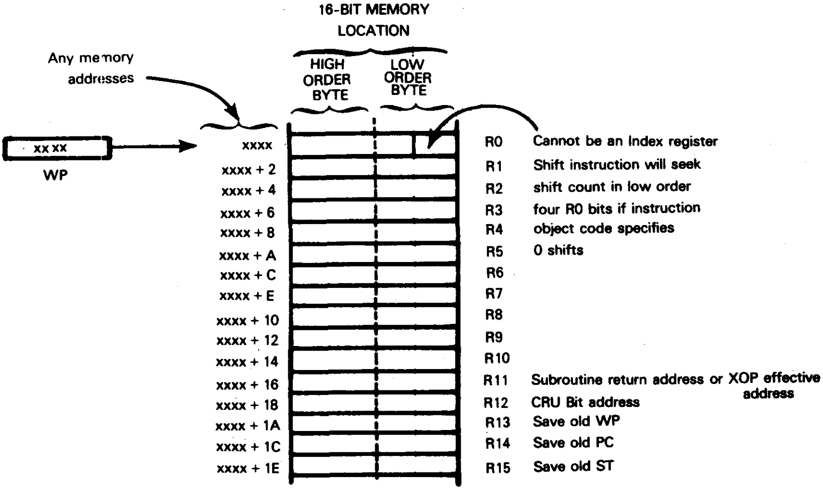
Let us first consider the manner in which the TMS9900 handles programmable registers.

TMS9900 PROGRAMMABLE REGISTERS

Within the logic of the TMS9900 itself, there are just three 16-bit programmable registers: a Program Counter, a Workspace register and a Status register.

The Program Counter and Status register are straightforward. The Program Counter always addresses the next instruction to be executed. The Status register maintains various statuses which we describe later in this chapter.

The Workspace register is a unique and powerful programming feature of the TMS9900. This register identifies the first of sixteen 16-bit memory locations which act as 16 General Purpose registers. This may be illustrated as follows:



Observe that the TMS9900 addresses memory in byte units; memory addresses that identify registers therefore increment in units of 2.

Each of the 16 General Purpose registers may be used to store data or addresses. Thus, **each General Purpose register may serve as an Accumulator or as a Data Counter.**

Registers R11 through R15 are used as special address storage buffers; we will be describing the way in which these registers are used as the chapter proceeds.

Having 16 General Purpose registers in read/write memory, rather than in the CPU, is the single most important feature of the TMS9900.
The advantage of having 16 General Purpose registers located anywhere in read/write memory is that you can have many sets of 16 General Purpose registers with the single limitation that they must fit within the available address space. For example, following an interrupt acknowledge, you no longer need to save the contents of General Purpose registers—all you need to do is save the contents of the Program Counter, the Workspace register and the Status register and that is done automatically by TMS9900 interrupt handling logic. By loading new values into the Program Counter and the Workspace register, you can begin executing a new program, accessing 32 new memory bytes—which will be treated as a new set of 16 General Purpose registers.

The disadvantage of having 16 General Purpose registers in read/write memory is that no TMS9900 microcomputer system can be configured without read/write memory; and if you are going to use many different sets of 16-bit registers, then you are going to require a significant amount of read/write memory. Furthermore, you lose the speed associated with executing register-to-register operations; there are no source and destination locations left in the CPU and every register access becomes a memory access.

TMS9900 CONTEXT SWITCH

TMS9900 literature refers to the process of switching from one set of General Purpose registers to another as a context switch.

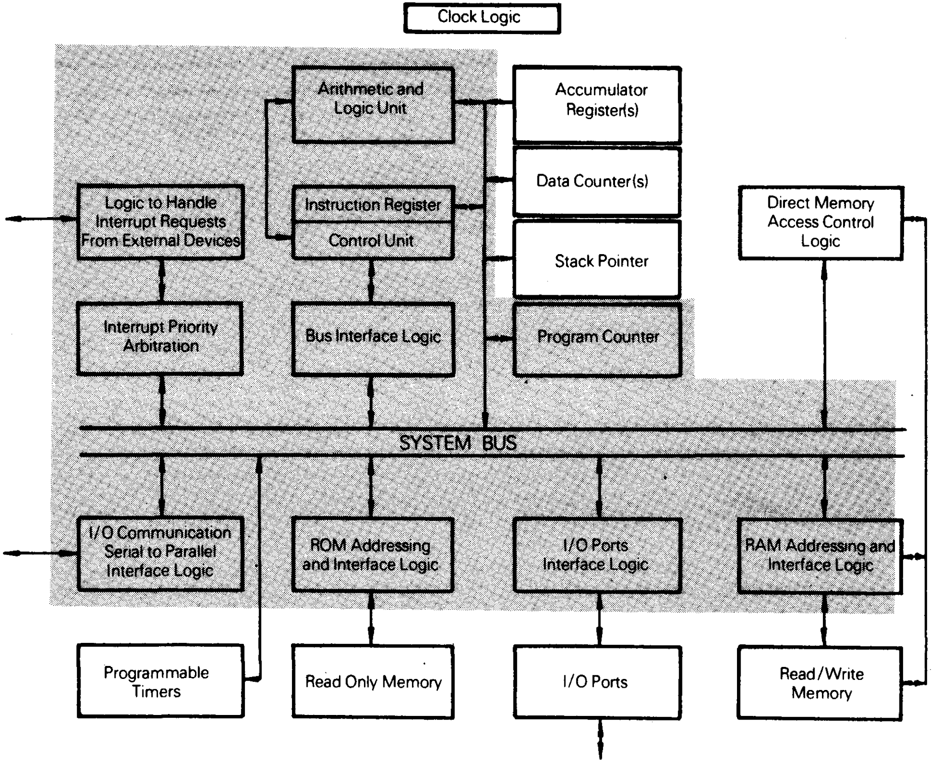

Clock Logic
Arithmetic and Logic Unit
Accumulator Register(s)
Data Counter(s)
Logic to Handle Interrupt Requests From External Devices
Instruction Register
Control Unit
Direct Memory Access Control Logic
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

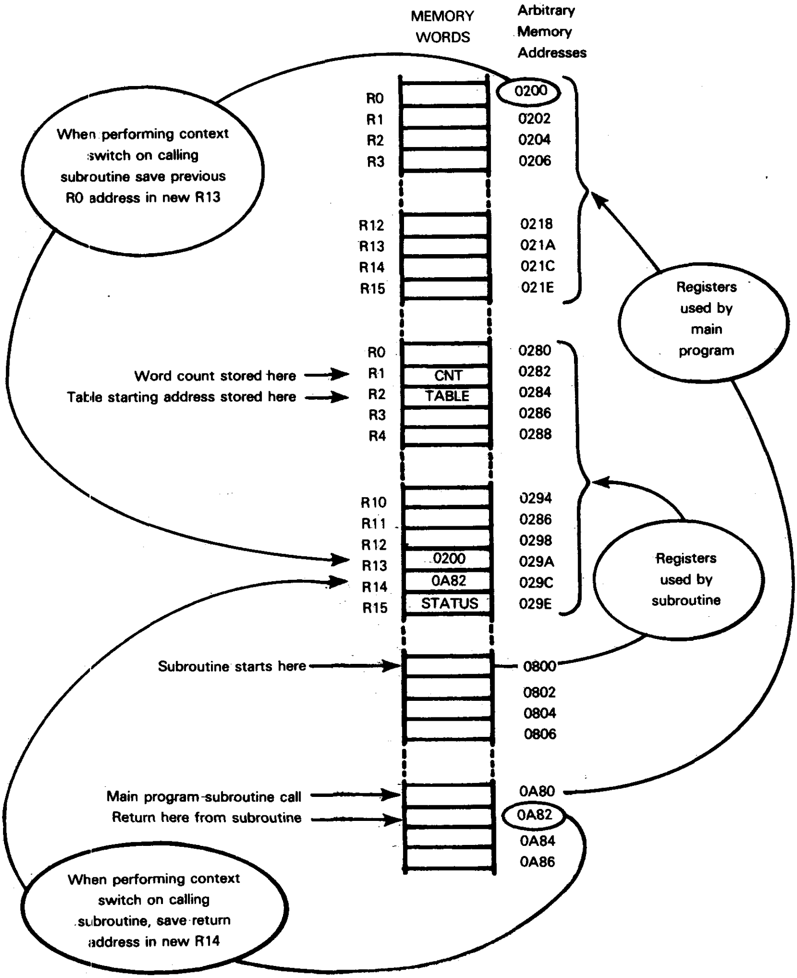
Figure 16-1. Logic Of The TMS9900 CPU

Special instructions allow you to perform a context switch or a backward context switch.

During a forward context switch, you load new values into the Workspace register and Program Counter, while simultaneously saving the old Workspace register, Program Counter and Status register contents in the new General Purpose Registers R13, R14 and R15.

A reverse context switch loads the current contents of General Purpose Registers R13, R14 and R15 into the Workspace register, Program Counter and Status register, respectively, thus returning you to your previous set of General Purpose registers.

You can perform context switches as often as you like and whenever you like. For example, a very effective way of using context switching is to group data into contiguous memory words which you can identify as a register set. Upon entering a subroutine, you can perform a context switch which automatically creates all necessary initial data and address values in appropriate General Purpose registers. This may be illustrated as follows:



MEMORY WORDS
Arbitrary Memory Addresses
When performing context switch on calling subroutine save previous R0 address in new R13
R0 0200
R1 0202
R2 0204
R3 0206
R12 0218
R13 021A
R14 021C
R15 021E
Registers used by main program
R0 0280
Word count stored here
R1 CNT 0282
Table starting address stored here
R2 TABLE 0284
R3 0286
R4 0288
R10 0294
R11 0286
R12 0298
R13 0200 029A
R14 0A82 029C
R15 STATUS 029E
Registers used by subroutine
Subroutine starts here
0800
0802
0804
0806
Main program-subroutine call
0A80
Return here from subroutine
0A82
0A84
0A86
When performing context switch on calling subroutine, save return address in new R14

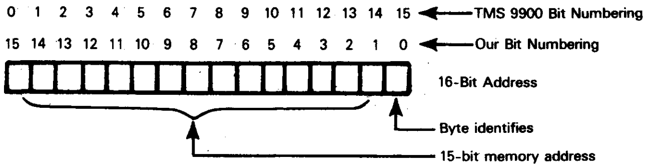
The illustration above sets up parameters as needed by the benchmark program we have used to illustrate instruction sets for all microcomputers in this book.

TMS9900 MEMORY ADDRESSING MODES

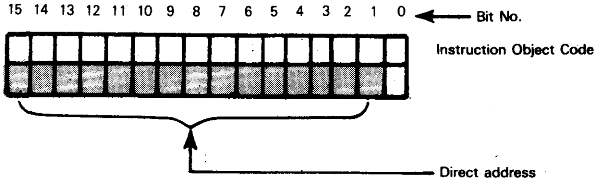
The TMS9900 provides these four methods of addressing memory:

1) **Direct memory addressing**
2) **Direct, indexed memory addressing**
3) **Implied memory addressing**
4) **Implied memory addressing with auto-increment**

The way in which the TMS9900 implements these four memory addressing modes is exactly as described in Volume I, Chapter 6. The important point to note is that the TMS9900 looks upon its address space as consisting of 32,768 16-bit memory words which are addressed using 15, rather than 16 Address Bus lines; yet all addresses are computed as 16-bit words. What happens is that when a 16-bit address is computed by logic within the TMS9900 CPU, the low order bit is interpreted as a byte discriminator and is not output. Thus, bits 1 through 15 become the effective memory address, while **instructions that operate on bytes discriminate between the high order and low order byte using the low order address bit.** This may be illustrated as follows:



Direct memory addressing instructions provide the memory address in the second word of instruction object code:

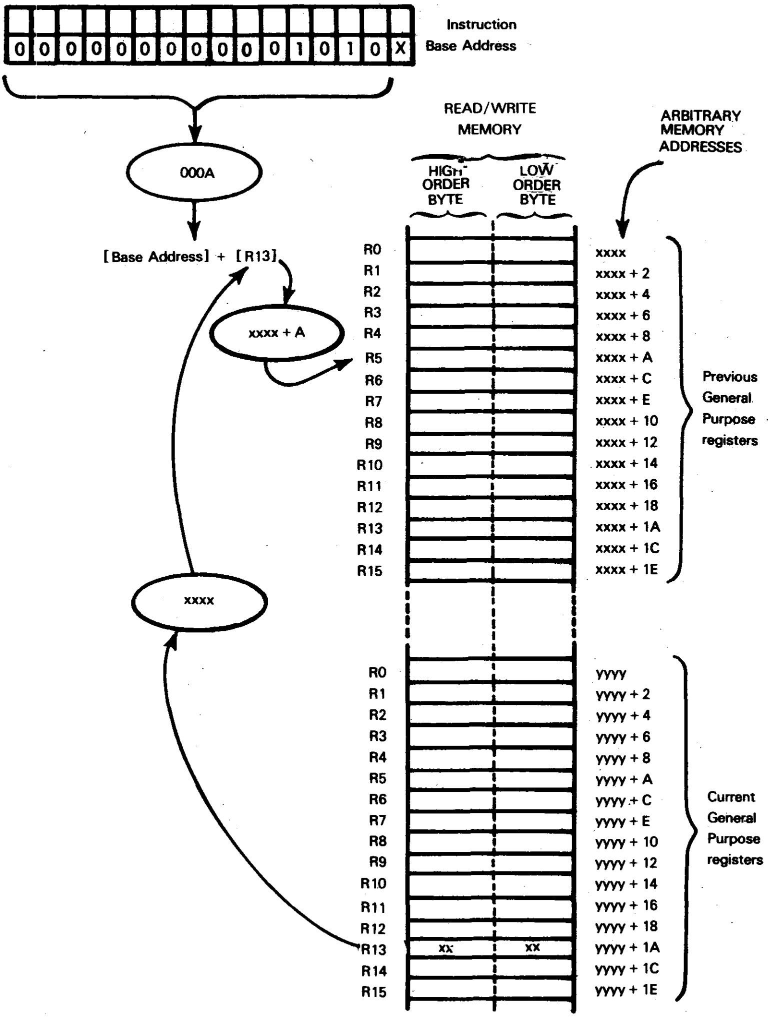


Direct, indexed memory addressing instructions provide a base address in the second object code word, but they also identify a General Purpose register whose contents are to be added to the base address. Again the low order bit of the computed address is discarded.

General Purpose Register R0 cannot be specified as an Index register.

Direct, indexed addressing makes more sense in a TMS9900 microcomputer than it does in a simpler logic replacement device, because the TMS9900 may well be used in applications that need a great deal of minicomputer-like programming. Also, you can address the previous set of General Purpose registers, following a context switch, without knowing where the previous registers were. Suppose you want to access the contents of the memory word which was being used as General Purpose Register R5 before you switched to your current set of General Purpose registers. Recall that the previous Workspace register contents are stored in your current General

Purpose Register R13. You could thus address the previous General Purpose Register R5, without having to know where this General Purpose register may have been, by using direct indexed addressing as follows:



An implied memory addressing instruction will specify one of the 16 current General Purpose registers as providing the effective memory address.

If you specify implied memory addressing with auto-increment, then the contents of the identified General Purpose register will be incremented after the memory access has been performed.

TMS9900 Jump instructions use program relative, direct addressing. These are one-word instructions, where the low order byte of the instruction object code provides an 8-bit, signed binary value, which is added to the incremented contents of the Program Counter. This is straightforward program relative, direct addressing.

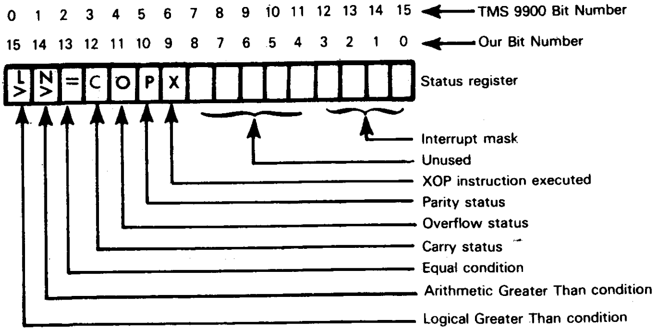
The TMS9900 provides two ways to address an I/O device:

1) As a memory location
2) As a separate bit field, up to 4,096 bits wide.

The second method of I/O addressing is very unusual as compared to other microcomputers described in this book. Since the bit field is accessed via special control signals, we will postpone a discussion of I/O addressing until later in the chapter after we have described CPU pins and signals.

TMS9900 STATUS FLAGS

The TMS9900 CPU has a 16-bit Status register which may be illustrated as follows:



The low order four bits of the Status register represent an interrupt mask which identifies the level of interrupt, if any, which is currently enabled. As the 4-bit interrupt mask would imply, 16 levels of interrupt are allowed. We will describe interrupt processing later in this chapter.

The X status is set to 1 while an XOP instruction is being executed. This instruction allows you to perform a software interrupt -- as described later in this chapter.

The P, O and C are standard Parity, Overflow and Carry statuses.

The Equal status (=) identifies conditions which will cause a Branch-if-Equal instruction to branch. An I/O bit to be tested also gets stored in the Equal status.

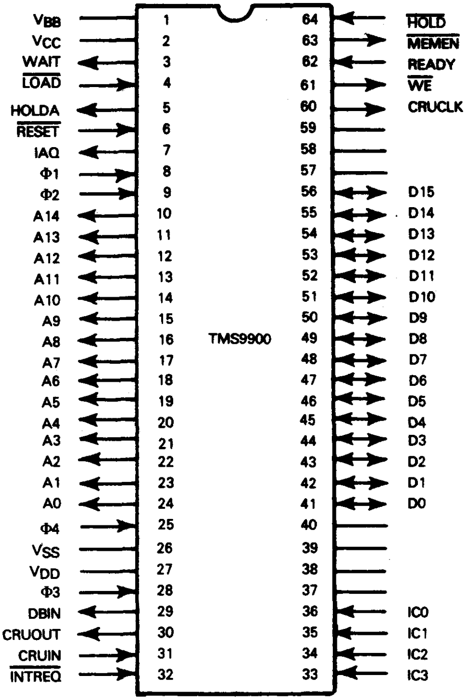
The Logical Greater Than and Arithmetic Greater Than statuses are set or reset following arithmetic logical or data move operations. **A Logical Greater Than treats the source data as simple, unsigned binary numbers. An arithmetic Greater Than interprets the operand as signed binary numbers.**

TMS9900 CPU PINS AND SIGNALS

Figure 16-2 illustrates the pins and signals of the TMS9900 CPU.

Being a 64-pin DIP, the TMS9900 can afford to have separate Address and Data Busses.

Pins A0 - A14 provide the 15-bit Address Bus. Note that Texas Instruments' literature numbers bits and pins from left to right -- therefore address line A0 represents the most significant address bit, whereas address line A14 represents the least significant address bit.



Pin Name	Description	Type
A0 - A4	Address Bus	Tristate, Output
D0 - D15	Data Bus	Tristate, Bidirectional
Φ1, Φ2, Φ3, Φ4	Clock Signals	Input
MEMEN	Memory Enable	Output
IAQ	Instruction Fetch	Output
DBIN	Data Bus In	Output
WE	Write Enable	Output
READY	Memory Ready	Input
WAIT	Wait State Indicator	Output
CRUCLK	I/O Clock	Output
CRUOUT	Serial I/O Out	Output
CRUIN	Serial I/O In	Input
INTREQ	Interrupt Request	Input
IC0 - IC3	Interrupt Code	Input
HOLD	DMA Request	Input
HOLDA	Hold Acknowledge	Output
LOAD	Load Interrupt	Input
RESET	Reset	Input
VBB, VCC, VDD, VSS	Power and Ground	

Figure 16-2. TMS9900 Signals And Pin Assignments

D0 - D15 provides a 16-bit bidirectional Data Bus. Once again, D0 represents the most significant data bit.

Remaining signals may be divided into bus control, interrupt control, and timing.

External logic must provide four clock signals, Φ1, Φ2, Φ3, and Φ4.

Any memory access operation begins with an address being output via the Address Bus. **The TMS9900 CPU identifies a stable address on the Address Bus by outputting $\overline{\text{MEMEN}}$ low.**

If the memory access operation is an instruction fetch, then IAQ is output high.

If the memory access is a read, then the TMS9900 outputs a high pulse via DBIN. Memory interface logic must interpret the high DBIN pulse as a signal to place data on the Data Bus.

If the memory access is a memory write, then the TMS9900 CPU outputs a low pulse via $\overline{\text{WE}}$. Memory interface logic must use the low $\overline{\text{WE}}$ pulse as a signal to read data off the Data Bus and store it in the addressed memory location.

When external logic cannot respond to a memory access in the available time, it requests a Wait state by inputting READY high. The CPU acknowledges by outputting WAIT high.

CRUCLK, CRUIN and CRUOUT are three signals used to implement single-bit or serial data transfers.

CRUOUT is used to output bits of data to the I/O devices and CRUIN is used to retrieve input data from the I/O devices. CRUCLK is active during output operations only and defines when data bits on CRUOUT are valid.

Let us now look at interrupt control signals.

There is a single interrupt request input, $\overline{\text{INTREQ}}$, which must be held low by any external device requesting an interrupt. External devices identify themselves via control signals IC0 - IC3. Thus, an interrupt request must be accompanied by the appropriate input at IC0 - IC3.

Observe that there is no interrupt acknowledge signal.

For DMA operations, external logic requests access to the System Bus by inputting $\overline{\text{HOLD}}$ low. The CPU acknowledges the Hold request by outputting HOLDA high.

$\overline{\text{LOAD}}$ is a nonmaskable interrupt.

$\overline{\text{RESET}}$ is a typical system Reset signal. However, TMS9900 Reset logic uses the device's interrupt capabilities; therefore we will describe the Reset operation in detail when discussing TMS9900 interrupt capabilities in general.

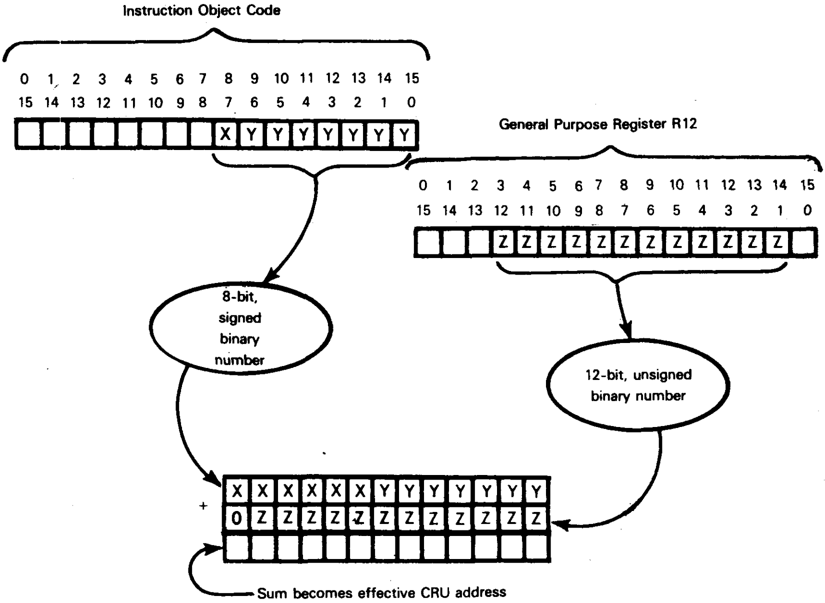
TMS9900 I/O ADDRESSING

As compared to other microcomputers described in this book, the TMS9900 exhibits unusual I/O addressing. **In addition to addressing I/O devices as memory locations, you can address a separate field of up to 4,096 bits. Texas Instruments' literature refers to this field as the "Communications Register Unit" (CRU).** If you are programming a TMS9900 microcomputer system that has already been configured by Texas Instruments, then it is justifiable to look upon the Communications Register Unit as a form of I/O port. If you are building your own interface to a TMS9900 CPU, then instructions that are supposed to access the Communications Register Unit, in reality simply make alternative use of the Data and Address Busses, in conjunction with three control signals, CRUCLK, CRUIN and CRUOUT.

There are two classes of TMS9900 CRU instructions. The first class accesses individual bits or signals while the second class accesses bit fields that may range between 1 and 16 bits.

There are three single-bit CRU instructions; they set, reset or test the identified CRU bit. This is equivalent to setting, resetting or testing an external signal, or single I/O port bit. When a bit is to be set or reset, the new level is output via CRUOUT, and a CRUCLK pulse indicates that a bit is being output. When the condition of a bit is to be tested, then external logic is required to return the level of the tested bit via CRUIN.

A CRU bit instruction outputs a 12-bit address which is computed as follows:



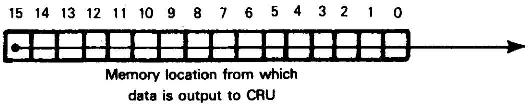
x, y and z represent any binary digits.

Now during the execution of a CRU bit instruction, the address which is output is supposed to be a bit address -- that is, an address identifying one bit in a possible 4096-bit field. So far as external interface logic is concerned, the address can be interpreted in any way. However, data output will occur via CRUOUT only, while data input is restricted to CRUIN, and thence to the Equal bit of the Status register.

There are two multibit CRU instructions: one loads data from any addressed CRU bit field, the other outputs data to any addressed CRU bit field. The address of the first CRU bit is specified by Register R12. Subsequently, this CRU bit address is incremented for each succeeding bit access. The multibit CRU instruction object code specifies the number of bits to be input or output, plus the data source or destination.

So far as external logic is concerned, there is no difference between a single-bit CRU instruction or a multibit CRU instruction. In either case, an address appears on the Address Bus. CRUCLK is plused high for output operations.

When a multibit CRU instruction is executed, bits are right-shifted onto the CRUOUT line during an output operation:



Thus bit 0, as bits are numbered above, will be output first. Any number of bits in the range 1 through 16 may be output by a single multibit CRU instruction.

During an input operation serial data is also right-shifted but now it must enter via the high order bit. Thus the first bit received is destined to become the low order bit.

Observe that data input via CRUIN during a multibit CRU instruction finishes up in a memory location -- not in the = bit of the Status register.

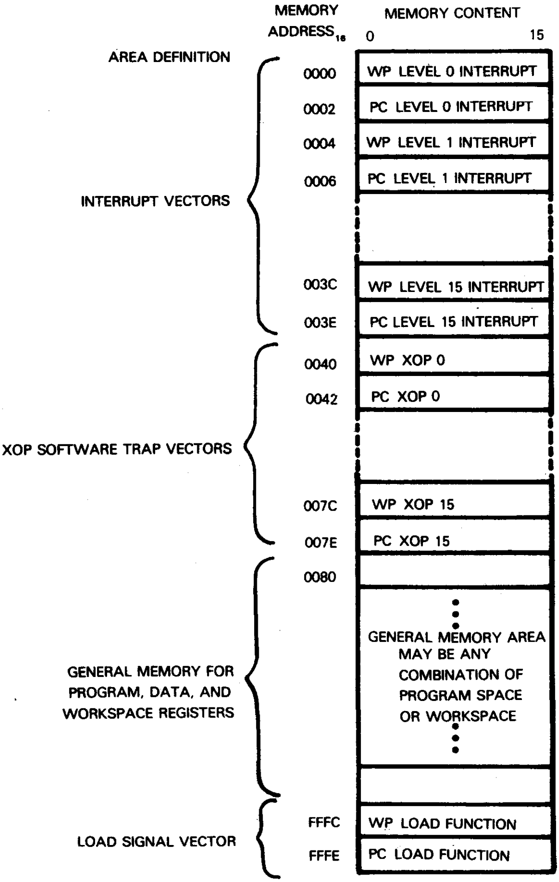


Figure 16-3. TMS9900 Memory Map

THE TMS9900 MEMORY MAP

Before discussing TMS9900 interrupt processing, it is appropriate that we should first look at the way in which the TMS9900 requires memory to be allocated. Figure 16-3 illustrates the TMS9900 memory map. Observe that for each of the 16 possible interrupt priority levels, four memory bytes are set aside to hold an initial Workspace register value and an initial Program Counter value.

Beyond the external interrupt vectors a further 32 memory bytes are set aside to hold Workspace register and Program Counter initial contents for 16 software interrupts. Software interrupts are created by executing an XOP instruction. The XOP instruction identifies one of the 16 software interrupts, then causes the appropriate context switch to occur.

An additional interrupt request is identified in Figure 16-3 as the LOAD; its Workspace register and Program Counter contents are stored in memory locations FFFC, FFFD, FFFE and FFFF. TMS9900 microcomputer systems manufactured by Texas Instruments connect the Load interrupt to a front panel switch which initiates execution of a bootstrap or cold start loading program. If you are building your own logic around a TMS9900 CPU, you can use the Load interrupt request in any other way. The only restriction is that LOAD must be synchronized with IAQ and must not stay active for more or less than one instruction execution time.

TMS9900 INTERRUPT PROCESSING LOGIC

Let us now look at the exact event sequence which will accompany any TMS9900 interrupts.

External logic capable of requesting an interrupt will do so by inputting a low signal via $\overline{\text{INTREQ}}$, together with an interrupt priority, input via IC0 - IC3.

Interrupt priorities range from 0000_2 for highest priority through 1111_2 for lowest priority. The 0000_2 priority is reserved for Resets; therefore 15 priority levels remain, with 0001_2 representing the highest external priority.

If more than one external source is simultaneously requesting an interrupt, then external logic must make sure that only the highest priority's identification code appears on the input signals, IC0 - IC3.

Logic within the TMS9900 CPU will acknowledge an interrupt request providing interrupts are enabled and no higher priority interrupt is being serviced. What the TMS9900 CPU logic does upon acknowledging an interrupt is to store the value input via IC0 - IC3 plus 1, in the four low order Status register bits. Subsequently, an interrupt will be acknowledged only if its priority level is equal to or greater than the value stored in the four low order status bits. Thus, interrupt priorities extend over the entire duration of the interrupt service routine and are not limited to the interrupt acknowledge process itself.

The TMS9900 has no interrupt acknowledge output control signal. However, as soon as an interrupt is acknowledged, a context switch is executed, during which the contents of the appropriate memory words are loaded into the Program Counter and Workspace register, while previous Program Counter and Workspace register contents are saved in the new General Purpose Registers R13 and R14. While this context switching occurs, memory addresses in the range 0000_{16} through $003F_{16}$ are placed on the Address Bus, since these are the memory locations within which Workspace register and Program Counter values for interrupt levels are stored. If you need an interrupt acknowledge signal, then you can create it by decoding 0s in address lines A0 - A10 while $\overline{\text{MEMEN}}$ is also low.

The interrupt service routine for the acknowledged level of interrupt will immediately be executed. You do not need to begin this interrupt service routine by saving the contents of any registers or status, since that is automatically done for you during the context switch. You should, however, terminate the interrupt service routine by executing a RTWP instruction. This instruction performs a reverse context switch and is equivalent to a subroutine Return instruction.

TMS9900 DMA LOGIC

Whereas the TMS9900 has extensive interrupt processing logic, it takes a simpler and more microcomputer-oriented approach to direct memory access.

Like most other microcomputers described in this book, there is a DMA request signal which causes the System Bus to be floated while a DMA acknowledge signal identifies the condition for external logic. External logic must provide all necessary DMA processing capabilities. All the TMS9900 CPU will do for you is float the System Bus.

THE TMS9900 INSTRUCTION SET

Table 16-1 illustrates the TMS9900 instruction set.

This is a very minicomputer-oriented instruction set.

The following symbols are used in Table 16-1:

AG	Arithmetic Greater Than status
C	Carry status
CNT	4-bit count field
CRUA	CRU address which is formed as described in text
D	Destination register. There are five possible options for the destination register. They are represented by these combinations of addressing modes: 00 Workspace Register D 01 Indirect through Workspace Register D 10 If D=0, word following instruction is address of destination. If D≠0, indexed address is calculated by adding word following instruction to Workspace Register D. 11 Indirect through Workspace Register D, auto-increment Workspace Register D.
DATA4	4-bit data unit
DATA16	16-bit data unit
DISP	8-bit signed displacement
EQ	Equal status
G	The AG and LG statuses
LG	Logical Greater Than status
OP	Odd Parity status
OV	Overflow status
PC	Program Counter
R	Any of the 16 Workspace registers
RXX	Workspace Register XX. For example, R15 is Workspace Register 15.
S	Source register. Addressing options identical to destination register.
ST	Status register
WP	Workspace register pointer

x<y,z> Bits y through z of the quantity x. For example, ([S] * [R])<31,16> represents the high order word of the product of the contents of the source register S and the Workspace register R.

[] Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.

* Multiplication

/ Division

Λ Logical AND

V Logical OR

∀ Logical Exclusive OR

← Data is transferred in the direction of the arrow.

Under the heading of STATUSES in Table 16-1, an X indicates statuses which are modified in the course of the instruction's execution. If there is no X, it means that the status maintains the value it had before the instruction was executed.

Table 16-1. TMS9900 Instruction Set Summary

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS					OPERATION PERFORMED
				G	EQ	C	OV	OP	
I/O	LDCR	S,CNT	2	X	X			X	[CRUA]←[S<CNT-1,0>] Transfer the specified number of bits from source register to the CRU.
	STCR	D,CNT	2	X	X			X	[D<CNT-1,0>]←[CRUA] Transfer the specified number of bits from the CRU to a register.
	SBO	DISP	2						[CRUA + DISP]←1 Set bit in CRU to 1.
	SBZ	DISP	2						[CRUA + DISP]←0 Set bit in CRU to 0.
	TB	DISP	2		X				If [CRUA + DISP]=0, then [EQ]=1; else [EQ]=0 Test bit in CRU.
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	A	S,D	2	X	X	X	X		[D]←[S] + [D] 16-bit addition of contents of source register to contents of destination register.
	AB	S,D	2	X	X	X	X	X	[D]←[S] + [D] 8-bit addition of contents of source register to contents of destination register.
	S	S,D	2	X	X	X	X		[D]←[D] - [S] 16-bit subtraction of contents of source register from contents of destination register.
	SB	S,D	2	X	X	X	X	X	[D]←[D] - [S] 8-bit subtraction of contents of source register from contents of destination register.
	MOV	S,D	2	X	X				[D]←[S] 16-bit move of contents of source register to destination register.
	MOVB	S,D	2	X	X			X	[D]←[S] 8-bit move of contents of source register to destination register.
	C	S,D	2	X	X				Set status flags based on 16-bit comparison of contents of source and destination registers.
	CB	S,D	2	X	X			X	Set status flags based on 8-bit comparison of contents of source register and contents of destination registers.
	XOR	S,R	2	X	X				[R]←[S]⊻[R] Exclusive-OR contents of source with Workspace Register R.

Table 16-1. TMS9900 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS G	STATUS EQ	STATUS C	STATUS OV	STATUS OP	OPERATION PERFORMED
SECONDARY MEMORY REFERENCE (MEMORY OPERATE) (CONTINUED)	MPY	S,R	2						[R]←[([S]*[R])<31,16>] [R + 1]←[([S]*[R])<15,0>] Multiply the contents of source register by those of Workspace Register R. Store most significant word of result in R. Store least significant word in Workspace Register R + 1.
	DIV	S,R	2				X		[R]←([R,R + 1]/[S])(quotient) [R + 1]←([R,R + 1]/[S])(remainder) Divide the 32-bit quantity represented by R (high order) concatenated with R + 1 (low order) by the contents of the source register. Store the quotient in R, the remainder in R + 1 and set overflow if quotient will exceed 16 bits.
	INC	D	2	X	X	X	X		[D]←[D] + 1 Increment contents of register by 1.
	INCT	D	2	X	X	X	X		[D]←[D] + 2 Increment contents of register by 2.
	DEC	D	2	X	X	X	X		[D]←[D] - 1 Decrement contents of register by 1.
	DECT	D	2	X	X	X	X		[D]←[D] - 2 Decrement contents of register by 2.
	CLR	D	2						[D]← 0000_{16} Clear the destination register.
	SETO	D	2						[D]← $FFFF_{16}$ Set all bits of register.
	INV	D	2	X	X				[D]← $\overline{[D]}$ Ones complement the destination register.
	NEG	D	2	X	X		X		[D]← $\overline{[D]}$ + 1 Twos complement the destination register.
	ABS	D	2	X	X	X	X		[D]←| [D] | Take the absolute value of the destination register's contents.
	SWPB	D	2						[D<15,8>]←→[D<7,0>] Exchange the high and low bytes of the register.

Table 16-1. T,MS9900 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS					OPERATION PERFORMED
				G	EQ	C	OV	OP	
SECONDARY MEMORY REFERENCE (MEMORY OPERATE) (CONTINUED)	SOC	S,D	2	X	X				If [S<i>]=1, then [D<i>]← 1 Set the bits in the destination register that correspond to 1s in the source register for all 16 bits.
	SOCB	S,D	2	X	X			X	If [S<i>]=1, then [D<i>]← 1 Set the bits in the destination register that correspond to 1s in the source register for 8 bits.
	SZC	S,D	2	X	X				If [S<i>]=1, then [D<i>]← 0 Clear the bits in the destination that correspond to 1s in the source register for all 16 bits.
	SZB	S,D	2	X	X			X	If [S<i>]=1, then [D<i>]← 0 Clear the bits in the destination register that correspond to 1s in the source register for 8 bits.
	COC	S,R		X					If for all [S<i>]=1, [R<i>]=1, then [EQ]← 1 If the bits in the Workspace Register R that correspond to the set bits in the source register are all 1s, set the EQUAL status.
	CZC	S,R	2		X				If for all [S<i>]=1, [R<i>]=0, then [EQ]=1 If the bits in the Workspace Register R that correspond to set bits in the source register are all 0s, set the EQUAL status.
IMMEDIATE	LI	R,DATA16	4	X	X				[R]← DATA16 Load immediate to Workspace Register R.
	LIMI	DATA4	4						[SR<3,0>]← DATA4 Load immediate data into the interrupt mask in the Status register.
	LWPI	DATA16	4						[WR]← DATA16 Load immediate to Workspace Pointer Register, WR.
IMMEDIATE OPERATE	CI	R,DATA16	4	X	X				Set the status flags based on 16-bit comparison between contents of Workspace Register R and immediate data.
	AI	R,DATA16	4	X	X	X	X		[R]←[R] + DATA16 Add immediate to Workspace Register R.
	ANDI	R,DATA16	4	X	X				[R]←[R] Λ DATA16 AND immediate to Workspace Register R.
	ORI	R,DATA16	4	X	X				[R]←[R] V DATA16 OR immediate to Workspace Register R.

Table 16-1. TMS9900 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS					OPERATION PERFORMED
				G	EQ	C	OV	OP	
JUMP	B	S	2						[PC]←[S] Branch unconditional to address in Source register.
	BL	S	2						[R11]←[PC] + 1 [PC]←[S] Branch to subroutine at address in source register.
	BLWP	S	2						[R13]←[WP] [R14]←[PC] [R15]←[S] [WP]←[S] [PC]←[S + 2] Branch to subroutine at address in source + 2, perform context switch to address contained in source.
	JMP	DISP	2						[PC]←[PC] + DISP Branch unconditional.
BRANCH ON CONDITION	JEQ	DISP	2						If [EQ]=1; then [PC]←[PC] + DISP Branch if equal.
	JNE	DISP	2						If [EQ]=0; then [PC]←[PC] + DISP Branch if not equal.
	JGT	DISP	2						If [AG]=1; then [PC]←[PC] + DISP Branch on arithmetic greater than.
	JLT	DISP	2						If [AG]=0and [EQ]=0; then [PC]←[PC] + DISP Branch on arithmetic less than.
	JHE	DISP	2						If [LG]=1 or [EQ]=1; then [PC]←[PC] + DISP Branch on logical greater than or equal.
	JH	DISP	2						If [LG]=1 and [EQ]=0; then [PC]←[PC] + DISP Branch on logical greater than.
	JL	DISP	2						If [LG]=0 and [EQ]=0; then [PC]←[PC] + DISP Branch on logical less than.

Table 16-1. TMS9900 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS					OPERATION PERFORMED
				G	EQ	C	OV	OP	
BRANCH ON CONDITION (CONTINUED)	JLE	DISP	2						If [EQ]=1 or [LG]=0; then [PC]←[PC]+DISP Branch on less than or equal.
	JNC	DISP	2						If [C]=0; then [PC]←[PC]+DISP Branch on carry reset.
	JNO	DISP	2						If [OV]=0; then [PC]←[PC]+DISP Branch on overflow reset.
	JOC	DISP	2						If [C]=1; then [PC]←[PC]+DISP Branch on carry set.
	JOP	DISP	2						If [OP]=1; then [PC]←[PC]+DISP Branch on odd parity set.
REGISTER OPERATE	SLA	R,CNT	2	X	X	X	X		Arithmetic shift the Workspace Register R left the specified number of bits.
	SRA	R,CNT	2	X	X	X			Arithmetic shift the Workspace Register R right the specified number of bits.
	SRL	R,CNT	2	X	X	X			Logical shift the Workspace Register R right the specified number of bits.
	SRC	R,CNT	2	X	X	X			Rotate the Workspace Register R right the specified number of bits.
STATUS	RTWP		2	X	X	X	X	X	[WP]←[R13] [PC]←[R14] [ST]←[R15] Perform a backward context switch.
	STST	R	2						[R]←[ST] Store the Status register into Workspace Register R.
	STWP	R	2						[R]←[WP] Store the Workspace Pointer into Workspace Register R.

Table 16-1. TMS9900 Ihstruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS					OPERATION PERFORMED
				G	EQ	C	OV	OP	
	X	S	2						Execute the instruction represented by the data in the source register.
	XOP	S,R	2					X	[R13]←[WP] [R14]←[PC] [R15]←[ST] [R11]←[S] [WP]←[40_{16}+(4*[R])] [PC]←[41_{16}+(4*[R])] Perform a context switch. This is the software interrupt.

Without analyzing the instruction set in great detail (and that is beyond the scope of the current discussion) it is worth noting that this instruction set owes its heritage to previous Texas Instruments minicomputers.

Of all the microcomputers described in this book, including the MicroNova discussed in Chapter 17, none has an instruction set which will be as effective as the TMS9900 when it comes to writing very large programs.

One point worth particular mention is the fact that the TMS9900 provides elementary multiplication and division as single assembly language instructions.

THE BENCHMARK PROGRAM

For the TMS9900, our benchmark program may be illustrated as follows:

```
        BLWP    MOVE              CONTEXT SWITCH TO APPROPRIATE REGISTERS
LOOP    MOV     @IOBUF(R1),*R2 +  LOAD NEXT INPUT BYTE IN NEXT TABLE BYTE
        DEC     R1                DECREMENT COUNT
        JNE     LOOP              RETURN FOR MORE
        RTWP                      RETURN FROM SUBROUTINE
```

Let us look at how our benchmark program can collapse to just five instructions.

We assume that there is some set of 16 General Purpose registers within which we store the word count and the address of the first free word in TABLE. We illustrated this idea when describing context switching earlier in the chapter.

Observe that Register R1 contains the word count, and is therefore used as an Index register, while Register R2 addresses the first free word in TABLE. Note that the contents of Register R2 are incremented automatically when the next byte is loaded into the table.

The BLWP instruction will branch to the program which performs the required data move, but simultaneously it loads the Workspace register with the appropriate initial address. We do not need to load any initial addresses or word counts into registers, since we have adopted the memory space where this data is stored to serve as our General Purpose registers.

After the move has been completed, we do not have to update any counters or pointers, because they were updated "in situ". All we have to do upon completing the move is store the contents of the current General Purpose Registers 13 and 14 to the Workspace register and Program Counter.

The following notation is used in Table 16-2:

aa	Two bits determining the addressing mode for the destination register.
bb	Two bits determining the addressing mode for the source register.
cccccccc	8-bit signed address displacement
dddd	Four bits used with aa to determine the destination register.
eeee	4-bit count field
rrrr	Four bits choosing the Workspace register.
ssss	Four bits used with bb to determine the source register.
XX	16 bits of immediate data.

If aa is 10_2 or bb is 10_2 then an additional 16-bit word used in computing the effective memory address of the operand will follow the instruction.

If aa is 10_2 and bb is 10_2 then two additional 16-bit words will follow the instruction: the first will be used in computing the destination address; the second will be used in computing the source address.

In the column labeled "Machine Cycles", two values may be given. If the values are separated by a slash (for example, 8/10), the second number represents execution time if the program branch is taken. Two values separated by a dash (such as 14-30) indicate that instruction execution time depends on the addressing mode.

Table 16-2. TMS9900 Instruction Set Object Codes

INSTRUCTION		OBJECT CODE	BYTES	MACHINE CYCLES*
A	S,D	1010aaddddbbssss	2	14-30
AB	S,D	1011aaddddbbssss	2	14-30
ABS	D	0000011101aadddd	2	12-20
AI	R,DATA16	000000100010rrrr	4	14
		XX		
ANDI	R,DATA16	000000100100rrrr	4	14
		XX		
B	S	0000010001bbssss	2	8-16
BL	S	0000011010bbssss	2	12-20
BLWP	S	0000010000bbssss	2	26-34
C	S,D	1000aaddddbbssss	2	14-30
CB	S,D	1001aaddddbbssss	2	14-30
CI	S,D	000000101000rrrr	4	14
		XX		
CLR	D	0000010011aadddd	2	
COC	S,R	001000rrrrbbssss	2	10-18
CZC	S,R	001001rrrrbbssss	2	14-22
DEC	D	0000011000aadddd	2	14-22
DECT	D	0000011001aadddd	2	10-18
DIV	S,R	001111rrrrbbssss	2	10-18
INC	D	0000010110aadddd		16-124
INCT	D	0000010111aadddd	2	10-18
INV	D	0000010101aaoddd	2	10-18
JEQ	DISP	00010011cccccccc	2	10-18
JGT	DISP	00010101cccccccc	2	8/10
JH	DISP	00011011cccccccc	2	8/10
JHE	DISP	00010100cccccccc	2	8/10
JL	DISP	00011010cccccccc	2	8/10
JLE	DISP	00010010cccccccc	2	8/10
JLT	DISP	00010001cccccccc	2	8/10
JMP	DISP	0001000cccccccc	2	10
JNC	DISP	00010111cccccccc	2	8/10
JNE	DISP	00010110cccccccc	2	8/10
JNO	DISP	00011001cccccccc	2	8/10
JOC	DISP	00011000cccccccc	2	8/10
JOP	DISP	00011100cccccccc	2	8/10
LDCR	S,CNT	001100eeeebbssss	2	22-52
LI	R,DATA16	000000100000rrrr	4	12
		XX		
LIMI	DATA4	0000001100000000	4	16
		XX		
LWPI	DATA16	0000001011100000	4	10
		XX		
MOV	S,D	1100aaddddbbssss	2	14-30
MOVB	S,D	1101aaddddbbssss	2	14-30

Table 16-2. TMS9900 Instruction Set Object Codes (Continued)

INSTRUCTION		OBJECT CODE	BYTES	MACHINE CYCLES*
MPY	S,R	001110rrrrbbssss	2	52-60
NEG	D	0000010100aadddd	2	12-20
ORI	R,DATA16	000000100110rrrr		14
		XX		
RTWP		0000001110000000	4	14
S	S,D	0110aaddddbbssss	2	14-30
SB	S,D	0111aaddddbbssss	2	14-30
SBO	DISP	00011101cccccccc	2	12
SBZ	DISP	00011110cccccccc	2	12
SETO	D	0000011100aadddd	2	10-18
SLA	R,CNT	00001010eeeerrrr	2	14-52
SOC	S,D	1110aaddddbbssss	2	14-30
SOCB	S,D	1111aaddddbbssss	2	14-30
SRA	R,CNT	00001000eeeerrrr	2	14-52
SRC	R,CNT	00001011eeeerrrr	2	14-52
SRL	R,CNT	00001001eeeerrrr	2	14-52
STCR	D,CNT	001101eeeeaadddd	2	42-60
STST	R	000000101100rrrr	2	8
STWP	R	000000101010rrrr	2	8
SWPB	D	0000011011aadddd	2	10-18
SZC	S,D	0100aaddddbbssss	2	14-30
SZCB	S,D	0101aaddddbbssss	2	14-30
TB	DISP	00011111cccccccc	2	12
X	S	0000010010bbssss	2	8-16**
XOP	S,R	001011rrrrbbssss	2	44-52
XOR	S,R	001010rrrrbbssss	2	14-22

* All instructions may require additional cycles if slow memory is accessed.

** Execution time is added to the execution time of the instruction located at the source address.

ELECTRICAL DATA

The following section contains specific electrical and timing characteristics for the TMS9900 CPU.

4. TMS 9900 ELECTRICAL AND MECHANICAL SPECIFICATIONS

4.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} (see Note 1)	−0.3 to 20 V
Supply voltage, V_{DD} (see Note 1)	−0.3 to 20 V
Supply voltage, V_{SS} (see Note 1)	−0.3 to 20 V
All input voltages (see Note 1)	−0.3 to 20 V
Output voltage (with respect to V_{SS})	−2 V to 7 V
Continuous power dissipation	1.2 W
Operating free-air temperature range	0°C to 70°C
Storage temperature range	−55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: Under absolute maximum ratings voltage values are with respect to the most negative supply, V_{BB} (substrate), unless otherwise noted. Throughout the remainder of this section, voltage values are with respect to V_{SS}.

4.2 RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB}	−5.25	−5	−4.75	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except clocks)		2.4		V
High-level clock input voltage, $V_{IH(\phi)}$		V_{DD}		V
Low-level input voltage, V_{IL} (all inputs except clocks)		0.4		V
Low-level clock input voltage, $V_{IL(\phi)}$		0.3		V
Operating free-air temperature, T_A	0		70	°C

4.3 ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input current	Data bus during DBIN	$V_I = V_{SS}$ to V_{CC}		±75		µA
		$\overline{WE}$, $\overline{MEMEN}$, DBIN during HOLDA	$V_I = V_{SS}$ to V_{CC}		±75		
		Clock	V_I = −1 V to 13.6 V		±75		
		Any other inputs	$V_I = V_{SS}$ to V_{CC}		±10		
V_{OH}	High-level output voltage		I_O = −0.4 mA		2.4		V
V_{OL}	Low-level output voltage		I_O = 3.2 mA		0.4		V
I_{BB}	Supply current from V_{BB}				1		mA
I_{CC}	Supply current from V_{CC}				125		mA
I_{DD}	Supply current from V_{DD}				30		mA
C_i	Input capacitance (any inputs except clock and data bus)		f = 1 MHz, unmeasured pins at V_{SS}		15		pF
$C_{i(\phi1)}$	Clock-1 input capacitance		f = 1 MHz, unmeasured pins at V_{SS}		100		pF
$C_{i(\phi2)}$	Clock-2 input capacitance		f = 1 MHz, unmeasured pins at V_{SS}		200		pF
$C_{i(\phi3)}$	Clock-3 input capacitance		f = 1 MHz, unmeasured pins at V_{SS}		100		pF
$C_{i(\phi4)}$	Clock-4 input capacitance		f = 1 MHz, unmeasured pins at V_{SS}		100		pF
C_{DB}	Data bus capacitance		f = 1 MHz, unmeasured pins at V_{SS}		25		pF
C_o	Output capacitance (any output except data bus)		f = 1 MHz, unmeasured pins at V_{SS}		15		pF

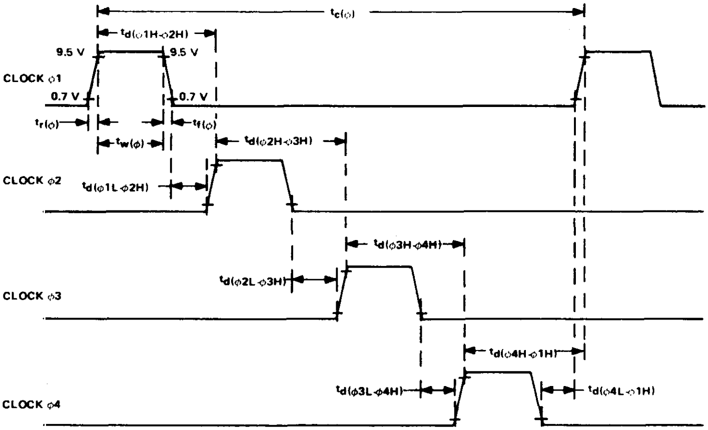
†All typical values are at T_A = 25°C and nominal voltages.

4.4 TIMING REQUIREMENTS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURES 12 AND 13)

PARAMETER		MIN	NOM	MAX	UNIT
$t_{c(\phi)}$	Clock cycle time		0.333		µs
$t_{r(\phi)}$	Clock rise time		12		ns
$t_{f(\phi)}$	Clock fall time		12		ns
$t_{w(\phi)}$	Pulse width, any clock high		45		ns
$t_{d(\phi1L\text{-}\phi2H)}$	Delay time, clock 1 low to clock 2 high (time between clock pulses)		5		ns
$t_{d(\phi2L\text{-}\phi3H)}$	Delay time, clock 2 low to clock 3 high (time between clock pulses)		5		ns
$t_{d(\phi3L\text{-}\phi4H)}$	Delay time, clock 3 low to clock 4 high (time between clock pulses)		5		ns
$t_{d(\phi4L\text{-}\phi1H)}$	Delay time, clock 4 low to clock 1 high (time between clock pulses)		5		ns
$t_{d(\phi1H\text{-}\phi2H)}$	Delay time, clock 1 high to clock 2 high (time between leading edges)		80		ns
$t_{d(\phi2H\text{-}\phi3H)}$	Delay time, clock 2 high to clock 3 high (time between leading edges)		80		ns
$t_{d(\phi3H\text{-}\phi4H)}$	Delay time, clock 3 high to clock 4 high (time between leading edges)		80		ns
$t_{d(\phi4H\text{-}\phi1H)}$	Delay time, clock 4 high to clock 1 high (time between leading edges)		80		ns
t_{su}	Data or control setup time before clock 1		40		ns
t_h	Data hold time after clock 1		10		ns

4.5 SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURE 13)

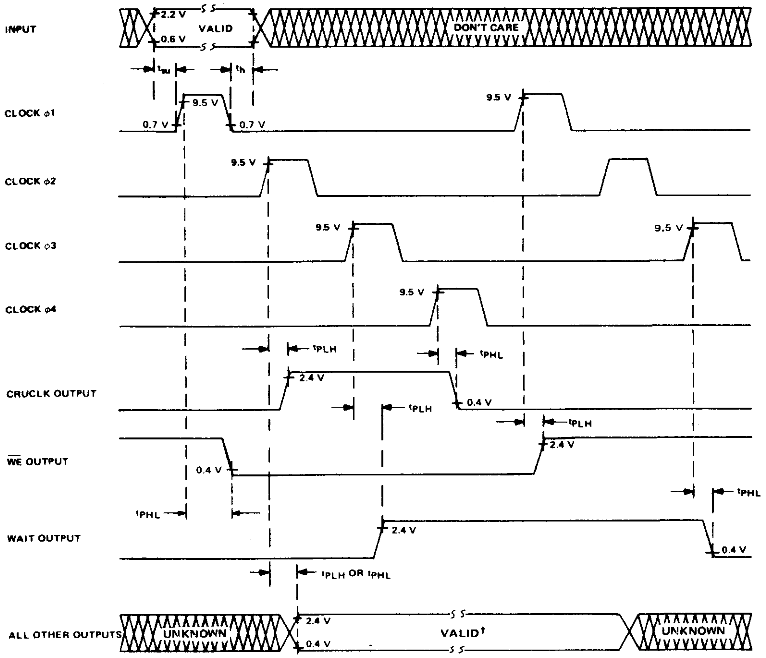
PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH} or t_{PHL}	Propagation delay time, clocks to outputs	C_L = 200 pF		20		ns



$t_{c(\phi)}$
$t_{d(\phi1H\text{-}\phi2H)}$
9.5 V
9.5 V
CLOCK φ1
0.7 V
0.7 V
$t_{r(\phi)}$
$t_{f(\phi)}$
$t_{w(\phi)}$
$t_{d(\phi2H\text{-}\phi3H)}$
CLOCK φ2
$t_{d(\phi1L\text{-}\phi2H)}$
$t_{d(\phi3H\text{-}\phi4H)}$
$t_{d(\phi2L\text{-}\phi3H)}$
CLOCK φ3
$t_{d(\phi4H\text{-}\phi1H)}$
$t_{d(\phi3L\text{-}\phi4H)}$
$t_{d(\phi4L\text{-}\phi1H)}$
CLOCK φ4

NOTE: All timing and voltage levels shown on φ1 applies to φ2, φ3, and φ4 in the same manner.

FIGURE 12 – CLOCK TIMING



† The number of cycles over which input/output data must will remain valid can be determined from Section 3.9. Note that in all cases data should not change during ϕ1.

FIGURE 13—SIGNAL TIMING

Chapter 17
SINGLE CHIP NOVA MINICOMPUTER CENTRAL PROCESSING UNITS

In this chapter we are going to look at two microprocessors which are the world's first single chip reproductions of established 16-bit minicomputers. We are going to describe two products which reproduce, on a single chip, the logic of a Nova Central Processing Unit. Nova minicomputers are built by Data General Corporation.

We will also explain how to extend the 9440 with logic that creates a standard Nova I/O interface bus and typical memory interface logic.

The Nova minicomputer was designed as a next generation enhancement of the PDP-8. The IM6100, which we have described in Chapter 12, is a single chip implementation of the PDP-8 Central Processing Unit.

If you compare the Nova architectures, which we describe in this chapter, with the IM6100 described in Chapter 12, the two products will indeed look very different. But conceptually they are similar. Both the Nova and the PDP-8 Central Processing Units have few addressable registers; for computing power they rely upon instructions which may perform complex sequences of operations. Similarities between the Nova and the PDP-8 will become more apparent if you compare these two devices with the CP1600 and the TMS9900 — which we have described in Chapters 15 and 16, respectively.

What is interesting about the Nova minicomputer is that it is one of the most popular in the world; and Data General Corporation is the second largest minicomputer manufacturer in the world, despite the fact that many aspects of the Nova Central Processing Unit may, on first inspection, appear to be very restricting.

Neither of the two products we are going to describe in this chapter will be available in commercial quantities until the latter part of 1977. The two products are:

1) The MicroNova, manufactured by:

DATA GENERAL CORPORATION
Mail Stop 6-58
Southborough, MA 01772

2) The 9440, manufactured by:

FAIRCHILD SEMICONDUCTOR
464 Ellis Street
Mountain View, CA 94040

The MicroNova and the 9440 are not the same; differences, however, are small.

The MicroNova is equivalent to the Nova 3 minicomputer. The Nova 3 is a low-end minicomputer recently introduced by Data General. Although it is a low-end product, it includes a number of features not found in the basic Nova architecture.

The 9440 reproduces basic Nova architecture — that is, the lowest common denominator of architectural features found in any Nova Central Processing Unit. As such, the 9440 lacks a number of logic features provided by the MicroNova. The 9440, however, has higher instruction execution speeds.

Because the MicroNova and the 9440 are very similar, we are going to describe them together in this chapter.

The MicroNova is manufactured using NMOS LSI technology. The 9440 is manufactured using Isoplanar integrated injection logic (I^3L) technology.

Both products are packaged as 40-pin DIPs.

The MicroNova requires four power supplies: -4.25V, +5V, +10V and +14V. The 9440 requires two power supplies: +5V and +180 mA.

Using a 240 nanosecond clock, the MicroNova executes instructions in 2.4 to 10 microseconds.Using a 100 nanosecond clock, 9440 instructions will execute in 1 to 2.5 microseconds.

A PRODUCT OVERVIEW

Figure 17-1 illustrates that part of our general microcomputer system logic which has been implemented by the MicrpNova and the 9440.

Note that only the MicroNova has a Stack Pointer, and DMA logic.

Most Nova minicomputers do not have a Stack; the 9440 is a reproduction of the basic Nova architecture, which is why the 9440 lacks a Stack.

The MicroNova and Nova 3 do contain Stacks, because the addition of the Stack is technologically straightforward, while the lack of a Stack had been one of the most distressing features of earlier Nova minicomputers.

Both the 9440 and the MicroNova have DMA request and DMA acknowledge signals; however, in response to a DMA request, the 9440 does nothing except float the System Bus. It is up to you to provide any and all external logic needed to actually perform a data transfer via direct memory access. The MicroNova, on the other hand, executes the required sequence of I/O operations to actually perform the DMA transfer. That is why in Figure 17-1 DMA logic is shown as being present on the MicroNova but not the 9440.

What about I/O ports? I/O ports interface logic is shown as absent in Figure 17-1. The I/O port is a microcomputer concept.

In any microcomputer configuration, you will look upon I/O ports as the ultimate interface between the microcomputer system and external logic. You need a conduit via which data bits or signals can be transferred to, or received from logic beyond the microcomputer system. Each conduit becomes an I/O port and an I/O port becomes a set of pins, which can be addressed as a unit on a support device. Minicomputers take a conceptually different approach to I/O operations. To begin with, data is generally transferred to or from the CPU — not signals. The data finishes up on a System Bus. Therefore a minicomputer's interface with the outside world consists of an I/O System Bus and a memory System Bus. In some cases the two busses are one; in other cases, such as the Nova minicomputers, these two are separate and distinct busses. Conceptually, what is important is the fact that the minicomputer anticipates transferring data via its I/O System Bus to line printers, disk units, or other substantial devices each of which is capable of having a significant amount of local logic. Thus the System Bus is as far as the minicomputer attempts to go when defining its interface to the outside world.

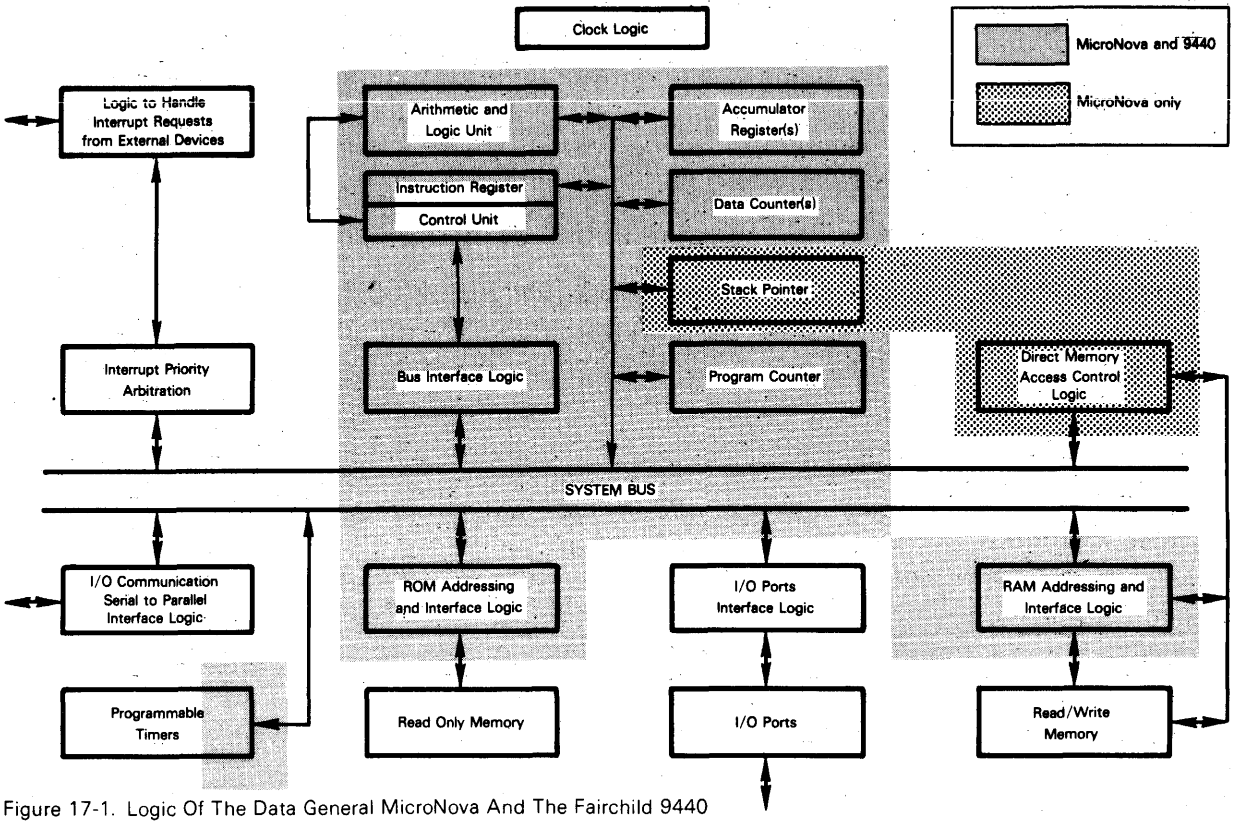

Clock Logic
MicroNova and 9440
MicroNova only
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 17-1. Logic Of The Data General MicroNova And The Fairchild 9440

Including bus interface logic in Figure 17-1, within the logic of the Central Processing Unit, needs some clarification. As we have just stated, the Nova minicomputer creates two separate System Busses: one for memory, the other for I/O devices. All the signals of these two busses originate at card edge pins. There is nothing very expensive about adding more pins to the edge of a card, as there is to adding more pins to a DIP. Therefore the Nova System Bus has 47 signals. Since neither the MicroNova nor the 9440 can have 47 signals, neither of these two devices creates standard Nova System Busses; but each device creates its own System Bus which could be used to drive external logic. That is why interface logic is shown as being present in Figure 17-1.

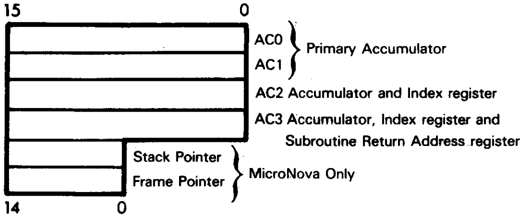
There is one further major difference between the MicroNova and the 9440 which is not evident from Figure 17-1. The MicroNova provides transparent dynamic memory refresh logic. The 9440 has no dynamic memory refresh logic.

The MicroNova, but not the 9440, contains an elementary interval timer capability. Providing interrupt timer logic is enabled, the MicroNova will generate an interrupt request every 20,000 instruction cycles. Using a standard 8.333 MHz clock, this translates to an interrupt request occurring every 2.4 msec.

Note that the MicroNova and the Nova 3 interval timer logic differ. The Nova 3 provides four programmable interval timer options; the MicroNova provides just one.

NOVA PROGRAMMABLE REGISTERS

These are the programmable registers of the MicroNova and the 9440:

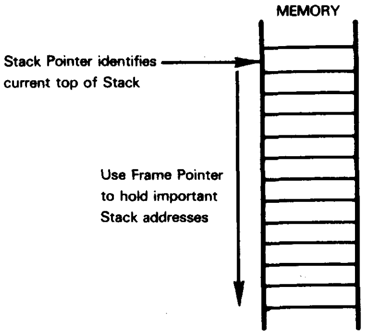

15
0
AC0
AC1
Primary Accumulator
AC2 Accumulator and Index register
AC3 Accumulator, Index register and
Subroutine Return Address register
Stack Pointer
Frame Pointer
MicroNova Only
14
0

Data General literature numbers registers and memory words from left to right, rather than as illustrated above, from right to left. Also Data General is one of the few minicomputer manufacturers that uses octal numbering. In order to remain consistent with the rest of this book, we will use hexadecimal numbers, and we will number registers from right to left; where confusions may arise, we will show both our standard numbers and Data General equivalents.

AC0 and AC1 are typical primary Accumulators. AC2 and AC3 may be used as Accumulators or as Index registers. The Jump-to-Subroutine instruction automatically stores the return address in AC3. If one subroutine is going to call another (i.e., you are nesting subroutines), then the calling subroutine must save the contents of AC3 before itself calling another subroutine.

Only the MicroNova has a Stack Pointer. The only instructions that access the Stack Pointer are "Push" and "Pop" instructions.

The MicroNova, but not the 9440, also contains a Frame Pointer register. The Frame Pointer register is an address buffer used to access the Stack. This may be illustrated as follows:


MEMORY
Stack Pointer identifies current top of Stack
Use Frame Pointer to hold important Stack addresses

The Frame Pointer is a buffer register; it is not a Data Counter. There are no instructions that access the memory location addressed by the Frame Pointer.

Observe that we show no programmable registers identified as Data Counters, even though in Figure 17-1 we show Data Counter logic as being present. This is because the Data Counter is another microcomputer concept — in effect, a subset of the Index register. If a memory reference instruction specifies direct, indexed addressing with a zero displacement, then Index Registers AC2 and AC3 are equivalent to Data Counters.

NOVA MEMORY ADDRESSING MODES

Both the MicroNova and the 9440 offer the following standard Nova memory addressing modes:

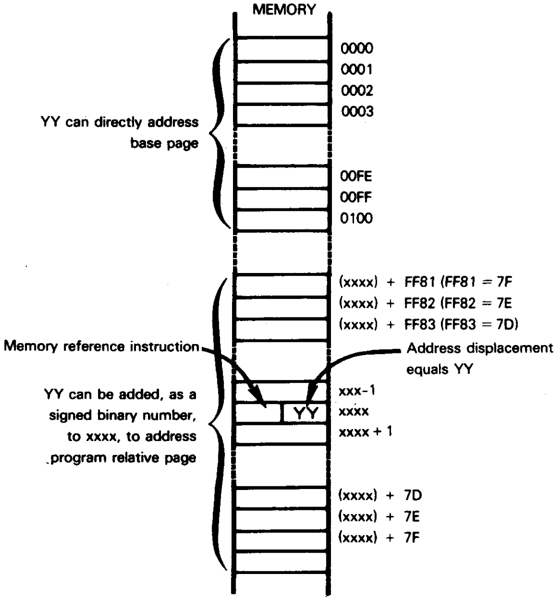
1) **Base page, direct addressing**
2) **Program relative, paged, direct addressing**
3) **Indirect addressing**
4) **Indirect addressing with auto-increment**
5) **Indirect addressing with auto-decrement**
6) **Direct, indexed addressing**
7) **Pre-indexed, indirect addressing**

These addressing modes have been described in Volume I, Chapter 6.

Nova memory addressing modes are heavily influenced by the fact that every Nova instruction generates a single 16-bit object code — just as the predecessor PDP-8 instructions each generated a single 12-bit object code. Even memory reference instructions are confined to 16 bits of object code; therefore the memory reference instruction can only provide a short address displacement. Whereas PDP-8 memory reference instructions provide a 7-bit address displacement, the Nova provides an 8-bit address displacement, which is handled in a much more intelligent fashion.

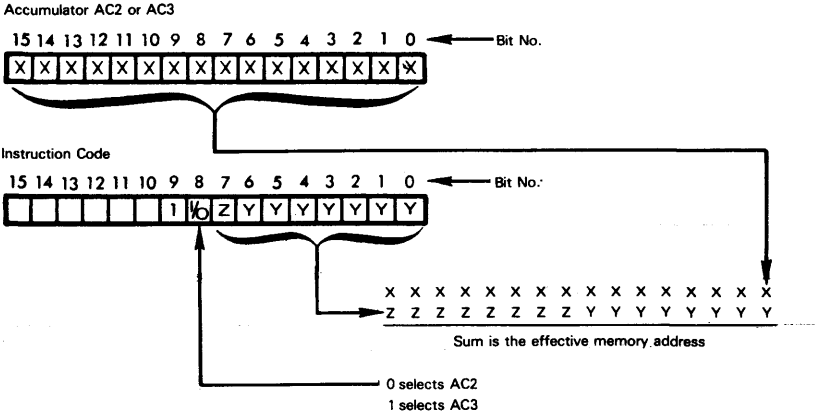
Nova instructions that use simple, direct addressing treat the 8-bit displacements as a direct, page zero address, or as a signed binary, program relative displace-

ment. Thus you can directly address the first 256 words of memory, or you can address any location within +127 to -128 words of the memory reference instruction itself:


MEMORY
0000
0001
0002
0003
00FE
00FF
0100
YY can directly address base page
(xxxx) + FF81 (FF81 = 7F
(xxxx) + FF82 (FF82 = 7E
(xxxx) + FF83 (FF83 = 7D)
Memory reference instruction
Address displacement equals YY
YY can be added, as a signed binary number, to xxxx, to address program relative page
YY
xxx-1
xxxx
xxxx + 1
(xxxx) + 7D
(xxxx) + 7E
(xxxx) + 7F

Remember, in microcomputer applications, program relative direct addressing is fine for Jump instructions, but is of limited value when accessing data memory. When a microcomputer program is stored in read-only memory, program relative, direct addressing can be used to read constant data only.

Nova instructions that specify direct, indexed addressing, compute the effective memory address as the contents of either AC2 or AC3, plus the 8-bit displacement provided by the instruction object code. The 8-bit displacement is treated as a signed binary number. Since the Index registers are 16 bits wide, direct indexed addressing allows you to address any memory word. This may be illustrated as follows:

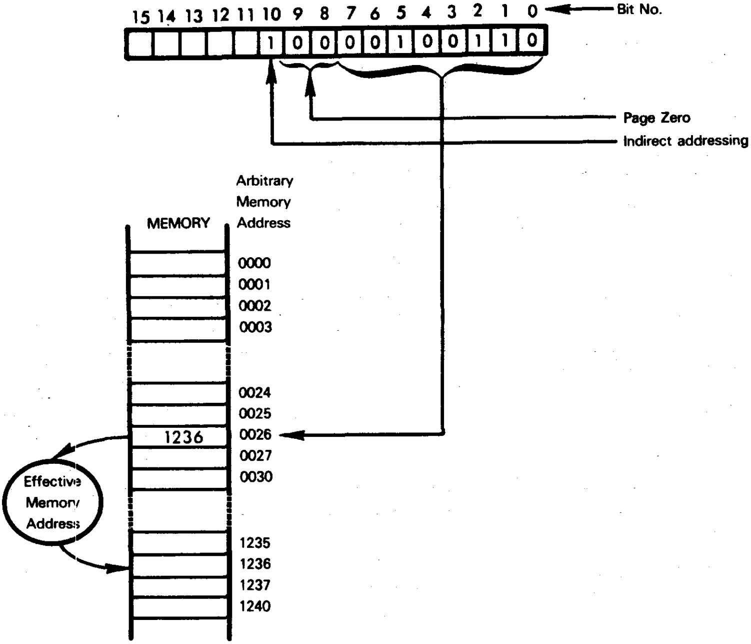


NOVA DIRECT MEMORY ADDRESSING

Indirect addressing may be superimposed on any of the memory addressing options described thus far. Indirect addressing is identified by a "1" in bit 10 of the Memory Reference instruction's object code. When indirect addressing is specified, the effective memory address is the contents of the directly addressed memory word.

NOVA INDIRECT PAGE ZERO ADDRESSING

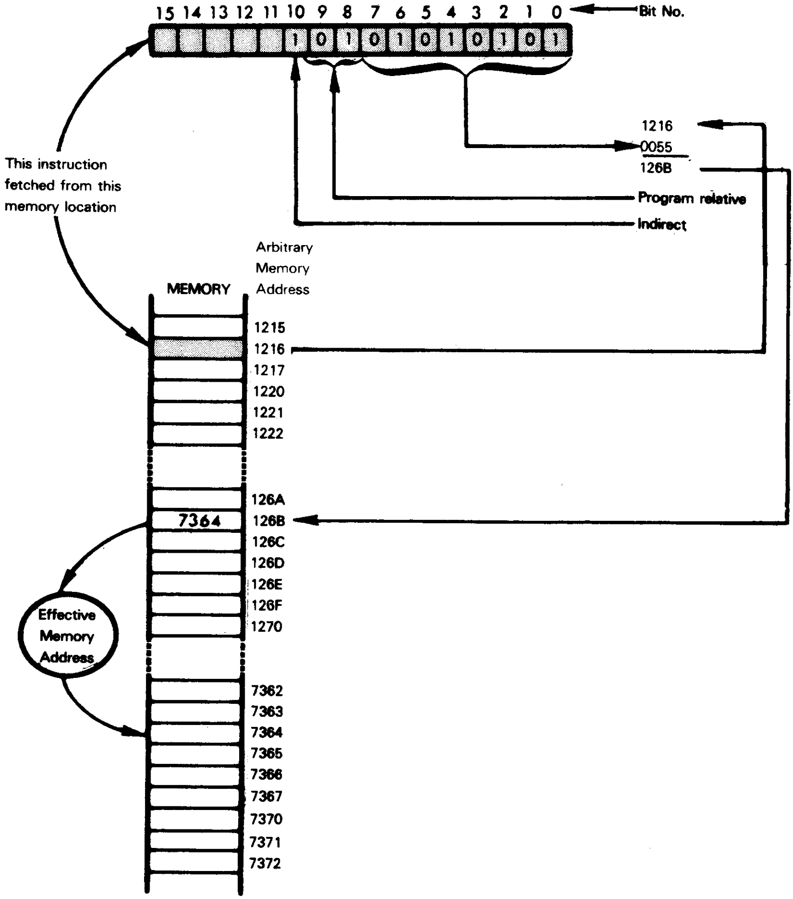
Let us examine the various indirect addressing options. First there is page zero indirect addressing:



In the illustration above, arbitrary, real memory addresses have been selected to make the illustration easier to understand.

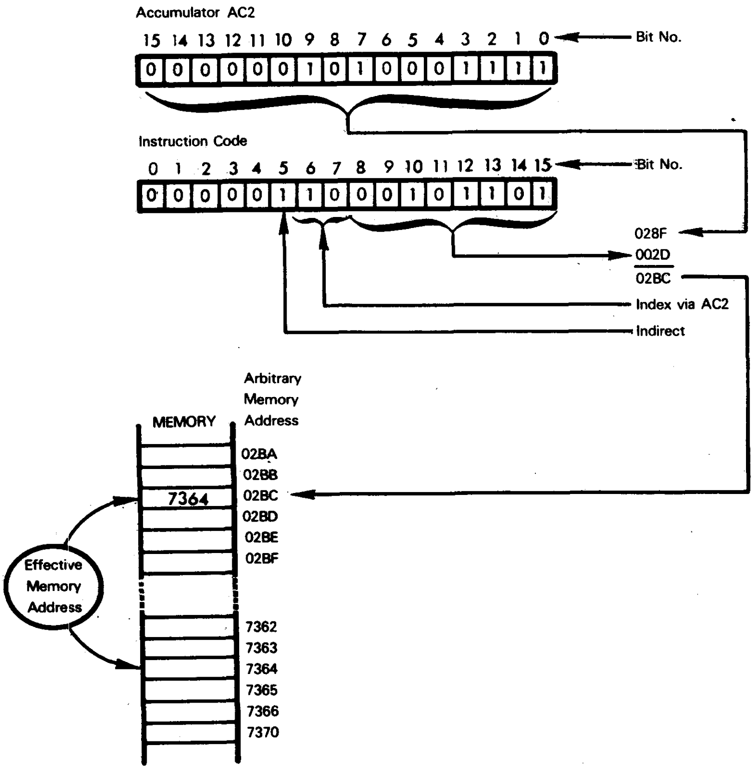
Program relative, indirect addressing may be illustrated as follows:

NOVA INDIRECT PROGRAM RELATIVE ADDRESSING



Indirect, indexed addressing may be illustrated as follows:

NOVA INDIRECT INDEXED ADDRESSING

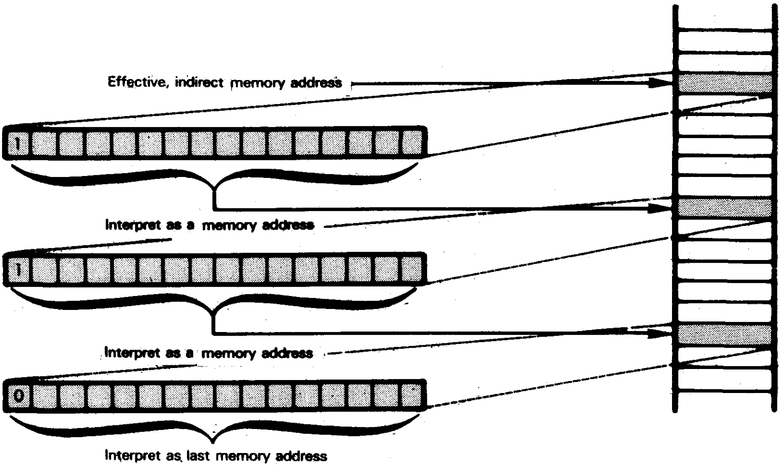


The illustration above arbitrarily uses indexed addressing via Accumulator AC2. Also the computed effective memory address is identical to that which was obtained in the indirect, program relative addressing illustration.

Observe that Nova indirect addressing logic results in pre-indexed indirect addressing. As described in Volume I, Chapter 6, this is less desirable than post-indexed indirect addressing.

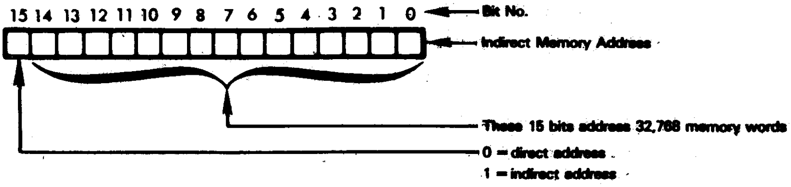
NOVA MULTIPLE INDIRECT ADDRESSING

If, and only if indirect addressing has been specified by a "1" in bit 10 of a Memory Reference instruction's object code, then the contents of the data fetched from memory are treated as a direct address, providing the high order bit of the direct address is 0. If the high order bit of the address is 1, then the address is treated as another indirect address pointer. This may be illustrated as follows:


Effective, indirect memory address
1
Interpret as a memory address
1
Interpret as a memory address
0
Interpret as last memory address

Note carefully that multilevel indirect addressing will occur only when indirect addressing is specified in the first place. If you execute a direct memory reference instruction, data will never be interpreted as an address.

The Nova indirect addressing logic means that, given a 16-bit indirect address, only 15 bits actually address memory; therefore you are limited to a 32,768 word memory address space:



The Nova minicomputers and microcomputers also provide indirect addressing with auto-increment and auto-decrement addressing. If you indirectly address one of the eight memory locations, 0010_{16} through 0017_{16}, then the contents of the addressed memory location are incremented at the beginning of the memory access. Thus you have indirect addressing with auto-increment.

If you indirectly address any one of the locations, 0018_{16} through $001F_{16}$ then the contents of the addressed memory location will be decremented at the beginning of the memory access. Thus you have indirect addressing with auto-decrement.

Neither the MicroNova nor the 9440 provide memory mapping logic. Memory mapping is a technique whereby more than 32,768 words of addressable memory may be accessed. The Nova 3 minicomputer is capable of supporting memory mapping as an option.

NOVA I/O DEVICE ADDRESSING

Nova minicomputers have separate memory and I/O device spaces. I/O instructions include six bits which identify one of 64 I/O devices. Because Nova minicomputers and microcomputers treat I/O devices in a manner that differs significantly from the typical microcomputer, we will defer our discussion of I/O addressing until we have looked at pins, signals and System Busses.

NOVA STATUS FLAGS

Nova minicomputers contain just one status flag, as we would define it, and that is the Carry status. Instructions are able to test for a zero or nonzero condition occurring at the conclusion of an instruction's execution, but no permanent zero status flag exists.

There are also these interrupt related status flags:

- Interrupt Enable
- Real Time Clock Enable
- Real Time Clock Request
- Stack Overflow Request

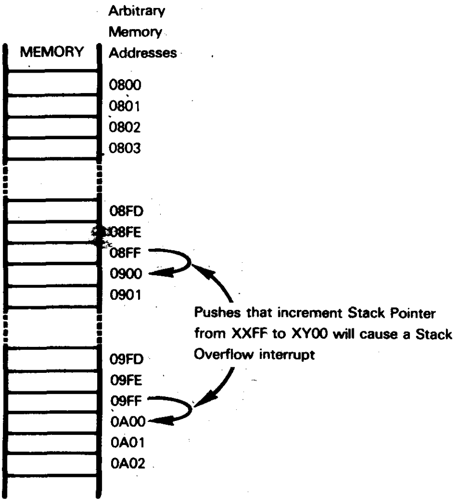
} MicroNova Only

The interrupt related status flags do not occur as addressable locations in any Status register; rather they represent flip-flops which are set or reset during the course of interrupt handling.

The interrupt enable bit is a master enable which is set to 1 in order to enable all interrupts. Specific instructions allow all interrupts to be enabled or disabled.

The MicroNova has a Real Time Clock interrupt enable bit and a Real Time Clock request bit. The Real Time Clock enable bit must be set to 1 in order to enable Real Time Clock interrupts; as soon as a Real Time Clock interrupt occurs, the Real Time Clock enable bit and the Real Time Clock request bit are reset to 0.

The Stack Overflow request bit is only present in the MicroNova, since only the MicroNova has a Stack. A Stack overflow condition occurs if, following a push operation, the incremented contents of the Stack register have zeros in the eight low order bits. What this implies is that the Stack must reside within a 256-word memory page:


MEMORY
Arbitrary Memory Addresses
0800
0801
0802
0803
08FD
08FE
08FF
0900
0901
09FD
09FE
09FF
0A00
0A01
0A02
Pushes that increment Stack Pointer from XXFF to XY00 will cause a Stack Overflow interrupt

When a Stack overflow occurs, the Stack Overflow request bit is set to 1 and an interrupt is requested.

MICRONOVA AND 9440 CPU PINS AND SIGNALS

As we stated earlier in this chapter, minicomputer Central Processing Units are implemented on cards, not DIPs; therefore they usually have System Busses containing more than 40 signals. The standard Nova System I/O Bus contains 47 signals; furthermore, the Nova System Bus is, in effect, two busses: one communicating with memory, while a separate and distinct bus communicates with I/O devices:

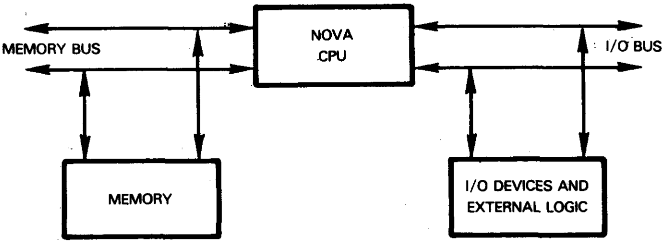


Table 17-1 briefly defines the functions of bus signals. The I/O Bus is standard for all Nova line computers, while the Memory Bus is different for each model. We give the Memory Bus signals of the Nova 2 in Table 17-1.

STANDARD NOVA SYSTEM I/O BUS

SIGNAL	DIRECTION	FUNCTION OR INDICATION
$\overline{DS0}$ - $\overline{DS5}$	To Device	Device selection
$\overline{DATA0}$ - $\overline{DATA15}$	Bidirectional	Data and address lines
DATOA	To Device	Data out to device's A buffer
DATIA	To Device	Data in from device's A buffer
DATOB	To Device	Data out to device's B buffer
DATIB	To Device	Data in from device's B buffer
DATOC	To Device	Data out to device's C buffer
DATIC	To Device	Data in from device's C buffer
STRT	To Device	Start device — clear Done flag, set Busy flag and clear device's INT REQ flip-flop
CLR	To Device	Clear device's Busy and Done flags and INT REQ flip-flop
IOPLS	To Device	I/O Pulse — user-defined function
$\overline{SELB}$	To Processor	Selected device's Busy flag is set
$\overline{SELD}$	To Processor	Selected device's Done flag is set
$\overline{RQENB}$	To Device	Enable interrupt or DMA requests
$\overline{INTR}$	To Processor	Interrupt request
$\overline{INTP}$	To Device	Interrupt priority
INTA	To Device	Interrupt acknowledge
$\overline{MSKO}$	To Device	Interrupt mask out
$\overline{DCHR}$	To Processor	Data channel request (DMA request)
$\overline{DCHP}$	To Device	Data channel priority
$\overline{DCHA}$	To Device	Data channel acknowledge
DCHM0,DCHM1	To Processor	Data channel mode:
		$\overline{DCHM0}$ $\overline{DCHM1}$
		1 1 Data out
		1 0 Increment memory
		0 1 Data in
		0 0 Add to memory
DCHI	To Device	Data channel in
DCHO	To Device	Data channel out
OVFLO	To Device	Overflow: result of memory increment or add exceeds $FFFF_{16}$
IORST	To Device	Clear all I/O devices

THE NOVA 2 MEMORY BUS

SIGNAL	DIRECTION	FUNCTION OR INDICATION
A0 - A14	To Memory	Memory address lines
DATA0 - DATA15	Bidirectional	Memory data lines
INHIBIT SELECT	To Memory	Inhibits selection of memory module
BMEMEN	To Memory	Starts memory cycle
WRITE	To Memory	Memory write
BRMW	To Memory	Causes pause between read and write
WE	To Memory	Enable write after pause in read-pause-write cycle
SYNC ENABLE	To Processor	CPU hold control
RELOAD DISABLE	To Memory	Inhibits loading of memory buffer
WAIT	To CPU	Disables other memory modules during write portion of memory cycle
MEM CLOCK	To Memory	Memory Clock
EXTERNAL SELECT	To Memory	Allows module to be selected despite contents of address lines
EXTERNAL MBLD	To Memory	Allows data to be stored in memory buffer without starting a memory cycle

Table 17-1. Nova System Bus Signals

If you are using the MicroNova or 9440 in a new product, then there is no reason why you should create the standard Nova System Busses. Providing the signals generated by the MicroNova or the 9440 are adequate for your needs, you can interface external logic directly to these two devices.

Let us first look at the MicroNova pins and signals, which are illustrated in Figure 17-2.

Two clock signals, Φ1 and Φ2, must be input to synchronize all MicroNova logic.

MICRONOVA MEMORY BUS

The Memory Bus consists of a 16-bit Address/Data Bus, plus three control signals: SAE, P and WE.

The Address/Data Bus connects to pins $\overline{\text{MB0}}$ - $\overline{\text{MB15}}$. P is a synchronization signal, SAE is a read enable and WE is a write enable.

MICRONOVA I/O BUS

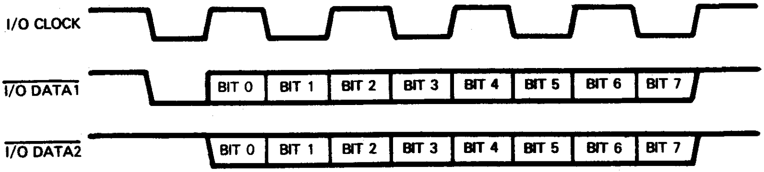
The I/O Bus consists of just four signals:

I/O CLOCK synchronizes I/O transfers.

$\overline{\text{I/O DATA1}}$ and $\overline{\text{I/O DATA2}}$ are bidirectional data and control signals.

$\overline{\text{I/O INPUT}}$ identifies the direction of data transfers occurring via $\overline{\text{I/O DATA1}}$ and $\overline{\text{I/O DATA2}}$.

As compared to other microcomputers described in this book, the MicroNova I/O interface is very unusual. 16-bit I/O data transfer occurs as two 8-bit serial units. This may be illustrated as follows:



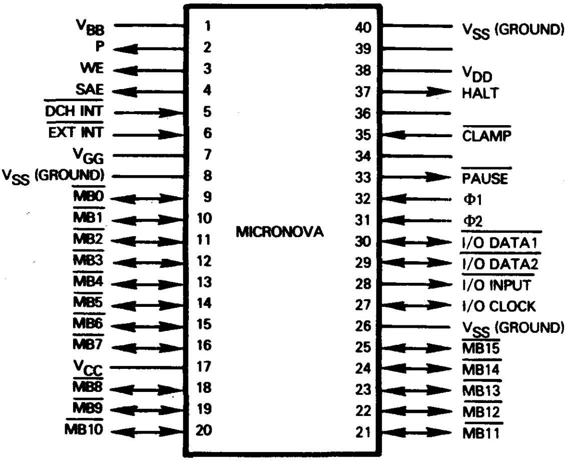
Eight serial bits are input in less than one microsecond; therefore this method of handling I/O is as fast as the parallel data input operations described for other microcomputers.

Each data transfer is preceded by one of four codes generated by levels output via $\overline{\text{I/O DATA1}}$ and $\overline{\text{I/O DATA2}}$. These are the four codes:

$\overline{\text{I/O DATA1}}$	$\overline{\text{I/O DATA2}}$	INTERPRETATION
1	1	Accompanying I/O low pulse may be used to synchronize interrupt requests and DMA requests.
1	0	DMA request acknowledge.
0	1	I/O data transfer. The transfer direction is specified by I/O INPUT.
0	0	I/O command out.

Thus every I/O operation will begin with $\overline{\text{I/O DATA1}}$ and $\overline{\text{I/O DATA2}}$ being output during a low I/O CLOCK pulse. $\overline{\text{I/O INPUT}}$ will be low at this time since data is being output via $\overline{\text{I/O DATA1}}$ and $\overline{\text{I/O DATA2}}$. Providing $\overline{\text{I/O DATA1}}$ and $\overline{\text{I/O DATA2}}$ specify a data transfer to follow, the actual data transfer will occur via $\overline{\text{I/O DATA1}}$ and $\overline{\text{I/O DATA2}}$ with $\overline{\text{I/O INPUT}}$ identifying the data transfer direction.

There are two CPU control signals which are not part of either the Memory Bus or the I/O Bus.



PIN NAME	DESCRIPTION	TYPE
Φ1, Φ2	Clock Signals	Input
MB0 - MB15	Address/Data Bus	Bidirectional
P	Memory Synchronization	Output
SAE	Memory Read Enable	Output
WE	Memory Write Enable	Output
I/O CLOCK	I/O Synchronization	Bidirectional
I/O DATA1, I/O DATA2	Data and Control	Bidirectional
I/O INPUT	Transfer Direction	Output
CLAMP	Power-On Reset	Input
HALT	CPU Halted	Output
DCH INT	DMA Request	Input
EXT INT	External Interrupt Request	Input
PAUSE	Memory Bus Grant	Output
V_{BB} V_{DD}, V_{GG}, V_{SS}	Power and Ground	

Figure 17-2. MicroNova CPU Signals And Pin Assignments

Following power-up, the MicroNova CPU will not perform any operation until a high input occurs at CLAMP. When CLAMP goes high, interrupts are enabled. Real Time Clock and Stack Overflow interrupt requests are cleared, and the CPU is halted. Once CLAMP has been input high, it is ignored until the MicroNova is powered down and then powered up again.

The HALT signal is output by the MicroNova as a high pulse while the MicroNova CPU has been halted — either in response to execution of a Halt instruction, or following CLAMP going high.

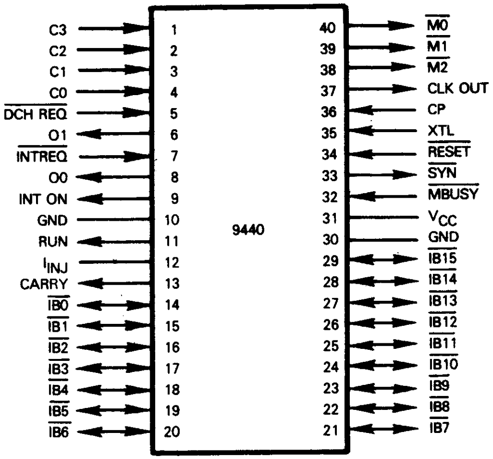
There are two MicroNova signals associated with interrupt logic. DMA requests are made via DCH INT while any external interrupt is requested via EXT INT. Both the DMA request and the interrupt request must be synchronized with instruction execution timing. This synchronization is provided by I/O DATA1 and I/O DATA2, as we have already described. The DMA acknowledge occurs via I/O DATA1 and I/O DATA2. There is no external interrupt acknowledge signal; however, such a signal can be derived from the Memory Bus, as we will describe later in this chapter.

PAUSE is output low by the CPU when devices other than the CPU are permitted to access memory.

Now look at 9440 pins and signals, which are illustrated in Figure 17-3.

These pins and signals create a single System Bus. No attempt is made to create separate Memory and I/O Busses.

You may connect a crystal across CP and XTL in order to create a master clock signal, or you may input a clock signal via CP



PIN NAME	DESCRIPTION	TYPE
XTL, CP	Clock Signals	Input
SYN	Synchronization Signal	Output
CLK OUT	System Clock	Output
IB0 - IB15	Data/Address Bus	Bidirectional
M0 - M2	Memory Controls	Output
MBUSY	Memory Busy	Input
O0, O1	I/O Control	Output
INTREQ	Interrupt Request	Input
INT ON	Interrupt Enable	Output
DCH REQ	DMA Request	Input
RUN	CPU Running	Output
CARRY	Carry Status	Output
C0 - C3	Front Panel/Console Control Signals	Input
RESET	Reset	Input
I_{INJ}, V_{CC}, GND	Power and Ground	

Figure 17-3. 9440 CPU Signals And Pin Assignments

9440 SYSTEM BUS

The 9440 generates a single synchronizing output (SYN). The CPU clock is output to the system via CLK OUT

IB0 - IB15 provides the 9440 with a multiplexed 16-bit Data Address Bus. This bus carries addresses to memory and I/O devices, and it carries bidirectional data between the CPU and memory or I/O devices. IB0 - IB15 is low true; a low signal level represents a 1 bit.

IB0 is the high order bus line while IB15 is the low order bus line. This agrees with Nova conventions. This chapter, and this whole book describe the low order bit as bit 0 — exactly the inverse of IB0 - IB15.

There are three control signals on the 9440 CPU-memory interface.

$\overline{\text{M0}}$ is output low to identify a memory read.
$\overline{\text{M1}}$ is output low to identify a memory write.
$\overline{\text{M2}}$ is output low to identify a memory address being output.

External memory interface logic inputs $\overline{\text{MBUSY}}$ low while it is responding to any memory access. MBUSY is equivalent to the WAIT signals that we have described for other microcomputers; it can be used to make the CPU wait for slow memory to respond to a CPU access request.

The 9440 has two I/O control signals O0 and O1. These two control signals define I/O and memory accesses as follows:

O0 = 0	O1 = 0	Instruction Fetch
O0 = 0	O1 = 1	Data Channel Access
O0 = 1	O1 = 0	I/O Operation
O0 = 1	O1 = 1	No I/O

There are two signals associated with 9440 interrupt logic.

An external interrupt is requested by inputting $\overline{\text{INTREQ}}$ low.

INT ON indicates whether or not interrupts are enabled. This signal is high when interrupts are enabled; if this signal is low, interrupts are disabled.

A DMA request is made by inputting $\overline{\text{DCH}}$ $\overline{\text{REQ}}$ low. The DMA request is acknowledged by O0 and O1 being output low and high, respectively.

There are six signals provided by the 9440 specifically to support a front panel or console.

Two of the front panel or console signals are outputs; these are the RUN and CARRY signals.

RUN is output high while the CPU is executing programs; it is output low while the CPU is halted. RUN is used to generate an appropriate front-panel display light; it is also equivalent to a Halt acknowledge, as described in this book for many other microcomputers.

CARRY represents the condition of the Carry status. This signal is output specifically to drive a front-panel light.

Four input control signals are provided for switches on a front-panel. These signals are C0, C1, C2 and C3; they perform the following operations:

C3	C2	C1	C0	FUNCTION
0	0	0	0	Display AC0 contents at console
0	0	0	1	Display AC1 contents at console
0	0	1	0	Display AC2 contents at console
0	0	1	1	Display AC3 contents at console
0	1	0	0	Load Program Counter from console switches
0	1	0	1	Display contents of addressed data memory word
0	1	1	0	Not Used
0	1	1	1	Halt
1	0	0	0	Deposit switches into AC0
1	0	0	1	Deposit switches into AC1
1	0	1	0	Deposit switches into AC2
1	0	1	1	Deposit switches into AC3
1	1	0	0	Load Program Counter from console switches
1	1	0	1	Load data memory from console switches
1	1	1	0	Continue/Run
1	1	1	1	No Operation

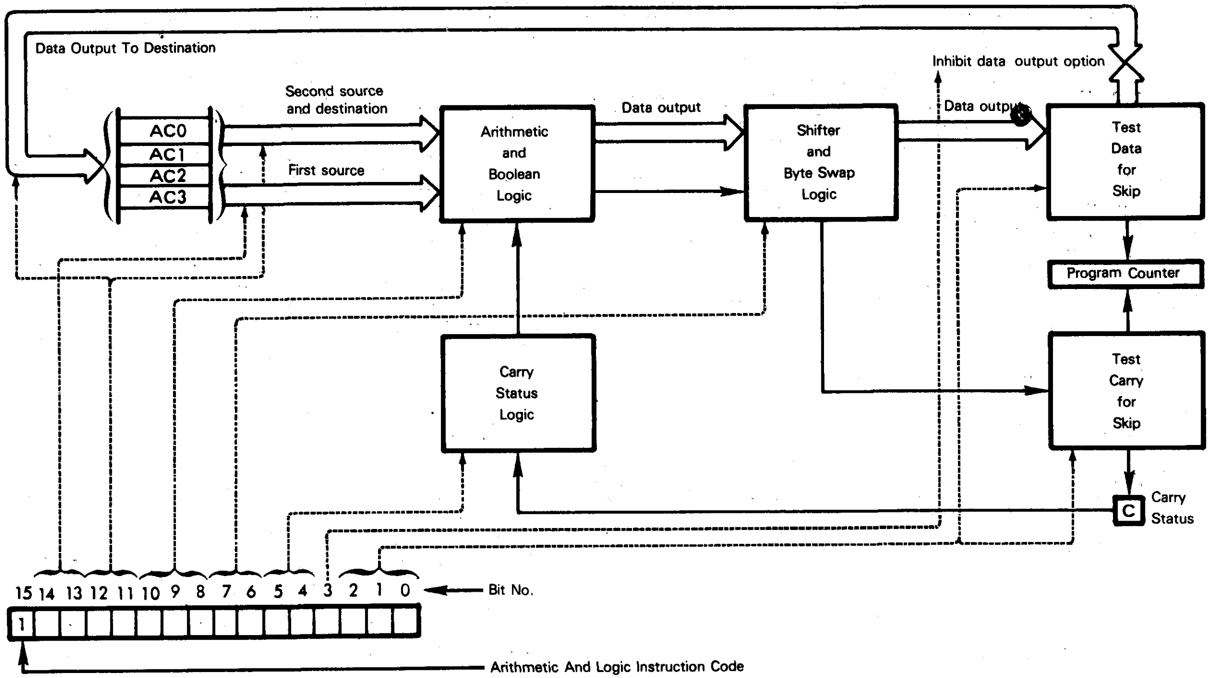


Figure 17-4. The Nova Arithmetic And Logic Unit

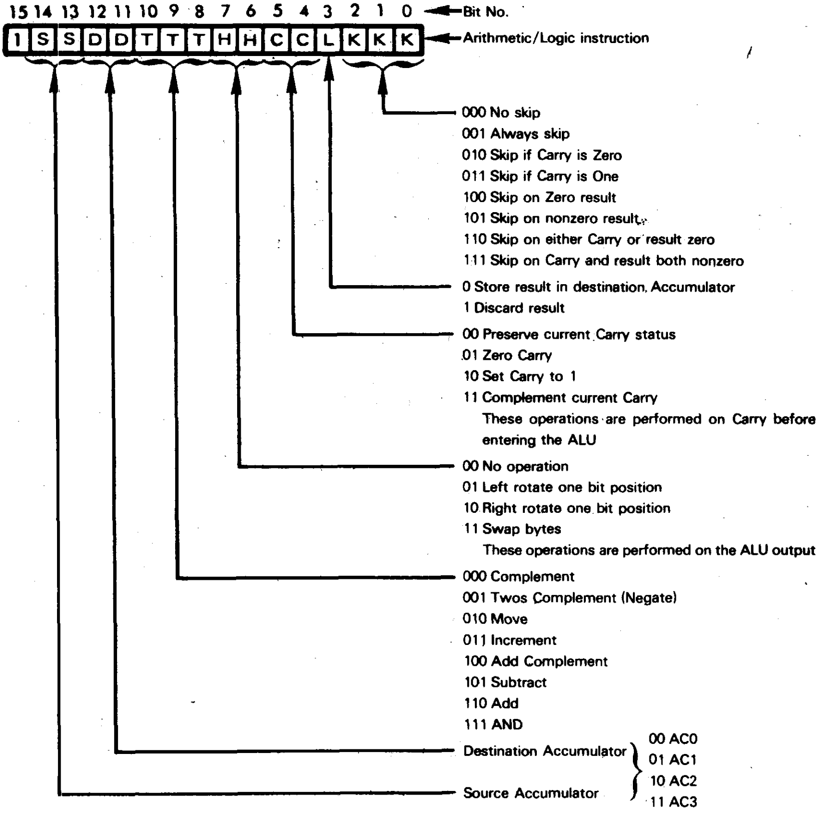

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 Bit No.
1 S S D D T T T H H C C L K K K Arithmetic/Logic instruction
000 No skip
001 Always skip
010 Skip if Carry is Zero
011 Skip if Carry is One
100 Skip on Zero result
101 Skip on nonzero result
110 Skip on either Carry or result zero
111 Skip on Carry and result both nonzero
0 Store result in destination. Accumulator
1 Discard result
00 Preserve current Carry status
01 Zero Carry
10 Set Carry to 1
11 Complement current Carry
These operations are performed on Carry before entering the ALU
00 No operation
01 Left rotate one bit position
10 Right rotate one bit position
11 Swap bytes
These operations are performed on the ALU output
000 Complement
001 Twos Complement (Negate)
010 Move
011 Increment
100 Add Complement
101 Subtract
110 Add
111 AND
Destination Accumulator
Source Accumulator
00 AC0
01 AC1
10 AC2
11 AC3

Figure 17-5. Arithmetic/Logic Instruction Object Code Interpretation

CPU LOGIC AND INSTRUCTION EXECUTION

The manner in which the Nova CPU executes instructions differs markedly from microcomputers described earlier in this book. We will therefore begin our discussion of CPU operations by looking at overall CPU architecture.

Our discussion of Nova CPU logic is tied to instruction object code bit patterns; this happens to be the simplest way of describing the Nova CPU. We will look at instructions from a programmer's perspective when we examine the Nova instruction set.

Nova instructions may be divided into these three groups:

1) Arithmetic, Boolean and logical operations which are essentially internal to the CPU.
2) Memory reference instructions which offer a variety of memory addressing modes and very little else.
3) I/O instructions which are designed to allow a considerable amount of intelligence in I/O devices.

Let us examine each group of instructions and associated CPU logic.

ARITHMETIC/LOGIC INSTRUCTIONS

The power of the Nova CPU lies in the fact that many logic functions are implemented sequentially along a single data path through the CPU. This is illustrated in Figure 17-4. This figure shows how individual bits of arithmetic and logic instruction object codes directly identify the many options available as data makes a single tour through the CPU. **Figure 17-5 provides specific arithmetic and logic instruction object code interpretations.**

Data to be operated on is always fetched from the Accumulators. Results are always returned to an Accumulator. For two operand instructions, such as binary addition, the Destination Accumulator also serves as the second Source Accumulator. For one operand instructions, such as a complement, there will be one Source Accumulator and one Destination Accumulator; the same Accumulator may serve as source and destination.

As the source and destination definitions would imply, the Nova has no Secondary Memory Reference (or Memory Operate) instructions as we define them; for example, you cannot directly add the contents of a memory word to the contents of an Accumulator.

In addition to one or two 16-bit data words, the Carry status is input to the Arithmetic and Boolean logic. You may input the Carry status as is, or you may complement it, reset it to 0 or set it to 1. If you modify the Carry status, then the modified Carry status becomes the new input to the Arithmetic and Boolean logic.

You may specify one of eight Arithmetic and Logic operations. The Move operation serves both as a Move and a No Operation. By specifying the same Accumulator as the source and destination for a Move, Arithmetic and Boolean logic is bypassed. Notice that only one Boolean operation, the AND, is provided. This is an inconvenience rather than a problem. As discussed in Volume I, Chapter 2, you can combine the AND and complement operations to generate an OR or an Exclusive-OR. The following Nova instruction sequences substitute for the OR and Exclusive-OR:

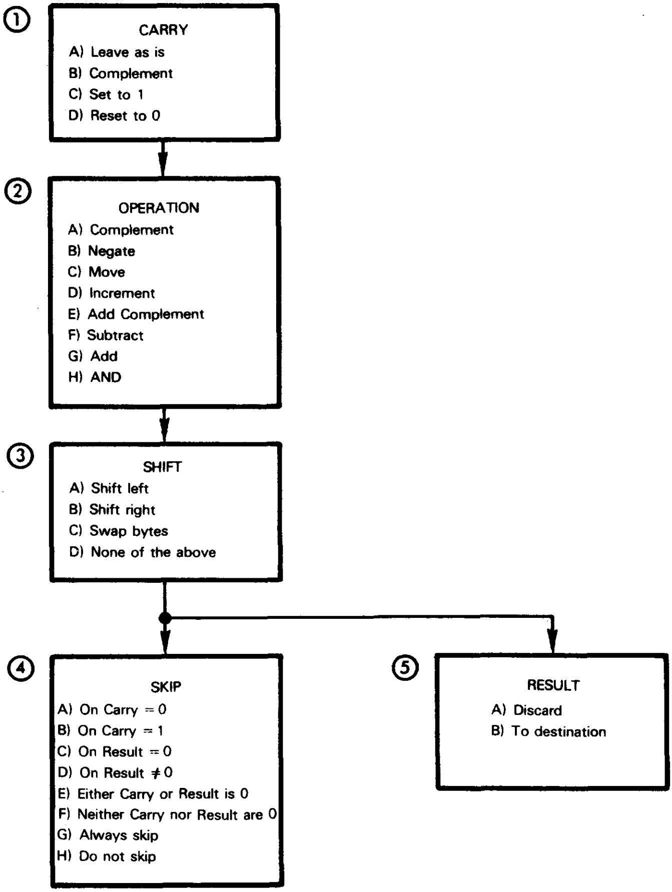
```
;OR the contents of ACX with ACY. Leave the result in ACY
        COM     ACX,ACX    Complement ACX
        AND     ACX,ACY    AND ACX with ACY. Result to ACY
        ADC     ACX,ACY    Add original ACX. Result to ACY
;Exclusive-OR ACX with ACY. Leave the result in ACY.
;ACZ is needed for temporary data storage
        MOV     ACY,ACZ    Save ACY in ACZ
        ANDZL   ACX,ACZ    Store twice ACX AND ACY in ACZ
        ADD     ACX,ACY    Add ACX to ACY
        SUB     ACZ,ACY    Subtract twice ACX AND ACY
```

The 16-bit output from the Arithmetic and Boolean logic, together with the Carry status, passes to the Shifter and Byte Swap logic; here the 17-bit data unit may be rotated left or right, high and low order bytes of the 16-bit data unit may be swapped, or this logic may be bypassed.

The Shifter and Byte Swap logic outputs 16 bits of data, plus the Carry status. The data and the Carry status may be tested separately, and based on one of eight identifiable conditions, the Program Counter contents may be incremented; this provides conditional skip logic. Figure 17-5 defines the eight conditions that may cause a skip.

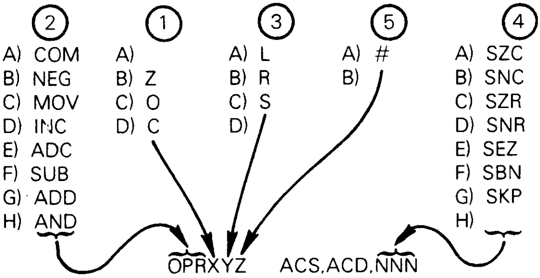
Finally you have the option of preventing results from being stored in the Destination register; this enables conditional branch logic without modifying the contents of any Accumulator.

In summary, the five operations that can be specified by a single arithmetic/logic instruction may be illustrated as follows:



It would take four or five typical microprocessor instructions to perform the same operations that a single Nova instruction can perform.

Arithmetic/logic instruction options are specified in the source program using compound mnemonics. The mnemonics are created as follows:



The numbers ①, ②, ③, ④ and ⑤ and the letters A), B), C), D), E), F), G) and H) are keyed to the previous illustration. ACS represents "Source Accumulator" while ACD represents "Destination Accumulator". Thus the instruction "set carry to 0, then add AC1 contents to AC2, shift the result left one bit, keep the result, but skip on carry set "will create the mnemonic:

ADDZL AC1,AC2,SNC

All logic associated with the execution of arithmetic/logic instructions is provided by the MicroNova and the 9440 chips.

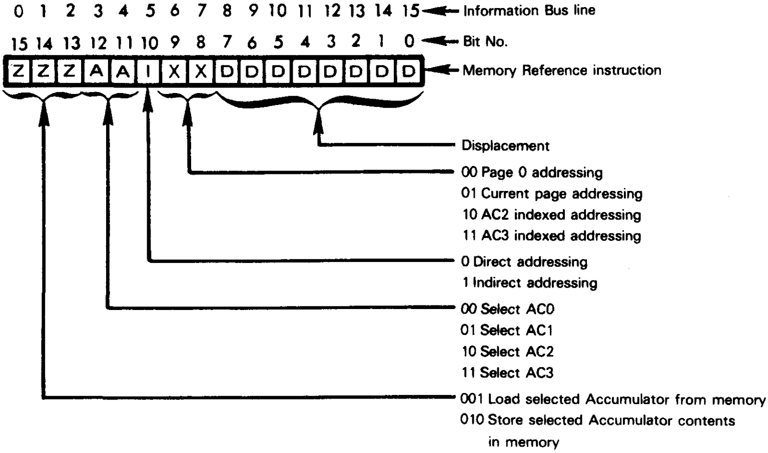

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Information Bus line
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bit No.
Z Z Z A A I X X D D D D D D D D
Memory Reference instruction
Displacement
00 Page 0 addressing
01 Current page addressing
10 AC2 indexed addressing
11 AC3 indexed addressing
0 Direct addressing
1 Indirect addressing
00 Select AC0
01 Select AC1
10 Select AC2
11 Select AC3
001 Load selected Accumulator from memory
010 Store selected Accumulator contents in memory

Figure 17-6. Load And Store Instruction Object Codes

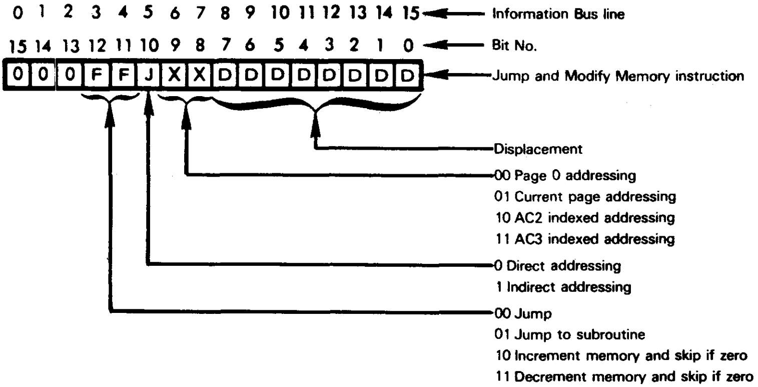

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Information Bus line
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bit No.
0 0 0 F F J X X D D D D D D D D
Jump and Modify Memory instruction
Displacement
00 Page 0 addressing
01 Current page addressing
10 AC2 indexed addressing
11 AC3 indexed addressing
0 Direct addressing
1 Indirect addressing
00 Jump
01 Jump to subroutine
10 Increment memory and skip if zero
11 Decrement memory and skip if zero

Figure 17-7. Jump And Modify Memory Instruction Object Codes

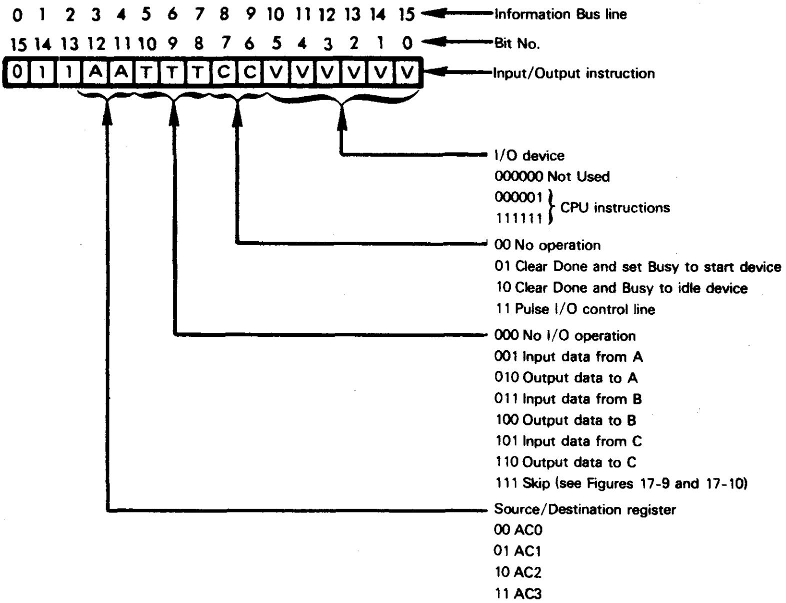


Figure 17-8. General Input/Output Instruction Object Code Interpretation

MEMORY REFERENCE INSTRUCTIONS

Since the four Accumulators of the Nova CPU must provide data sources and destinations for all arithmetic and logic instructions, you will constantly move data between memory and one of the four Accumulators. We have already described the Nova addressing modes. **Figure 17-6 illustrates memory reference instruction object codes and addressing mode specifications.** You can load data into any Accumulator, or you can store the contents of any Accumulator in memory.

There are four Jump and Modify Memory instructions. Object codes are given in Figure 17-7. The memory addressing options described earlier in the chapter apply also to the Jump and Modify Memory instructions.

The Jump-to-Subroutine instruction requires special mention; this instruction **stores the subroutine return address in Accumulator AC3.** If you are going to nest subroutines then **you must write your own subroutine to create a software stack.** Note that **even the MicroNova, which has a stack, does not use it when a Jump-to-Subroutine instruction is executed.**

MicroNova and 9440 chips provide all effective memory address computation logic and reduce memory reference instructions, as external logic sees them, to typical address and data transmissions with accompanying control strobe signals.

But remember, there is no such thing as a "standard" Nova memory bus.

INPUT/OUTPUT INSTRUCTIONS

Figure 17-8 illustrates input/output instruction object code interpretations.

NOVA I/O DEVICE BUSY AND DONE STATUS

Every I/O device that communicates with a Nova minicomputer must have a Busy status and Done status. These are bidirectional statuses; they are modified by the CPU to control the I/O device and they are modified by the I/O device to indicate the status of the I/O operation. **This is how the Busy and Done statuses are interpreted:**

BUSY	DONE	
0	0	Device Idle
1	0	CPU "starts" device by setting Busy to 1.
0	1	Device resets Busy to 0 and sets Done to 1 when device operation is complete.
1	0	CPU resets Done to idle device, or sets Busy for next device operation.
1	1	Illegal

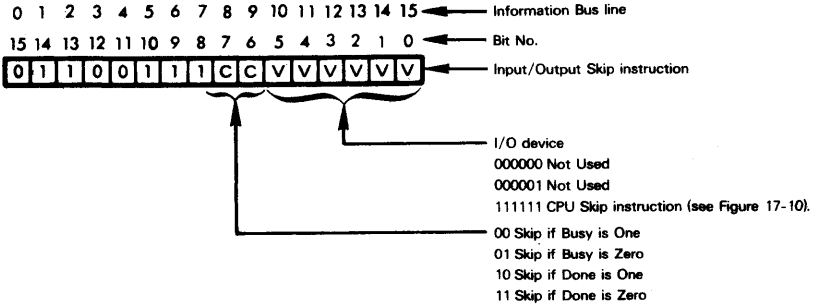


Figure 17-9. Input/Output Skip Instruction Object Code Interpretation

You start and stop I/O devices by manipulating device Busy and Done statuses.

NOVA I/O DEVICE REGISTERS

Every I/O device may optionally have up to three individually addressable registers, referred to as Registers A, B and C.

You transfer data between one of the four CPU Accumulators and one of the three I/O device registers.

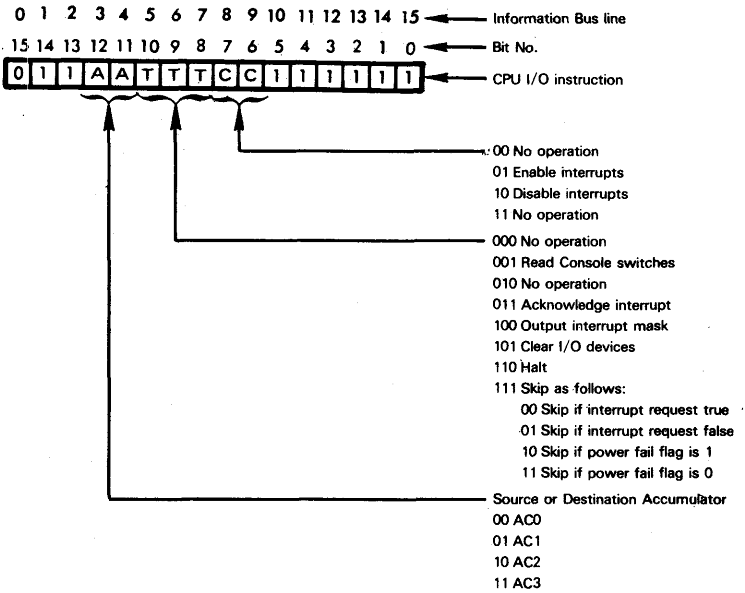


Figure 17-10. CPU Device $3F_{16}$ Input/Output Instruction Object Code Interpretation

Both a status manipulation and a data transfer may be specified by a single I/O instruction; these two operations occur in parallel and are supported by appropriate control signals on the I/O bus.

The Nova CPU must be able to poll the Busy and Done statuses of an I/O device, just as most microprocessors read the contents of an I/O device Status register. The Nova CPU responds to status condition tests by optionally performing a Skip (which means the Program Counter contents are incremented). **This variation of I/O instructions is illustrated in Figure 17-9.**

NOVA I/O DEVICE ADDRESS SPACE

Six bits of every I/O instruction object code are used to identify the I/O device being addressed. This gives you a total of 64 devices in the I/O device address space. But in order to enhance its instruction set, the Nova uses selected I/O device numbers to encode instructions internal to the CPU. I/O device numbers 0, 1 and $3F_{16}$ are reserved for this purpose. **I/O device $3F_{16}$ selects a number of interrupt related instructions whose object codes are defined in Figure 7-10. I/O device numbers 0 and 1 implement instructions illustrated in Figure 17-11.**

You will have to add considerable logic beyond the 9440, or the MicroNova, if you are going to execute all I/O instructions described in Figures 17-8, 17-9, 17-10 and 17-11. The only logic provided by the CPU chips themselves support that part of the I/O operation which is exclusively internal to the CPU — and that is not much. The CPU will route data to or from the selected Accumulator, if needed, and it will increment the Program Counter in response to a Skip true condition. Everything else is the responsibility of logic beyond the CPU chip.

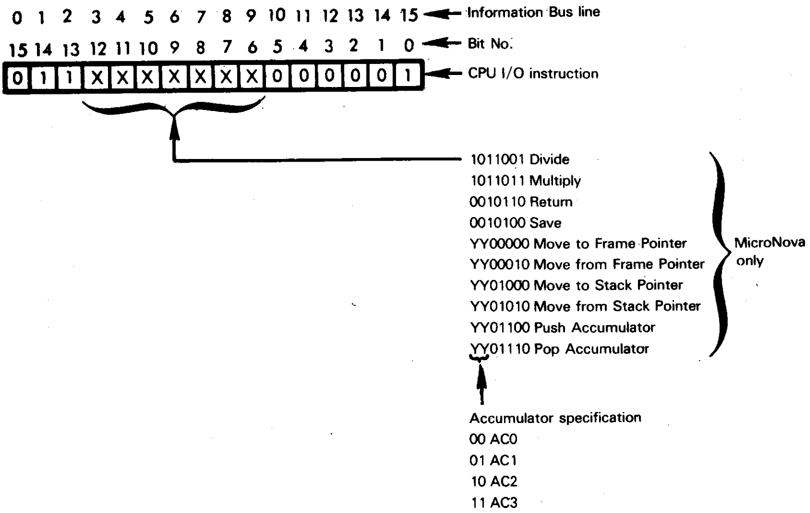


Figure 17-11. CPU Device 1 Input/Output Instruction Object Code Interpretation

A NOVA CPU SUMMARY

If you compare Nova CPU logic with microprocessors described earlier in this book, a number of minicomputer characteristics become self-evident. These characteristics have important implications when we look at bus signals, interfaces and timing; therefore they must be clearly defined.

Minicomputer Central Processing Units are more complex than their microprocessor counterparts. Look at the number of operations which may be performed during execution of a single Nova instruction; only the SMS300 makes any attempt to provide such serial logic. The microprocessor CPU architect has been severely restricted by the fact that only a limited amount of logic can be put on a chip without drastically affecting chip yield — and therefore the price of the microprocessor. When minicomputers were designed, making CPU logic more complex increased the size of the CPU card, or cards, which had some effect on eventual product price, but nothing like the microprocessor price escalations that result from low chip yields.

Thus unconstrained by logic limitations, minicomputer CPU architects also designed complex system busses, requiring equivalently complex logic within I/O devices attached to the system busses. For example, consider the fact that Figure 17-5 defines 32,768 different Register-Register Operate instructions, while the instruction format in Figure 17-8 assumes an I/O System Bus that can simultaneously manipulate I/O device status while transferring data.

These are formidable burdens placed on the designer of a chip which is supposed to reproduce the Nova CPU — with the result that chip designers have elected to tackle only part of the task. Both the MicroNova and the 9440 terminate at 40-pin DIPs; their busses are, in consequence, less than the standard Nova System Busses.

9440 TIMING AND INSTRUCTION EXECUTION

We will now examine 9440 instruction timing in detail.

9440 instructions and internal logic are timed by a master 10 MHz clock signal. Instructions are executed in machine cycles. This is the number of clock periods per machine cycle:

Memory read/instruction fetch	-	15 clock periods	Depends on actual memory timing
Memory write	-	14 clock periods	
I/O data in	-	10 clock periods	
I/O data out	-	10 clock periods	

Let us begin by looking at timing for an instruction fetch or a memory read; these two machine cycles have the timing illustrated in Figure 17-12.

9440 INSTRUCTION FETCH

9440 MEMORY READ

At the end of clock period 2, the three memory control signals $\overline{M0}$, $\overline{M1}$ and $\overline{M2}$ are output with levels that identify the memory access which will be performed during the current machine cycle. For a memory read or instruction fetch, $\overline{M0}$ and $\overline{M2}$ are output low while $\overline{M1}$ remains high.

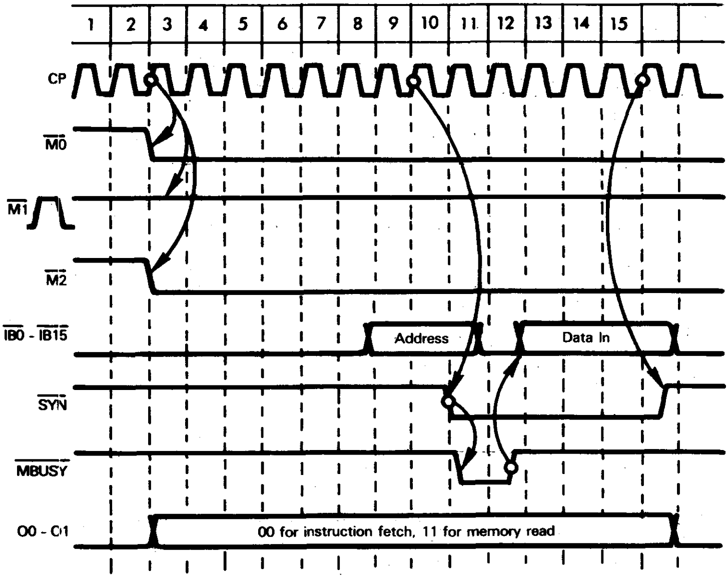
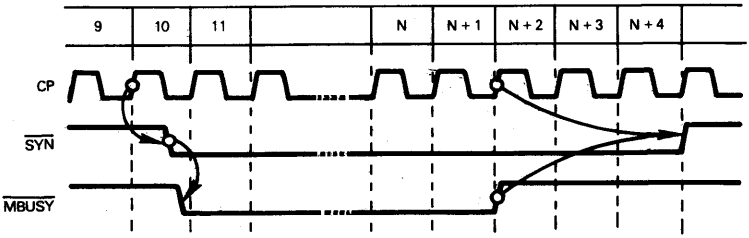


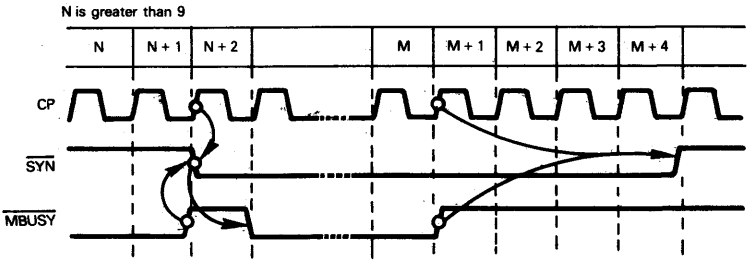
Figure 17-12. 9440 Memory Read/Instruction Fetch Timing

An instruction fetch and a memory read are differentiated by signals O0 and O1; these signals are both low for an instruction fetch and both high for a memory read. The address of the memory location to be accessed is output on the Information Bus (IB0 - IB15) beginning at the end of clock period 8. At the end of clock period 9 $\overline{\text{SYN}}$ is output low; external logic must use the high-to-low transition of $\overline{\text{SYN}}$ as a strobe to latch an address off the Information Bus. External logic must also use the high-to-low transition of $\overline{\text{SYN}}$ as a trigger to input $\overline{\text{MBUSY}}$ low to the 9440. $\overline{\text{MBUSY}}$ must be input low until addressed data has been read from memory and is stable on the Information Bus. At that time $\overline{\text{MBUSY}}$ goes high again. When $\overline{\text{MBUSY}}$ goes high, the 9440 will read data off the Information Bus. If the Memory Read machine cycle is to execute in the minimum 15 clock periods, then $\overline{\text{MBUSY}}$ must be low for two clock periods only.

$\overline{\text{MBUSY}}$ is a signal used by external memory interface logic to synchronize itself with the CPU. If $\overline{\text{MBUSY}}$ is low while $\overline{\text{SYN}}$ is high, early in any memory access machine cycle, then the high-to-low transition of $\overline{\text{SYN}}$ will be delayed until $\overline{\text{MBUSY}}$ goes high. For a Memory Read or Instruction Fetch machine cycle, the trailing edge of the low $\overline{\text{MBUSY}}$ pulse also acts as an end-of-machine-cycle trigger. Three clock periods after $\overline{\text{MBUSY}}$'s low-to-high transition, the machine cycle ends and $\overline{\text{SYN}}$ goes high again. Here is an example of $\overline{\text{MBUSY}}$ and $\overline{\text{SYN}}$ interaction during termination of a Memory Read or Instruction Fetch machine cycle:



$\overline{\text{MBUSY}}$ and $\overline{\text{SYN}}$ interaction at the high-to-low $\overline{\text{SYN}}$ transition may be illustrated as follows:



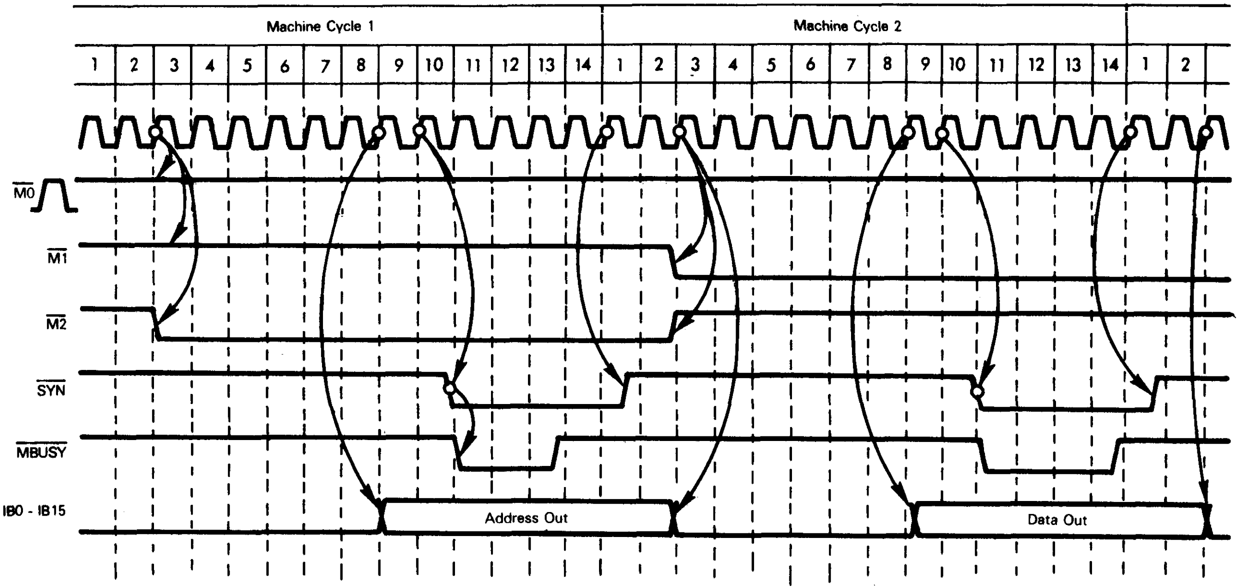


Figure 17-13. 9440 Memory Write Timing

Every instruction's execution will begin with an instruction fetch machine cycle. This machine cycle will be followed by internal operations, another memory read, a memory write, an I/O read, or an I/O write.

If the instruction to be executed requires operations only, that is, it is an arithmetic/logic instruction, then **internal operations are executed during clock periods 1 through 8 of the next machine cycle** — which must be another instruction fetch machine cycle.

If a memory read operation is to be performed, then another machine cycle is executed, exactly equivalent to Figure 17-12.

If a memory write is to be performed, then two machine cycles must follow the instruction fetch. During the first machine cycle the external memory address is output. During the second machine cycle data to be written to memory is output. Timing is illustrated in Figure 17-13. This figure is self-evident. During the first machine cycle only $\overline{\text{M2}}$ is low since a memory address is being output without a read or a write operation occurring during the same machine cycle. During the second machine cycle only $\overline{\text{M1}}$ is output low since a memory write operation alone will occur.

During both machine cycles of a Memory Write operation, $\overline{\text{MBUSY}}$ acts as a synchronizing signal, however only the high-to-low transition of $\overline{\text{MBUSY}}$ can modify instruction execution time. If $\overline{\text{MBUSY}}$ is low prior to $\overline{\text{SYN}}$ making its high-to-low transition, then the $\overline{\text{SYN}}$ high-to-low transition will be delayed until $\overline{\text{MBUSY}}$ goes high. Once $\overline{\text{SYN}}$ goes low, however, the memory write machine cycles will terminate five clock periods later. The subsequent low-to-high transition of $\overline{\text{MBUSY}}$ has no effect on the $\overline{\text{SYN}}$ signal, or on internal CPU operations.

The only memory addressing modes that change instruction execution time are indirect addressing and indirect addressing with auto-increment or auto-decrement.

Each level of indirect addressing is equivalent to an additional memory read and an additional memory write. In order to compute instruction execution times for memory references with indirect addressing, therefore, add one memory read machine cycle and one memory write machine cycle for each level of indirection.

Recall that memory locations 10_{16} through $1F_{16}$ are used to store addresses which, when accessed indirectly, will be incremented or decremented. When you use indirect addressing and specify a memory location from 10_{16} through 17_{16}, the address fetched from the specified location will be incremented. An indirect address fetched from locations 18_{16} through $1F_{16}$ will be decremented. The increment or decrement operation requires the memory address to be loaded into the CPU, incremented or decremented, then written back out. Loading the address into the CPU is a routine part of any indirect addressing sequence; however, writing the address back out represents an additional step requiring two additional memory write machine cycles. This may be illustrated as follows:

Machine Cycle 1 Instruction fetch	Machine Cycle 2 Fetch address from location 10_{16} - $1F_{16}$	Machine Cycle 3 Increment or decrement address and write address back	Machine Cycle 4 Perform memory access (read or write)
		Memory Write	

The increment or decrement and Skip-if-Zero instructions require an instruction fetch, a memory read and a memory write machine cycle. Timing may be illustrated for direct memory addressing as follows:

Machine Cycle 1 Instruction fetch	Machine Cycle 2 Fetch data from memory	Machine Cycle 3 Increment or decrement data and write data back	Machine Cycle 4 Increment Program Counter if needed
		Memory Write	

Let us now look at I/O instruction execution.

There are no special I/O device select or control signals output by the 9440, rather external I/O devices must have select logic which is created by decoding instruction object codes on the Information Bus. This is done by decoding the three high order Information Bus lines during an instruction fetch, as characterized by O0 and O1 both low. The three high order Information Bus lines will at this time be 011 if the instruction to be executed is an I/O instruction. If these conditions are met, then the six low order Information Bus lines must be decoded by device select logic. If the device code is $3F_{16}$, then all I/O devices must be selected simultaneously; for this to occur a special overriding device select signal must be created in response to device code 3F. If device code 00_{16} occurs, then no device should be selected; this requires no special select logic, rather it means that no external device should have the address 00_{16}. If any device code other than 00_{16}, or $3F_{16}$ appears on the six low order Information Bus lines, then one external device's select logic should go true.

An actual example of I/O device logic is given later in this chapter.

If device code $3F_{16}$ has been output, then one of the operations defined by Figure 17-10 is about to occur. A significant amount of external logic associated with execution of these instructions may be required. A specific implementation consistent with standard Nova 1200 I/O interface logic is given later in this chapter. Alternatively, you may create a variety of individual control signals unrelated to the standard Nova I/O bus by suitably decoding I/O instruction object code bits 10 through 6.

An I/O instruction which identifies a specific device further identifies the I/O operations which are to occur, via bits 10 through 6 of the instruction object code (Information Bus lines $\overline{\text{IB5}}$ through $\overline{\text{IB9}}$). Figures 17-8 and 17-9 provide the I/O operations which may be specified. **If data is to be input or output, then timing will conform to Figures 17-14 and 17-15.** But a significant amount of parallel control logic will accompany any I/O data transfer. We will shortly describe logic which implements a typical I/O device interface.

An I/O Skip on Busy or Done instruction, as illustrated in Figure 17-9, requires the addressed I/O device to return Busy and Done statuses to the CPU. The addressed I/O device returns these statuses on the two high order Information Bus lines $\overline{\text{IB0}}$ and $\overline{\text{IB1}}$, with timing conforming to Figure 17-14.

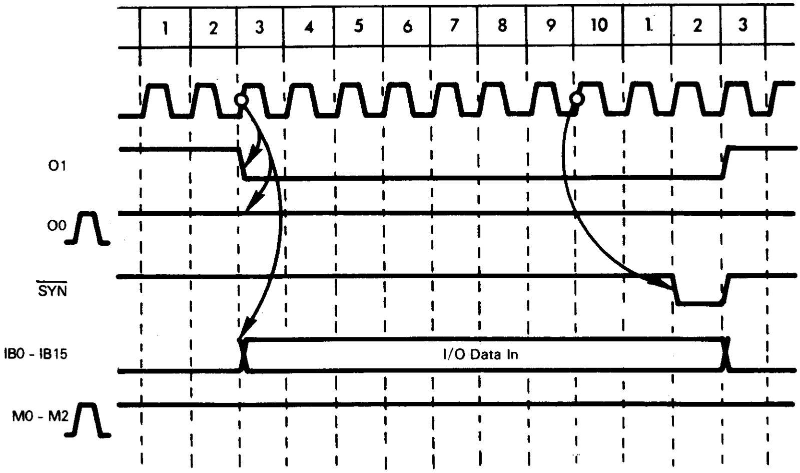


Figure 17-14. 9440 I/O Data Input Timing

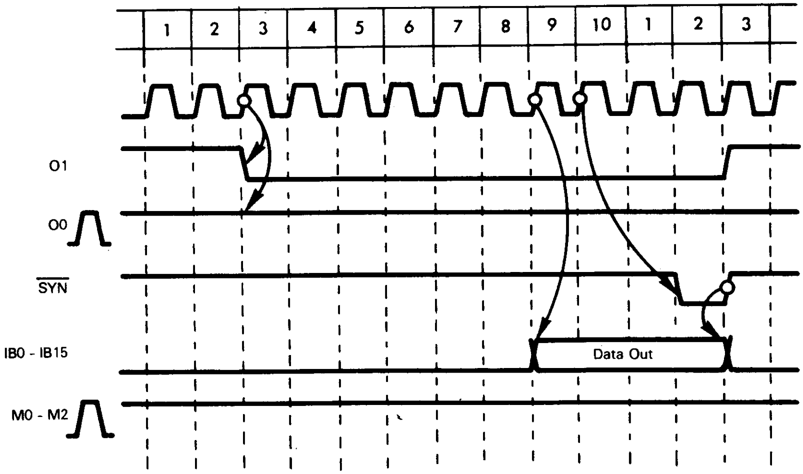


Figure 17-15. 9440 I/O Data Output Timing

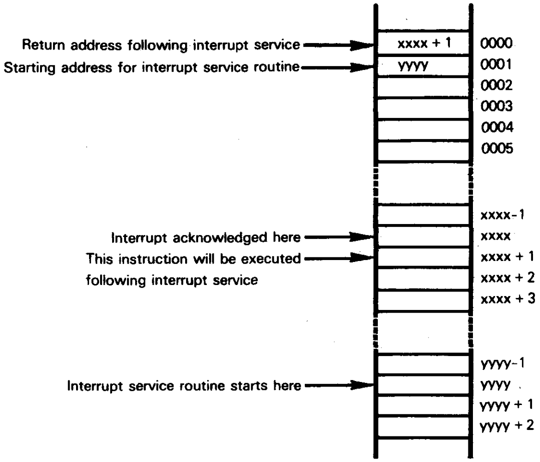
MICRONOVA AND 9440 INTERRUPT PROCESSING

At the most elementary level, the MicroNova and the 9440 respond to interrupts in a very simple way.

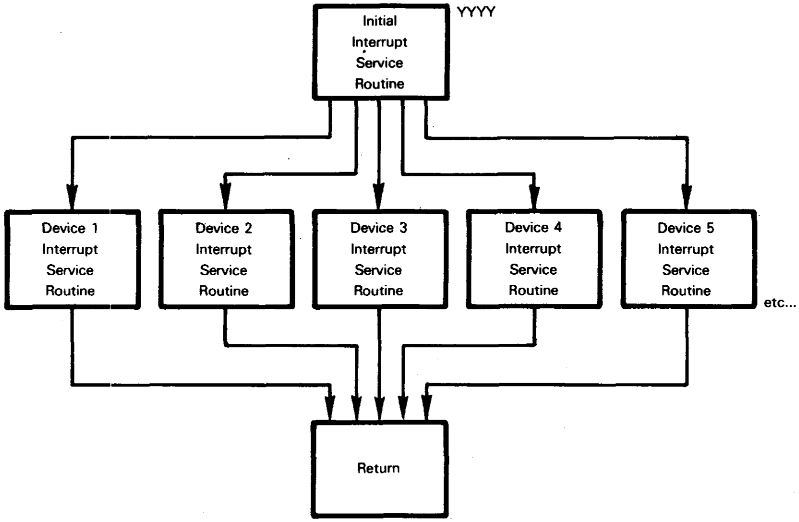
External logic requests an interrupt by inputting a low signal via $\overline{\text{INTREQ}}$.

Providing interrupts are enabled, the CPU acknowledges the interrupt upon completing execution of the current instruction; the CPU disables its own interrupt logic, saves the Program Counter contents in memory location 0000, then jumps

indirect to location 0001. Thus memory location 0001 must contain the address of the first interrupt service routine instruction.


Return address following interrupt service
Starting address for interrupt service routine
xxxx + 1
yyyy
0000
0001
0002
0003
0004
0005
xxxx-1
xxxx
xxxx + 1
xxxx + 2
xxxx + 3
Interrupt acknowledged here
This instruction will be executed following interrupt service
yyyy-1
yyyy
yyyy + 1
yyyy + 2
Interrupt service routine starts here

A single interrupt service routine will be executed in response to any external interrupt. In order to discriminate between interrupts, the interrupt service routine must identify the source of the interrupt, then jump to an appropriate individual program. This may be illustrated as follows:


YYYY
Initial Interrupt Service Routine
Device 1 Interrupt Service Routine
Device 2 Interrupt Service Routine
Device 3 Interrupt Service Routine
Device 4 Interrupt Service Routine
Device 5 Interrupt Service Routine
etc...
Return

There will be a separate device interrupt service routine for every I/O device capable of representing an interrupt.

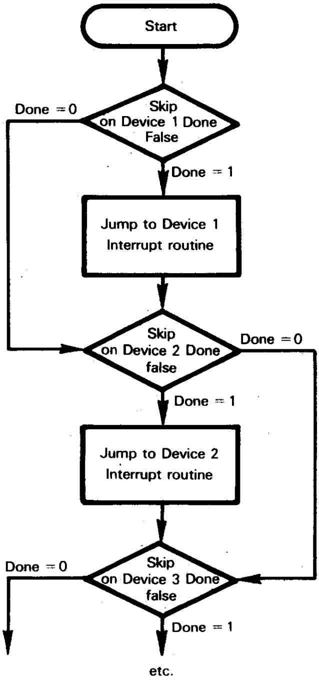
There are many ways in which the initial interrupt service routine may identify the interrupting I/O device in a multiple interrupt configuration.

The most primitive method used to identify an interrupting I/O device **is to test the device's Done status.** Standard Nova protocol requires an I/O device to request an interrupt when it sets its Done status. This may be illustrated as follows:

Interrupt Request	Busy	Done	
False	0	0	Device idle
False	1	0	Start I/O operation
True	0	1	End I/O operation

Primitive I/O device interface logic will request an interrupt by applying a low signal at $\overline{\text{INTREQ}}$ when it sets its Done status high. Now the initial interrupt service routine will execute a sequence of "Skip on Done False" instructions in order to identify the highest priority interrupting device. This may be illustrated as follows:

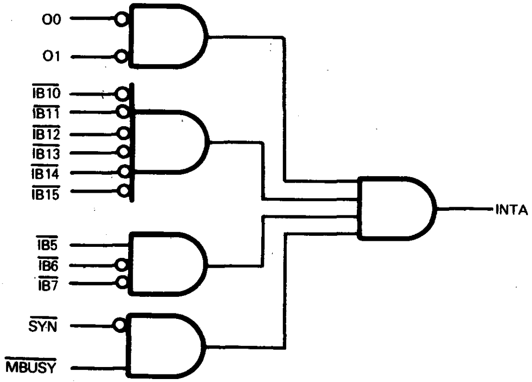
```
SKPDZ     DEV1

JMP       IDEV1

SKPDZ     DEV2

JMP       IDEV2

SKPDZ     DEV3

etc
```



Start
Done = 0
Skip on Device 1 Done False
Done = 1
Jump to Device 1 Interrupt routine
Skip on Device 2 Done false
Done = 0
Done = 1
Jump to Device 2 Interrupt routine
Done = 0
Skip on Device 3 Done false
Done = 1
etc.

The order in which the initial interrupt service routine program logic tests device Done statuses becomes interrupt priority. You can modify this priority sequence at any time simply by changing the program.

A faster method of identifying an interrupting device is to daisy chain the interrupting devices. Daisy chain logic has been described in Volume I, and again in Chapter 6 of this book (in conjunction with the 8048). Daisy chains are resolved by an interrupt acknowledge signal; but there is no interrupt acknowledge signal output by the MicroNova or the 9440; rather an interrupt acknowledge instruction is executed. This is an I/O instruction addressing device $3F_{16}$; bits 10 through 6 ($\overline{IB5}$ through $\overline{IB9}$) of the instruction object code must be decoded in order to create an interrupt acknowledge signal. Here is appropriate logic:



Recall that the Information Bus is low true; that is, a low logic level represents a bit value of 1. To ensure that INTA is generated only when a valid instruction code is on the Information Bus, it should be qualified by $\overline{SYN}$ low and $\overline{MBUSY}$ high. This is illustrated in Figure 17-16.

The highest priority interrupting device identifies itself by placing its device code on the Information Bus lines. The CPU stores the device number in one of the four Accumulators. Thus the interrupt acknowledge instruction is an I/O Data In instruction. Interrupt acknowledge timing is illustrated in Figure 17-16.

Interrupt enable and disable logic exists separately at the CPU and at external I/O devices.

At the CPU all interrupts are disabled as soon as an interrupt is detected. You can disable interrupts at any other time by executing a disable interrupt instruction (NIOC CPU).

In order to enable interrupts you must execute an interrupt enable instruction (NIOS CPU); when an NIOS CPU instruction is executed, interrupts are enabled following execution of the next instruction. This next instruction will usually be a Return instruction:

```
    -
    -
    -
    NIOS    CPU     ;Enable interrupts
    JMP     @0      ;Return from interrupt service routine
                    ;Interrupts are now enabled
```

When nested interrupts are not allowed, all interrupts are disabled following the interrupt detection; interrupts remain disabled until the end of the interrupt service routine. You terminate the interrupt service routine with the two instructions illustrated above; one re-enables interrupts, the other returns from the interrupt service routine. Interrupts are not actually re-enabled until after the Return instruction has been executed; this prevents pending interrupts from being acknowledged before you have finally exited the current interrupt service routine.

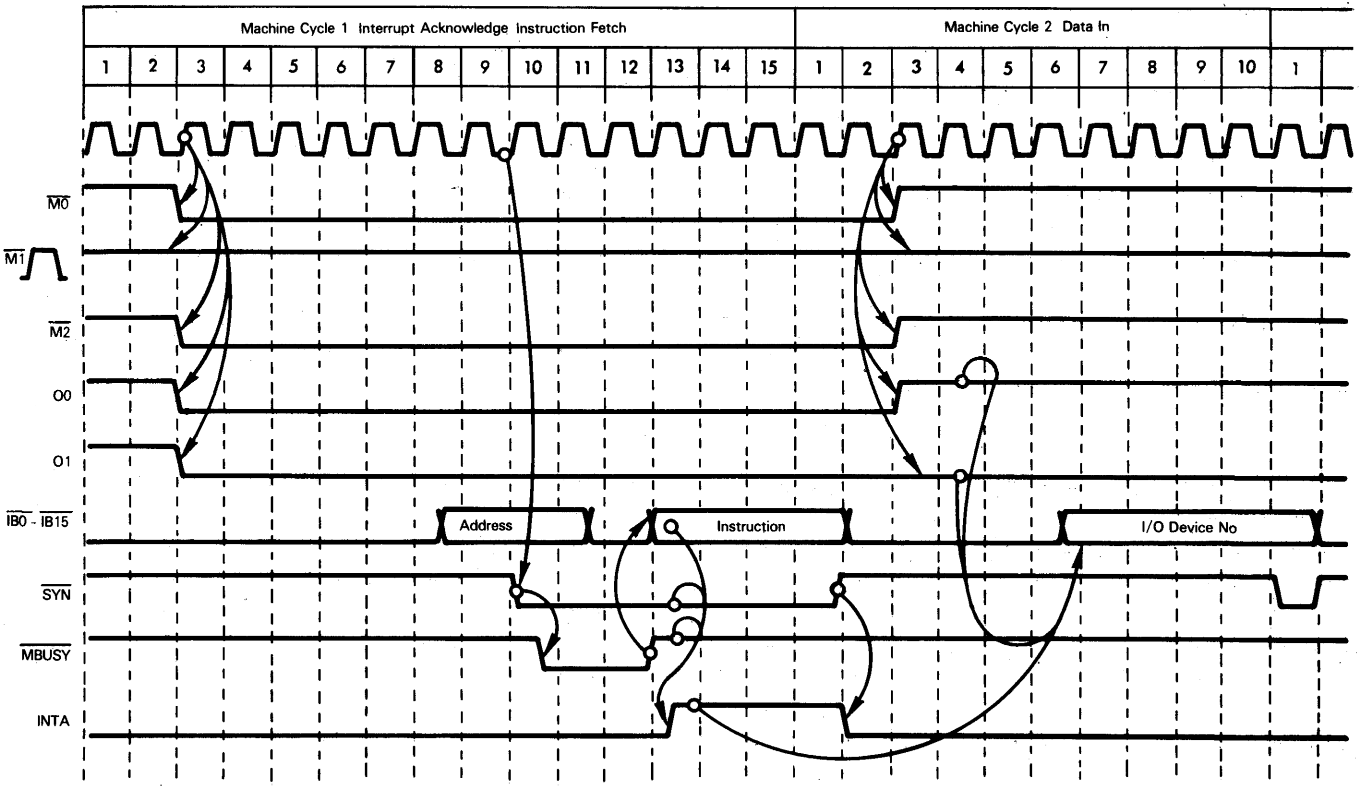


Figure 17-16. 9440 Interrupt Acknowledge Instruction Execution Timing

If you want to nest interrupts then you must execute an interrupt enable instruction within the interruptable interrupt service routine. But make sure that you do not re-enable interrupts until the initial interrupt service routine has executed; remember, the initial interrupt service routine is determining the source of the interrupt — and it makes no sense to allow another interrupt to occur until this determination has been completed.

You can disable interrupts selectively at external devices that have local interrupt disable logic. This is done using the Mask Out instruction (MSKO); MSKO is another I/O instruction addressing device $3F_{16}$. The MSKO instruction outputs data from one of the CPU Accumulators onto the Information Bus. Every I/O device capable of having its interrupt logic disabled must be connected to one of the Information Bus lines. When the MSKO instruction is executed, the I/O device must first decode the MSKO instruction in order to activate its interrupt disable logic; subsequently, if the Information Bus line to which device interrupt disable logic is connected is low, then interrupt request logic must be disabled locally. Timing is illustrated in Figure 17-17.

In order to re-enable interrupts at any external device you output a new mask with a high level on the Information Bus line to which the device's interrupt disable logic is connected.

Interrupt logic again demonstrates the minicomputer emphasis of the Nova. We have assumed that an external device capable of requesting interrupts can decode I/O instruction object codes on the Information Bus and have a considerable amount of logic associated with Busy, Done and Interrupt request flags. We will shortly look at a 9440 implementation of suitable interrupt logic on an I/O device controller.

MICRONOVA AND 9440 DIRECT MEMORY ACCESS LOGIC

MicroNova and 9440 direct memory access logic differ markedly.

In both cases external logic represents a DMA access by inputting a low signal via $\overline{\text{DCH REQ}}$.

The MicroNova responds by acknowledging the DMA request. This is done by outputting a high $\overline{\text{I/O DATA1}}$ with a low $\overline{\text{I/O DATA2}}$ signal. External logic then identifies the direction of the data transfer via the $\overline{\text{I/O INPUT}}$ control signal. Subsequently, MicroNova logic performs the entire DMA transfer by creating appropriate I/O Bus and Memory Bus signal sequences — but only data input or data output is allowed.

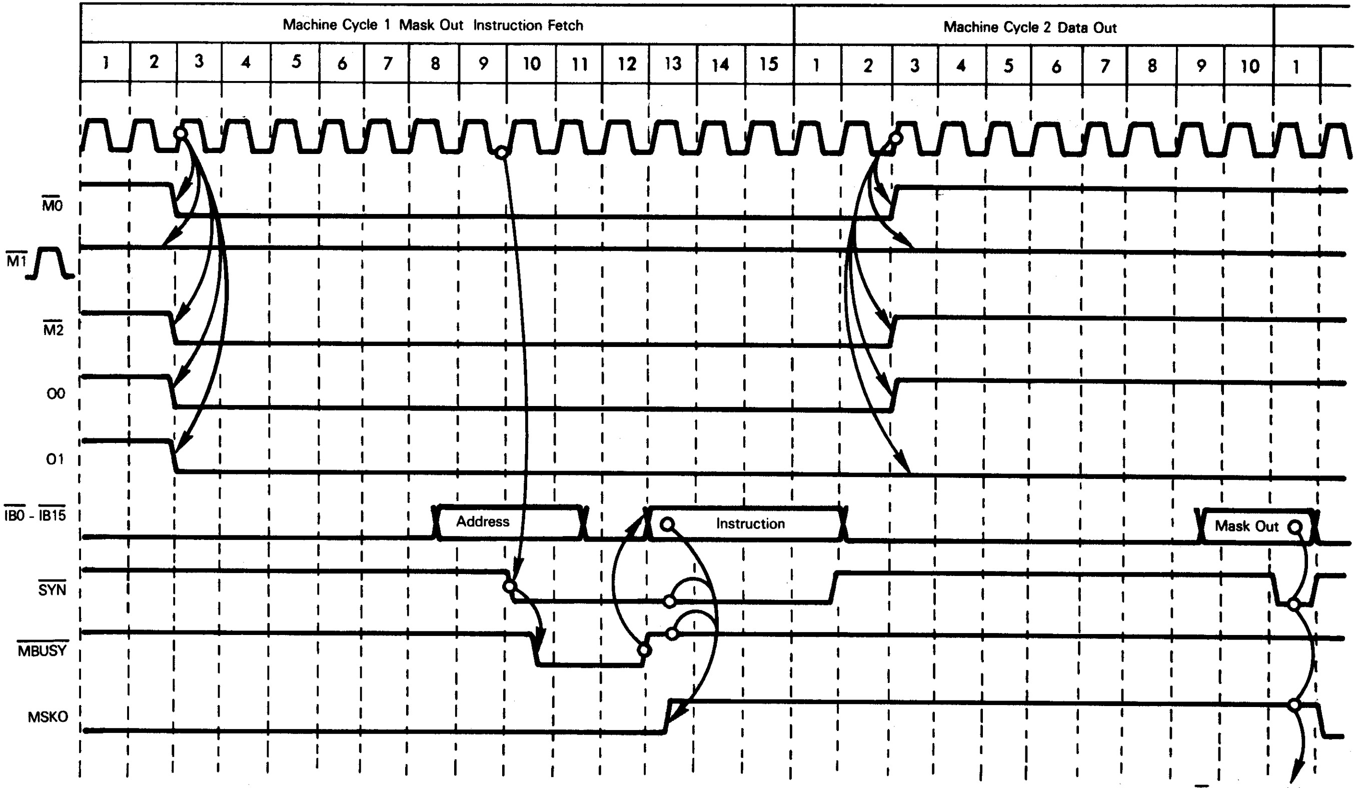
The 9440 has a more primitive DMA capability. It responds to $\overline{\text{DCH INT}}$ by outputting lines O0 and O1 low and high, respectively, and floating the Data Bus. External logic must implement the actual DMA transfer.

Standard Nova protocol allows four DMA operations to be defined by external logic via the DCHM0 and DCHM1 I/O bus signals. These are the four DMA operations that may be defined:

$\overline{\text{DCHM0}}$	$\overline{\text{DCHM1}}$	
0	0	Add to memory
0	1	Data in
1	0	Increment memory
1	1	Data out

The MicroNova, as we have already stated, handles data in and data out only; increment memory and add to memory are not available.

The 9440 on the other hand, **does nothing in response to a DMA request other than float the Information Bus. All external logic associated with DMA operations must exist outside the 9440 chip. We will describe suitable logic later in this chapter.**



Disable interrupt if $\overline{IB}$ line is low (1)
Enable interrupt if $\overline{IB}$ line is high (0)

Figure 17-17. 9440 Mask Out Instruction Execution Timing

THE MICRONOVA AND 9440 INSTRUCTION SETS

Table 17-2 summarizes the instruction sets for the MicroNova and the 9440. Observe that there are some instructions available with MicroNova that the 9440 lacks.

The power of the Nova instruction set is derived from the fact that many instructions perform multiple operations. Register Operate instructions, for example, allow you to set, or reset or complement a Carry status before the specified operation is performed. Primary Memory Reference and Register Operate instructions allow you to also perform data shifts, or to swap the high and low order bytes of the data word being moved or generated.

Primary Memory Reference and Register Operate instructions also allow you to perform a conditional skip based on the results of the operation.

It is the ability of the Nova instruction set to perform a combination of operations, during a single instruction's execution, that makes the instruction set so effective.

THE BENCHMARK PROGRAM

Our benchmark program may be illustrated as follows for the MicroNova and the 9440:

```
        LDA   2,CNT      LOAD WORD COUNT COMPLEMENT INTO AC2
        LDA   0,IOBUF    LOAD IOBUF BASE ADDRESS INTO AUTO-
        STA   0,10       INCREMENT LOCATION
        LDA   0,@TABLE   LOAD ADDRESS OF FIRST FREE TABLE WORD
        STA   0,11       INTO AUTO-INCREMENT LOCATION
LOOP    LDA   0,@10      LOAD NEXT BYTE FROM IOBUF
        STA   0,@11      STORE IN NEXT TABLE WORD
        INC   2,2,SZR    INCREMENT WORD COUNT SKIP IF ZERO
        JMP   LOOP       RETURN FOR MORE
        LDA   0,21       RETURN NEW ADDRESS OF FIRST FREE TABLE
        STA   0,@TABLE   WORD
```

This benchmark program uses indirect addressing with auto-incrementing in order to sequentially access IOBUF and TABLE. We begin the program by loading the word count (CNT) into Accumulator 2, and table base addresses into memory words 20_8 and 21_8. We assume that the address of the first free word in TABLE is stored in the first word of TABLE; thus we can fetch the address of the first free TABLE word by executing a load to Register 0 with indirect addressing.

Data is moved by a four-instruction loop. Two instructions load data from IOBUF and store data in TABLE using indirect addressing with auto-increment. Next we increment the counter stored in Register 2 and skip the following instruction upon detecting a zero count. The following instruction is a jump back to the beginning of the loop.

The final two instructions simply restore the new address for the first free TABLE word into the first word of the TABLE.

The benchmark program makes no assumptions. The source and destination tables may be any size and any number of data words may be transferred, limited only by the available memory space.

The following notation is used in Table 17-2.

An "X" in the column labeled "9440" indicates that the instruction is available on the 9440 CPU.

AC   Any of the four Accumulators.

ACX   A specific Accumulator. For example, AC1 is Accumulator 1.

C Carry status

D An Accumulator which serves as the destination for the results of an operation.

DEV A 6-bit device code.

DEVX A specific device register. For example, DEVA is Device Register A.

DEVBD Device Busy-Done flags.

EA Effective address determined by @DISP (,IX).

FP Frame Pointer (not present in 9440).

ION Interrupt ON flag

PC Program Counter

PM Priority Mask

S An Accumulator which serves as the source of an operand.

SP Stack Pointer (not present in 9440).

(CS#) Represents three options which are used by the Register-Register operations.

C is a 2-bit field which determines the carry state prior to the ALU operation.

Coded Character	Result Bits	Operation
option omitted	00	No operation
Z	01	Set carry to 0
O	10	Set carry to 1
C	11	Complement carry

For example, ADDO 2,2 would set carry to 1 before adding AC2 to AC2.

S is a 2-bit field which determines how the result of the ALU will be shifted.

Coded Character	Result Bits	Operation
option omitted	00	No shift
L	01	Shift result and carry left one bit
R	10	Shift result and carry right one bit
S	11	Swap result bytes

For example, MOVS 1,2 would swap the bytes of AC1 and store into AC2.

is a 1-bit field which determines whether the result is stored in ACD.

Coded Character	Result Bits	Operation
option omitted	0	Load result into ACD
#	1	Do not load result into ACD

For example, NEGOL# 1,2 would set carry to 1 then negate AC1, shift the result and carry left one bit, but would not store into AC2.

(f) A 2-bit I/O command whose meaning depends on whether the CPU or another device is being referenced.

CPU	f	Device
No operation	00	No operation
Set Interrupt On to 1	01	Start device by setting Busy to 1 and Done to 0
Set Interrupt On to 0	10	Idle device by setting Busy to 0 and Done to 0
No operation	11	Pulse a special device dependent line

(,SKCND) A 3-bit skip-on-condition field which is used by the Register-Register Operate instructions.

Coded Character	Result Bits	Operation
option omitted	000	No operation
SKP	001	Always skip
SZC	010	Skip if Carry = 0
SNZ	011	Skip if Carry = 1
SZR	100	Skip if result = 0
SNR	101	Skip if result ≠ 0
SEZ	110	Skip if either carry or result = 0
SBN	111	Skip if both carry and result ≠ 0

(@) DISP (,IX) Generates the address EA

@ is the indirect bit. If @=1 then indirection is specified.

DISP is an 8-bit address value.

(IX) is a 2-bit field which indicates the addressing Mode:

Bits are	Mode
00	Zero page addressing. DISP is an unsigned address between 0 and 256. EA = DISP
01	PC relative addressing. DISP is a signed two's complement address displacement. EA = DISP+[PC]
10	Indexed addressing via AC2. DISP is a signed two's complement address displacement. EA = DISP+[AC2]
11	Indexed addressing via AC3. DISP is a signed two's complement address displacement. EA = DISP+[AC3]

(t) A 2-bit I/O test field whose meaning depends on whether the CPU or another device is referenced.

CPU	t	Device
Test for Interrupt On=1	00	Test for Busy=1
Test for Interrupt On=0	01	Test for Busy=0
Never skip	10	Test for Done=1
Always skip	11	Test for Done=0

x<y,z> Bits y through z of the quantity x. [AC]<5,0> is the low six bits of the specified Accumulator.

[ ] Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.

[[ ]] Implied memory addressing; the contents of the memory location designated by the contents of a register.

Λ Logical AND

← Data is transferred in the direction of the arrow.

Under the heading of STATUS in Table 17-2, an X indicates statuses which are modified in the course of the instruction's execution. If there is no X, it means that the status maintains the value it had before the instruction was executed.

Table 17-2. MicroNova And 9440 Instruction Set Summary

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	9440	OPERATION PERFORMED
				C		
I/O	NIO (f)	DEV	2		X	[DEVBD] ← f Set the device's Busy and Done flags according to I/O command.
	DIA (f)	AC,DEV	2		X	[AC] ← [DEVA] [DEVBD] ← f Read device's A buffer into Accumulator. Set the device Busy and Done flags.
	DIB (f)	AC,DEV	2		X	[AC] ← [DEVB] [DEVBD] ← f Read device's B buffer into Accumulator. Set the device Busy and Done flags.
	DIC (f)	AC,DEV	2		X	[AC] ← [DEVC] [DEVBD] ← f Read device's C buffer into Accumulator. Set the device Busy and Done flags.
	DOA (f)	AC,DEV	2		X	[DEVA] ← [AC] [DEVBD] ← f Write Accumulator into device's A buffer. Set the device Busy and Done flags.
	DOB (f)	AC,DEV	2		X	[DEVB] ← [AC] [DEVBD] ← f Write Accumulator into device's B buffer. Set the device Busy and Done flags.
	DOC (f)	AC,DEV	2		X	[DEVC] ← [AC] [DEVBD] ← f Write the Accumulator into device's C buffer. Set the Busy and Done flags.
	SKP (t)	DEV	2		X	If T is true for DEV, [PC] ← [PC] + 1 Skip if I/O test true.
	IORST				X	[PM] ← 0 [ION] ← 10_2 The Busy and Done flags in all I/O devices are set to 0. The Priority Mask is set to 0 and interrupts are turned on.

Table 17-2. MicroNova And 9440 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	9440	OPERATION PERFORMED
				C		
PRIMARY MEMORY REFERENCE	LDA	AC,(@) DISP (,IX)	2		X	[AC] ← [EA] Load contents of memory to Accumulator.
	STA	AC,(@) DISP (,IX)	2		X	[EA] ← [AC] Store contents of Accumulator into memory.
REGISTER-REGISTER OPERATE	ADD (CS #)	S,D (SKCND)	2	X	X	[D] ← [D] + [S] Add contents of Source register to contents of Destination register. Perform the specified options.
	SUB (CS #)	S,D (,SKCND)	2	X	X	[D] ← [D] - [S] Subtract contents of Source register from contents of Destination register. Perform the specified options.
	NEG (CS #)	S,D (,SKCND)	2	X	X	[D] ← [$\overline{S}$] + 1 (twos complement) Place twos complement of the Source register contents in the Destination register. Perform the specified options.
	ADC (CS #)	S,D (,SKCND)	2	X	X	[D] ← [D] + [$\overline{S}$] Add the ones complement of the Source register contents to contents of Destination register. Perform the specified option.
	MOV (CS #)	S,D (,SKCND)	2	X	X	[D] ← [S] Move contents of Source register to Destination register. Perform the specified options.
	INC (CS #)	S,D (,SKCND)	2	X	X	[D] ← [S] + 1 Place incremented Source register contents into Destination register. Perform specified options.
	COM (CS #)	S,D (,SKCND)	2	X	X	[D] ← [$\overline{S}$] Complement the Source register contents, then move to Destination register. Perform specified options.
	AND (CS #)	S,D (,SKCND)	2	X	X	[D] ← [D] ∧ [S] AND the Source register contents with the Destination register contents. Perform specified options.

Table 17-2. MicroNova And 9440 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERA'D(S)	BYTES	STATUS	9440	OPERATION PERFORMED
				C		
REGISTER-REGISTER OPERATE (CONTINUED)	MUL		2			[AC0] ←(([AC1] * [AC2]) + [AC0]) <31,16> [AC1] ←(([AC1] * [AC2]) + [AC0]) <15,0> Multiply contents of AC1 by contents of AC2 and add contents of AC0 to result.
	DIV		2	X		[AC1] ←([AC0], [AC1])/ [AC2] (quotient) [AC0] ←([AC0], [AC1])/ [AC2] (remainder) Divide the 32-bit quantity contained in AC0 (high order) and AC1 (low order) by the contents of AC2.
STACK	PSHA	AC	2			[SP] ← [SP] + 1; [[SP]] ← [AC] Push the Accumulator onto the Stack.
	POPA	AC	2			[AC] ← [[SP]; [SP] ← [SP] - 1 Pop the top of the Stack to the Accumulator.
	SAV		2			[[SP] + 1] ← [AC0] [[SP] + 2] ← [AC1] [[SP] + 3] ← [AC2] [[SP] + 4] ← [AC3] [[SP] + 5] <14,0> ← [PC] [[SP] + 5] <15> ← [C] [SP] ← [SP] + 5 [FP] ← [SP] Save a return block in the Stack.
	MTSP	AC	2			[SP] ← [AC] <14,0> Move the low 15 bits of the Accumulator to the Stack Pointer.
	MTFP	AC	2			[FP] ← [AC] <14,0> Move the low 15 bits of the Accumulator to the Frame Pointer.

Table 17-2. MicroNova And 9440 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	9440	OPERATION PERFORMED
				C		
STACK (CONTINUED)	MFSP	AC	2			[AC] <14,0> ← [SP] [AC] <15> ←0 Move the Stack Pointer to low 15 bits of Accumulator.
	MFFP	AC	2			[AC] <14,0> ← [FP] [AC] <15> ←0 Move the Frame Pointer to the Accumulator.
JUMP	JMP	(″) DISP (,IX)	2		X	[PC] ← [EA] Branch unconditional.
	JMP	(″) DISP (,IX)	2		X	[AC3] ← [PC] + 1 [PC] ← [EA] Branch to subroutine.
	RET		2	X		[SP] ← [FP] [C] ← [[SP]] <15> [PC] ← [[SP]] <14,0> [AC3] ← [[SP] - 1] [AC2] ← [[SP] - 2] [AC1] ← [[SP] - 3] [AC2] ← [[SP] - 4] [SP] ← [SP] - 5 Return from subroutine and pop a return block off the Stack.
REAL TIME CLOCK	RTCEN (f)		2		X	[ION] ←f Enable Real Time Clock then set ION via I/O command.
	RTCDS (f)		2		X	[ION] ←f Disable Real Time Clock then set ION via I/O command.

Table 17-2. MicroNova And 9440 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS	9440	OPERATION PERFORMED
				C		
MEMORY OPERATE AND SKIP-ON-CONDITION	ISZ	(@) DISP (,IX)	2		X	[EA] ← [EA] + 1 If [EA] = 0 then [PC] ← [PC] + 1 Increment memory contents and skip if zero.
	DSZ	(@) DISP (,IX)	2		X	[EA] ← [EA] - 1 If [EA] = 0 then [PC] ← [PC] + 1 Decrement memory contents and skip if zero.
INTERRUPT	INTEN		2		X	[ION] ← 1 Enable interrupts.
	INTDS		2		X	[ION] ← 0 Disable interrupts.
	INTA (f)	AC	2		X	[AC] <5,0> ← DEV [ION] ← f The 6-bit device code of the device closest to the CPU that is requesting an interrupt is loaded into the low six bits of the Accumulator. Set ION via I/O command.
	MSKO (f)	AC			X	[PM] ← [AC] [ION] ← f Move contents of Accumulator to Priority Mask. Set ION via I/O command.
	TRAP		2			[26_{16}] ← [PC] [PC] ← [27_{16}] Performs a software interrupt.
	SKP (t)	CPU	2		X	If t is true, [PC] ← [PC] + 1 If interrupt or power fail condition satisfied, skip next instruction.
	HALT (f)		2		X	[ION] ← f Set ION via I/O command, then halt.

Table 17-3. MicroNova And 9440 Instruction Set Object Codes

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS	9440
ADC(CS #)	S,D (,SKCND)	1ssdd100rrccnwww	2	5/7	X
ADD(CS #)	S,D (,SKCND)	1ssdd110rrccnwww	2	5/7	X
AND(CS #)	S,D (,SKCND)	1ssdd111rrccnwww	2	5/7	X
COM(CS #)	S,D (,SKCND)	1ssdd000rrccnwww	2	5/7	X
DIAf	AC,DEV	011aa001ffpppppp	2	15	X
DIBf	AC,DEV	011aa011ffpppppp	2	15	X
DICf	AC,DEV	011aa101ffpppppp	2	15	X
DIV		7641	2	123	
DOAf	AC,DEV	011aa010ffpppppp	2	10	X
DOBf	AC,DEV	011aa100ffpppppp	2	10	X
DOCf	AC,DEV	011aa110ffpppppp	2	10	X
DSZ	(@) DISP (,IX)	00011ixxbbbbbbbb	2	8/10*	X
HALTf		011aa110ff111111	2	10	X
INC(CS #)	S,D (,SKCND)	1ssdd011rrccnwww	2	5/7	X
INTAf	AC	011aa011ff111111	2	15	X
INTDS		60BF	2	10	X
INTEN		607F	2	10	X
IORST		011aa010ff111111	2	10	X
ISZ	(@) DISP (,IX)	00010ixxbbbbbbbb	2	8/10*	X
JMP	(@) DISP (,IX)	00000ixxbbbbbbbb	2	6/8*	X
JSR	(@) DISP (,IX)	00001ixxbbbbbbbb	2	7/9*	X
LDA	AC (@),DISP (,IX)	011aaixxbbbbbbbb	2	6/8*	X
MFFP	AC	011aa00010000001	2	8	
MFSP	AC	011aa01010000001	2	7	
MOV(CS #)	S,D (,SKCND)	1ssdd010rrccnwww	2	5/7	X
MSKOf	AC	011aa100ff111111	2	10	X
MTFP	AC	011aa00000000001	2	6	
MTSP	AC	011aa01000000001	2	6	
MUL		76C1	2	86	
NEG(CS #)	S,D (,SKCND)	1ssdd001rrccnwww	2	5/7	X
NIOf	DEV	01100000ffpppppp	2	10	X
POPA	AC	011aa01110000001	2	7	
PSHA	AC	011aa01100000001	2	7	
RET		6581	2	15	
RTCDSf		01101010ff111111	2	10	X
RTCENf		01110010ff111111	2	10	X
SAV		6501	2	16	
SKPt		01100111ttpppppp	2	15/17	X
SKPT	DEV	01100111tt111111	2	15/17	X
STA	CPU	010aaixxbbbbbbbb	2	6/8*	X
SUB(CS #)	AC,(@) DISP (,IX)	1ssdd101rrccnwww	2	5/7	X
TRAP	S,D (,SKCND)	1ssddqqqqqqq1000	2	9	

*Direct addressing. For indirect addressing, add two clock periods for each level of indirection. For auto-increment or auto-decrement locations, add three clock periods, plus two for each level of indirection.

The following symbols are used in Table 17-3:

aa	Two bits selecting an Accumulator
bbbbbbbb	8-bit signed two's complement address displacement
cc	Two bits selecting the carry option
dd	Two bits selecting the destination Accumulator
ff	Two bits selecting the I/O command
i	One bit selecting indirect addressing
n	One bit choosing the no load option

pppppp	Six-bit device number
rr	Two bits determining the shift option
ss	Two bits choosing the source Accumulator
tt	Two bits choosing the I/O test
www	Three bits selecting the skip-on-condition option
xx	Two bits selecting the index option

Execution times shown are for MicroNova. Where two execution times are shown (for example, 5/7), the second is the instruction time if the skip or branch is taken.

9440 — NOVA BUS INTERFACE

We will now examine logic which expands the 9440 pins and signals to the standard Nova I/O bus and to a typical microcomputer memory bus. Table 17-1 identifies the Nova I/O bus that is created.

We will also illustrate that part of I/O device interface logic which is common to any I/O device — that is, logic associated with Busy, Done and Interrupt flags.

Our discussion of logic needed to create a Nova memory bus is quite superficial, reflecting the fact that there is no standard Nova memory bus. We will therefore limit ourselves to demonstrating, in general, how typical Nova memory bus signals may be created from 9440 signals. But we will be specific in describing logic that expands the 9440 interface to a standard Nova I/O bus.

The 9440-Nova bus interface description is divided into three parts:

1) Expansion of the Information Bus into various Address and Data Busses required by the I/O and memory references.
2) Creation of memory interface control signals.
3) Creation of I/O interface control signals.

We will examine each of the three logic expansions in turn.

9440 INFORMATION BUS EXPANSION

These four busses must be created out of the bidirectional 16-bit Information Bus:

1) A bidirectional, 16-bit Memory Data Bus.
2) An output only, 15-bit Memory Address Bus.
3) A bidirectional, 16-bit I/O Data Bus.
4) An output only, 6-bit I/O Device Address Bus.

We must also latch I/O instruction object codes into a buffer out of which I/O instruction code bits can be read by I/O control signal logic.

The 9440 Information Bus is low true; this means a low signal level represents a binary 1, while a high signal level represents a binary 0. Standard Nova I/O Data and Address Busses are also low true; we therefore do not need to invert signals during multiplexing and demultiplexing.

There are many ways in which the 9440 Information Bus may be multiplexed to create the four required busses. One possibility is illustrated in Figure 17-18. This logic shows 74LS245 8-bit bidirectional tristate buffers generating the two bidirectional Data Busses, while 74LS364 8-bit, edge-triggered flip-flops create the Address Busses and the I/O instruction object code register.

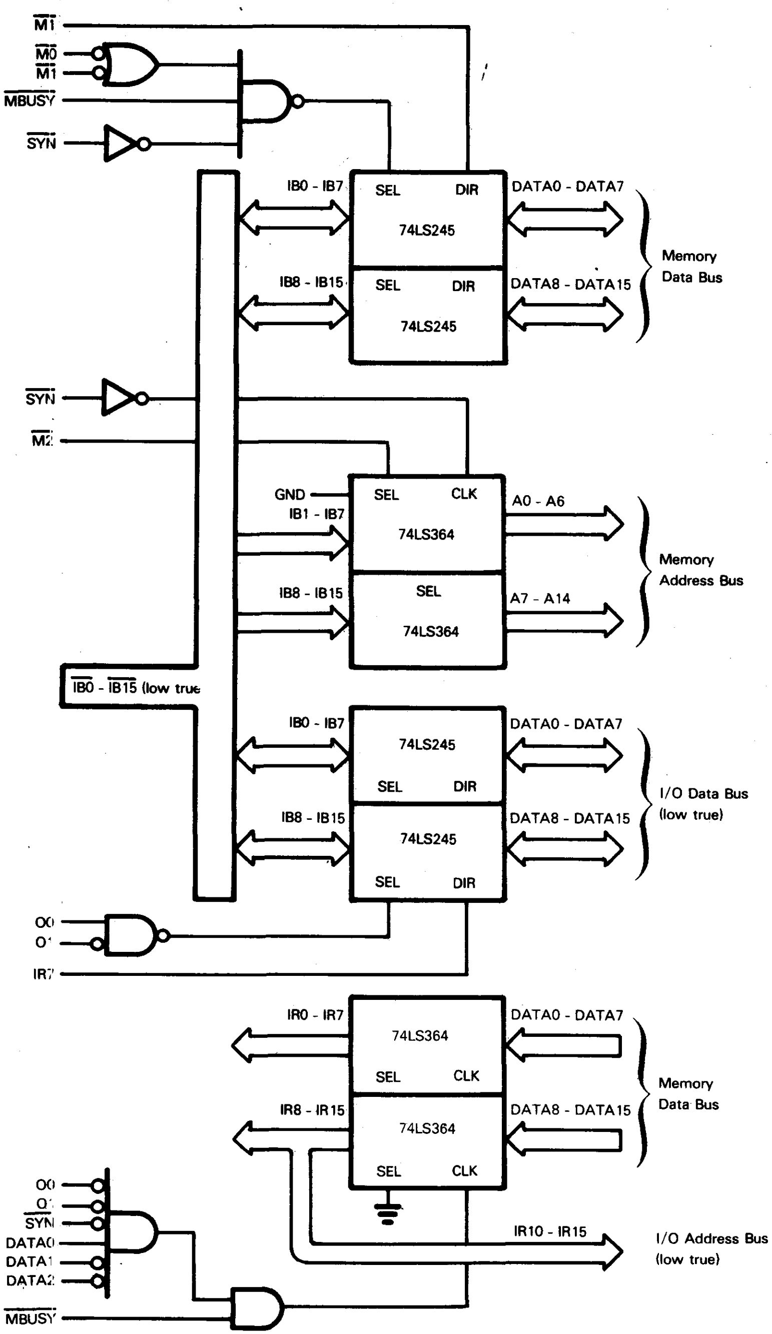


Figure 17-18. 9440 Information Bus Demultiplexing Logic

The Data Bus buffers each have a select input and a data direction input. The select inputs are low true. Logic shown in Figure 17-18 selects the 74LS245 buffers while valid memory data or valid I/O data can exist. Within these select periods a data direction control signal is created to ensure that data flows in the correct direction.

For the Memory Data Bus, the 74LS245 buffers must be selected either during a read or a write operation, as identified by $\overline{\text{M0}}$ or $\overline{\text{M1}}$. But these two signals span addresses and data occurring on the Information Bus. The period when valid data exists on the Information Bus is identified by $\overline{\text{MBUSY}}$ high while $\overline{\text{SYN}}$ is low. This timing is illustrated in Figure 17-12 and 17-13.

$\overline{\text{M1}}$ is used as the Memory Data Bus data direction control.

The I/O Data Bus buffer logic is somewhat simpler. The Information Bus is dedicated to transferring I/O data for the entire duration of a data input or data output machine cycle, as defined by O0 high and O1 low; these two signals are therefore used to create select logic. The direction of the I/O data transfer is taken from IR7; this bit of the I/O instruction object code defines the direction of an I/O data transfer, as illustrated in Figure 17-8.

For the Address Busses we do not use buffers, rather we use edge-triggered flip-flops. This allows the addresses being output to be held stable on the Memory Address Bus, or the I/O Address Bus after it is no longer on the Information Bus.

In the case of the Memory Address Bus, select logic is tied to $\overline{\text{M2}}$, which will be low whenever a memory address is being output on the Information Bus. The high-to-low transition of $\overline{\text{SYN}}$ is intended to act as a memory address strobe, therefore it is inverted to clock the Memory Address Bus flip-flops. Observe that there are only fifteen lines on the Memory Address Bus; the high order bit of a 16-bit memory address is reserved to indicate an indirect address.

Two 74LS364 flip-flops are used to latch I/O instruction object codes off the Memory Data Bus, to create the I/O Address Bus. These flip-flops constitute the Instruction register. The six low order output lines create the I/O Address Bus. The Instruction register flip-flops are constantly selected, which means that the I/O Address Bus will always hold the address of the most recently selected I/O device. The I/O device address, you will recall, is provided by the low order six bits of I/O instruction object codes. The Instruction register flip-flops are clocked by a signal which makes a low-to-high transition whenever an I/O instruction is on the Memory Data Bus. This condition is guaranteed by O0 and O1 both low, identifying an instruction fetch, while the DATA0, DATA1 and DATA2 are 01 and 1, respectively, identifying an I/O instruction object code. The instruction object code is actually clocked off the Memory Data Bus by the low-to-high transition of $\overline{\text{MBUSY}}$

Let us now examine I/O bus control signal logic.

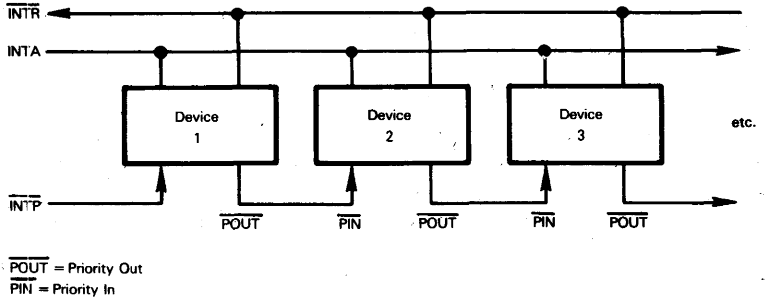
9440-NOVA I/O BUS INTERRUPT SIGNALS

Three signals on the standard Nova I/O bus are used by interrupt logic: $\overline{\text{INTR}}$,INTA and $\overline{\text{INTP}}$

$\overline{\text{INTR}}$ is the standard interrupt request signal. This signal can be tied directly to the 9440 $\overline{\text{INTREQ}}$ input.

The interrupt acknowledge signal INTA is created in response to execution of the interrupt acknowledge instruction. We have already described logic which creates INTA along with other I/O bus control signals.

$\overline{\text{INTP}}$ is the initial input to the highest priority device in an interrupt daisy chain. This may be illustrated as follows:



$\overline{\text{INTP}}$ may be connected to the complement of the 9440 output INT ON, in which case priorities within a daisy chain will not be resolved while interrupts are disabled. Frequently the initial $\overline{\text{PIN}}$ input to a daisy chain will be tied to ground and $\overline{\text{INTP}}$ will not be used. Now interrupt priorities will be arbitrated whether or not interrupts have been enabled.

As you will see it takes very little logic to expand the 9440 interrupt signals to standard Nova /O bus interrupt lines. But a considerable amount of interrupt-related logic must be present at external device controllers — logic which we will describe later in this chapter.

9440-NOVA DMA CONTROL SIGNALS

The only DMA logic provided by the 9440 consists of a DMA request signal $\overline{\text{DCH REQ}}$. When input low, this signal causes the 9440 to complete the instruction currently being executed, then to disable interrupts and wait. The DMA request is acknowledged by outputting O0 low and O1 high.

All logic which actually implements any DMA transfer must be implemented external to the 9440. A discussion of this logic is deferred to the next revision of Volume II.

9440-NOVA I/O BUS CONTROL SIGNALS CREATION

Standard Nova I/O bus control signals are created by the 9LS139 decoders illustrated in Figure 17-19. This signal logic directly interprets I/O instruction object code bits illustrated in Figures 17-8, 17-9 and 17-10.

Instruction object code bits are continuously read out of the Instruction register (IR0 - IR15); but I/O control signals are created only while I/OXEQ is high.

I/OXEQ is a master enable for all I/O control signals; it is the complement of $\overline{\text{SYN}}$, qualified by O0=1, O1=0, IR0=0, IR1=1 and IR2=1. This qualification guarantees that I/OXEQ will be active only during an I/O Data In or an I/O Data Out machine cycle.

Notice that we must create I/OXEQ using the I/O instruction object code bits latched in the Instruction register. Instruction register logic is illustrated in Figure 17-18. If Information Bus lines were used to create I/OXEQ, then I/OXEQ could be active only while the instruction object code is stable on the Information Bus. This would not be very helpful since it is during the next machine cycle that the I/O instruction is actually executed and I/O control signals must be created. The Instruction register illustrated in

Figure 17-18 latches I/O instruction object codes off the Information Bus during the instruction fetch machine cycle, then maintains the I/O instruction object code until the next I/O instruction is executed. That is why IR0 through IR15 are shown as inputs in Figure 17-19.

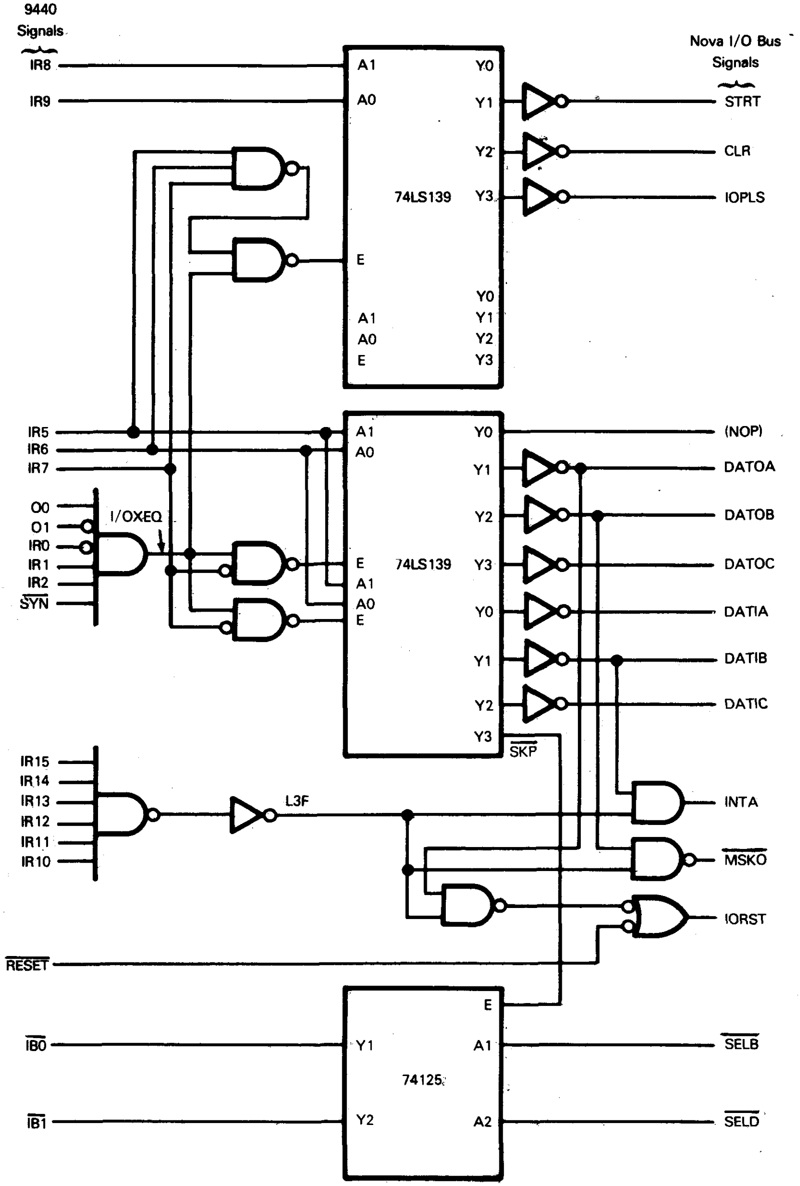


Figure 17-19. Creation Of Nova I/O Bus Control Signals From 9440 Signals

The logic of Figure 17-19 may be divided into these four sections:

1) Creation of control signals STRT, CLR and I/OPLS.
2) Creation of simple data transfer control signals.
3) Creation of I/O skip logic.
4) Creation of interrupt control signals.

Let us first consider logic needed to create STRT, CLR and I/OPLS.

These control signals are created in a very elementary way by the 74LS139 decoder at the top of Figure 17-19. This device decodes I/O instruction object code bits IR8 and IR9, providing an I/O Skip instruction is not being executed. The I/O Skip instruction is identified by IR5, IR6 and IR7 all high; therefore this combination is used as an inhibit input to the decoder enable.

Simple data transfer control signals consist of DATIA, DATIB, DATIC, DATOA, DATOB and DATOC; they are created by the lower 74LS139 decoder in Figure 17-19.

If you look at Figure 17-8 you will see that IR5 and IR6 select one of the three registers that may exist at an I/O device, while IR7 differentiates between I/O data input and I/O data output. IR7, and its complement, are therefore used in conjunction with I/OXEQ to enable the two halves of the 74LS139 decoder.

The Skip control $\overline{\mathrm{SKP}}$ is used to enable $\overline{\mathrm{SELB}}$ and $\overline{\mathrm{SELD}}$ onto Information Bus lines $\overline{\mathrm{IB0}}$ and $\overline{\mathrm{IB1}}$. This is done using a 74125 three-state buffer; this device is enabled by $\overline{\mathrm{SKP}}$ low. $\overline{\mathrm{SELB}}$ and $\overline{\mathrm{SELD}}$ are inputs to this buffer, while the outputs are connected to Information Bus lines $\overline{\mathrm{IB0}}$ and $\overline{\mathrm{IB1}}$. We assume that as soon as any I/O device is selected, it immediately connects its Busy and Done statuses to the $\overline{\mathrm{SELB}}$ and $\overline{\mathrm{SELD}}$ control lines of the I/O bus. However $\overline{\mathrm{SELB}}$ and $\overline{\mathrm{SELD}}$ will not appear on Information Bus lines $\overline{\mathrm{IB0}}$ and $\overline{\mathrm{IB1}}$ unless a Skip I/O instruction has been executed.

When an I/O instruction is executed specifying device $3F_{16}$, a set of interrupt-related I/O instructions are executed, as illustrated in Figure 17-10. Most of the instructions illustrated in this figure specify events internal to the CPU. For example, "enable interrupts" and "disable interrupts" apply to CPU interrupt logic; moreover, the Skip instructions interrogate interrupt request status and power fail status within the CPU. **"Acknowledge Interrupt" (INTA), "Output Interrupt Mask" ($\overline{\mathrm{MSKO}}$) and "Clear All I/O Devices" (IORST) are the only instructions which require control signals to be generated on the I/O bus. These control signals are generated by qualifying equivalent control signals from Figure 17-8 with a device $3F_{16}$ select code.** The device $3F_{16}$ select code, L3F, is a high true signal created by ANDing the low order six Instruction register bits (IR10 through IR15). Thus the gates producing INTA,$\overline{\mathrm{MSKO}}$ and IORST are effectively switched on and off by L3F. Note that IORST is generated either by execution of a "Clear I/O Devices" instruction, or by the master system $\overline{\mathrm{RESET}}$ signal.

NOVA I/O DEVICE CONTROLLER LOGIC

Interface logic which an external device must have, to connect to the standard Nova I/O bus, depends on the nature of the external device. A minicomputer device controller may be very complex, even costing more than the minicomputer itself; that is because minicomputer devices that connect to the I/O bus are peripherals, such as printers, disks, etc. When we reduce the Nova to microprocessor terms, however, external devices connected to the I/O bus reduce to such primitive elements as parallel I/O ports or serial data lines. Within this reduced context we can synthesize the minimum necessary elements of an I/O interface as consisting of three status flags: a Busy, a Done and an Interrupt request. We can implement these three status flags using three 7474 flip-flops, as

illustrated in Figure 17-20. Device select logic in this figure is limited to showing a select signal which will be generated true when the appropriate device code appears on the I/O device Address Bus. We have discussed I/O device select logic at various points earlier in this chapter.

Let us look at the BUSY and the DONE status logic. These are the operations which may affect the condition of the BUSY and DONE statuses for input:

1) At the start of an input operation BUSY must be set while DONE is clear. This condition is identified by 01 in bits IR8 and IR9 of the I/O instruction object code, which generates the STRT control signal of the I/O bus.
2) At the completion of an operation BUSY is cleared and DONE is set. This change in status setting must be implemented automatically by I/O device interface logic; it alone knows when the I/O operation has been completed.
3) BUSY and DONE may be cleared by the CPU. This is specified by 10 in bits IR8 and IR9 of the I/O instruction, which generates the CLR control signal on the I/O bus.
4) There is a "Clear All I/O Devices" instruction. This instruction generates IORST on the I/O bus; it clears BUSY and DONE statuses at all I/O devices.
5) A Master Reset must also clear the BUSY and DONE statuses. This Master Reset signal can also create IORST, as illustrated in Figure 17-19.

Two D-type flip-flops implement the BUSY and DONE status logic. These two D-type flip-flops are clocked by an "I/O Complete" signal which local device logic must generate. The BUSY and DONE statuses are generated by the flip-flop Q outputs which must connect to $\overline{\text{SELB}}$ and $\overline{\text{SELD}}$, as required by I/O skip logic, which we have already described.

The BUSY flip-flop uses its Set and Clear logic to control the BUSY status. The BUSY status is set by STRT • SELECT This combination of STRT and SELECT sets the device BUSY status high while it resets the DONE status low.

IORST
CLR
STRT

The BUSY status clear input is the NOR of IORST and CLR • SELECT. Both of these inputs will be false (that is, low) and the flip-flop clear input will therefore be high when BUSY is set high by STRT • SELECT Subsequently, when STRT • SELECT goes false, BUSY will stay high until it is reset low by "I/O Complete" or by a low clear input, which will occur when either IORST or CLR • SELECT goes high.

The DONE status is set high by the "I/O Complete" pulse after BUSY is set high since $\overline{\text{BUSY}}$ contributes to the NAND gate which provides the D input to the DONE flip-flop. The other input to this NAND gate is $\overline{\text{DONE}}$; this second input is there to preserve the "on" condition of the flip-flop until it is cleared by a low clear input. The low clear input is generated by a NOR gate which receives three inputs as follows:

1) The complement of the BUSY flip-flop set input; this ensures that the DONE status will be reset at the instant the BUSY status is being set.
2) The master Reset, IORST
3) CLR • SELECT which clears the DONE status.

The device interrupt may be individually disabled by a Mask Out instruction's execution which creates the $\overline{\text{MSKO}}$ control signal used to clock the interrupt status flip-flops. Accompanying execution of the Mask Out instruction, a 16-bit data value is output on the I/O Data Bus. Device interrupt logic is tied to bit N of this mask, which translates into I/O Data Bus line DATAN. Therefore DATAN becomes the D input to the interrupt flip-flop. If bit N is high then the interrupt flip-flop goes high; this sets the Q output high and at the same time prevents $\overline{\text{INTR}}$ from going low.

The bottom flip-flop in Figure 17-20 implements interrupt logic for the I/O interface. Let us summarize the conditions that can affect I/O interface interrupt logic.

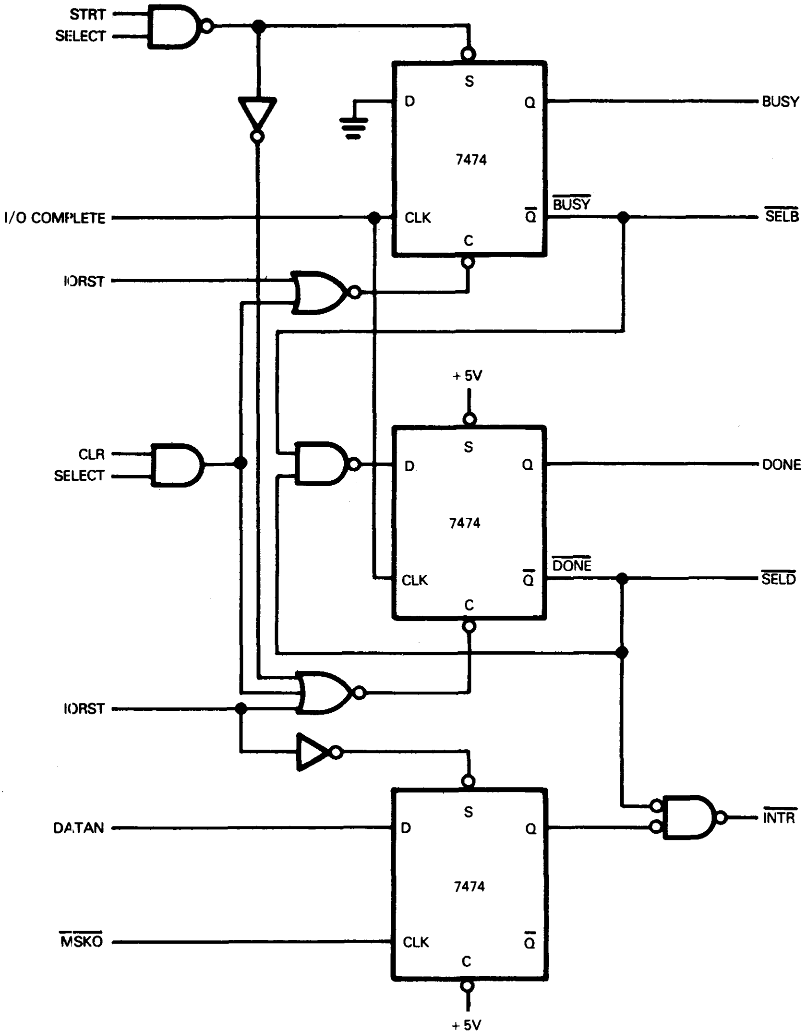


Figure 17-20. Busy, Done And Interrupt Status Logic Required By I/O Device Controllers On The Nova I/O Bus

Providing interrupts are enabled at the I/O interface, an interrupt will be requested whenever an I/O operation is completed, as identified by the DONE status going true. This is enabled by the OR gate preceding $\overline{\text{INTR}}$. As soon as the DONE status goes low, $\overline{\text{INTR}}$ will go low.

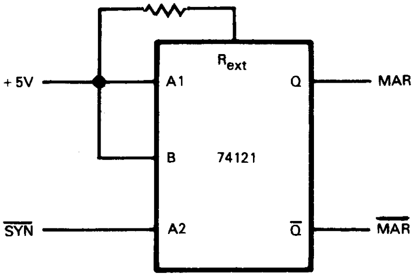
Interrupt logic may be enabled by a master I/O reset; therefore IORST is connected to the flip-flop set input.

9440 MEMORY BUS

There being no standard Nova Memory Bus, we will look at the signals available to you when you interface memory to the 9440.

First return to Figure 17-18. This figure shows how stable Data and Memory Busses may be demultiplexed off the 9440 Information Bus. In order to create a Memory Bus of any type, all you need is control signals to accompany the Memory Data Bus and the Memory Address Bus.

A valid memory address is always identified by a high-to-low transition of $\overline{\text{SYN}}$; therefore **a "Valid Memory Address" strobe may be created using $\overline{\text{SYN}}$ to trigger a one-shot.** Of the innumerable possibilities, here is one:



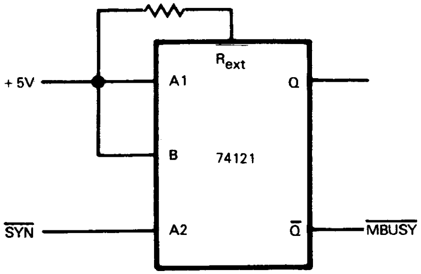
A "Memory Read" control signal is easily created by NORing $\overline{\text{M0}}$ with $\overline{\text{SYN}}$. This is easily deduced from Figure 17-12.

Figure 17-13 similarly illustrates how you can create a "Memory Write" control signal by NORing $\overline{\text{M1}}$ with $\overline{\text{SYN}}$.

A "Memory Reference" control signal may be created by $\overline{\text{M0}}$ or $\overline{\text{M1}}$.

An "Instruction Fetch" control signal is given by O0 and O1 both low.

$\overline{\text{MBUSY}}$ is a control input available to you when using slow memories. If your memory is fast enough to respond in the allowed time, then you can create $\overline{\text{MBUSY}}$ from $\overline{\text{SYN}}$ by using $\overline{\text{SYN}}$ to trigger a one-shot, as follows:



By modifying the duration of the one-shot you can extend the low $\overline{\text{MBUSY}}$ pulse, and therefore accommodate slow memories.

A complex memory interface can use $\overline{\text{MBUSY}}$ to lock out memory accesses while memory is busy — for example, while memory is responding to a direct memory access. The interaction of $\overline{\text{MBUSY}}$ and $\overline{\text{SYN}}$ is discussed in the text following Figure 17-12.

DATA SHEETS

The following section contains electrical data for the MicroNova CPU.

ABSOLUTE MAXIMUM RATINGS*

Supply Voltage Range V_{BB}	-2	to	-7	Volts
Supply Voltage Range V_{CC}	-0.3	to	+7	Volts
Supply Voltage Range V_{DD}	-0.3	to	+13	Volts
Supply Voltage Range V_{GG}	-0.3	to	+17	Volts
Input Voltage Range V_I	-0.3	to	+7	Volts
Input Current Range I_I	0	to	6	mAmps
Operating Temperature Range T_A	0	to	+70	°C
Storage Temperature Range T_{stg}	-55	to	+125	°C
Average Power Dissipation			1	Watt

NOTES *All voltages in this document are referenced to V_{SS} (ground).*

**Subjecting a circuit to conditions either outside these limits or at these limits for an extended period of time may cause irreparable damage to the circuit. As such, these ratings are not intended to be used during the operation of the circuit. Operating specifications are given in the DC (STATIC) CHARACTERISTICS TABLE.*

D. C. (STATIC) CHARACTERISTICS
mN601

OPERATING SPECIFICATIONS

T_A	range 0 to 70 °C	V_{GG}	= 14 ± 1.0 Volts	I_{CC}	= 20 mAmps Average	I_{BB}	= -.1 mAmps Average
V_{CC}	5 ± 0.25 Volts	V_{BB}	= -4.25 ± .25 Volts	I_{DD}	= 50 mAmps Average	I_{SS}	= -150 mAmps Average
V_{DD}	= 10 ± 1.0 Volts	V_{SS}	= 0 ± 0.0 Volts	I_{GG}	= 20 mAmps Average		

CHARACTERISTIC	SYMBOL	UNITS	PINS	LIMITS	
				MIN.	MAX.
INPUT LOW VOLTAGE	V_{IL}	Volts	α1,3 and α2,4	-2.0	+0.5
			MB 0-15, $\overline{\text{CLAMP}}$ EXTINT, DCH INT	-1.0	+1.0
			I O CLOCK, I O DATA 1, I O DATA 2	-1.0	+0.5
INPUT CURRENT FOR LOW STATE	I_{IL}	mAmps	α1,3 and α2,4	-	+.01
			MB 0-15	0	-2.0
			$\overline{\text{EXTINT}}$, $\overline{\text{DCH INT}}$, $\overline{\text{CLAMP}}$	-2.0	-4.0
			I O CLOCK, I O DATA 1, I O DATA 2	-2.0	-4.0
INPUT HIGH VOLTAGE	V_{IH}	Volts	α1,3 and α2,4	+13.0	+15.0
			MB 0-15, $\overline{\text{CLAMP}}$ $\overline{\text{EXTINT}}$, $\overline{\text{DCH INT}}$	+4.25	+5.8
			I O CLOCK, I O DATA 1, I O DATA 2	+2.5	+5.8
INPUT CURRENT FOR HIGH STATE	I_{IH}	mAmps	α1,3 and α2,4	-	-.01
			MB 0-15	-	-.06
			I O CLOCK, I O DATA 1, I O DATA 2	-	-1.0
			$\overline{\text{EXTINT}}$, $\overline{\text{DCH INT}}$	-	-.02
			$\overline{\text{CLAMP}}$	-	+.001
OUTPUT LOW VOLTAGE	V_{OL}	Volts	HALT	-	+3.0
			MB 0-15, I O INPUT, $\overline{\text{PAUSE}}$, SAEG, WEG, PG	-	+0.4
			I/O CLOCK, I/O DATA 1, I/O DATA 2		+0.5
OUTPUT CURRENT FOR LOW STATE	I_{OL}	mAmps	PG, I O INPUT	+4.0	
			MB 0-15, I O CLOCK I O DATA 1, I O DATA 2 $\overline{\text{PAUSE}}$, SAEG, PG, HALT	+2.0	-
OUTPUT HIGH VOLTAGE	V_{OH}	Volts	MB 0-15 I O CLOCK, I O DATA 1, I O DATA 2 I O INPUT, PAUSE, SAEG, WEG, PG	+4.25	-
			HALT	V_{CC}-0.5	
OUTPUT CURRENT FOR HIGH STATE	I_{OH}	mAmps	HALT	-	-.01
			MB 0-15	-	-.06
			I O INPUT, PG	-	-.02
			I O CLOCK, I O DATA 1, I O DATA 2, PAUSE SAEG, WEG	-	-.01
INPUT CAPACITANCE	C_I	pF	α1,3 and α2,4	-	100
			$\overline{\text{CLAMP}}$ MB 0-15, I O CLOCK I O DATA 1, I O DATA 2 $\overline{\text{EXTINT}}$, $\overline{\text{DCH INT}}$	-	10

NOTE

Logic "1" is defined as the more positive voltage as are the maximum figures given under voltage limits. Logic "0" is defined as the more negative voltage as are the minimum figures given under voltage limits.

Positive current, in the conventional sense, is defined as flowing into the pin.

On power-up, V_{BB} must be within its specified operating range (with respect to V_{SS}) before any of the other power supply voltages are applied to the circuit.

Chapter 18
2900 SERIES AND 6700 SERIES CHIP SLICE PRODUCTS

In the next two chapters of this book we are going to summarize chip slice logic products.

We begin with the 2900 and 6700 series 4-bit slice products which conform very closely to the general chip slice logic description given in Volume I, Chapter 4. The 6700 series product came first and the 2900 series represents a relatively small enhancement.

The MC10800 series chip slice products described in Chapter 19 represent a very powerful and significant next step in chip slice products; MC10800 series products are likely to account for a significant share of the chip slice market over the next few years.

3000 series chip slice logic, which was first introduced by Intel, is not described in this book. For the typical application, the 3000 series product line is obsolete. Providing a 2-bit slice rather than a 4-bit slice, the 3000 product line will require a significantly higher chip count than the 2900 or 6700 series, resulting in increased cost with no compensating performance.

Since the 2900 and 6700 series devices are very similar, this chapter is going to concentrate on the 2900 series — the more recent product. Differences between the 2900 series and 6700 series products, where they exist, will be identified.

This chapter is relatively superficial. We are going to describe the general capabilities of the various devices, relying upon Volume I, Chapter 4 to provide basic concepts. If you do not already have a basic understanding of chip slice products, then you should refer to Volume I, Chapter 4 before proceeding with this chapter.

All 2900 series and 6700 series devices use bipolar LSI technology.

The 2900 series microinstruction execution time is 100 nanoseconds; the 6700 series microinstruction execution time is 200 nanoseconds.

The primary source for the 2900 series chip slice logic is:

ADVANCED MICRO DEVICES
901 Thompson Place
Sunnyvale, CA 94086

There are two second sources for the 2900 series logic:

MOTOROLA SEMICONDUCTOR
Box 20912
Phoenix, AZ 85036

RAYTHEON SEMICONDUCTOR
350 Ellis Street
Mountain View, CA 94042

The primary source for the 6700 series chip slice logic is:

MONOLITHIC MEMORIES
1165 East Arques Avenue
Sunnyvale, CA 94086

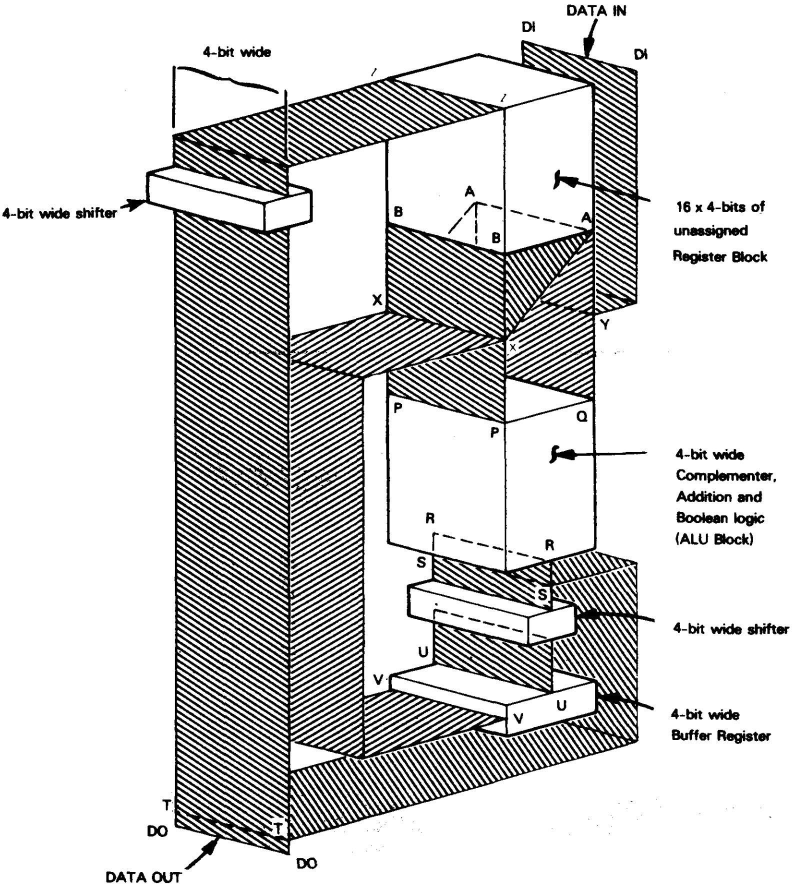
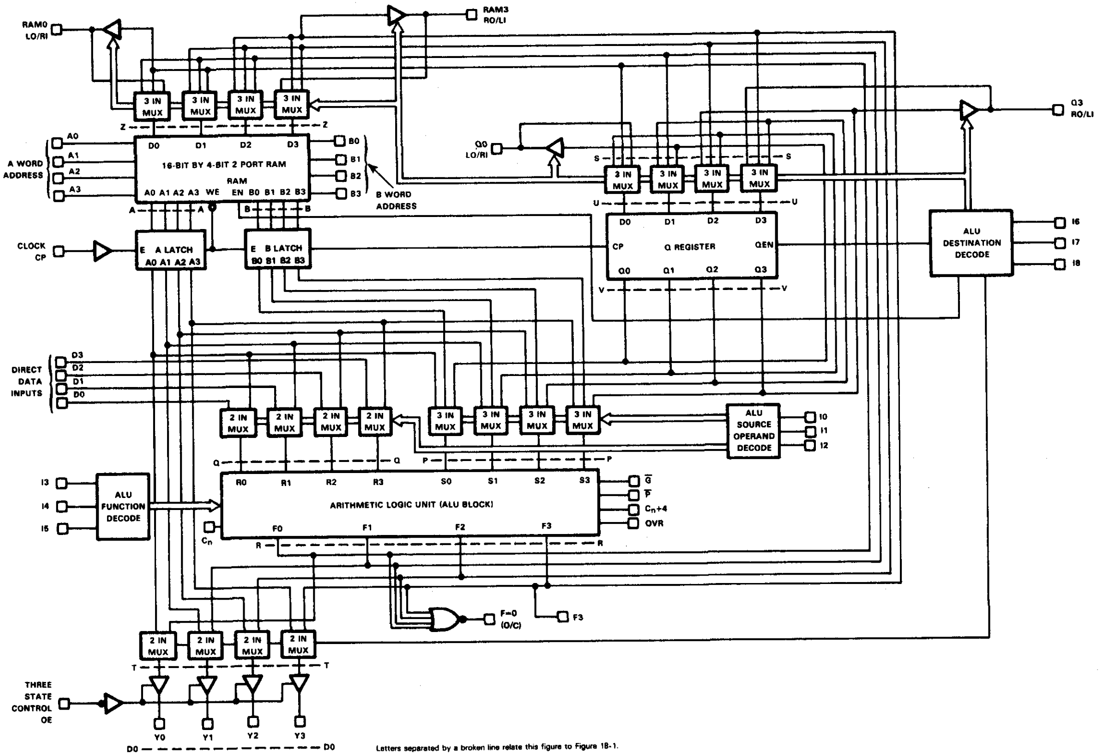

DATA IN
DI
DI
4-bit wide
4-bit wide shifter
A
B
B
A
16 x 4-bits of unassigned Register Block
X
Y
X
P
Q
P
4-bit wide Complementer, Addition and Boolean logic (ALU Block)
R
R
S
S
4-bit wide shifter
U
V
V
U
4-bit wide Buffer Register
T
DO
T
DO
DATA OUT

Figure 18-1. The 2901/6701 Arithmetic And Logic Unit

THE 2901/6701 ARITHMETIC AND LOGIC UNIT (ALU)

These devices constitute the center of any chip slice logic product.

Figure 18-1 illustrates the logic provided by a 2901 or 6701 ALU. Figure 18-1 is a reproduction of Figure 4-3 from Volume I, except that the AA and BB Register Block ports have been switched in order to become compatible with 2901 literature.



Letters separated by a broken line relate this figure to Figure 18-1.

Figure 18-2. 2901 ALU Logic

Table 18-1. 2901 ALU Function Control

MICRO CODE				ALU FUNCTION	SYMBOL
I5	I4	I3	OCTAL CODE		
L	L	L	0	R Plus S	R + S
L	L	H	1	S Minus R	S-R
L	H	L	2	R Minus S	R-S
L	H	H	3	R OR S	$R \vee S$
H	L	L	4	R AND S	$R \wedge S$
H	L	H	5	$\bar{R}$ AND S	$\bar{R} \wedge S$*
H	H	L	6	R EX-OR S	$R \veebar S$
H	H	H	7	R EX-NOR S	$\overline{R \veebar S}$*

*2901 ONLY

Table 18-2. ALU Source Operand Control

MICRO CODE				ALU SOURCE OPERANDS	
I2	I1	I0	OCTAL CODE	R	S
L	L	L	0	A	Q
L	L	H	1	A	B
L	H	L	2	O	Q
L	H	H	3	O	B
H	L	L	4	O	A
H	L	H	5	D	A/B*
H	H	L	6	D	Q
H	H	H	7	D	O

*A for 2901
B for 6701

The first thing to notice about the 2901/6701 ALU is the fact that it represents a 4-bit slice through the arithmetic and logic unit of a typical central processing unit. But being a discrete logic device, it must provide more than simple arithmetic and Boolean logic; it must provide some method of identifying data sources and destinations. Also, as we saw in Volume I, Chapter 4, an ALU chip slice is going to acquire some additional responsibilities toward its neighbors. Within this context, **let us examine the pins and signals of the 2901 and 6701, as illustrated in Figure 18-2.**

First of all, note that the ALU device receives two types of input:

1) Status and control signals via which it communicates with its neighbors.

2) An instruction code, plus data, via which it is sequenced by a Control Unit.

2901 ALU OPERATIONS SPECIFICATION

The focus of attention for the 2901/6701 ALU is the logic which actually performs arithmetic and logic operations — the ALU Block. This block of logic performs **eight operations** which **are specified by instruction signal inputs I3, I4 and I5, as defined in Table 18-1.** Observe that these eight functions consist of three arithmetic functions and four Boolean functions. Shift logic is missing. Shift logic is taken out of the ALU and placed within source and destination data paths.

2901 ALU SOURCE SPECIFICATION

There are two ALU Block sources, shown in Figure 18-2 as the R and S inputs; they are the P---P and Q---Q inputs of Figure 18-1. Each source is four bits wide, since we are dealing with a 4-bit chip slice.

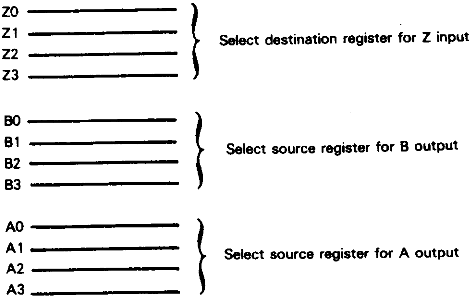
Inputs may consist of:

1) External data transmitted from the control unit to data pins D0 - D3.
2) Temporary data extracted from a small, 16 x 4-bit read/write memory within the 2901/6701.
3) The output of a shifter or temporary 4-bit register, identified in Figures 18-1 and 18-2 as the Q register. In reality, the Q register is a short circuit from ALU logic to ALU logic input.

The results of ALU operations may be output directly from the 2901/6701 via the Y0 - Y3 output pins (DO---DO in Figure 18-1); alternatively, the data may be routed to the Register Block, or the Q register.

The three instruction code bits, I0 - I2, define what the R and S inputs to the ALU will be. Table 18-2 defines how I0 - I2 will be interpreted.

Now take a look at the 16 x 4-bit read/write memory. We have seen that 2901/6701 logic allows the contents of any two 4-bit registers to be output; also, data is input to any one of the sixteen registers. External logic must define input and output registers using select pins. Ideally, three 4-bit select codes would be required:


Z0
Z1
Z2
Z3
Select destination register for Z input
B0
B1
B2
B3
Select source register for B output
A0
A1
A2
A3
Select source register for A output

But that is going to require 12 pins — and that is too many pins; therefore, the B output address code does double duty, also providing the Z input address; this eliminates the four Z pins, but reduces your options.

In summary, these 11 signals constitute a complete set of inputs:

Three microinstruction signals, I0, I1 and I2.

Two 4-bit register select codes, B0 - B3 and A0 - A3.

External logic must provide all 11 signal inputs simultaneously for every microinstruction's execution, simply to define the data entering the ALU Block.

We have not yet defined destination logic within the 2901/6701 ALU because the shifter and destination logic are combined. **The single ALU 4-bit result can go to one of three places:**

1) The output pins Y0 - Y3
2) The Q register
3) The 4-bit read/write memory register addressed by the B input

Of these three destinations, two -- the Q register and the read/write memory -- are optionally preceded by shifter logic.

You could specify a variety of destination options for the ALU Block results which are output via R---R. This data may be transmitted to one, two or all three of the identified destinations; and for two of the destinations data may optionally be shifted. It would require five pins simply to define the possible combinations of shifting and destination; but there are additional options. Observe that the contents of the 4-bit register addressed by the A address lines may be output directly to

DO---DO. This data path is an important one, since we must have some means of outputting shifting data without transmitting it through the ALU Block. To enable all destination combinations would require too many pins; not only would more pins be expensive in terms of device packaging, but each pin must be backed up by a microinstruction -- and if you increase the size of the microinstruction word, you will also increase the size of the control read only memory within which the microprogram must be stored. Therefore, a judicious subset of the possible destination combinations is selected via the three microinstruction input pins I6, I7 and I8. Table 18-3 defines the way in which these three microinstructions are decoded.

Let us then summarize the signals which must be input to a 2901/6701 simply to identify a single microinstruction.

The actual microinstruction object code must be input via the nine pins I0 - I8.

Two 4-bit register select codes must be input via A0 - A3 and B0 - B3. These address inputs must occur with the execution of every microinstruction.

A 4-bit direct data nibble may or may not be needed. If it is needed, it must be input along with the microinstruction object code via pins B0 - B3.

The principal output created following the execution of each microinstruction appears via the pins Y0 - Y3.

A number of timing and status signals remain to be described.

Timing is controlled by a single clock signal input via CP.

The ALU Block has two sets of status signals. One set allows normal CPU statuses to be created, the other set enables carry look ahead logic. The carry look ahead signals have been described in Volume I, Chapter 4.

These are the normal status signals provided:

ZERO STATUS IN CHIP SLICE LOGIC

1) A Zero status shown in Figure 18-2 as F0. This signal is the NOR of ALU Block outputs. For a number of 2901/6701 devices, you can create an overall Zero status by a wire-OR of the F0 outputs.

SIGN STATUS IN CHIP SLICE LOGIC

2) The high order bit of the ALU Block output appears as the F3 status in Figure 18-2. This status, when output by the high order 4-bit ALU slice, can be used to create a Sign status. **The 6701 does not provide this status.**

OVERFLOW AND CARRY STATUS IN CHIP SLICE LOGIC

$C_n + 4$ and OVR are outputs which, when taken from the high order slice, can be used to generate Carry and Overflow statuses, respectively.

Each of the two shifters has a shift-in and a shift-out pin so that shifts may be rippled from one ALU slice to the next.

THE 2909 MICROPROGRAM SEQUENCER

A group of 2901/6701 ALU slices must be driven by microprograms which will be stored in read only memory. The read only memory requires address logic. The responsibility of the address logic is to ensure that microinstructions are fetched in the correct sequence, so that in response to an instruction's object code, the ALU logic will perform necessary operations.

The 2909 microprogram sequencer provides you with the logic needed to create any address sequence for instructions stored in a microprogram ROM. Figure 18-3 illustrates the logic of the 2909 microprogram sequencer.

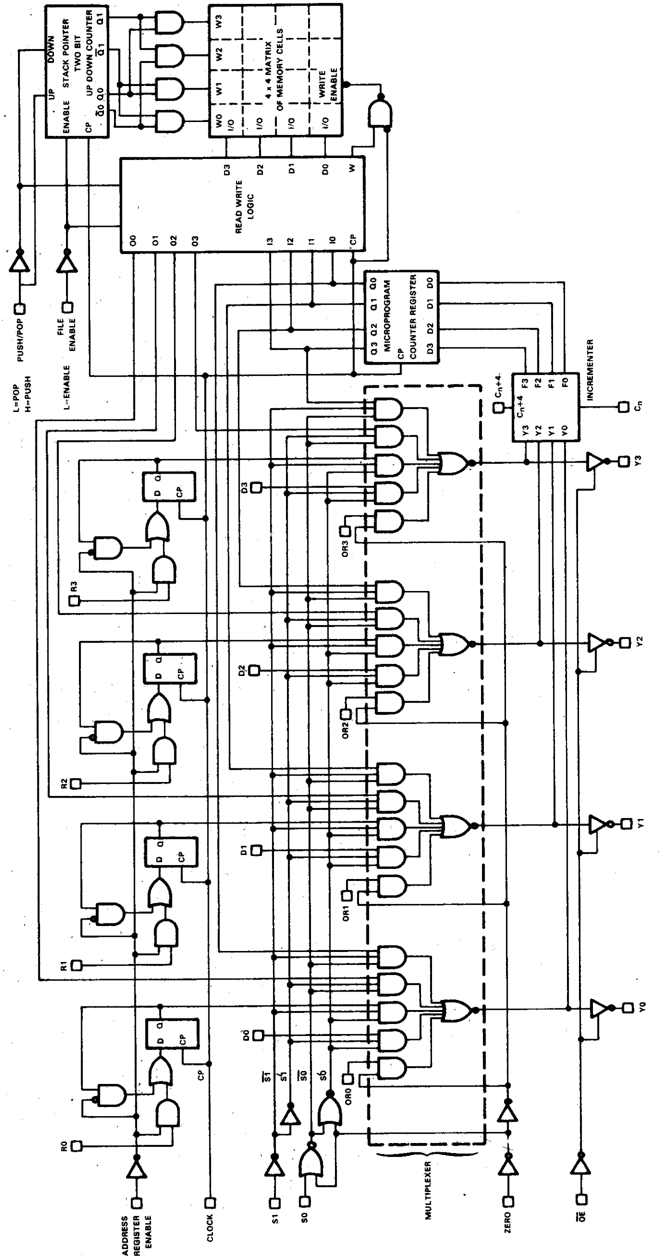


Figure 18-3. 2909 Microprogram Sequencer Block Diagram

Table 18-3. ALU Destination Control

MICRO CODE				RAM FUNCTION		Q-REG. FUNCTION		Y OUTPUT	RAM SHIFTER		Q SHIFTER		
I8	I7	I6	OCTAL CODE	SHIFT	LOAD	SHIFT	LOAD		RAM0 LO/RI	RAM3 LI/RO	Q0 LO/RI	Q3 LI/RO	
L	L	L	0	-	-	NONE	ALU (F_i)	F	X	X	X	X	
L	L	H	1	-	-	-	-	F	X	X	X	X	2901 ONLY
L	L	H	1	-	ALU (F_i)	-	-	B	X	X	X	X	6701 ONLY
L	H	L	2	NONE	ALU (F_i)	-	-	A	X	X	X	X	
L	H	H	3	NONE	ALU (F_i)	-	-	F	X	X	X	X	
H	L	L	4	LEFT (DOWN)	ALU ($F_i + 1$)	LEFT (DOWN)	Q-REG ($Q_i + 1$)	F	F0	IN3	Q0	IN3	
H	L	H	5	LEFT (DOWN)	ALU ($F_i + 1$)	-	-	F	F0	IN3	Q0	X	
H	H	L	6	RIGHT (UP)	ALU (F_{i-1})	RIGHT (UP)	Q-REG (Q_{i-1})	F	IN0	F3	IN0	Q3	
H	H	H	7	RIGHT (UP)	ALU (F_{i-1})	-	-	F	IN0	F3	X	Q3	

X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state.

Monolithic Memories provides the 67110 Control Unit which performs the same functions as the 2909, but in a substantially different way. The 67110 is not described in this chapter.

The most important thing to note about microprogram sequencer logic is that it bears a striking resemblance to the program memory addressing logic which will be provided on any microprocessor CPU. The principal difference is that microprogram sequencer logic is more elementary and therefore can execute faster — a necessary prerequisite if microprogram instruction executions are to concatenate in order to generate macroprogram instruction executions.

The next important point to note is that the 2909 microprogram sequencer logic, like the 2901 ALU, is a chip slice product. Each 2909 is a 4-bit chip slice. One 2909 device is capable of generating four address lines — addressing just sixteen microinstructions stored in ROM. By having two 2909 devices in parallel, you can create an 8-bit address which will access 256 microinstructions stored in program ROM. Each additional 2909 will increase the size of the address by four bits; and the number of microinstructions that can be accessed will increase accordingly.

Let us take a look at Figure 18-3. You should begin by looking at the multiplexer. This logic selects and outputs one of four possible address inputs. The two control signals, S0 and S1, determine which of the four addresses will be output.

The four lines of the address which is selected for output are ORed individually with external inputs OR0 - OR3, then the result of the OR is ANDed with a possible zero input.

The reason for having the individual OR0 - OR3 inputs is to allow branch logic to unilaterally modify an address which is being created. This is the point at which you would implement logic associated with a conditional branch.

The AND with zero allows you to unilaterally zero the output address — which you might want to do in response to a RESTART or other initialization.

These are the four possible address inputs:

1) A direct address input via the pins D0 - D3. This is an input which you would use initially to start the execution of a microinstruction sequence, after decoding a macroinstruction object code. You could also use these inputs subsequently to force a unilateral branch.

2) The incremented contents of the Microprogram Counter register. The Microprogram Counter register serves exactly the same function as the Program Counter register in a microcomputer. You would initially load a starting address into the Microprogram Counter register. Subsequently the Microprogram Counter register is going to be the normal location from which the multiplexer chooses its output address. After each address from the Microprogram Counter register is selected, the address will be incremented and returned, just as it would be in any microprocessor Program Counter. But there is a difference; since we are dealing with a chip slice product, the total Microprogram Counter register will consist of a number of 4-bit sections. There will accordingly be a carry-in pin and a carry-out pin, so that incrementing can ripple down from one 4-bit section to the next.

   There is an additional feature of Microprogram Counter register logic. As described in Volume I, Chapter 4, it is frequently necessary to re-execute the same microinstruction many times. For example, you may execute a no operation code a number of times in order to maintain synchronization between microinstructions and the macroinstruction system clock. You may also re-execute a Shift or Rotate microinstruction many times to perform multiple shifts or rotates. In order to save on microprogram ROM, you can inhibit the Microprogram Counter register increment logic by inputting a high value at the CI input to the low order four bits of the Microprogram Counter register. Clearly, this carry input must be zero in the normal course of events, since there is no lower shift that could possibly generate a legitimate carry input.

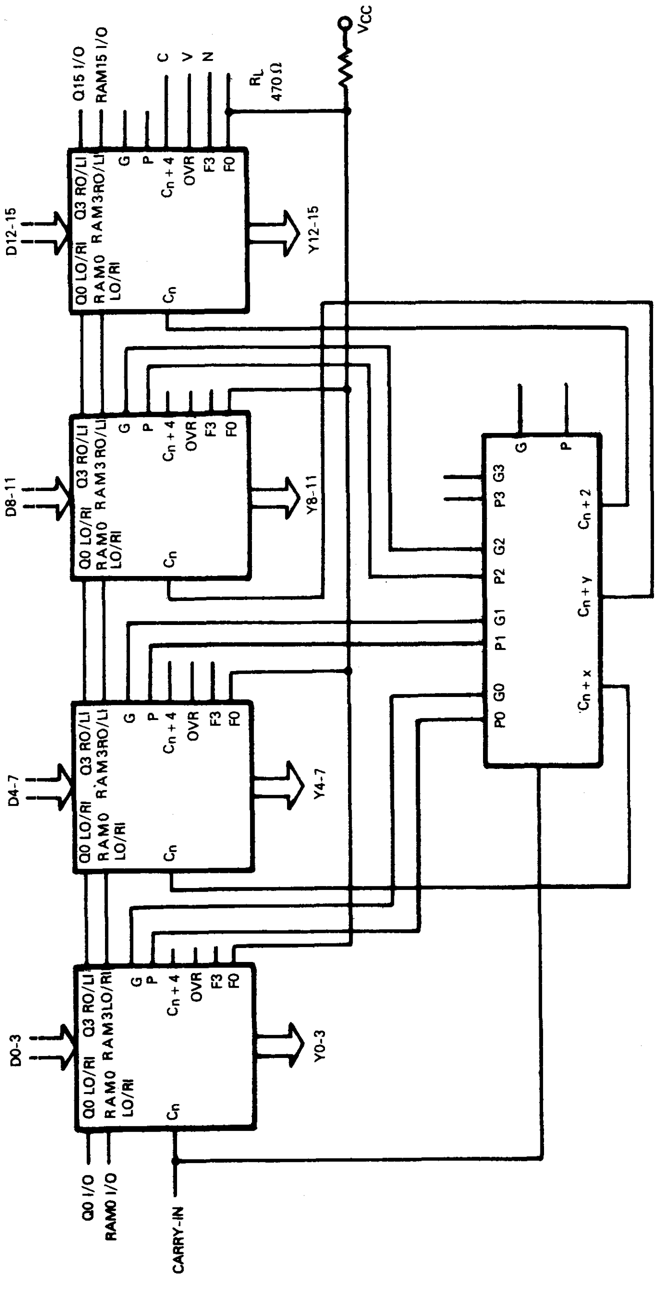


Figure 18-4. Four 2901's In A 16-Bit CPU Using The 2902 For Carry Lookahead

3) Just as assembly language programs can contain subroutines, so a microinstruction program can also contain subroutines. From our discussion of microprogramming in Chapter 4 of Volume I, you will recall that having subroutines in a microprogram is a very desirable feature. For example, large portions of an instruction fetch, a memory read and a memory write will be implemented via exactly the same microinstruction sequences. By including these microinstruction sequences in a microprogram subroutine, you can save significant amounts of microprogram memory. Microprogram subroutines are just as useful and memory-saving devices as assembly language subroutines. However, since microprograms are likely to be shorter than assembly language programs, the 2909 provides a four-level subroutine Stack. This means that you can nest microprogram subroutines to a depth of four. By inputting FILE ENABLE low, you can pop the top of the four-deep Stack into the multiplexer, or you can push the Microprogram Counter contents into the top of the Stack. Signal PUP, when high, forces the push; when low, PUP forces a pop.

4) The fourth possible input for the multiplexer address is the contents of the Address register. You can at any time input an address to the Address register via the R0 - R4 pins.

The $\overline{\text{OE}}$ control input allows you to disconnect the microprogram sequencer from the Address Bus. Thus, address outputs may be floated.

Observe that although the 2909 microprogram sequencer provides a good deal of the logic needed in order to create address sequences, a great deal of additional logic must still be provided in order to access microprogram sequencer logic appropriately.

THE 2902 CARRY LOOK AHEAD

This device serves just one function: when performing binary addition it creates parallel carry inputs for those 4-bit slices that are going to need a carry. Carry look ahead logic has been described in detail, in Volume I, Chapter 4. We will therefore provide a simple summary of this device in this chapter.

Suppose two 16-bit binary numbers are to be added. If each 16-bit number is implemented in four 4-bit slices, then how are you going to generate the carry for the second, third and fourth 4-bit slice? You could perform the binary addition in four steps — in which case at the conclusion of each step you would generate the necessary carry for the next step. This is an unsatisfactory method of performing binary addition when using chip slice logic because it is slow. The whole purpose of chip slice logic is to obtain maximum execution speed. The alternative is to create a device which will anticipate the carry that would be generated and provided so that all four segments of the 16-bit binary addition can be performed simultaneously. That is exactly what the 2902 device does.

Figure 18-4 illustrates the way in which a 2902 Carry Look Ahead device will connect to 2901 ALU slices.

As illustrated in this figure, the 2902 device can compute carry look ahead for up to three 4-bit slices — which means that it will support a 16-bit word; remember, the low order slice does not need any carry look ahead.

You can generate carry look ahead for larger words by using a number of 2902s together.

In order to generate carry look ahead, the 2902 receives, as inputs, the Carry Generate and Carry Propagate signals from the 2901/6701 ALU slices. For a discussion of this carry look ahead logic see Volume I, Chapter 4.

DATA SHEETS

The following pages contain specific electrical and timing data for the 2900 series devices described in this chapter.

MICRO CODE				ALU SOURCE OPERANDS	
I_2	I_1	I_0	Octal Code	R	S
L	L	L	0	A	Q
L	L	H	1	A	B
L	H	L	2	O	Q
L	H	H	3	O	B
H	L	L	4	O	A
H	L	H	5	D	A
H	H	L	6	D	Q
H	H	H	7	D	O

Figure 2. ALU Source Operand Control.

MICRO CODE				ALU Function	Symbol
I_5	I_4	I_3	Octal Code		
L	L	L	0	R Plus S	$R + S$
L	L	H	1	S Minus R	$S - R$
L	H	L	2	R Minus S	$R - S$
L	H	H	3	R OR S	$R \vee S$
H	L	L	4	R AND S	$R \wedge S$
H	L	H	5	$\overline{R}$ AND S	$\overline{R} \wedge S$
H	H	L	6	R EX-OR S	$R \veebar S$
H	H	H	7	R EX-NOR S	$\overline{R \veebar S}$

Figure 3. ALU Function Control.

MICRO CODE				RAM FUNCTION		Q-REG. FUNCTION		Y OUTPUT	RAM SHIFTER		Q SHIFTER	
I_8	I_7	I_6	Octal Code	Shift	Load	Shift	Load		RAM_0 LO/RI	RAM_3 LI/RO	Q_0 LO/RI	Q_3 LI/RO
L	L	L	0	–	–	NONE	ALU (F_i)	F	X	X	X	X
L	L	H	1	–	–	–	–	F	X	X	X	X
L	H	L	2	NONE	ALU (F_i)	–	–	A	X	X	X	X
L	H	H	3	NONE	ALU (F_i)	–	–	F	X	X	X	X
H	L	L	4	LEFT (DOWN)	ALU (F_{i+1})	LEFT (DOWN)	Q-REG (Q_{i+1})	F	F_0	IN_3	Q_0	IN_3
H	L	H	5	LEFT (DOWN)	ALU (F_{i+1})	–	–	F	F_0	IN_3	Q_0	X
H	H	L	6	RIGHT (UP)	ALU (F_{i-1})	RIGHT (UP)	Q-REG (Q_{i-1})	F	IN_0	F_3	IN_0	Q_3
H	H	H	7	RIGHT (UP)	ALU (F_{i-1})	–	–	F	IN_0	F_3	X	Q_3

X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state.

Figure 4. ALU Destination Control.

OCTAL I_{543}	I_{210} OCTAL	0	1	2	3	4	5	6	7
	ALU Source / ALU Function	A, Q	A, B	O, Q	O, B	O, A	D, A	D, Q	D, O
0	C_n = L R Plus S C_n = H	A+Q A+Q+1	A+B A+B+1	Q Q+1	B B+1	A A+1	D+A D+A+1	D+Q D+Q+1	D D+1
1	C_n = L S Minus R C_n = H	Q–A–1 Q–A	B–A–1 B–A	Q–1 Q	B–1 B	A–1 A	A–D–1 A–D	Q–D–1 Q–D	–D–1 –D
2	C_n = L R Minus S C_n = H	A–Q–1 A–Q	A–B–1 A–B	–Q–1 –Q	–B–1 –B	–A–1 –A	D–A–1 D–A	D–Q–1 D–Q	D–1 D
3	R OR S	$A \vee Q$	$A \vee B$	Q	B	A	$D \vee A$	$D \vee Q$	D
4	R AND S	$A \wedge Q$	$A \wedge B$	0	0	0	$D \wedge A$	$D \wedge Q$	0
5	$\overline{R}$ AND S	$\overline{A} \wedge Q$	$\overline{A} \wedge B$	Q	B	A	$\overline{D} \wedge A$	$\overline{D} \wedge Q$	0
6	R EX-OR S	$A \veebar Q$	$A \veebar B$	Q	B	A	$D \veebar A$	$D \veebar Q$	D
7	R EX-NOR S	$\overline{A \veebar Q}$	$\overline{A \veebar B}$	$\overline{Q}$	$\overline{B}$	$\overline{A}$	$\overline{D \veebar A}$	$\overline{D \veebar Q}$	$\overline{D}$

+ = Plus; – = Minus; $\vee$ = OR; $\wedge$ = AND; $\veebar$ = EX-OR

Figure 5. Source Operand and ALU Function Matrix.

MAXIMUM RATINGS (Above which the useful life may be impaired)

Storage Temperature	−65°C to +150°C
Temperature (Ambient) Under Bias	−55°C to +125°C
Supply Voltage to Ground Potential	−0.5 V to +6.3 V
DC Voltage Applied to Outputs for HIGH Output State	−0.5 V to +V_{CC} max.
DC Input Voltage	−0.5 V to +5.5 V
DC Output Current, Into Outputs	30 mA
DC Input Current	−30 mA to +5.0 mA

OPERATING RANGE

P/N	Ambient Temperature	V_{CC}
Am2901PC, DC	0°C to +70°C	4.75 V to 5.25 V
Am2901DM, FM	−55°C to +125°C	4.50 V to 5.50 V

STANDARD SCREENING

(Conforms to MIL-STD-883 for Class C Parts)

Step	MIL-STD-883 Method	Conditions	Level	
			Am2901PC, DC	Am2901DM, FM
Pre-Seal Visual Inspection	2010	B	100%	100%
Stabilization Bake	1008	C 24-hour 150°C	100%	100%
Temperature Cycle	1010	C −65°C to +150°C 10 cycles	100%	100%
Centrifuge	2001	B 10,000 G	100% *	100%
Fine Leak	1014	A 5 x 10 $^{-8}$ atm-cc/cm3	100% *	100%
Gross Leak	1014	C2 Fluorocarbon	100% *	100%
Electrical Test Subgroups 1 and 7	5004	See below for definitions of subgroups	100%	100%
Insert Additional Screening here for Class B Parts				

Group A Sample Tests
Subgroup 1
Subgroup 2
Subgroup 3
Subgroup 7
Subgroup 8
Subgroup 9
	5005	See below for definitions of subgroups	
			LTPD = 5
			LTPD = 7
			LTPD = 7
			LTPD = 7
			LTPD = 7
			LTPD = 7
				
				LTPD = 5
				LTPD = 7
				LTPD = 7
				LTPD = 7
				LTPD = 7
				LTPD = 7

*Not applicable for Am2901PC

ADDITIONAL SCREENING FOR CLASS B PARTS

Step	MIL-STD-883 Method	Conditions	Level
			Am2901DMB, FMB
Burn-In	1015	D 125°C 160 hours min.	100%
Electrical Test Subgroup 1 Subgroup 2 Subgroup 3 Subgroup 7 Subgroup 9	5004		 100% 100% 100% 100% 100%
Return to Group A Tests in Standard Screening			

ORDERING INFORMATION

Package Type	Temperature Range	Order Number
Molded DIP	0°C to +70°C	AM2901PC
Hermetic DIP	0°C to +70°C	AM2901DC
Hermetic DIP	−55°C to +125°C	AM2901DM
Hermetic Flat Pack	−55°C to +125°C	AM2901FM
Dice	0°C to +70°C	AM2901XC

GROUP A SUBGROUPS

(as defined in MIL-STD-883, method 5005)

Subgroup	Parameter	Temperature
1	DC	25°C
2	DC	Maximum rated temperature
3	DC	Minimum rated temperature
7	Function	25°C
8	Function	Maximum and minimum rated temperature
9	Switching	25°C
10	Switching	Maximum Rated Temeperature
11	Switching	Minimum Rated Temperature

ELECTRICAL CHARACTERISTICS OVER OPERATING RANGE (Unless Otherwise Noted)

(Group A, Subgroups 1, 2 and 3)

Parameters	Description	Test Conditions (Note 1)			Min.	Typ. (Note 2)	Max.	Units
V_{OH}	Output HIGH Voltage	V_{CC} = MIN. V_{IN} = V_{IH} or V_{IL}	I_{OH} = −1.6mA Y_0, Y_1, Y_2, Y_3		2.4			Volts
			I_{OH} = −1.0mA, C_{n+4}		2.4			
			I_{OH} = −800μA, OVR, $\overline{P}$		2.4			
			I_{OH} = −600μA, F_3		2.4			
			I_{OH} = −600μA All RO/LI, LO/RI		2.4			
			I_{OH} = −1.6mA, $\overline{G}$		2.4			
I_{CEX}	Output Leakage Current for F = 0 Output	V_{CC} = MIN., V_{OH} = 5.5V V_{IN} = V_{IH} or V_{IL}					250	μA
V_{OL}	Output LOW Voltage	V_{CC} = MIN., V_{IN} = V_{IH} or V_{IL}	I_{OL} = 16mA Y_0, Y_1, Y_2, Y_3				0.5	Volts
			I_{OL} = 10mA, C_{n+4}, F = 0				0.5	
			I_{OL} = 8.0mA, OVR, $\overline{P}$				0.5	
			I_{OL} = 6.0mA, F_3 All RO/LI, LO/RI				0.5	
V_{IH}	Input HIGH Level	Guaranteed input logical HIGH voltage for all inputs			2.0			Volts
V_{IL}	Input LOW Level	Guaranteed input logical LOW voltage for all inputs	Military				0.7	Volts
			Commercial				0.8	
V_I	Input Clamp Voltage	V_{CC} = MIN., I_{IN} = −18mA					−1.5	Volts
I_{IL} (Note 3)	Input LOW Current	V_{CC} = MAX. V_{IN} = 0.5V	Clock, $\overline{OE}$				−0.36	mA
			A_0, A_1, A_2, A_3				−0.36	
			B_0, B_1, B_2, B_3				−0.36	
			D_0, D_1, D_2, D_3				−0.72	
			I_0, I_1, I_2, I_6, I_8				−0.36	
			I_3, I_4, I_5, I_7				−0.72	
			All LO/RI, RO/LI (Note 4)				−0.8	
			C_n				−3.6	
I_{IH} (Note 3)	Input HIGH Current	V_{CC} = MAX. V_{IN} = 2.7V	Clock, $\overline{OE}$				20	μA
			A_0, A_1, A_2, A_3				20	
			B_0, B_1, B_2, B_3				20	
			D_0, D_1, D_2, D_3				40	
			I_0, I_1, I_2, I_6, I_8				20	
			I_3, I_4, I_5, I_7				40	
			All LO/RI, RO/LI (Note 4)				100	
			C_n				200	
I_I	Input HIGH Current	V_{CC} = MAX., V_{IN} = 5.5V					1.0	mA
I_{OZ}	Off State (High-Impedance) Output Current	V_{CC} = MAX.	Y_0, Y_1, Y_2, Y_3	V_O = 2.4V			50	μA
				V_O = 0.5V			−50	
			All LO/RI, RO/LI	V_O = 2.4V (Note 5)			100	
				V_O = 0.5V (Note 5)			−800	
I_{SC}	Output Short Circuit Current (Note 4)		$Y_0, Y_1, Y_2, Y_3, \overline{G}$		−6.0		−40	mA
			C_{n+4}		−6.0		−40	
			OVR, $\overline{P}$		−6.0		−40	
			F_3		−6.0		−40	
			All RO/LI, LO/RI		−6.0		−40	
I_{CC}	Power Supply Current	V_{CC} = MAX.	Military			185	280	mA
			Commercial			185	280	

Notes: 1. For conditions shown as MIN or MAX, use the appropriate value specified under Electrical Characteristics for the applicable device type.

2. Typical limits are at V_{CC} = 5.0V, 25°C ambient and maximum loading.

3. Not more than one output should be shorted at a time. Duration of the short circuit test should not exceed one second.

4. LO/RI and RO/LI are three-state outputs internally connected to TTL inputs. Input characteristics are measured with I_{678} in a state such that the three-state output is OFF.

GUARANTEED OPERATING CONDITIONS

Tables I, II, and III below define the timing requirements of the Am2901 in a system. The Am2901 is guaranteed to function correctly over the operating range when used within the delay and set-up time constraints of these tables for the appropriate device type. The tables are divided into three types of parameters; clock characteristics, combinational delays from inputs to outputs, and set-up and hold time requirements. The latter table defines the time prior to the end of the cycle (i.e., clock LOW-to-HIGH transition) that each input must be stable to guarantee that the correct data is written into one of the internal registers.

The performance of the Am2901 within the limits of these tables is guaranteed by the testing defined as "Group A, Subgroup 9" Electrical Testing. For a copy of the tests and limits used for subgroup 9, contact Advanced Micro Devices' Product Marketing.

TABLE I

CYCLE TIME AND CLOCK CHARACTERISTICS

TIME	Am2901DC	Am2901DM
Minimum Read-Modify-Write Cycle (time from selection of A, B registers to end of cycle)	105 ns	120 ns
Maximum Clock Frequency to Shift Q Register (50% duty cycle)	9.5 MHz	8.3 MHz
Minimum Clock LOW Time	30 ns	30 ns
Minimum Clock HIGH Time	30 ns	30 ns
Minimum Clock Period	105 ns	120 ns

TABLE II

MAXIMUM COMBINATIONAL PROPAGATION DELAYS (all in ns, $C_L \leq 15pF$)

	Am2901DC								Am2901DM							
To Output / From Input	Y	F_3	C_{n+4}	$\overline{G}, \overline{P}$	F=0 R_L= 470	OVR	RO, LO		Y	F_3	C_{n+4}	$\overline{G}, \overline{P}$	F=0 R_L= 470	OVR	RO, LO	
							RAM	Q							RAM	Q
Clock ↑	115	85	100	100	110	95	105	60	125	95	110	110	120	105	115	65
A, B	110	85	80	80	110	75	110	–	120	95	90	90	120	85	120	–
D	100	70	70	70	100	60	60	–	110	80	75	75	110	65	65	–
C_n	55	35	30	–	50	40	55	–	60	40	30	–	55	45	60	–
I_{012}	85	65	65	65	80	65	80	–	90	70	70	70	85	70	85	–
I_{345}	70	55	60	60	70	60	65	–	75	60	65	65	75	65	70	–
I_{678}	55	–	–	–	–	–	45	45	60	–	–	–	–	–	50	50
$\overline{OE}$ Enable/Disable	40/25	–	–	–	–	–	–	–	40/25	–	–	–	–	–	–	–
A bypassing ALU (I = 2xx)	60	–	–	–	–	–	–	–	65	–	–	–	–	–	–	–

SET-UP AND HOLD TIMES (minimum cycles from each input)

Set-up and hold times are defined relative to the clock LOW-to-HIGH edge. Inputs must be steady at all times from the set-up time prior to the clock until the hold time after the clock. The set-up times allow sufficient time to perform the correct operation on the correct data so that the correct ALU data can be written into one of the registers.

TABLE III

Set-Up and Hold Times (all in ns) (Note 1)

From Input	Notes	Am2901DC		Am2901DM	
		Set-Up Time	Hold Time	Set-Up Time	Hold Time
A, B Source	2, 3, 4	105 $t_{pw}L + 30$	0	120 $t_{pw}L + 30$	0
B Dest.	2, 4	$t_{pw}L + 15$	0	$t_{pw}L + 15$	0
D		100	0	110	0
C_n		55	0	60	0
I_{012}		85	0	90	0
I_{345}		70	0	75	0
I_{678}	4	$t_{pw}L + 15$	0	$t_{pw}L + 15$	0
RI, LI (RAM or Q)		30	0	30	0

Notes: 1. See Figure 11 and 12.
2. If the B address is used as a source operand, allow for the "A, B source" set-up time; if it is used only for the destination address, use the "B dest." set-up time.
3. Where two numbers are shown, both must be met.
4. "$t_{pw}L$" is the clock LOW time.

MAXIMUM RATINGS (Above which the useful life may be impaired)

Storage Temperature	–65°C to +150°C
Temperature (Ambient) Under Bias	–55°C to +125°C
Supply Voltage to Ground Potential	–0.5 V to +7.0 V
DC Voltage Applied to Outputs for HIGH Output State	–0.5 V to +V_{CC} max.
DC Input Voltage	–0.5 V to +7.0 V
DC Output Current, Into Outputs	30 mA
DC Input Current	–30 mA to +5.0 mA

OPERATING RANGE

P/N	Ambient Temperature	V_{CC}
Am2909PC, DC	0°C to +70°C	4.75 V to 5.25 V
Am2909DM,FM	–55°C to +125°C	4.50 V to 5.50 V

STANDARD SCREENING

(Conforms to MIL-STD-883 for Class C Parts)

Step	MIL-STD-883 Method	Conditions	Level	
			Am2909PC, DC	Am2909DM, FM
Pre-Seal Visual Inspection	2010	B	100%	100%
Stabilization Bake	1008	C 24-hour 150°C	100%	100%
Temperature Cycle	1010	C –65°C to +150°C 10 cycles	100%	100%
Centrifuge	2001	B 10,000 G	100% *	100%
Fine Leak	1014	A 5 x 10^{-8} atm-cc/cm3	100% *	100%
Gross Leak	1014	C2 Fluorocarbon	100% *	100%
Electrical Test Subgroups 1 and 7	5004	See below for definitions of subgroups	100%	100%
Insert Additional Screening here for Class B Parts				

Group A Sample Tests
Subgroup 1
Subgroup 2
Subgroup 3
Subgroup 7
Subgroup 8
Subgroup 9
	5005	See below for definitions of subgroups	
			LTPD = 5
			LTPD = 7
			LTPD = 7
			LTPD = 7
			LTPD = 7
			LTPD = 7
				
				LTPD = 5
				LTPD = 7
				LTPD = 7
				LTPD = 7
				LTPD = 7
				LTPD = 7

*Not applicable for Am2909PC

ADDITIONAL SCREENING FOR CLASS B PARTS

Step	MIL-STD-883 Method	Conditions	Level
			Am2909DMB, FMB
Burn-In	1015	D 125°C 160 hours min.	100%
Electrical Test Subgroup 1 Subgroup 2 Subgroup 3 Subgroup 7 Subgroup 9	5004		 100% 100% 100% 100% 100%
Return to Group A Tests in Standard Screening			

ORDERING INFORMATION

Package Type	Temperature Range	Order Number
Molded DIP	0°C to +70°C	AM2909PC
Hermetic DIP	0°C to +70°C	AM2909DC
Hermetic DIP	–55°C to +125°C	AM2909DM
Hermetic Flat Pack	–55°C to +125°C	AM2909FM
Dice	0°C to +70°C	AM2909XC

GROUP A SUBGROUPS

(as defined in MIL-STD-883, method 5005)

Subgroup	Parameter	Temperature
1	DC	25°C
2	DC	Maximum rated temperature
3	DC	Minimum rated temperature
7	Function	25°C
8	Function	Maximum and minimum rated temperature
9	Switching	25°C
10	Switching	Maximum Rated Temeperature
11	Switching	Minimum Rated Temperature

ELECTRICAL CHARACTERISTICS OVER OPERATING TEMPERATURE RANGE (Unless Otherwise Noted)

Am2909XC	T_A = 0°C to +70°C	V_{CC} = 5.0V ±5% (COM'L)	MIN. = 4.75V	MAX. = 5.25V
Am2909XM	T_A = −55°C to +125°C	V_{CC} = 5.0V ±10% (MIL)	MIN. = 4.50V	MAX. = 5.50V

Parameters	Description	Test Conditions (Note 1)			Min.	Typ. (Note 2)	Max.	Units
V_{OH}	Output HIGH Voltage	V_{CC} = MIN., V_{IN} = V_{IH} or V_{IL}	MIL	I_{OH} = −1.0mA	2.4			Volts
			COM'L	I_{OH} = −2.6mA	2.4			
V_{OL}	Output LOW Voltage	V_{CC} = MIN., V_{IN} = V_{IH} or V_{IL}	I_{OL} = 4.0mA				0.4	Volts
			I_{OL} = 8.0mA				0.45	
			I_{OL} = 12mA (Note 5)				0.5	
V_{IH}	Input HIGH Level	Guaranteed input logical HIGH voltage for all inputs			2.0			Volts
V_{IL}	Input LOW Level	Guaranteed input logical LOW voltage for all inputs		MIL			0.7	Volts
				COM'L			0.8	
V_I	Input Clamp Voltage	V_{CC} = MIN., I_{IN} = −18mA					−1.5	Volts
I_{IL}	Input LOW Current	V_{CC} = MAX., V_{IN} = 0.4V	C_n				−1.08	mA
			Push/Pop, OE				−0.72	
			Others				−0.36	
I_{IH}	Input HIGH Current	V_{CC} = MAX., V_{IN} = 2.7V	C_n				40	μA
			Push/Pop				40	
			Others				20	
I_I	Input HIGH Current	V_{CC} = MAX., V_{IN} = 7.0V	C_n, Push/Pop				0.2	mA
			Others				0.1	
I_{SC}	Output Short Circuit Current (Note 3)	V_{CC} = MAX.			−30		−85	mA
I_{CC}	Power Supply Current	V_{CC} = MAX. (Note 4)				80	130	mA
I_{OZ}	Output OFF Current	V_{CC} = MAX., $\overline{OE}$ = 2.7V	V_{OUT} = 0.4V				−20	μA
			V_{OUT} = 2.7V				20	

Notes: 1. For conditions shown as MIN. or MAX., use the appropriate value specified under Electrical Characteristics for the applicable device type.
2. Typical limits are at V_{CC} = 5.0V, 25°C ambient and maximum loading.
3. Not more than one output should be shorted at a time. Duration of the short circuit test should not exceed one second.
4. Apply GND to C_n, R_0, R_1, R_2, R_3, OR_0, OR_1, OR_2, OR_3, D_0, D_1, D_2, and D_3. Other inputs open. All outputs open. Measured after a LOW-to-HIGH clock transition.
5. The 12mA output applies only to Y_0, Y_1, Y_2 and Y_3.

TYPICAL Am2909 AC CHARACTERISTICS (All in ns)

TABLE I
CLOCK REQUIREMENTS

$t_{pw}L$	30
$t_{pw}H$	30

TABLE II
COMBINATORIAL PROPAGATION DELAYS

OUTPUTS / INPUTS	Y_i	C_{n+4}
$\overline{OE}$	15	–
$\overline{ZERO}$	20	30
OR_i	10	20
S_0, S_1	20	30
D_i	10	20
C_n	–	10

TABLE III
CLOCK CONTROLLED PROPAGATION DELAYS

FUNCTIONAL PATH	CLOCK TO Y_i	CLOCK TO C_{n+4}
VIA INSTRUCTION REG	25	30
VIA μ PROGRAM COUNTER	25	30
VIA FILE-PUSH MODE	35	40
VIA FILE-POP MODE	40	45

$T_A = 25°C$ $R_L = 2k\Omega$ $C_L = 15pF$

TABLE IV
SET-UP AND HOLD TIME REQUIREMENTS

EXTERNAL INPUTS	t_s	t_h
$\overline{RE}$	15	0
R_i	10	0
PUSH/POP	10	0
$\overline{FE}$	10	0
C_n	10	0
D_i	15	0
OR_i	15	0
S_0, S_1	25	0
$\overline{ZERO}$	25	0

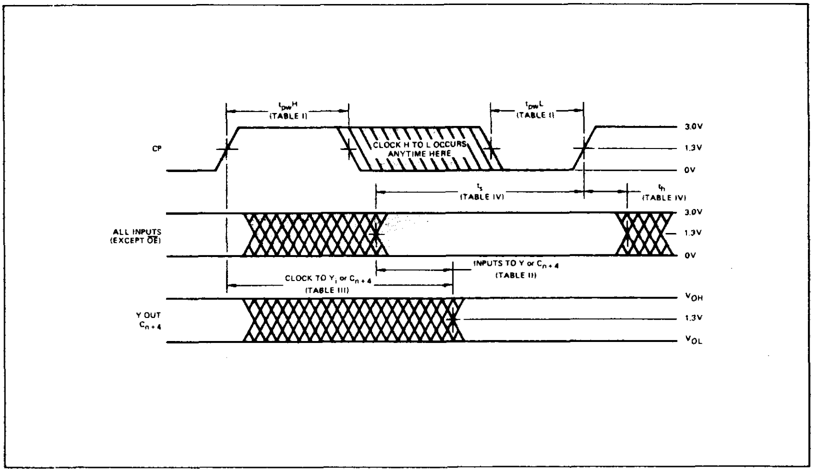


Figure 12. Switching Waveforms. See Tables for Specific Values.

Chapter 19
THE MC10800 SERIES CHIP SLICE LOGIC

The MC10800 chip slice logic devices manufactured by Motorola Semiconductor represent the most recently introduced chip slice logic products. These devices will be available in commercial quantities by early 1977, offering three very significant enhancements over the 2900/6700 logic which we described in Chapter 18:

1) **MC10800 devices are manufactured using Emitter Coupled Logic (ECL),** which creates very high speed devices. We cannot exactly define microinstruction execution times, because you have great flexibility in determining events which will occur within a microinstruction; however, individual events within the arithmetic and logic unit are executed in the range 5 to 50 nanoseconds.
2) If you are familiar with central processing unit design, it will be apparent to you that the 2900 series and 6700 series chip slice products are going to require a significant amount of additional interface logic. This additional interface logic must be created out of standard digital logic packages. **The MC10800 series covers a substantially greater portion of normal central processing unit logic;** thus in most cases it will result in smaller chip counts.
3) **MC10800 series devices are more flexible and more powerful than 2900 series or 6700 series.** They are more flexible because they allow more data flow options. They are more powerful because they enable more CPU operations — and they distribute logic in a manner better suited to central processing unit design. As a result, however, the MC10800 series chip slice products will use a larger microinstruction object code than the 2900 or 6700 chip slice products; and that will translate into a larger control read-only memory.

Figure 19-1 illustrates the devices which constitute the MC10800 device set and the way in which they connect in order to generate a central processing unit.

The MC10800 ALU represents a 4-bit slice through an Arithmetic and Logic Unit. In contrast to the 2901 and 6701, the MC10800 does not include read/write memory for registers. You create a register file with external memory; that allows you to design central processing units with a substantial number of programmable registers. Combining the register file and the MC10800 ALU chip slice, you have logic equivalent to an ALU chip slice as described in Volume I, Chapter 4.

The microprogram which drives the entire system will be stored in a control memory which is sequenced by the MC10801 Microprogram Control Unit. Each MC10801 provides a 4-bit slice of the total control memory sequencing logic. Conceptually the MC10801 does not differ from the description given in Volume I, Chapter 4; however, in implementation it does. The MC10801 Microprogram Control Unit has 16 instruction codes which allow you to sequence microinstructions within control memory using branch-on-condition, subroutine and interrupt logic. There is no limit to the manner in which you create microinstruction execution sequence logic; sequences may depend on status conditions created within, or beyond the ALU; moreover, external interrupts may directly influence microinstruction sequences.

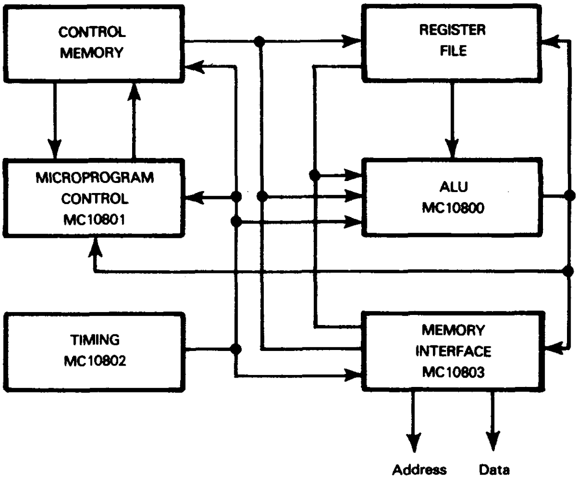


Figure 19-1. MC10800 Series Devices In A Central Processing Unit Configuration

The MC10803 Memory Interface device performs all of the memory addressing operations required to address program and data memory. This device contains its own small arithmetic and logic unit, so that computations needed by indexed addressing or any other addressing scheme do not use the ALU logic — and can therefore proceed in parallel. The MC10803 is also a 4-bit slice product which can be cascaded to any word size; it has sufficient capabilities to handle all common memory addressing schemes, ranging from the simplest microcomputer to the largest mainframe computer. Using a 2901 or 6701 ALU slice product, you must perform memory addressing operations using ALU logic and the 16 RAM locations provided within the ALU slice itself.

The MC10802 timing device can, under program control, create four timing signals, with any required signal interactions or interdependences. This is not a chip slice part, but if four timing signals are insufficient, additional MC10802 devices will be needed to create additional timing signals.

If you look at Figure 19-1, you will see that the various devices described cover a substantial portion of the logic within any central processing unit. This is in marked contrast to 2900 series or 6700 series devices which leave undefined a great deal of memory interface logic, timing logic and control memory - ALU interface logic undefined.

We will now look at each of the MC10800 series devices in overview. These overviews will summarize the capabilities of devices; however, refer to manufacturer's literature for detailed information.

MC10800 series devices are manufactured by:

MOTOROLA SEMICONDUCTOR
Box 20912
Phoenix, Arizona 85036

At the present time there is no second source.

All devices are manufactured using Emitter Coupled Logic. Devices are fabricated using unique Quad Inline Packages, which provide two sets of parallel pins on each side of the package. This configuration will require special modifications to existing PC cards; however, it results in very compact packaging.

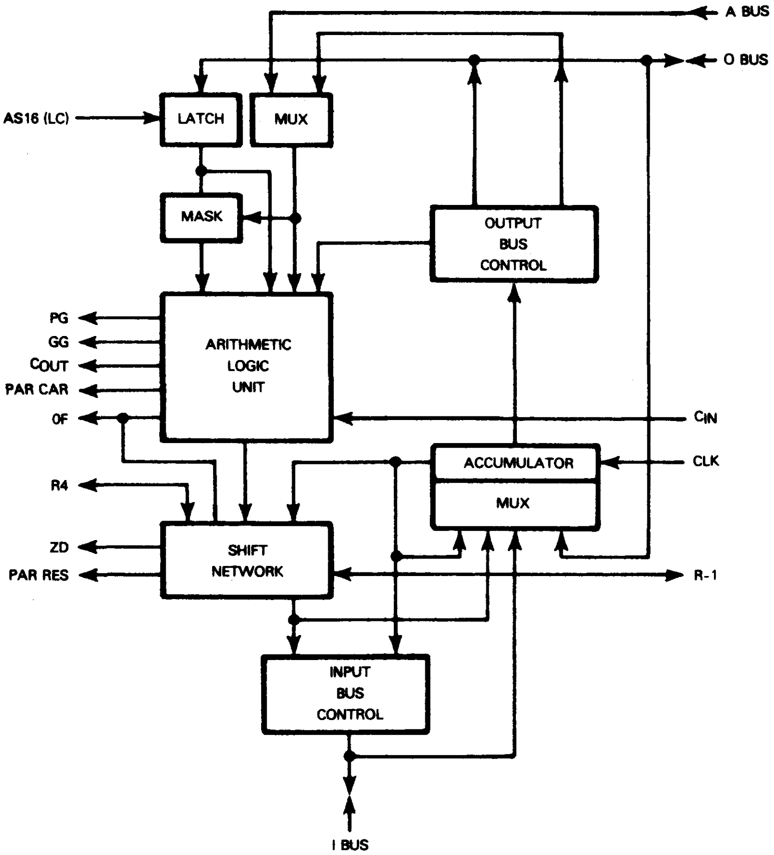


Figure 19-2. The MC10800 ALU Slice Functional Diagram

THE MC10800 ARITHMETIC AND LOGIC UNIT SLICE

Figure 19-2 illustrates the data flows and logic functions of the MC10800 ALU slice.

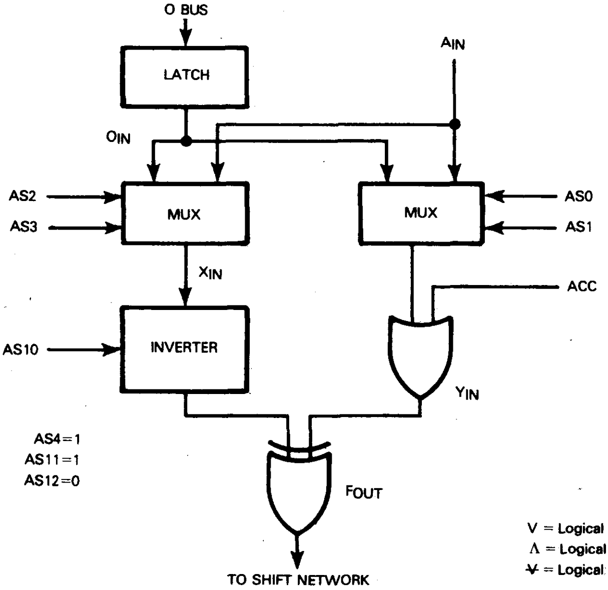
The Arithmetic and Logic Unit block is similar to logic with the same name, as described for the 2901/6701 — or the general case ALU logic, as described in Volume I, Chapter 4.

Functions which the Arithmetic and Logic Unit can perform include:

- Binary and BCD addition and subtraction
- Boolean operations, AND, OR and Exclusive-OR
- Status signals generated by the Arithmetic and Logic Unit are comprehensive and adequate. PG and GG are standard carry look ahead signals. OF is an Overflow status. Carry In (CIN) and Carry Out (COUT) allow arithmetic operations to be performed by a number of cascaded ALU slices. Notice that there is a Parity status.
- ALU logic automatically creates statuses appropriate for BCD or binary arithmetic.

Table 19-1. MC10800 ALU Logical Operations

Y MUX		X MUX		INVERT	ACC	FUNCTION
AS0	AS1	AS2	AS3	AS10	$\overline{AS5} \wedge AS6$	
0	1	0	1	1	0	LOGIC 0
0	0	1	0	1	0	A
0	0	0	1	1	0	0
0	0	1	0	0	0	$\overline{A}$
0	0	0	1	0	0	$\overline{0}$
0	0	1	1	1	0	$A \vee O$
0	1	0	0	0	0	$A \vee \overline{O}$
1	0	0	0	0	0	$\overline{A} \vee O$
0	0	0	0	1	0	$A \wedge O$
0	1	1	1	1	0	$A \wedge O$
0	1	0	0	1	0	$A \wedge O$
0	1	1	0	1	0	$A \veebar O$
0	1	1	0	0	0	$\overline{A \veebar O}$
0	0	0	0	0	0	$\overline{A \wedge O}$
0	0	1	1	0	0	$\overline{A \vee O}$
0	1	0	1	0	0	LOGIC 1
1	0	1	0	1	1	$ACC \wedge \overline{A}$
0	1	0	1	1	1	$ACC \wedge \overline{O}$
1	0	1	0	0	1	$\overline{ACC} \vee A$
0	1	0	1	0	1	$\overline{ACC} \vee O$
0	0	1	0	1	1	$ACC \veebar \overline{A}$
0	0	1	0	0	1	$ACC \veebar A$
0	0	0	1	1	1	$ACC \veebar \overline{O}$
0	0	0	1	0	1	$ACC \veebar O$
0	0	0	0	1	1	$ACC \veebar A \wedge O$
0	0	0	0	0	1	$ACC \veebar \overline{A \wedge O}$
0	0	1	1	1	1	$ACC \veebar A \vee O$
0	0	1	1	0	1	$ACC \veebar \overline{A \vee O}$

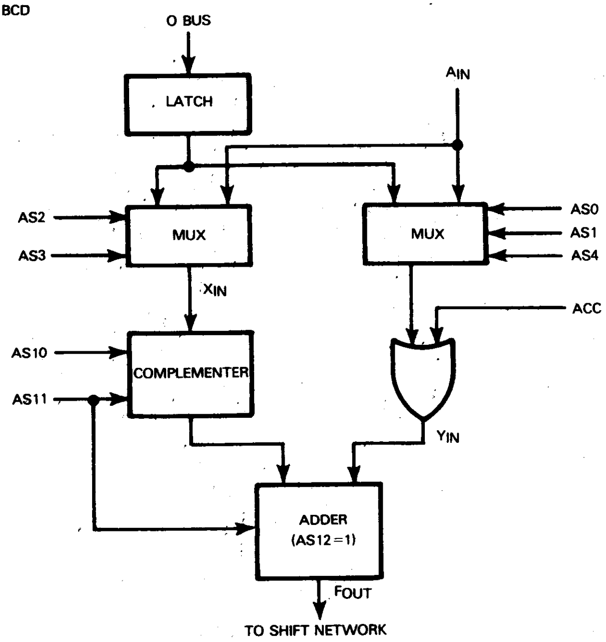


V = Logical Inclusive OR
Λ = Logical AND
⊻ = Logical Exclusive OR

Table 19-2. MC10800 Arithmetic Operations

Y MUX		X MUX		±2	COMP.	ACC	BINARY FUNCTION (PLUS C_{IN})	BCD FUNCTION (PLUS C_{IN})
AS0	AS1	AS2	AS3	AS4	AS10	$\overline{AS5} \wedge AS6$	AS11=1	AS11=0
1	0	0	1	1	1	0	A PLUS O	A PLUS O
1	0	0	1	1	0	0	A PLUS $\overline{O}$	A PLUS 9s COMP. O
0	1	1	0	1	0	0	O PLUS $\overline{A}$	O PLUS 9s COMP. A
0	0	1	0	1	1	0	A	A
0	0	0	1	1	1	0	O	O
0	0	1	0	1	0	0	$\overline{A}$	9s COMP. A
0	0	0	1	1	0	0	$\overline{O}$	9s COMP. O
1	1	1	0	1	1	0	-1 PLUS A	*
1	1	0	1	1	1	0	-1 PLUS O	*
1	1	1	0	0	1	0	-2 PLUS A	*
1	1	0	1	0	1	0	-2 PLUS O	*
0	0	1	0	0	1	0	+2 PLUS A	+2 PLUS A
0	0	0	1	0	1	0	+2 PLUS O	+2 PLUS O
1	0	1	0	1	1	0	A PLUS A	A PLUS A
0	1	0	1	1	1	0	O PLUS O	O PLUS O
0	0	1	0	1	1	1	ACC PLUS A	ACC PLUS A
0	0	0	1	1	1	1	ACC PLUS O	ACC PLUS O
0	0	1	0	1	0	1	ACC PLUS $\overline{A}$	ACC PLUS 9s COMP. A
0	0	0	1	1	0	1	ACC PLUS $\overline{O}$	ACC PLUS 9s COMP. O
0	0	0	0	1	1	1	ACC PLUS $A \wedge O$	ACC PLUS $A \wedge O$
0	0	0	0	1	0	1	ACC PLUS $\overline{A \wedge O}$	ACC PLUS 9s COMP. $A \wedge O$
0	0	1	1	1	1	1	ACC PLUS $A \vee O$	*
0	0	1	1	1	0	1	ACC PLUS $\overline{A \vee O}$	*

*Not defined in BCD



MC10800 shift logic is confined to one location; it is on the Arithmetic and Logic Unit output path. However, data paths do allow you to bypass the Arithmetic and Logic Unit if a simple shift operation is to be performed.

The most important aspect of the MC10800 ALU is the freedom you have to move data via innumerable paths. Overall, there are three busses: the A, I and O Busses. Each is a 4-bit bus. The A Bus is input only, while the I and O Busses are bidirectional. The various data paths are illustrated in Figure 19-2.

The mask logic needs definition.

When activated, this logic allows one of the ALU inputs to be the AND or OR of the A and I Bus inputs.

The various operations which can be performed by the MC10800 ALU are summarized, along with the microinstruction codes, in Tables 19-1 and 19-2.

Table 19-1 defines the logical operations which may be performed while Table 19-2 defines the arithmetic operations which may be performed.

THE MC10801 MICROPROGRAM CONTROL UNIT

The MC10801 Microprogram Control Unit consists of "Next Address" logic, plus eight 4-bit registers. Four of the 4-bit registers are organized as a Stack, while the remaining four may be defined as follows:

R0	Microprogram Counter
R1	Retry, post-interrupt R0 buffer or cycle counter
R2	General purpose storage or microinstruction data storage
R3	Status register

Program sequencing is controlled by 16 instructions which are specified via a 4-bit instruction code, input to the next address logic as four signals. The 16 instructions listed below refer to the contents of the four programmable registers, two branch flags, plus two external 4-bit inputs.

The external 4-bit inputs are referred to as the "NA inputs" and the "O Bus".

The NA (Next Address) inputs, along with the 4-bit instruction code, must come from the control read-only memory, along with microinstruction code. Thus, as each microinstruction code is being executed, the address for the next microinstruction code is being computed.

The O Bus represents external data which may be input from any source.

One of the two branch flags is common to all MC10801 slices in a unit; the other branch flag may be individually created for each MC10801 slice.

Any instruction listed below which causes a branch- or jump-to-subroutine automatically pushes the contents of R0 onto the Stack before loading the microsubroutine starting address into R0. Any instruction which causes a return from subroutine pops the Stack into R0.

By combining the 16 instructions listed below in various ways, it is possible to create virtually any type of branch logic to access the control read-only memory.

INC	-	Increment
JMP	-	Jump to NA inputs
JIB	-	Jump to I Bus
JIN	-	Jump to I Bus and load R2
JPI	-	Jump to R2
JEP	-	Jump to External Bus (O Bus)

JL2	-	Jump to NA inputs and load R2
JLA	-	Jump to NA inputs and load R1
JSR	-	Jump to Subroutine
RTN	-	Return from Subroutine
RSR	-	Repeat subroutine (load R1 from NA inputs)
RPI	-	Repeat Instruction (jump to R2)
BRC	-	Branch to NA inputs on condition; otherwise increment
BSR	-	Branch to Subroutine on condition; otherwise increment
ROC	-	Return from Subroutine on condition or jump to NA inputs
BRM	-	Branch and Modify address with branch inputs (multiway branches)

THE MC10802 TIMING DEVICE

This device is capable of creating four output clock signals. Clock signals can be output in 1, 2, 3 or 4 phases; and the duration of any single clock phase may be doubled by inputting an appropriate control signal. This allows single slow steps in asynchronous logic sequences to be accommodated.

A number of MC10802 devices may be cascaded together if more than four clock signals are required.

A number of control signals allow the MC10802 to be started and stopped on demand. Thus, almost any timing sequence that you require may be generated.

THE MC10803 MEMORY INTERFACE DEVICE

You will use this device in a CPU configuration in order to create memory addresses. Program Counter logic, Data Counter logic, indexed addressing, indirect addressing or any other address computations can be handled by this device. Logic is illustrated in Figure 19-3. Each MC10803 memory interface device is a 4-bit slice. You can create memory addresses of any width by cascading MC10803 4-bit slices.

In order to provide total address creation flexibility, the MC10803 has five separate 4-bit data ports, five 4-bit registers, plus an arithmetic and logic unit.

Let us first look at the MC10803 data ports.

The two 4-bit bidirectional O and I Busses connect to internal CPU logic busses. Via these two busses data is transferred to or from other parts of the CPU. The bidirectional 4-bit Data Bus allows data transfers between the memory interface device and logic beyond the CPU. The final effective memory address is output via the 4-bit Address Bus. There is a further 4-bit Pointer input bus via which data can be input directly to the arithmetic and logic unit.

The arithmetic and logic unit is capable of performing binary addition or subtraction, logical AND, OR or Exclusive-OR, plus shift left and shift right operations. The possible sources and destinations for ALU operations are illustrated in Figure 19-3.

There are six 4-bit registers within the MC10803 memory interface device.

MAR is the Memory Address register, out of which the final effective memory address will be output to external memory.

MDR is a 4-bit Data register within which temporary data can be held in any way.

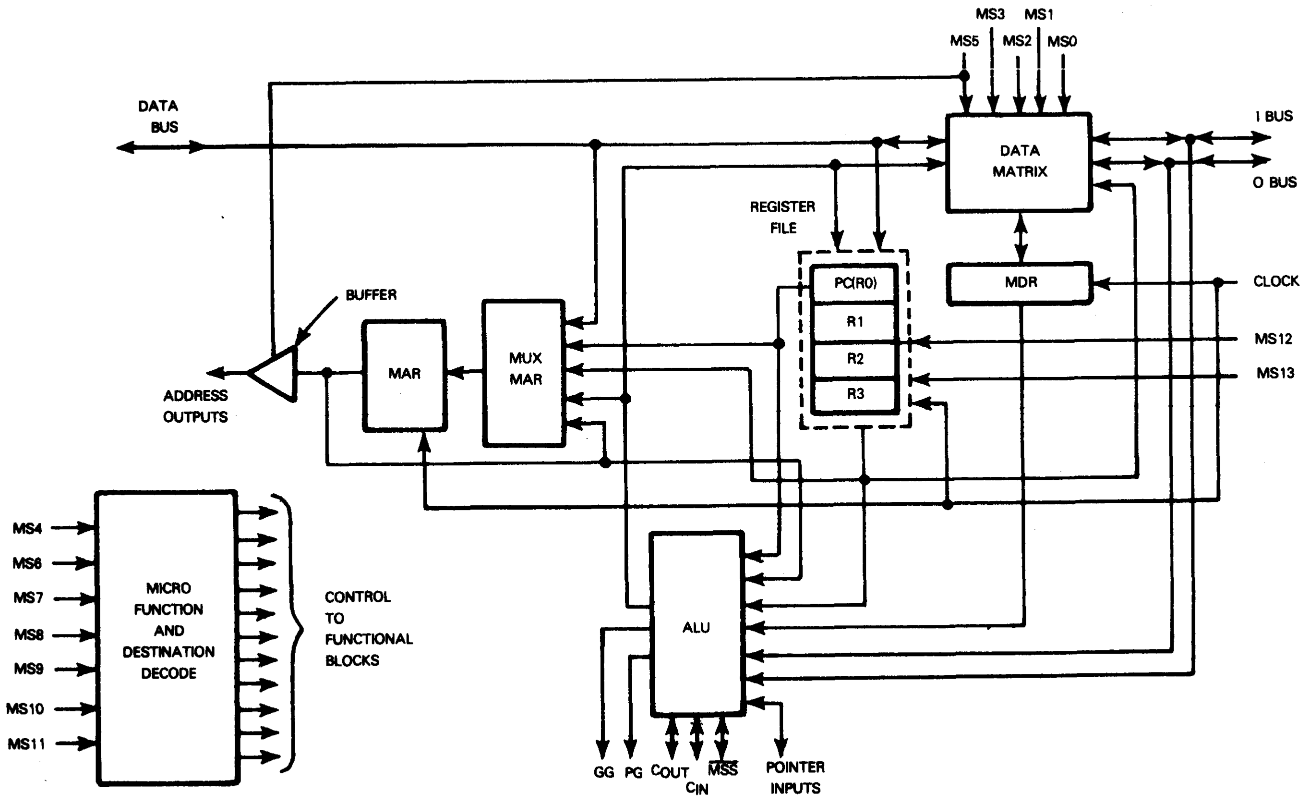


Figure 19-3. MC10803 Memory Interface Device Block Diagram

Four 4-bit registers in a register file are used to hold frequently needed data. For example, R0 will invariably become the Program Counter. R1, R2 and R3 can be used to hold indexes, base addresses or other similar data. Note that the four registers in the register file represent additional logic over and above the external register file, which can have any size. Thus, a number of Index registers could be maintained in the external register file, while data that is being frequently manipulated will be maintained in the MC10803 internal register file.

The data matrix has a number of control signals which usually will be input from the microprogram control read-only memory; the control signals allow one of the following data transfers to be specified:

FOB	-	Register File to O Bus
ROB	-	Data Register to O Bus
RDB	-	Register File to Data Bus
ODB	-	O Bus to Data Bus
RDB	-	Data Register to Data Bus
ADR	-	ALU to Data Register
BDR	-	Data Bus to Data Register
AIB	-	ALU to I Bus
BIB	-	Data Bus to I Bus
IDR	-	I Bus to Data Register
ODR	-	O Bus to Data Register
NOP	-	No Operation
BRF	-	Data Bus to Register File
BAR	-	Data Bus to Address Register
MDR	-	Modify Data Register (I Bus to Data Register and Data Register to O Bus)
PFB	-	Pipeline from Data Bus (Data Bus to Register File and Data Register to O Bus)
PTB	-	Pipeline to Data Bus (I Bus to Data Register and Data Register to Data Bus)

DATA SHEETS

The following pages contain specific electrical data for the MC10800 ALU.

SET UP AND HOLD TIMES (NANOSECONDS AT 25°C)

PATH	LONGEST PATH						SHORTEST PATH					
	SET UP			HOLD			SET UP			HOLD		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A BUS→ACC (Via ALU)		31.0			−17.0			15.0			− 7.5	
ø BUS→ACC (via ALU)		32.5			−16.5			16.0			− 7.5	
ø BUS→ACC (DIRECT)		3.5			+ 5.5			3.5			+ 4.5	
I BUS→ACC (DIRECT)		4.0			+ 4.5			4.0			+ 5.0	
AS0, AS1, AS4→ACC		28.5			−18.5			12.0			− 5.0	
AS2, AS3→ACC		31.0			−21.0			17.5			− 8.0	
AS5, AS6→ACC		40.0			−23.5			19.0			− 6.0	
AS10, AS11→ACC		35.5			−26.0			17.5			− 5.0	
AS12→ACC		23.0			−10.5			13.0			− 3.0	
R-1, R4→ACC		6.5			+ 1.0			5.5			+ 2.5	
AS7→ACC		7.5			+ 4.5			7.5			+ 4.5	
AS13, AS14→ACC		12.5			+ 4.5			8.0			0.0	
AS9, AS15→ACC		5.0			+ 1.5			4.5			+ 7.0	
AS16→ACC		35.5			−18.5			19.0			− 9.5	
CIN→ACC		19.0			− 8.0			10.0			− 2.5	
ø BUS→LATCH		1.0			+ 4.5			1.5			+ 5.0	

PROPAGATION DELAY TIMES (NANOSECONDS AT 25°C)

	OUTPUT			I BUS			P_G, G_G			C_{OUT}			OF, ZD, R-1, R4			P_C, P_R			ø BUS		
INPUT	VIA	MODE	FUNCTION	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A BUS, ø BUS	ALU	ARITH	SUBTRACT		33.0			18.0			19.5			33.0			30.5			–	
CIN	ALU	ARITH	ADDITION		19.0			–			7.5			15.5			18.5			–	
AS*	ALU	ARITH	SUBTRACT ACC		40.0			23.5			25.5			39.5			37.5			-	
AS16	ALU	ARITH	SUBTRACT		35.0			20.5			22.0			35.0			33.0			–	
R-1, R4	SHIFT	SHIFT LEFT SHIFT RIGHT			8.0			.			–			–			–			–	
AS7, AS13, AS14	SHIFT	SHIFT LEFT SHIFT RIGHT			13.5			–			–			–			–			–	
AS9, AS15	DIRECT	SHIFT ACC	–		9.0			–			–			–			–			–	
AS8	DIRECT	ENABLE DISABLE			7.5			–			–			–			–			..	
AS5, AS6	DIRECT	ENABLE DISABLE	.		–			–			–			–			–			7.5	
CLK	A BUS ALU	ARITH	SUBTRACT ACC		47.5			35.0			36.5			41.0			42.0			.	
CLK	ALU	ARITH	ADD ACC		41.5			–			–			–			–			..	
CLK	SHIFT	AS7 0	MULTIPLE SHIFT		19.0			–			–			19.5			20.5		_		
CLK	DIRECT	-	ACC TO I, ø BUS		12.0			-			–			–			–			11.5	
ø BUS	ALU (MASK)	LOGIC	WITHOUT COMPLEMENT		31.5			–			–			–			–			–	
CLK	A BUS (MASK)	LOGIC	WITHOUT COMPLEMENT		42.0			–			–			–			–			–	
CLK	A BUS (MASK)	LOGIC	WITH COMPLEMENT		44.5			–			–			–			–			–	
CLK	A BUS (Y MUX)	LOGIC	WITHOUT COMPLEMENT		37.5			-			–			34.5			–			–	
OUTPUT RISE TIME, t_r (20% to 80%)					3.5			3.5			3.5			3.5			3.5			3.5	
OUTPUT FALL TIME, t_f (20% to 80%)					2.5			2.5			2.5			2.5			2.5			2.5	

AS* AS0, AS1, AS2, AS3, AS4, AS5, AS6, AS10, AS11, AS12

RECOMMENDED OPERATING CONDITIONS

PARAMETER	SYMBOL	VALUE	UNIT
Supply Voltage (V_{CC} = 0 Volts)	V_{TT} V_{EE}	−1.9 to −2.2 −4.68 to −5.72	Vdc Vdc
Operating Temp. (Functional)	T_A	−30 to +85	°C
Output Drive	—	50Ω to −2.0 Vdc	—
Maximum Clock Input Rise and Fall Time (20% to 80%)	t_r, t_f	10	ns
Minimum Clock Pulse Width	PW	5	ns

ELECTRICAL CHARACTERISTICS

Each MECL 10,000 series circuit has been designed to meet the dc specifications shown in the test table, after thermal equilibrium has been established. The circuit is in a test socket or mounted on a printed circuit board and transverse air flow greater than 500 linear fpm is maintained. Outputs are terminated through a 50-ohm resistor to −2.0 volts. Test procedures are shown for only one input, or for one set of input conditions. Other inputs tested in the same manner.

@Test Temperature	TEST VOLTAGE VALUES					
	Volts					
	V_{IHmax}	V_{ILmin}	V_{IHAmin}	V_{ILAmax}	V_{EE}	V_{TT}
−30°C	−.890	−1.890	−1.205	−1.500	−5.2	−2
+25°C	−.810	−1.85	−1.105	−1.475	−5.2	−2
+85°C	−.700	−1.825	−1.035	−1.440	−5.2	−2

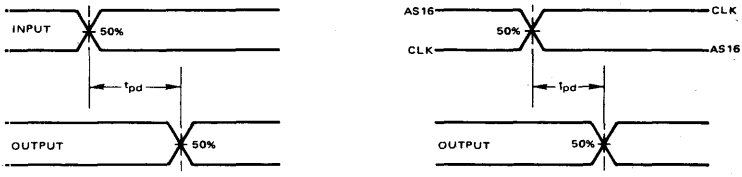
Characteristic	Symbol	Pin Under Test	TEST LIMITS								VOLTAGE APPLIED TO PINS LISTED BELOW:						(V_{CC}) Gnd
			−30°C		+25°C			+85°C									
			Min	Max	Min	Typ	Max	Min	Max	Unit	V_{IHmax}	V_{ILmin}	V_{IHAmin}	V_{ILAmax}	V_{EE}	V_{TT}	
Power Supply Drain Current	I_{EE} I_{TT}	1, 24 25, 48		— —	— —	195 180	— —	— —	— —	mAdc					1, 24	25, 48	12, 36 7, 17
Input Current	I_{inH}	23 31 27		— — —	— — —	— — —	65 350 435	— — —	— — —	μAdc	23 31 27				↓	↓	↓
	I_{inL}	31		—	0.5	—	—	—	—	μAdc		31			↓	↓	↓
Logic "0" Output Voltage	V_{OH}	13 10	−1.060 −1.060	−.89 −.89	−.960 −.960	— —	−.810 −.810	−.890 −.890	−.700 −.700	Vdc Vdc	8, 26, 46, 47 *				↓	↓	↓
Logic "1" ** Output Voltage	V_{OL}	13 10	−1.94 −1.89	−1.675 −1.675	−1.90 −1.85	— —	−1.65 −1.65	−1.875 −1.825	−1.615 −1.615	Vdc Vdc	26, 46, 47 *, 47				↓	↓	↓
Logic "0" Threshold Voltage	V_{OHA}	13 10	−1.08 −1.08	— —	−.980 −.980	— —	— —	−.910 −.910	— —	Vdc Vdc	26, 46, 47 *		8	47	↓	↓	↓
Logic "1" Threshold Voltage	V_{OLA}	13 10	— —	−1.655 −1.655	— —	— —	−1.63 −1.63	— —	−1.595 −1.595	Vdc Vdc	26, 46, 47 *		47	8	↓	↓	↓

* V_{IH} on pins 19, 26, 30, 31, 32, 33, 34, 35, 37

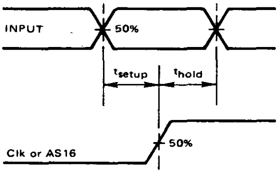
** The bi-directional outputs are specified at −1.90 volts for V_{OL} min. This is lower than the normal V_{OL} min. output to give increased noise margin for bussing applications.

SWITCHING WAVEFORMS

PROPAGATION DELAYS



SETUP AND HOLD



TEST PROCEDURE:

a) Establish setup time with long t_{hold}.

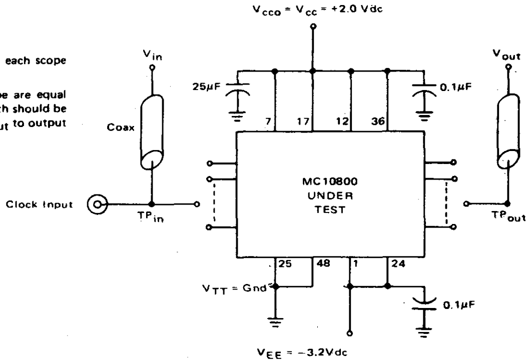
b) Keeping the leading edge of the input constant (t_{setup}) vary the trailing edge of the input to determine t_{hold}.

NOTE: t_{setup} and t_{hold} as defined are positive. Internal delays in the data path may result in a shift of the data waveform to the left, with respect to the clock, resulting in negative hold times.

SWITCHING TIME TEST CIRCUIT

50 ohm termination to ground located in each scope channel input.

All input and output cables to the scope are equal lengths of 50 ohm coaxial cable. Wire length should be <¼ inch from TP_{in} to input pin and TP_{out} to output pin.



Chapter 20
THE HEWLETT PACKARD MC2

The MC2 is the first microprocessor manufactured by Hewlett Packard. The MC2 is a 16-bit microprocessor designed specifically for process control applications; it has been designed for internal use within the many instruments and electronic products manufactured by Hewlett Packard. The MC2 is most unlikely to be sold as a single chip in the foreseeable future; even its availability as a computer card is not guaranteed at the present time.

The most important aspect of the MC2 is its technology; it is built using CMOS logic with Silicon On Sapphire (SOS technology). The CMOS logic gives the MC2 typical CMOS low power requirements and noise insensitivity, while Silicon On Sapphire technology gives it high speed.

Using a +12V power supply the MC2 operates with a 125 nanosecond clock and executes instructions in 4 to 12 clock cycles. Typical instructions are therefore executed in less than one microsecond.

The MC2 is packaged on a squared, 48-pin leadless ceramic substrate.

The sole manufacturer of the MC2 is:

HEWLETT PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, CA 95014

A second source for this microprocessor is unlikely in the foreseeable future.

AN MC2 SYSTEM OVERVIEW

Logic implemented on the MC2 CPU chip is illustrated in Figure 20-1. A number of support devices for the MC2 have been manufactured, but no information on these support devices is available at the present time.

Clock logic is external to the MC2 chip; however a simple single waveform external clock signal will suffice.

Figure 20-1 shows I/O interface logic as being implemented on the MC2 CPU chip. This reflects the unusual way in which the MC2 handles external devices. The MC2 CPU has eight 16-bit general-purpose programmable registers. Every I/O device is assumed to have a CPU equivalent set of eight programmable registers. Instructions and control signals of the MC2 treat registers of the CPU and I/O devices similarly, which means that no special I/O interface logic is required.

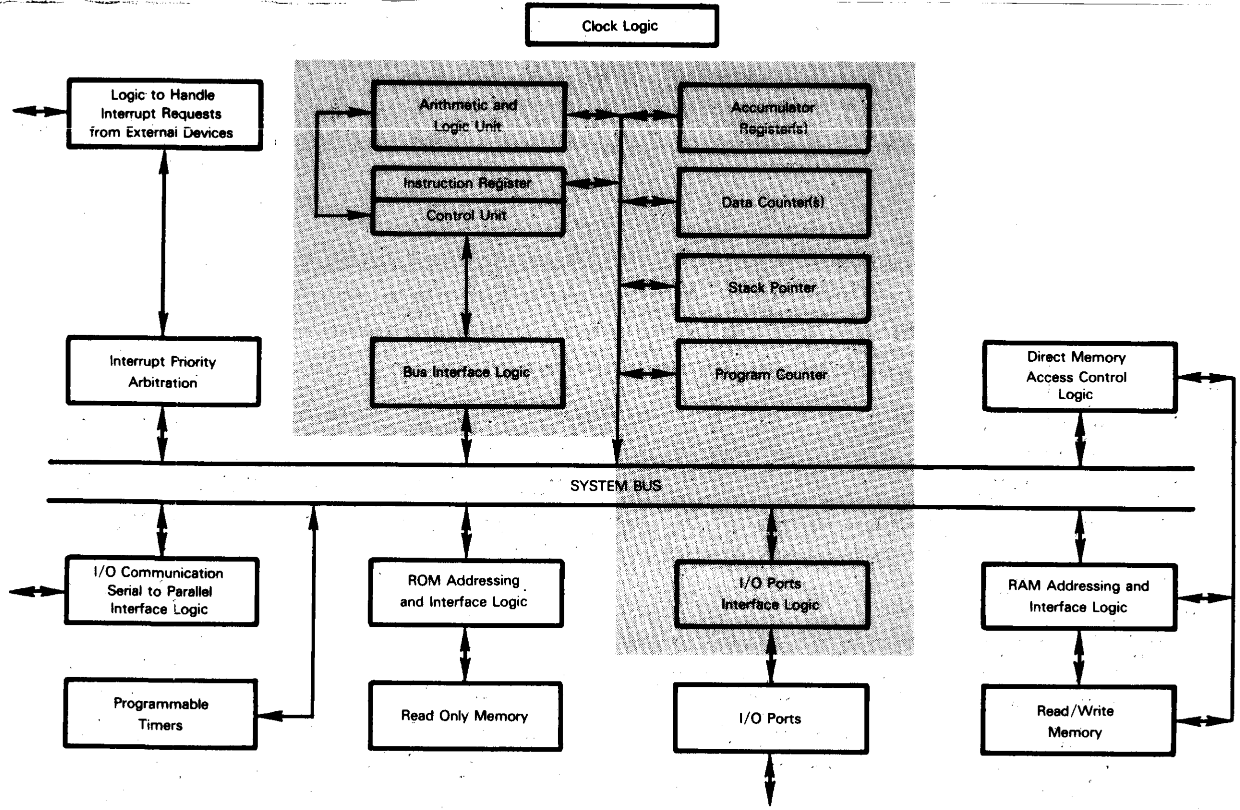
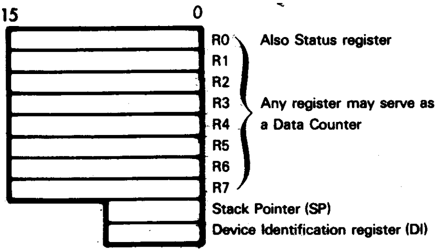

Clock Logic
Logic to Handle Interrupt Requests from External Devices
Arithmetic and Logic Unit
Accumulator Register(s)
Instruction Register
Control Unit
Data Counter(s)
Stack Pointer
Interrupt Priority Arbitration
Bus Interface Logic
Program Counter
Direct Memory Access Control Logic
SYSTEM BUS
I/O Communication Serial to Parallel Interface Logic
ROM Addressing and Interface Logic
I/O Ports Interface Logic
RAM Addressing and Interface Logic
Programmable Timers
Read Only Memory
I/O Ports
Read/Write Memory

Figure 20-1. Logic Of The Hewlett Packard MC2 Microprocessor

MC2 PROGRAMMABLE REGISTERS AND STATUS

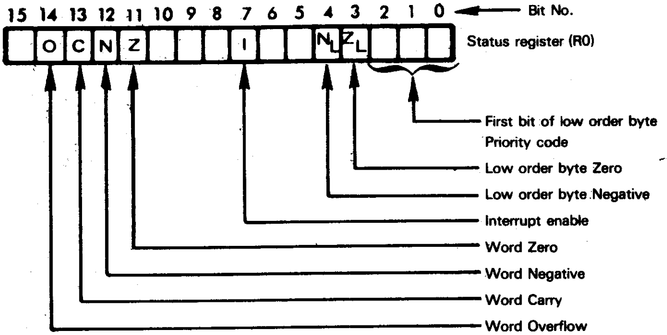
The MC2 has eight 16-bit programmable registers and an 8-bit Stack Pointer. In addition there is an 8-bit I/O Device Identification register. Registers may be illustrated as follows:



15
0
R0
R1
R2
R3
R4
R5
R6
R7
Also Status register
Any register may serve as a Data Counter
Stack Pointer (SP)
Device Identification register (DI)

Any one of the eight 16-bit registers may be used as a Data Counter.

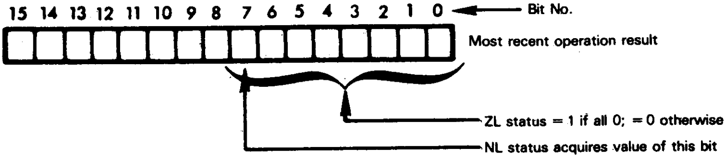
Register R0 serves as the Status register. Its contents are interpreted as follows:



The Zero, Negative, Carry and Overflow statuses in bits 11, 12, 13 and 14, respectively, apply to the 16-bit result of the most recent data operation performed. Zero, Carry and Overflow are standard statuses, as described in Volume I, Chapter 6. The Negative status reflects the sign of the 16-bit result; that is to say, it is set to the value of the result's high order bit.

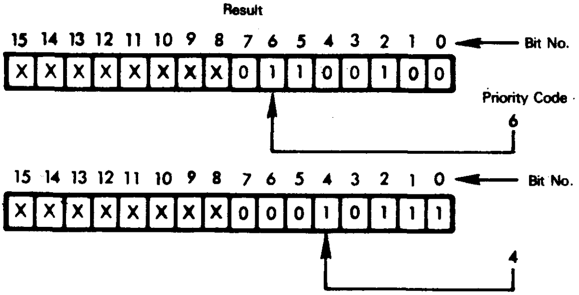
The Interrupt Enable flag in Status register bit 7 is set to 1 in order to enable interrupts. It is reset to 0 in order to disable interrupts.

The low order byte Zero and Negative statuses are identical to standard Zero and Negative statuses, except that they reflect the low order 8 bits of the most recent operation's result. This may be illustrated as follows:



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bit No.
Most recent operation result
ZL status = 1 if all 0; = 0 otherwise
NL status acquires value of this bit

The low order three Status register bits are referred to as a Priority Code. This Priority Code identifies the highest order 1 bit in the low order byte of the most recent operation's result. The Priority Code has the value of the bit position being identified. Here are some examples of Priority Codes:



The Stack Pointer enables a 256-word Stack. The Stack occupies the first 256 words of read/write memory, with memory word 0 being the top of the Stack.

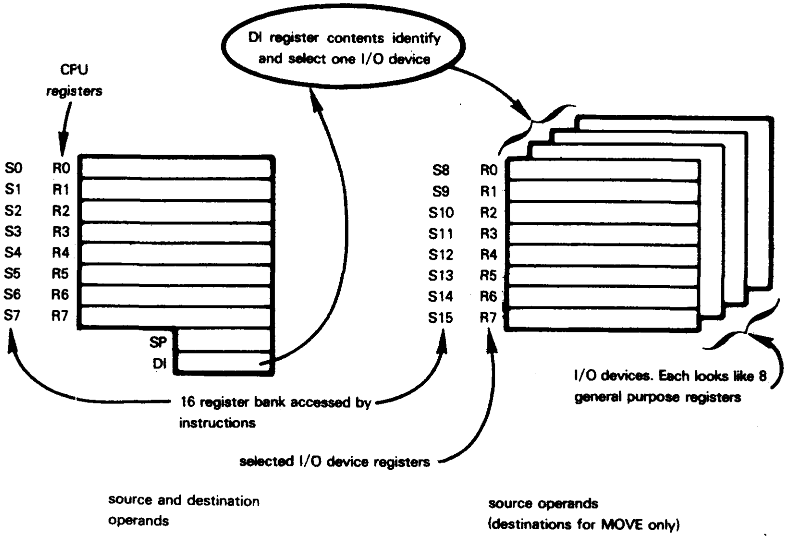

DI register contents identify and select one I/O device
CPU registers
S0 R0
S1 R1
S2 R2
S3 R3
S4 R4
S5 R5
S6 R6
S7 R7
SP
DI
S8 R0
S9 R1
S10 R2
S11 R3
S12 R4
S13 R5
S14 R6
S15 R7
I/O devices. Each looks like 8 general purpose registers
16 register bank accessed by instructions
selected I/O device registers
source and destination operands
source operands (destinations for MOVE only)

Figure 20-2. CPU And I/O Device Registers' Organization For The MC2

The I/O Device Identification register, also referred to as a Base register, identifies one of 256 possible external I/O devices. The identified external I/O device will be interpreted as consisting of eight 16-bit registers. **When executing Register-Register instructions, there is little differentiation made between the eight CPU registers and the eight registers of the identified external device.** The two sets of eight registers constitute a 16-register bank out of which two operands are selected. The two operands may or may not come from the same register. The destination, which is the first identified operand register, is usually one of the CPU registers (R0 - R7); only the Register-Register Move instruction permits an external register (R8 - R15) to be the destination. **This scheme is illustrated in Figure 20-2.**

MC2 MEMORY ADDRESSING MODES

The MC2 is quite limited in its memory reference capabilities. Instructions allow you to load data from memory into a CPU register, or to store data from CPU registers to memory. **Data access instructions use direct memory addressing or implied memory addressing.**

Direct memory addressing instructions are two words long; the second instruction object code word provides the 16-bit direct memory address.

Instructions that use implied memory addressing allow any one of the eight CPU registers to specify the 16-bit memory address.

Conditional Branch instructions and Subroutine Call instructions allow direct and indirect addressing; however direct addressing is program relative and the displacement is an 8-bit signed binary number.

HARDWARE ASPECTS OF THE MC2

We are not going to describe pins and signals of the MC2 because Hewlett Packard has not made sufficient information available at the present time. Also, such information will be irrelevant until you can buy the MC2 as a chip. **Instead we will provide a brief summary of principal MC2 hardware characteristics.**

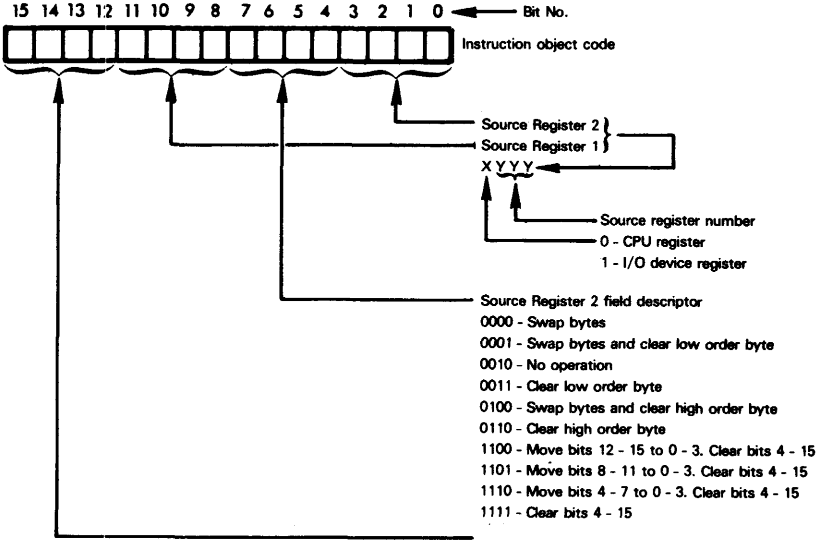
The MC2 is packaged as a 48-pin package. This allows separate 16-bit Data and Address Busses, together with an adequate set of control signals. Control logic on the System Bus is asynchronous, having a request/acknowledge control philosophy. This simplifies multiple CPU configurations. Execution of a "No Operation" instruction puts any CPU into a slave state on the System Bus, at which time its internal registers may be accessed by some other "master" CPU. A slave CPU may be converted into the master via an interrupt request.

MC2 interrupt logic is primitive but effective. There is one interrupt request line; when an interrupt is acknowledged, a subroutine call to memory location $FFFE_{16}$ is executed. Memory location $FFFE_{16}$ must contain the beginning address for the interrupt service routine.

THE MC2 INSTRUCTION SET

The MC2 instruction set is characterized by a lack of distinction between CPU registers and I/O devices. Most instructions that operate on data or move data are Register-Register instructions; as illustrated in Figure 20-2, each register may be an internal CPU register or a register out of an I/O device. Thus there is no difference between Move instructions that access two CPU registers, one CPU register and an I/O device or two registers from the same I/O device. This similarity between Register-Register and I/O instructions is manifest by the way in which the MC2 instruction set has been defined in Table 20-1.

For a better understanding of MC2 instructions you must understand the way in which Register-Register instruction object codes are defined. This may be illustrated as follows:


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bit No.
Instruction object code
Source Register 2
Source Register 1
XYYY
Source register number
0 - CPU register
1 - I/O device register
Source Register 2 field descriptor
0000 - Swap bytes
0001 - Swap bytes and clear low order byte
0010 - No operation
0011 - Clear low order byte
0100 - Swap bytes and clear high order byte
0110 - Clear high order byte
1100 - Move bits 12 - 15 to 0 - 3. Clear bits 4 - 15
1101 - Move bits 8 - 11 to 0 - 3. Clear bits 4 - 15
1110 - Move bits 4 - 7 to 0 - 3. Clear bits 4 - 15
1111 - Clear bits 4 - 15

Source registers 1 and 2 may each be identified as any one of the eight CPU 16-bit registers, or any one of the eight I/O device 16-bit registers. The particular I/O device which is currently selected is defined by the contents of the I/O Device Identification register.

Source register 1 also becomes the Destination register for the result of the operation. In all instructions except MOVE, Source register 1 must be one of the eight CPU registers.

The contents of Source register 2 are manipulated before becoming an operand for the operation to be performed. The manipulation performed on Source register 2 contents is defined by the field descriptor. Register-Register instructions therefore perform an operation identified by bits 12 through 15 of the instruction object code; the operation uses two 16-bit operands as inputs. These two operands may come from the CPU registers, or the currently selected I/O device's registers. One 16-bit operand may be manipulated as defined by the field descriptor before it becomes an input to the operation specified by the instruction code.

THE BENCHMARK PROGRAM

For the Hewlett Packard MC2 our benchmark program may be illustrated as follows:

```
        LDWI    R1 = IOBUF    LOAD INPUT BUFFER ADDRESS IN REGISTER 1
        LOAD    R2 = COUNT    LOAD BUFFER SIZE IN REGISTER 2
        LOAD    R3 = TABLE    LOAD NEXT FREE TABLE LOCATION IN REGISTER 3
LOOP:   LOAD    R4 = (R1)     INPUT DATA WORD TO REGISTER 4
        STOR    (R3) = R4     STORE WORD TO NEXT FREE TABLE LOCATION
        ADDI    R1,1          INCREMENT BUFFER ADDRESS
        ADDI    R3,1          INCREMENT TABLE ADDRESS
        SUBI    R2,1          DECREMENT WORD COUNT
        CBR     LOOP IF G     RE-ITERATE IF COUNT STILL GREATER THAN ZERO
        STOR    TABLE = R3    SAVE ADDR OF NEXT TABLE WORD
```

This benchmark program makes very few assumptions. Memory is addressed in 16-bit units (rather than 8-bit bytes), and data is transferred 16 bits at a time. The input table IOBUF and the data table TABLE can have any length, and can reside anywhere in memory. The address of the first free word in TABLE is stored in the first word of the TABLE

The following notation is used in Table 20-1:

BYTE — 8 bits of immediate data — the lower byte of the instruction word.

CDST
CREG
CSRC — Any of the CPU registers (Registers 0 through 7).

DST — Any of the 16 registers, used as the destination of a move or result.

DI — The I/O Device Identification register (Base register).

F — Fill specification for register shift: if F is 0, the bit is reset to 0; if F is 1, the bit is set to 1.

<,FD> — Optional field descriptor:

SWB	Swap bytes
LJL	Left justify lower byte
LJU	Left justify upper byte
RJU	Right justify upper byte
RJL	Right justify lower byte
RJ0	Right justify high order nibble
RJ1	Right justify next-to-high order nibble
RJ2	Right justify next-to-low order nibble
RJ3	Right justify low order nibble

REG (FD) — The result of the operation of the field descriptor on the specified register.

SRC (K) — Operand field specification of one bit of the register. Register may be any of the 16 registers; K may be any number from 0 to 15.

SRC<K> — Bit K of Register SRC.

<,I> — Optional indirection specification. When I is present, the address used is the contents of the memory location addressed.

<.IF CC> Optional Condition Code representing a linear combination of the Zero and Negative status flags and "Greater Than":

N	Z	$\overline{NVZ}$	
0	0	0	Not true
0	0	1	G - greater than 0
0	1	0	E - equal to 0
0	1	1	GE - greater than or equal to 0
1	0	0	L - less than 0
1	0	1	LG - not equal to 0
1	1	0	LE - less than or equal to 0
1	1	1	Unconditional branch

LABEL A 16-bit address; it may be the second word in the instruction, or its lower byte may be the lower byte of a one-word instruction.

REG Any of the CPU or external registers.

<(REG<,FC>)> Optional indexing specification. For example, the instruction:

```
 IBR        TABLE(9,RJL)
```

will calculate an address by clearing the upper byte of the contents of Register 9 and adding the result to the 16-bit word TABLE. Then the contents of the location thus addressed will be the address at which instruction execution continues.

R0 Register 0, the Status register, as described earlier in this chapter.

PC The Program Counter.

SP The Stack Pointer.

SRC Any of the 16 registers, used as the source of an operand.

STATUSES The following status flags are affected by the instructions:

O The Overflow status
C The Carry status
N The Negative or Sign status
Z The Zero status
L The lower byte statuses NL and ZL

The following symbols are used in the STATUSES columns:

A blank means the status flag is not affected by the operation.
X The operation affects the status flag in a meaningful manner.
? The operation affects the status flag, but it is meaningless.

WORD 16 bits of immediate data.

x <y,z> Bits y through z of the Register x. For example, PC<15,8> represents the upper byte of the Program Counter.

[ ] Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.

[[ ]] Implied memory addressing; the contents of the memory location designated by the contents of a register.

$\wedge$ Logical AND

$\vee$ Logical OR

$\veebar$ Logical Exclusive-OR

← Data is transferred in the direction of the arrow.

** Exponentiation. 2**K represents a 16-bit word with a 1 in bit K, and 0 in all the other bits.

Table 20-1. A Summary Of The MC2 Instruction Set

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES					OPERATION PERFORMED
				O	C	N	Z	L	
PRIMARY MEMORY REFERENCE	LOAD	CDST = LABEL <(CSRC <,FD >)>	4						[CDST]←[LABEL + [CSRC(FD)]] Load CPU register from memory using direct addressing or indexed addressing via a CPU register.
	LOAD	CDST = (CSRC)	2						[CDST]←[[CSRC]] Load CPU register from memory using implied addressing via a CPU register.
	STOR	LABEL <(CDST <,FD >)> = CSRC	4						[LABEL + [CDST(FD)]]←[CSRC] Store CPU register to memory using direct addressing or indexed addressing via a CPU register.
	STOR	(CDST) = CSRC	2						[[CDST]]←[CSRC] Store CPU register to memory using implied addressing via a CPU register.
I/O AND IMMEDIATE	LDWI	CDST = WORD	4						[CDST]← WORD Load immediate 16 bits to CPU register.
	LDBI	REG = BYTE	2	?	?	X	X	X	[REG <7,0 >]← BYTE Load immediate 8 bits to CPU or external register.
IMMEDIATE OPERATE	ADDI	CDST,BYTE	2	X	X	X	X	X	[CDST]←[CDST] + BYTE Add immediate 8 bits to lower half of CPU register.
	SUBI	CDST,BYTE	2	X	X	X	X	X	[CDST]←[CDST] - BYTE Subtract immediate 8 bits from lower half of CPU register.
	CMPRI	CREG,BYTE	2	X	X	X	X	X	[CREG] - BYTE Compare immediate 8 bits with lower half of CPU register. Only the statuses are affected.
I/O, JUMP, AND BRANCH ON CONDITION	BR	REG <,FD >	2						[PC]←[REG(FD)] Branch to memory location addressed by register contents or by some operation on the register's contents.
	IBR	LABEL <(REG <,FD >)>	4						[PC]←[LABEL + [REG(FD)]] Branch using direct addressing or indexed addressing via any CPU or external register.

Table 20-1. A Summary Of The MC2 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES					OPERATION PERFORMED
				O	C	N	Z	L	
I/O, JUMP, AND BRANCH ON CONDITION (CONTINUED)	CALL	LABEL <,I> <IF CC>	2	?	?	?	?	?	If CC is true then [[SP]]←[R0] [SP]←[SP] + 1 [R0]←[PC] [PC]←[PC] <15,8>LABEL <7,0> or [PC]←[[PC] <15,8>LABEL <7,0>] if I is specified Subroutine call — may be conditional or unconditional. If condition is not satisfied, Program Counter is incremented and next instruction is executed. If condition is satisfied, statuses are saved on the stack, the incremented Program Counter is saved in Register 0, and the lower 8 bits of the Program Counter are replaced by the second byte of the instruction. Subroutine starting location must be within 256 words of CALL instruction location.
	RTN	CREG	2						[PC]←[CREG] [CREG]←[[SP] - 1] [SP]←[SP] - 1 Subroutine return — get return address from specified CPU register and pop the stack into that register.
	CBR	LABEL <,I> <IF CC>	2						If CC is true then [PC]←[PC] <15,8>LABEL <7,0> or [PC]←[[PC] <15,8>LABEL <7,0>] if I is specified Conditional branch — if condition is satisfied, then replace lower 8 bits of Program Counter with lower 8 bits of instruction. Branch destination must be within 256 words of CBR instruction.
I/O AND REGISTER-REGISTER MOVE	MOVE	DST = SRC <,FD>	2	?	?	X	X	X	[DST]←[SRC(FD)] Move data from register to register, optionally operating on source word.
	STRB	CDST							[CDST]←[DI] Move contents of Device Identification register into specified CPU register.
	LDRB	CSRC <,FD>	2						[DI]←[CSRC(FD)] Load Device Identification register with contents of a CPU register, or with some function of those contents.

Table 20-1. A Summary Of The MC2 Instruction Set (Continued)

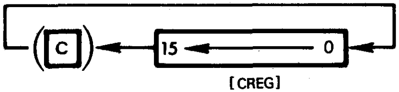
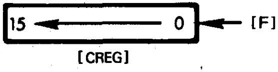
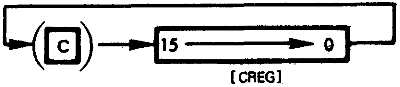
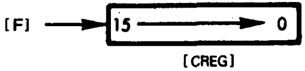
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES					OPERATION PERFORMED
				O	C	N	Z	L	
I/O AND REGISTER-REGISTER OPERATE	ADD	CDST,SRC<,FD>	2	X	X	X	X	X	[CDST]←[SRC(FD)] + [CDST] Add contents of CPU register and any register; deposit result in CPU register.
	SUBR	CDST,SRC<,FD>	2	X	X	X	X	X	[CDST]←[SRC(FD)] - [CDST] Subtract contents of CPU register from any register's contents; deposit result in CPU register.
	AND	CDST,SRC<,FD>	2	?	?	X	X	X	[CDST]←[SRC(FD)] Λ [CDST] AND CPU register contents with any register's contents; deposit result in CPU register.
	OR	CDST,SRC<,FD>	2	?	?	X	X	X	[CDST]←[SRC(FD)] V [CDST] OR CPU register contents with any register's contents; deposit result in CPU register.
	XOR	CDST,SRC<,FD>	2	?	?	X	X	X	[CDST]←[SRC(FD)] ⊻ [CDST] Exclusive-OR CPU register contents with any register's contents; deposit result in CPU register.
	CMPR	CDST,SRC<,FD>	2	X	X	X	X	X	[SRC(FD)] - [CDST] Compare contents of CPU register with those of any register. Only the statuses are affected.
REGISTER OPERATE	ADDC	CREG	2	X	X	X	X	X	[CREG]←[CREG] + [C] Add Carry bit to contents of CPU register.
	NEG	CREG	2	X	X	X	X	X	[CREG]←0 - [CREG] Negate contents of CPU register (twos complement).
	CMPL	CREG	2	?	?	X	X	X	[CREG]←[$\overline{CREG}$] Complement contents of CPU register (ones complement).
	SHFTL	RRL,CREG<,C>	2	X	X	X	X	X	 Rotate CPU register contents left one bit position, through Carry if specified.
	SHFTL	LSL,CREG,F	2	X	X	X	X	X	 Shift CPU register contents left one bit position, filling bit 0 according to F.

Table 20-1. A Summary Of The MC2 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES					OPERATION PERFORMED
				O	C	N	Z	L	
REGISTER OPERATE (CONTINUED)	SHFTR	RRR,CREG<,C>	2	?	X	X	X	X	 Rotate CPU register contents right one bit position, through Carry if specified.
	SHFTR	LSR,CREG,F	2	?	X	X	X	X	 Shift CPU register contents right one bit position, filling bit 15 according to F.
I/O AND BIT MANIPULATION	SBIT	CDST,SRC(K)	2	?	?	X	X	X	[SRC<K>]←1 [CDST]←[SRC] Set the specified bit of the specified register to 1, then deposit result in a CPU register.
	RBIT	CDST,SRC(K)	2	?	?	X	X	X	[SRC<K>]←0 [CDST]←[SRC] Reset the specified bit of the specified register to 0, then deposit result in a CPU register.
	CBIT	CDST,SRC(K)	2	?	?	X	X	X	[SRC<K>]←$\overline{\text{[SRC<K>]}}$ [CDST]←[SRC] Complement the specified bit of the specified register, then deposit result in a CPU register.
	TBIT	CDST,SRC(K)	2	?	?	X	X	X	[CDST]←[SRC] Λ 2**K Set all bits of the specified register, except the specified bit, to 0; deposit result in a CPU register.
STACK	PUSH	CREG	2						[[SP]]←[CREG] [SP]←[SP]+1 Store CPU register's contents on top of stack.
	POP	CREG	2						[CREG]←[[SP] - 1]; [SP]←[SP] - 1 Load CPU register from top of stack.
	HALT		2						CPU enters idle state.

minicomputer buyer will be interested in buying a great deal of support in addition to hardware and will not be quite so influenced by small dollar differences going from one product to another.

It is in the area of discrete logic replacement that we may expect to see the greatest volatility among microcomputer manufacturers. The microcomputer user in this market will usually be buying in huge volumes with very little front end programming expense; therefore, this user has a much greater incentive to switch from one microcomputer to another, based solely on pricing considerations. This being the case, the logic device replacement market is the one which will be hardest for established microcomputer manufacturers to defend, and the most attractive to latecomers into the field.

It is quite probable that a microcomputer manufacturer who has not established a market for minicomputer-like devices within the next two or three years will have no further opportunity to do so, however interesting the products he designs. No such window exists in the logic replacement market, where ten years from now a manufacturer who is able to sell microcomputer devices for 10¢ each, where the going rate has been 25¢ each, will be able to establish himself.

In conclusion, we predict that microcomputer devices will separate into minicomputer look-alikes and logic device replacements. The minicomputer look-alike market will become increasingly harder to break into and will stabilize fairly quickly. The logic device replacement market will continue to spawn products that look nothing like minicomputers and will continue to be extremely volatile until prices have been driven so low that there is simply no room left for further economies.

(We have not changed a word of this prediction from the first edition).

Chapter 21
SELECTING A MICROCOMPUTER

So you have a product and you may want to build it using microcomputer devices. You have two decisions to make:

1) **Should you use a microcomputer at all?**
2) **And if so, which?**

Of course, both decisions must be based on minimizing costs.

The eventual unit price for any product, whether or not it includes a microcomputer, is given by the equation:

$$P = \frac{F}{N} + V$$

In the above equation, P represents unit price, F represents fixed costs, V represents variable costs and N represents the number of units you plan to build.

FIXED COST CONTRIBUTING FACTORS

Fixed costs are the front-end expense which is essentially insensitive to the number of units you plan to build. Fixed costs include the following items:

1) Product evaluation expense, including preliminary market research.
2) Product advertising and promotion expense.
3) The cost of doing a competitive analysis to select a microcomputer.
4) The cost of going from specification to product.

VARIABLE COST CONTRIBUTING FACTORS

Variable costs are the costs that must be incurred for every unit built. These are the contributors to variable costs:

1) The cost of logic components and particularly, whether you have access to second sources for all logic components. A product without a second source may be a product that becomes significantly more expensive as time goes by.
2) Assembly line expenses.
3) Product testing expenses.

While you are still deciding whether to use microcomputer logic or discrete logic, two further considerations must be taken into account:

1) **Subsequent product modification**
2) **After sales servicing**

CONTINUING ENGINEERING COSTS

Remember, if your product is built around a microcomputer you can make very drastic changes to the product simply by rewriting the microcomputer program. That will result in a single ROM or PROM device having to be replaced. Were the product completely implemented using discrete logic, a similar product change may require one or more boards of logic to be completely replaced.

AFTER SALES SERVICE

The cost of servicing a product built around a microcomputer is significantly less than the cost of servicing a product that uses discrete logic. There are two reasons why this is so.

First, a product that is built around a microcomputer is likely to have far fewer components than the same product implemented entirely out of discrete logic. This means that not only are there fewer parts to malfunction, but when a part does malfunction, it is easier to locate.

The second reason that servicing a microcomputer-based product is cheaper than servicing the same product implemented in discrete logic is that you can write a diagnostic program to test every logic device on a card. Suppose there are 200 logic devices on a large card that includes a microcomputer system. Give each device a number, and place eight light-emitting diodes at the card edge. Then write a program which systematically tests every single device on the card to ensure that it is functioning correctly. Any malfunctioning device may be identified using the eight light-emitting diodes to display a binary device number. There are, of course, numerous applications where this simplistic approach to diagnostics will not work, but in many applications the concept has potential.

In order to determine whether you should use a microcomputer at all, you must estimate costs, based on fixed and variable expenses, for a product built around a microcomputer, and for a product built entirely out of discrete logic. You must then consider continuing engineering and after sales service economies that may accrue when you build your product around a microcomputer.

Assuming that you are going to use a microcomputer, which should you use? Let us examine the impact that microcomputer selection has on fixed and variable costs. First consider variable costs.

VARIABLE COSTS

It may not be immediately apparent, but a microcomputer's instruction set and execution speed will usually have very little impact on variable costs, being overwhelmed by simple pricing considerations. For example, the F8 has an instruction set that will invariably generate longer object programs than an equivalent 8080A system. On the other hand, by combining the 3850 CPU with a 3851 PSU, you have a two-device system which includes a CPU, 1024 bytes of ROM, 64 bytes of RAM, four I/O ports (each of which are 8 bits wide), a programmable timer and a single external interrupt line. Providing your application is simple enough to fit into this small configuration, the fact that the 8080A instruction set is superior, or that 8080A programs execute faster, becomes irrelevant. If your F8 program fits in the minimum 1K bytes of memory, memory savings become irrelevant, and it would take seven 8080A devices to give you the same functional capacity as the two F8 devices. Clearly, the seven 8080A devices will cost considerably more.

If the F8-based product provides lower parts costs (Variable Costs), but the 8080A-based product costs less to develop (Fixed Costs), at what point will fixed costs become more important than variable costs?

FIXED COSTS

Let us answer the question by looking more closely at factors that contribute to fixed costs.

Of the four cost factors that contribute to fixed costs, only the fourth, the cost of going from specification to product, can be critically evaluated.

We will begin by summarizing, without comment, the steps that lead from a specification to an end product. Using this sequence of events as a framework, we will describe the decisions you must make, and the basis on which you should make these decisions.

DESIGNING LOGIC WITH MICROCOMPUTERS — A SEQUENCE OF EVENTS

Any microcomputer development project will involve some minor variations of these nine steps:

1) **Specify the product.** The need for clear and accurate specifications is more critical when a product is to be built around a microcomputer than it would be if the product were to be built around discrete logic.

   Remember that designing with discrete logic involves a single set of choices — the selection of discrete logic components. When you use a microcomputer, there are two sets of choices: having decided on the CPU and the devices that will surround the CPU, you still have to create a program, which means that enormous flexibilities and variables remain unresolved even after the hardware has been selected. In other words, you are going to be faced with constant hardware/software tradeoffs. Unless an excellent specification defines the product which is to be built, the process of compromising between hardware and software will be difficult at best, and will result in unforeseen errors at worst.

2) **Prepare a preliminary hardware design.** Even before you have selected a microcomputer, you will lay out a preliminary product design, leaving as "black boxes" those parts of the product which will eventually become the microcomputer system.

3) **Specify the microcomputer requirements.** The "black boxes" from step 2 can now be expanded into a set of performance criteria upon which the selection of a microcomputer will be based. Some iteration between steps 2 and 3 may be required.

If the first time you perform step 3, you discover that no microcomputer on the market can meet your performance criteria, redo step 2; relax the demands placed on the microcomputer by identifying the critical steps that the microcomputers cannot handle, then implement these critical steps using discrete logic. In other words, shrink the "black box".

If you find that virtually every microcomputer provides overkill for your job as specified, it is worth going back to step 2 to see if some additional logic functions can be performed by the microcomputer. In other words, expand the "black box".

A frequently made error, when specifying the logic that must be implemented by a microcomputer, is to needlessly include steps that demand extremely fast logic. Remember, none of the microcomputers described in this book are capable of performing real-time events in much less than ten microseconds.

Erring in the other direction, another common mistake is to underestimate what a microcomputer can do. A microcomputer can handle large volumes of data, and can perform complex data manipulations, providing program execution speed is unimportant.

4) **Depending on your history as a microcomputer user, it may or may not be worthwhile doing a competitive analysis of microcomputer products. If it is worthwhile, you will, at this point, narrow the field to two or three products.**

5) **Write source programs. You must now make a major decision: do you write source programs in assembly language or in a higher level language?**

   Most microcomputer manufacturers now allow you to write source programs in FORTRAN. Intel's new programming language, PL/M, is also being adopted by a number of microcomputer manufacturers. In all probability, a growing number of higher level languages will be made available to microcomputer users. As we will discuss later in this chapter, however, using assembly language is frequently less expensive.

6) **Convert the source program to an object program.** This step may be handled in one of two ways: you may use a time-sharing computer service or you may use a microcomputer development system.

   A microcomputer development system looks like a minicomputer system, but is built around the microcomputer of your choice.

   Until you have made a final microcomputer selection, you will likely use a time-sharing service to convert your source programs to object program format. Eventually you are going to need access to a microcomputer development system (for step 7); therefore, it makes sense to get off the time-sharing service and to start using a microcomputer development system as soon as you have made your final microcomputer selection.

7) **Convert the object program into Programmable Read Only Memory devices (PROMs).** You will do this using the microcomputer development system that supports the microcomputer of your final choice.

8) **Build a prototype of your product.** Now is the time to ensure that all conceivable errors have been detected and corrected, both in the programs driving the microcomputer and in the logic supporting it. Correcting programming errors and logic design errors will require constant iteration between development steps, perhaps as far back as step 2.

9) **Create a ROM mask.** Unless your product is a low volume item, or is still being developed, economics dictate that you stop using PROMs and start using ROMs.

   When you are certain that all your programs are correct, you will define ROM masks for your read only memory devices. ROMs will likely be created for you by the microcomputer manufacturer.

   Programs that drive your microcomputer now become nothing more than logic devices, and will be handled routinely on the production line, like any other logic device.

Within the framework of these nine steps, we are now in a position to explain how you go about estimating product development costs.

The most important factor determining microcomputer-based product development costs is the type of assistance you receive, either from the microcomputer manufacturer, or from an alternative source. Product development assistance can be divided into development hardware and system software.

Development hardware consists of a minicomputer-like device which you will use to implement some or all of steps 6 through 9. System software consists of programs that make the hardware usable.

We will describe microcomputer development hardware first, and system software next.

MICROCOMPUTER DEVELOPMENT HARDWARE

At the center of any hardware development system, there will be a box that looks like a minicomputer.

In its simplest form, this box closely parallels a minicomputer. Its Central Processing Unit is a microcomputer, which is surrounded by read/write memory, I/O interface, and logic to support the various options available with the microcomputer. All this is packaged in a minicomputer-like box, with a power supply and a front console. This "micro-minicomputer" will have minicomputer-style peripherals, including an input device, an output device and bulk storage devices.

SIMPLE MICROCOMPUTER DEVELOPMENT SYSTEMS

A very simple micro-minicomputer system will consist of the microcomputer box and a teletype. The teletype keyboard is the input device, the teletype printer is the output device, and the teletype paper tape reader/punch is the bulk storage device.

Source programs and any other human readable documentation will be printed by the teletype printer.

The source program you enter and the object program which the computer creates will both be output by the teletype paper tape punch. Subsequently, these paper tapes may be input via the teletype paper tape reader.

The first enhancement of this very simple hardware development system will be to stop using the teletype paper tape reader/punch as the bulk storage device, replacing it with a tape cartridge or floppy disk system, which is much faster and easier to handle.

The next enhancements will be to replace the teletype keyboard with a CRT terminal, and the teletype printer with a line printer.

At this point your microcomputer development system looks remarkably like a small minicomputer system, and you will use it, just as you would use a minicomputer system, to create source programs, and to convert source programs into object programs.

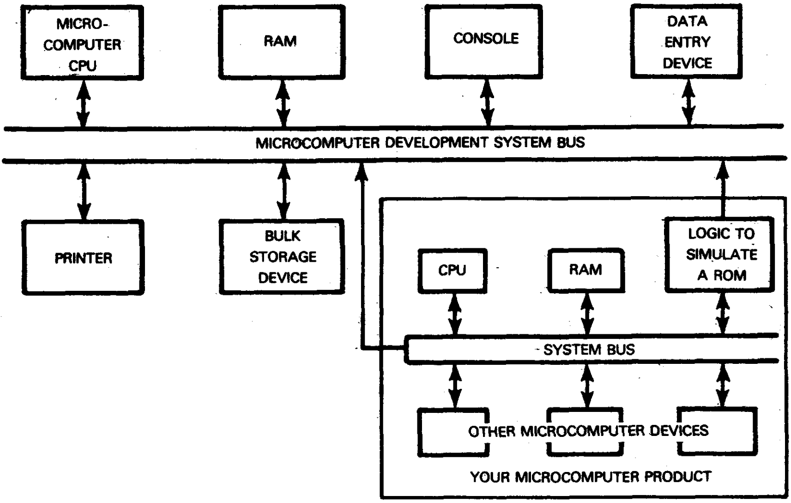
However, your microcomputer development system will have one feature which no minicomputer ever had: on the console of the microcomputer box there will be a plug, into which you can insert unused Programmable Read Only Memory devices. The development system will give you the ability to write any part of your object program into a PROM, via the console plug. You may take the PROM, plug it into a prototype board, and test the prototype product in the traditional way.

Every microcomputer manufacturer provides a straightforward microcomputer development system, as described above. The oldest and most popular microcomputers, such as the Intel 8080, now have more sophisticated development systems available. These more sophisticated development systems are produced not only by the microcomputer manufacturer, but by a number of independent companies who are rapidly entering the microcomputer development products business.

SIMULATING MICROCOMPUTER DEVELOPMENT SYSTEMS

The first enhancement of the straightforward microcomputer development product, as described above, is a product that allows you to include a hardware simulation of the logic you are developing, within the microcomputer

development system. Conceptually, such a system may be illustrated as follows:



Since there is no established term to describe microcomputer development systems as illustrated above, we will call it a "Simulating Microcomputer Development System". In reality, the only parts of your system that will indeed be simulated are read only memory, interrupts, direct memory access and I/O.

If read only memory can be accurately simulated within the development system, then you will be able to bypass the Programmable Read Only Memory creation step, at least until you are certain (to the extent that you can be certain) that your programs are error-free.

By allowing the product you are developing to be handled as though it is an external device, the microcomputer development system serves the double purpose of allowing you to create object programs and, at the same time, of allowing you to check that the object programs, together with your external digital logic, perform as required. In theory, the microcomputer development system can now take you right up to the point where you can define your ROMs and organize a production line.

Another development system enhancement that is appearing with greater frequency is the system that can handle more than one microcomputer. Intel, for example, sells not only the Intel 8080A, which is described in this book, but also the Intel 8008, two 4-bit microprocessors and the 3000 Series chip slice. You can use Intel's ICE microcomputer development system to develop logic around any of the microcomputers sold by Intel.

Independent manufacturers of microcomputer development products are attracted to the idea of a microcomputer development system that can be used with more than one microcomputer, since this gives them the flexibility of selling into more than one manufacturer's market.

MICROCOMPUTER SYSTEM SOFTWARE

Neither a time-sharing computer service nor a microcomputer development system is of any value without programs that give you access to the capabilities of the system. We refer to these programs collectively as system software.

We have described in Volume I, Chapter 6 how a program must first be written in a programming language using pencil and paper. The program is then converted into a sequence of binary digits, stored in computer memory. Microcomputer systems demand an additional step, that is, the creation of a read only memory device, within which the object program is implemented.

EDITOR

Figure 21-1 illustrates the components of system software which are routinely found in microcomputer systems. The Editor, Assembler and Compiler have already been described in Volume I, Chapter 6. Referring to Figure 21-1, step 1 shows how an Editor program is loaded into computer memory and is used to create a source program, which is then stored in a computer-readable form on paper tape, magnetic cartridge or disk.

MONITOR

The Monitor is a small resident program that simply lets you identify and load individual system software modules.

ASSEMBLER

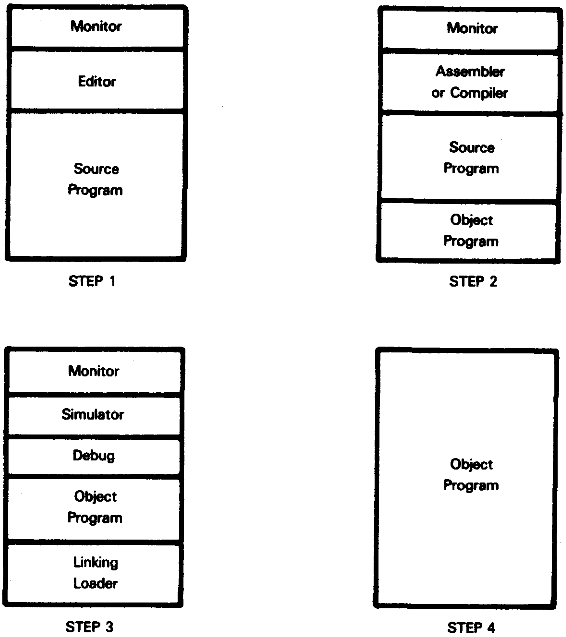
In Figure 21-1, step 2, the source program is either assembled or compiled, depending upon whether the source program was written in assembly language or a higher level language. An object program is created.

A number of aspects of source and object program creation are not self-evident. The first and most obvious question to ask is whether the amount of memory available in the microcomputer development system for source and object programs will be sufficient. In Figure 21-1, step 2, memory is illustrated holding, at one time, source programs, object programs, an Assembler or Compiler, and a Monitor. What if the source program and object program are simply too big to fit into memory as illustrated?

There is another potential problem; the object program developed in step 2 is almost certain to contain errors, and it is not unreasonable for a source program to be corrected and re-assembled ten, twenty or more times before an error-free object program results. **How long will it take to load the Editor for step 1, then reload the Assembler for step 2?**

EDITOR/
ASSEMBLER
COMBINED

Let us first consider those problems associated with the need to constantly edit and re-assemble a source program, while detecting and correcting program errors. Are there any hidden problems to watch out for in this process?

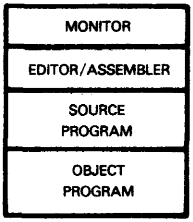


Boxes represent microcomputer memory space.

Figure 21-1. System Software Modules

If your development system uses magnetic tape cassettes or floppy disks as the bulk storage device, you will have no problems; it will just take a few seconds to load either the Assembler or Editor into memory; therefore, an inconsequential amount of time will be wasted shuttling between steps 1 and 2 of Figure 21-1.

On the other hand, **if you are working with a very low budget and your development system uses the teletype paper tape reader/punch as the storage medium for all programs, you could be faced with a very severe problem;** it could take as much as half an hour simply to load the Editor and Assembler into memory. This being the case, you will waste a very substantial amount of time and money watching the teletype paper tape reader monotonously load and punch paper tapes. **Some microcomputer manufacturers get around this problem by combining an Editor and Assembler into one program.** By breaking up your application into sufficiently small modules, you can generate a single memory load as follows:

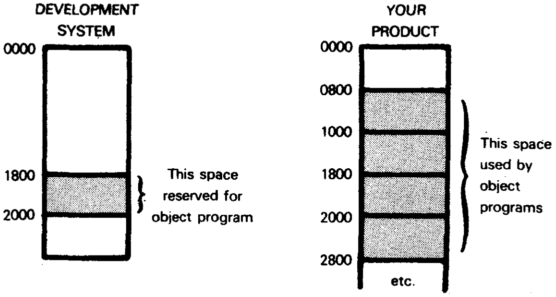


Now you do not need to waste time reloading the Editor, and then the Assembler, every time you wish to make a correction to your source program.

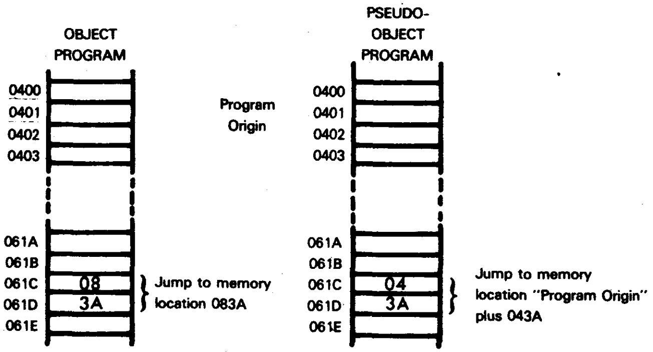
RELOCATABLE OBJECT PROGRAMS

Let us first describe how you go about developing programs which are too big to constitute a single memory load, as illustrated in step 2 of Figure 21-1. The solution is self-evident: create the program in pieces. Implementation of this solution is not quite so straightforward.

If the program is to be developed in pieces, then clearly the pieces will each occupy different areas of memory. On the other hand, one specific area of memory may be assigned to object programs by the Assembler. This is the situation which arises:


DEVELOPMENT SYSTEM
0000
1800
2000
This space reserved for object program
YOUR PRODUCT
0000
0800
1000
1800
2000
2800
etc.
This space used by object programs

The necessary solution is to create object programs which are one step removed from being truly executable. In these "pseudo-object programs", every object program byte that encodes an absolute memory address will instead encode a displacement from the beginning of the object program. This may be illustrated as follows:



RELOCATABLE LOADER

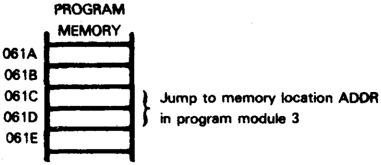
This pseudo-object program will be loaded into memory by a system software program referred to as a "Relocatable Loader"; the Relocatable Loader acquires its name from the fact that it can relocate the pseudo-object program anywhere in memory, changing all the displacement addresses to reflect a new origin.

RELOCATING ASSEMBLER

An Assembler which is able to create pseudo-object programs as described above is called a Relocating Assembler.

PROGRAM LINKING

If programs have been written in pieces and the pieces must be loaded into memory to form a unit, then it is quite possible that a memory reference instruction in one piece of program may reference a label in another piece of program:



LINKING LOADER

A loader that can relocate program modules and, in addition, link memory references from one module to another is called a "Linking Loader".

A Linking Loader works in conjunction with an Assembler that generates linkable object program modules.

LABEL TABLE

A Relocating Assembler will replace every absolute address in the object program with a code which represents a label number. Then, at the end of the object program, **the Assembler will generate a Label Table identifying every label number** as representing a specific object program byte in a defined object program module.

When the Linking Loader loads object program modules, it will identify the real memory address into which every object program byte which owns a label actually gets loaded. Now the Linking Loader can replace label numbers in object programs with the actual memory address that happens to correspond to the label number. For example, the Jump instruction illustrated above may get encoded by the Assembler as three bytes which say:

Jump to label number 4 in program module number 3.

At the end of the object program, the Assembler will, in some coded fashion, identify label number 4 as corresponding to byte number $32A_{16}$ of program module 3.

The Linking Loader will wait until program module number 3 has been loaded into memory, at which time it can determine the exact memory address for byte number $32A_{16}$ of program module 3. This memory address is equal to the origin of program module 3, plus $32A_{16}$. This becomes the address which the Linking Loader inserts into the Jump instruction.

The only thing that is important to you, as a microcomputer programmer, is to realize that, given a Relocating Assembler and a Linking Loader, you can write programs in small modules and not have to worry about changing object code depending upon where each module resides in memory.

It is very important to ascertain whether a microcomputer development system offers Relocating Assemblers and Linking Loaders, because they will make the task of developing object programs much simpler.

DEBUG

Once an object program has been created, it must be executed in order to check it for errors. Another system software module, referred to as a Debug program, will always be required at this point. The Debug program allows you to conditionally execute your object code, stopping at will to examine the contents of memory or programmable registers, or to temporarily make changes to the object program as a means of determining what went wrong.

While you are debugging your object programs, there are certain parts of your system which do not exist and whose presence must therefore be simulated. If you have a Simulating Microcomputer Development System, then the Simulator program only has to simulate interrupts, direct memory access, and external devices communicating through I/O ports.

If you have a simple microcomputer development system, then it must have a Simulator capable of representing the entire environment beyond your microcomputer.

UTILITIES

There is one further set of software modules which is extremely important in the world of minicomputers, but less important in the world of microcomputers; these are Utility and Input/Output routines.

SUBROUTINE LIBRARY

There are a number of programming procedures which virtually every microcomputer application is going to encounter; these include routines to move data around memory, to transfer data between memory and external devices, or to perform arithmetic operations. In the world of minicomputers, such programs are bundled up as a package so that a minicomputer programmer never has to write programs to perform such basic operations. Instead, a minicomputer program will simply call subroutines out of a subroutine library in order to perform standard operations.

Is the same idea feasible in the world of microcomputers? Unfortunately, not always.

The concept of an I/O subroutine library is doubtful, since from one application to the next, you cannot even be sure that I/O will be implemented in the same way, let alone that external devices will be similar enough to allow any form of general purpose program to control input/output operations. Remember that we are no longer dealing with a CPU that interfaces with standard peripheral devices, such as disk, line printer, card reader etc; we are dealing with a microcomputer that is connected to various and sundry discrete logic systems.

Even such routine operations as multibyte arithmetic frequently cannot be standardized. One microcomputer system may have a total of 512 bytes of memory; another may have 4096 bytes of memory. In each case, saving bytes will be extremely important. Any type of generalized program will be unacceptable if generality is bought at the price of extra memory bytes. An application that will never require more than 16 binary digit numbers cannot efficiently use a multibyte addition subroutine which has been written to handle multibyte numbers of indefinite length. The fact that someone else has already written that very general purpose multibyte addition program will not prevent you from rewriting your own addition program to serve your very limited needs — and nothing more. Your highly specialized addition program may only require half as much memory and in a product that may be reproduced thousands of times.

A microcomputer program written making liberal use of subroutines out of a library may well finish up using twice as much memory as a program written to meet the immediate needs of a single application. Suppose writing your own program allows you to reduce program memory from 2K bytes to 1K bytes of ROM. **Realistically, your programming expenses may be increased $3,000 or $4,000 because you did not use an existing subroutine library (presuming that such a library exists). However, your product does not have to have a very large volume before the extra programming expense becomes trivial compared to the money spent on extra memory devices, larger PC cards, more power supply and higher assembly expenses.**

The very same argument determines whether you will write your source programs in assembly language or in a higher level language. A higher level language will result in object programs that are anywhere from 1.4 to 2 times as long as the object program would have been had the source program been written in assembly language.

On the other hand, it will probably take twice as long to develop programs in assembly language as it would to develop the same programs in a higher level language. You may have to deduct from the time saved, time your programmers spend learning a new language. In any event, it is clear that for very low volume systems, programming in a higher level language has always got to be more economical. In high volume systems, programming in assembly language has always got to be more economical and, depending upon individual circumstances, it becomes a tossup at some intermediate level.

AN ECONOMIC EXAMPLE

We will now give substance to the discussion of microcomputer development economics by looking at some hypothetical but realistic numbers. Table 21-1 lists possible numbers for three different microcomputers. If we assume that fixed costs consist of programming expense and product development expense only, while variable costs consist of CPU and support device costs only, then Table 21-2 shows how unit costs will vary as a function of product volume.

Observe from Table 21-2 that **at very low volume, higher language program development is less expensive. If you are building more than a thousand units, on the other hand, in almost every case it will be cheaper to use assembly language programming.**

Costs associated with products A, B and C have been purposely skewed to demonstrate the impact of fixed and variable costs. Notice that product C, having lower fixed costs, generates the smallest unit price at low volume even though the cost of the microcomputer devices themselves is high.

The problem with Table 21-2 is that it oversimplifies the factors which influence eventual unit price. You should look at Table 21-2 as an illustration of general price versus volume relationships and nothing more.

Table 21-1. Some Typical Microcomputer Based Product And Development Costs

SOURCE OF EXPENSE	MICROCOMPUTER SELECTED		
	PRODUCT A	PRODUCT B	PRODUCT C
Microcomputer CPU, plus support devices and logic ($/unit)	63	78	91
Cost of extra memory if programs are written in higher level language ($/unit)	10	8	3
Cost of writing programs ($ total):			
a) In assembly language	8000	7500	6500
b) In higher level language	5500	5000	3000
Cost of developing prototype ($ total)	42000	40000	40000

Table 21-2. Unit Prices For Microcomputer Based Products

VOLUME	UNIT PRICE ($)					
	PRODUCT A ASSEMBLY	PRODUCT A HIGHER	PRODUCT B ASSEMBLY	PRODUCT B HIGHER	PRODUCT C ASSEMBLY	PRODUCT C HIGHER
100	563.00	548.00	553.00	536.00	556.00	524.00
500	163.00	168.00	173.00	176.00	184.00	180.00
1000	113.00	120.50	125.30	131.00	137.50	137.00
5000	73.00	82.50	87.50	95.00	100.30	102.60
10000	68.00	77.75	82.75	90.95	95.65	98.30

Assembly = Assembly language programming
Higher = Higher level language programming

A LOOK AT THE FUTURE

Let us take a moment to gaze in a crystal ball.

What types of microcomputer product can we expect to see in the future, and what impact will they have on the minicomputer market?

If there is one key aspect of microcomputer design which was not immediately apparent but is becoming more apparent every day, it is that the way logic is distributed among various devices of a microcomputer chip set is fundamentally the most important feature of any microcomputer product. Assembly language instruction sets, addressing modes and even instruction execution times are all of secondary importance in that they become inconsequential providing they meet modest criteria of sufficiency. The logic designer using microcomputers is likely to be far more influenced by control signals on the system bus and by the number of devices he has to work with, rather than by the complexity of the instruction set or its addressing modes. And this we believe is the key to a future drift into two types of microcomputer product: the logic device and the computer.

If there will be a branch of the microcomputer industry which builds minicomputer look-alikes, what impact will this have on the microcomputer industry? In truth, most manufacturers of computers, mini or larger, are already scrambling to build their central processing units and support logic out of large scale integration devices; therefore, we may conclude that within ten years every computer will be a microcomputer in that every computer will be built out of large scale integrated logic. This does not mean that the microcomputer manufacturers of today will overwhelm the minicomputer and large computer manufacturers of yesterday. This is because programming expenses constitute an already expended front end fixed cost for most users of minicomputers and larger computers; the hardware savings that might be gained by switching from a minicomputer to a microcomputer are simply insignificant when compared to reprogramming expenses.

Therefore, those minicomputer manufacturers who can defend their current sales with existing software are likely to be impacted very little by microcomputers. Those minicomputer manufacturers who are essentially selling components are likely to be eliminated from the component market entirely, unless they are able to scale down their minicomputers into microcomputers and survive as component suppliers at the new microcomputer price levels. It is this reduction in prices that opens a window for new products such as the National Semiconductor and Signetics microcomputers to attack markets that look characteristically like minicomputer markets. These are markets which were suited to minicomputer-type products, but in the past could not use minicomputers because of pricing considerations. Now that minicomputer-like devices are available for a few hundred dollars, a large number of new markets open up, none of which have used minicomputers before and none of which have invested in the front end program development fixed costs; the new markets are therefore equally likely candidates for the old minicomputer manufacturer's product or the new microcomputer manufacturer's product. This pseudo-minicomputer buyer will be interested in buying a great deal of support in addition to hardware and will not be quite so influenced by small dollar differences going from one product to another.

It is in the area of discrete logic replacement that we may expect to see the greatest volatility among microcomputer manufacturers. The microcomputer user in this market will usually be buying in huge volumes with very little front end programming expense; therefore, this user has a much greater incentive to switch from one microcomputer to another, based solely on pricing considerations. This being the case, the logic device replacement market is the one which will be hardest for established microcomputer manufacturers to defend, and the most attractive to latecomers into the field.

It is quite probable that a microcomputer manufacturer who has not established a market for minicomputer-like devices within the next two or three years will have no further opportunity to do so, however interesting the products he designs. No such window exists in the logic replacement market, where ten years from now a manufacturer who is able to sell microcomputer devices for 10¢ each, where the going rate has been 25¢ each, will be able to establish himself.

In conclusion, we predict that microcomputer devices will separate into minicomputer look-alike and logic device replacements. The minicomputer look-alike market will become increasingly harder to break into and will stabilize fairly quickly. The logic device replacement market will continue to spawn products that look nothing like minicomputers and will continue to be extremely volatile until prices have been driven so low that there is simply no room left for further economies.

(We have not changed a word of this prediction from the first edition.)

AN
INTRODUCTION
TO
MICROCOMPUTERS

VOL. II
SOME REAL
PRODUCTS

OSBORNE
A.