GENERAL INSTRUMENT

PIC SERIES MICROCOMPUTER DATA MANUAL

Microelectronics Division
General Instrument Corporation

GENERAL INSTRUMENT

PIC SERIES MICROCOMPUTER DATA MANUAL

ARCHITECTURE

INSTRUCTION SET

PRODUCTION CYCLE

ROUTINES

APPLICATIONS

TABLE OF CONTENTS—PAGE 2

APRIL 1983

Table of Contents

1 INTRODUCTION

1.1 Description

The General Instrument PIC Family is a series of MOS/LSI 8-bit microcomputers manufactured to meet the requirements of the cost-competitive controller market. The PIC microcomputer contains RAM, I/O and a central processing unit, as well as customer-defined ROM on a single chip. The 8-bit input/output registers provide latched lines for interfacing to a limitless variety of applications including:

- ☐ Industrial timing
- ☐ Radio/TV Tuning
- ☐ Consumer appliances
- ☐ Motor control
- ☐ Display control
- ☐ Repertory dialers
- ☐ Vending machines
- ☐ Security devices
- ☐ Automotive dashboard

1.2 Features

The architecture of each device in the PIC Family is based on a register file concept with a concise yet powerful instruction set designed to perform bit, byte and register transfer operations. The architectural features of the PIC family are outlined below:

PART	ROM	RAM	I/O LINES	INSTR. SPEED	INTERRUPT	PACKAGE
PIC1650A	512 x 12	32 x 8	32	4 μsec	NO	40 PIN
PIC1654	512 x 12	32 x 8	12	2 μsec	NO	18 PIN
PIC1655A	512 x 12	32 x 8	20	4 μsec	NO	28 PIN
PIC16C58	512 x 12	32 x 8	20	4.5 μsec	NO	28 PIN
PIC1656	512 x 12	32 x 8	20	4 μsec	YES	28 PIN
PIC1670	1024 x 13	64 x 8	32	2 μsec	YES	40 PIN
PIC1672	2048 x 13	64 x 8	32	2 μsec	YES	40 PIN

The PIC microcomputer was designed to be an efficient control processor as well as an arithmetic processor. It has an instruction set which allows the user to directly set, reset and to test and skip on the status of any RAM bit, including I/O lines. The "wide" instruction word (12 or 13 bits) gives the PIC Family capabilities which are not found in other 8-bit microcomputers:

- ☐ All Instructions Single Word
- ☐ All Registers Directly Addressable
- ☐ Registers Indirectly Addressable
- ☐ Set, Clear any Bit in any Register
- ☐ Test and Skip on Bit Status
- ☐ 2 Destinations for ALU Operations
- ☐ More Than 20 I/O Instructions

These added capabilities allow the user to produce compact and efficient code. In other words, many functions requiring a 1024 x 8 bit ROM may very well be programmed into a 512 x 12 bit ROM resident in the PIC1650 and at a lower cost.

1.3 Support

Hardware and software development support is provided by a wide range of products available from General Instrument. These support products include the ROMless development microcomputer, the PIC In-Circuit Emulation System (PICES II), the PIC Field Demo System (PFD), and the PIC Cross-Assembler (PICAL).

- The PIC1664 DEVELOPMENT MICROCOMPUTER is designed as a useful tool for engineering prototyping and field trial demonstration. The contents of the program counter (ROM address) and the instruction word lines (ROM data) are brought out to pins for connection to external RAM or EPROM. The addition of a HALT pin enables single-stepping of the development program.
- The PIC IN-CIRCUIT EMULATION SYSTEM allows the user to load his PIC program into RAM and test it in the actual environment of his hardware application. A powerful interactive debugging program (PICBUG) is provided for easy troubleshooting and program corrections. The PICES system is provided complete with its own enclosure and power supply for stand-alone or peripheral applications.
- The PICAL CROSS-ASSEMBLER PROGRAM converts symbolic source programs into object code for the PIC family of microcomputers. PICAL, coded in FORTRAN IV or BASIC, is intended for use on minicomputer, larger main-frame computers, and time-sharing systems.

1.4 Microcomputer Fundamentals

A microcomputer provides, on a single-chip, all of the functional elements of a minicomputer or a large main-frame computer. Basically, these functional elements include a central processing unit (CPU), program memory (ROM), data memory (RAM), and an input/output interface (I/O). It also provides the means for implementing many combinations of arithmetic and logical operations. By selecting the proper combinations of operations relevant to a particular application, a microcomputer can be used to perform logical processing, basic code conversions, formatting, and to generate fundamental timing and control signals for I/O devices.

A microcomputer is best suited for applications in which the cost of developing and manufacturing customized controller hardware would exceed the cost and/or space requirements of a general-purpose microcomputer with a specially-designed control program.

1.4.1 BASIC MICROCOMPUTER ARCHITECTURE

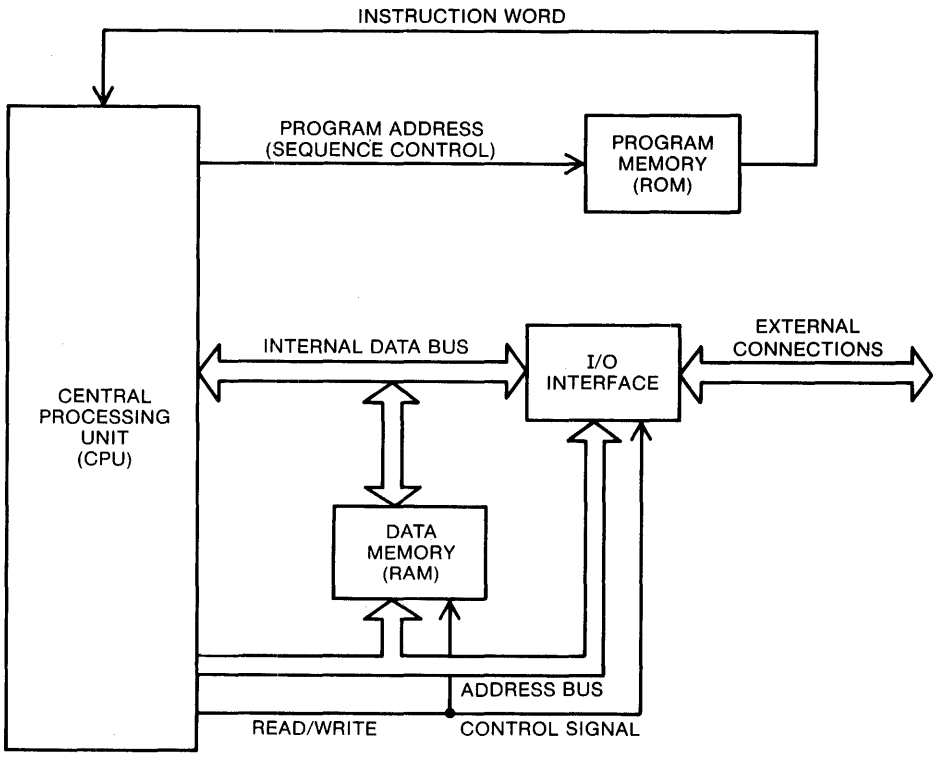
Figure 1 is a functional block diagram of a typical microcomputer, including the CPU, ROM, RAM, and I/O.

■ **Central Processing Unit.** The CPU performs the processing and control functions of the microcomputer. The CPU fetches instruction words from memory, decodes them, and generates the appropriate signals that cause the instruction to be executed. The CPU implements its various arithmetic and logical operations on operands obtained from memory. It also tests the results of arithmetic and logical operations, and as a result of these tests, chooses between alternate branches in the program.

■ **Program Memory.** The instructions for the CPU to execute are stored in a Read-Only Memory (ROM). The ROM provides permanent non-volatile storage of the program. Power interruptions or equipment shutdown will not alter the contents of the ROM.

Program memory size is defined by the number of addressable locations available for program storage and by the size of the word (number of bits) stored in each location.

Fig. 1 TYPICAL MICROCOMPUTER BLOCK DIAGRAM



For example, the notation 512 x 12 specifies that there are 512 addressable locations for program words and each word has 12 bits.

Each instruction word is selected from the program memory (ROM) by a combination of 1's and 0's on the address bus. Each unique combination of 1's and 0's addresses a unique location in program memory; the number of locations that can be addressed is therefore determined by the number of combinations of 1's and 0's available. This, of course, is a function of the number of address lines (i.e., the number of bits in the program counter).

■ **Data Memory.** The data memory provides temporary storage for data processed by the CPU. Data memory is usually a Random-Access Memory (RAM). Any data stored can be obtained from the same location (address) in which it was stored. RAM is volatile, which means that after equipment shutdown or a power interruption, valid data is no longer present. Since the information stored is usually of a temporary nature anyway, volatility is not a serious consideration. The size of data memory (RAM) is defined using the same notation described in the program memory (ROM) discussion.

The same address is used with RAM to both read data from and write data to a particular memory location. A read/write signal determines the direction of data transfer.

■ **I/O Interface.** The I/O interface permits the microcomputer to communicate with external devices. A typical interface consists of one or more I/O registers communicating with an external I/O bus or individual I/O lines. The CPU can address these I/O registers and either input data from an external device or output the result of its processing to an external device. In order for information to be transferred between the I/O ports and the external devices at appropriate times, the program logic must be written so as to anticipate significant external events.

For example, to determine if data is present, one or more bits of an input port can be tested periodically. When data from a particular input device is made available, the corresponding "flag" bit can be set by the external device. This "flag" bit could then be reset via an output port once the input data has been stored.

In some microcomputers, "interrupt" logic is incorporated in the CPU to direct the control function when an external device signals for service by activating an external interrupt line.

1.4.2 CPU FUNCTIONAL DESCRIPTION

Figure 2 is a functional block diagram of a typical CPU. The typical CPU consists of an instruction decode and control unit, an arithmetic logic unit (ALU), and several special purpose registers. These registers usually include at least an accumulator, a status register, a program counter, and a stack or stack pointer.

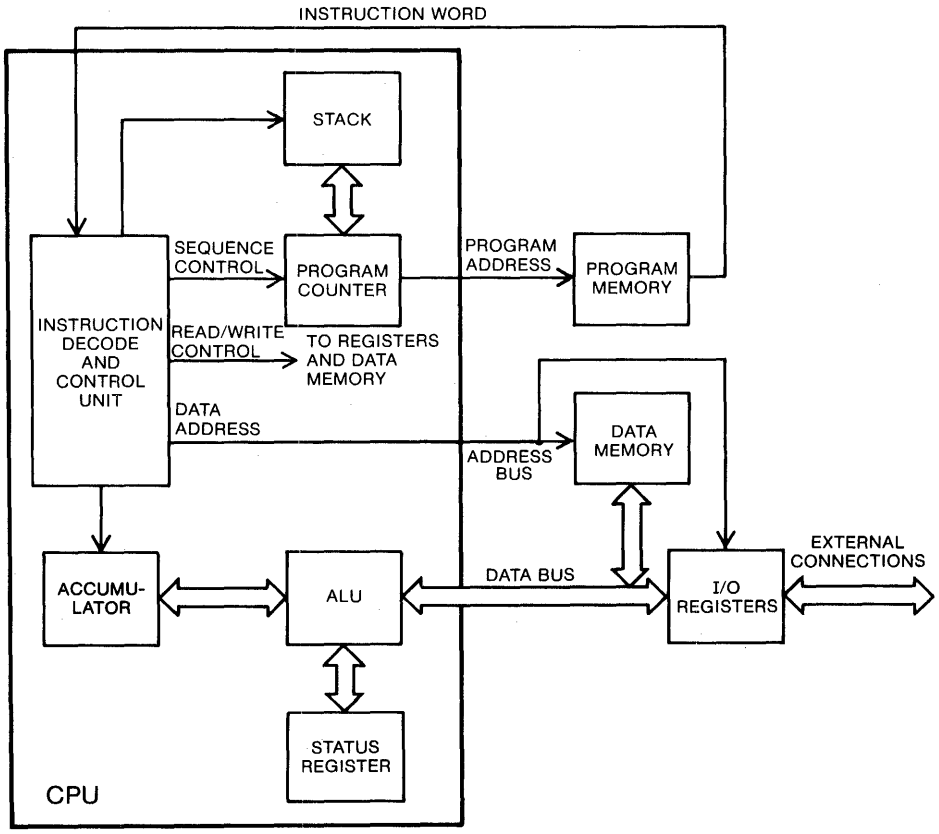
■ **Instruction Decode and Control Unit.**The instruction decode and control unit fetches an instruction word from the program memory, decodes it, and generates the appropriate signals that cause the desired operations to take place. The instruction decode and control unit also controls the program counter.

■ **Program Counter.** The program counter is a register that holds the address of the next instruction to be fetched out of program memory. Since a program is usually executed in the order in which it is written, the program counter is automatically incremented by one after the execution of every instruction, except for the following operations:

—A conditional jump instruction whose criteria have been met
—An unconditional jump instruction
—A jump to subroutine (call) instruction
—A return from subroutine instruction
—An interrupt

■ **Stack.** The stack is a group of registers used for temporary storage of program addresses required for returns from subroutines. A subroutine is a frequently used group of instructions that, for convenience and for program economy, is written once and is located in a separate part of program memory. Whenever this group of instructions is to be executed, the subroutine is called. This is accomplished by storing the contents of the program counter plus one (PC+1) in the top element of the stack and placing the starting address of the subroutine into the program counter. When the subroutine has been executed, the program must return to the next instruction following that which called the subroutine. This is accomplished by transferring the contents of the top element of the stack (PC+1) back into the program counter.

Fig. 2 TYPICAL CPU ARCHITECTURE



It is not uncommon for one subroutine to call a second subroutine, and perhaps the second subroutine to call a third subroutine, and so forth, in a process called nesting. To provide for the proper execution of nested subroutines and the subsequent return to the main program, the last subroutine, after it has executed, must return to the preceding subroutine from which it had been called. After the preceding subroutine has finished executing, it in turn must return to the subroutine from which it had been called. This sequence continues until the first subroutine has executed fully and the program counter is returned to the main program.

In order for this sequence to be implemented, there must be enough elements in the stack to accommodate each of the return addresses. As each subroutine is called, its return address is pushed onto the stack.The previous return addresses can be pushed down to accommodate each new address until the stack is filled. The stack is a LIFO (last-in, first-out) storage device. As each nested subroutine is executed, the last return address at the top of the stack is popped off and placed into the program counter. The next to last return address pops to the top of the stack ready to be transferred to the program counter when the next to last subroutine is finished executing. This process continues until the first return address is at the top of the stack and is finally transferred to the program counter.

If the CPU architecture does not provide a stack for return addresses, the programmer must allocate a block of data memory to serve as a stack. When the stack is part of memory, it must be addressed when data is to be pushed on or popped off. Therefore, a register is provided that points to the stack location in the same manner that the program counter points to the location in program memory of the next instruction word. This register is known as the stack pointer. The stack pointer points to the next stack location at which a return address will be pushed or popped.

If no stack or stack pointer is provided in the CPU and the program must be written with nested subroutines, a portion of memory may be allocated for a stack pointer and stack (software stack).

- **Arithmetic Logic Unit (ALU).** The ALU implements various binary arithmetic and logical operations utilizing one or two operands. The arithmetic operations include binary addition and subtraction. Boolean logic operations include AND, OR, Complement and Exclusive OR.

- **Accumulator.** The accumulator is a register that provides temporary storage for one of the operands to be manipulated by the ALU. The other operand is usually located in memory. The results of the operations may be stored in the accumulator.

- **Status Register.** A status register is provided to store the condition of the most recent ALU operation. Conditions such as a zero or non-zero result, carry or digit-carry will be stored. The contents of the status register can be interrogated under program control to determine the program sequence to be performed next. Depending upon the contents of the status register, a jump, skip, or subroutine call may be executed.

1.4.3 THE PROGRAM

The microcomputer has the capability of performing many different data manipulations and transactions. However, it requires a program to direct it to perform even the simplest of operations.

The program is a series of instruction words that direct internal processing functions and the transfer of data between the external devices and the CPU.

The instruction word for any CPU contains a fixed number of bits that is determined by the instruction format of the particular CPU. Some of these bits are used for an OP Code (operation). The OP Code is a description of the operation to be performed. The remaining bits following the OP Code are the operand(s) and contain either a literal, an address from which data can be obtained, or the address of the next instruction.

The complete sequence of operations required to carry out a single instruction is referred to as an instruction cycle. Each instruction cycle consists of two parts: a fetch cycle and an execute cycle. In the fetch cycle, the address in the program counter accesses a location in program memory. The program memory releases the instruction word stored in the addressed location. The CPU stores this instruction in an instruction word register. During the execute cycle, this word is decoded and control signals are generated to direct activities during the remainder of the execute cycle.

The program becomes operational when power is turned on. The program counter is set to an address that holds the first line (instruction word) of the program. This first instruction word is fetched and is executed by the instruction decode and control unit. The program counter is then incremented by one count and the next instruction word is fetched from memory. This orderly progression through the program continues until an instruction is fetched or an interrupt occurs that causes a jump or a branch to another location in program memory.

An instruction may specify an unconditional jump or branch to another address, in which case the contents of the program counter are changed. The instruction may specify that a particular bit in the status register be interrogated for a particular condition. Based upon the results of the status bit interrogation, the program counter may be incremented to the next instruction, it may skip the next instruction, or it may be changed to the address of a different area of the program.

If an instruction calls a subroutine, the contents of the program counter plus one (PC+1) are placed on the stack, and the starting address of the subroutine is loaded into the program counter. If the contents of the accumulator, status register, or other registers are needed later and cannot be kept in their present locations while the subroutine is being executed, these contents will have to be temporarily stored elsewhere. This may be accomplished as part of the subroutine, or other subroutines may be called to store and then replace the contents of these registers.

Depending upon the nature of the data to be processed and the number of alternate branches and subroutines available to the program, a program may rarely repeat the same sequence of instructions, or it may rarely deviate from the same sequence of instructions. The complexity of the program is dependent upon the application in which the microcomputer is being used.

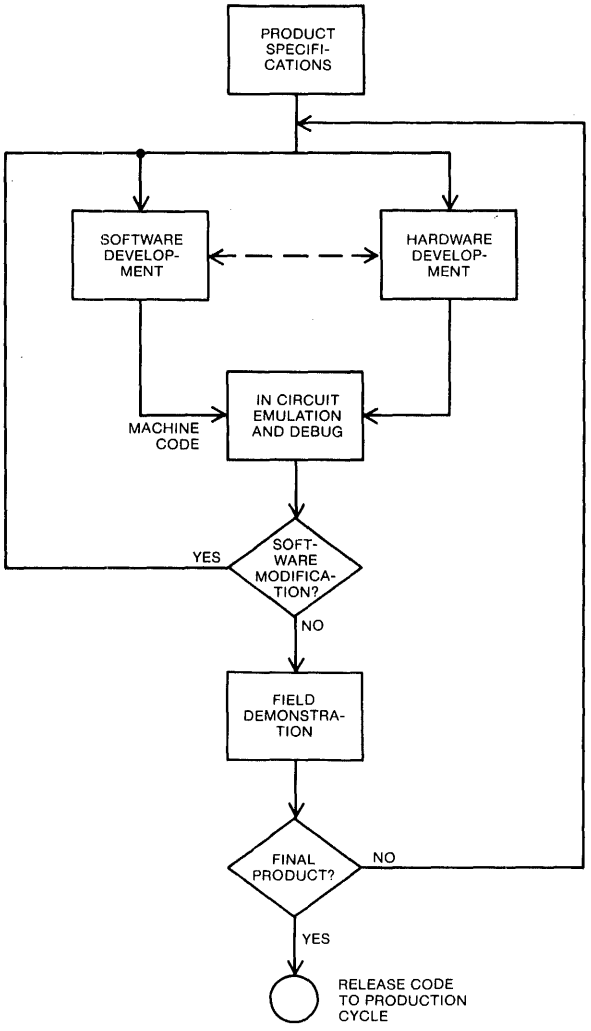
1.5 Development Cycle

As a prerequisite to the development of a product utilizing microcomputer control and/or processing, a product specification providing functional details of the product and its hardware and software requirements must be generated.

Once a microcomputer has been selected that can satisfy the software requirements and interface successfully with the hardware, the development process can be undertaken.

As shown in Figure 3, the development cycle consists of hardware and software development, in-circuit emulation and debugging, and field demonstration. The final objective of the development cycle is to generate an application program in a binary format (PIC object code) on paper tape that can be used to directly mask program the PIC microcomputer chips during the production cycle. This program is also known as the object program.

Fig. 3 DEVELOPMENT CYCLE



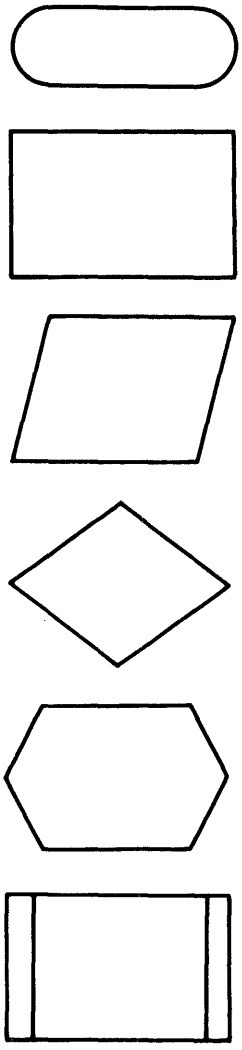
1.5.1 SOFTWARE DEVELOPMENT

Before proceeding with any coding, the hardware and software specifications must be analyzed. Once the programmer fully understands the software requirements of the particular application, he can then proceed to develop his application program by performing the following steps:

a. Flow charting
b. Code writing
c. Assembling
d. Editing (debugging).

Flow charting enables the programmer to list the major logical sequences of his program and to graphically depict input/output operations, decision points, branches, and instruction modifications to change the program and initialize a routine.

Basic symbols used in flow charting are:



START, STOP, HALT, INTERRUPT

INTERNAL PROCESSING FUNCTION

INPUT OR OUTPUT OPERATION

DECISION

PREPARATION (SET FLAG, INITIALIZE FILE SELECT REGISTER, INITIALIZE A ROUTINE, ETC.)

ONE OR MORE NAMED PROGRAM STEPS SPECIFIED IN A SUB-ROUTINE OR ANOTHER SET OF FLOW CHARTS

An example of a flow chart utilizing these symbols is shown in Figure 4.

The program that is written by the programmer to eventually be translated into the object program is known as the source program. There are many different ways to write a source program. One way is to write the program in binary code and directly punch this code onto paper tape. Although this method is direct, it is virtually impossible to implement. Writing a program in object code makes it exceedingly difficult to locate and correct mistakes. It is very difficult to analyze the program logic and make changes mandated by application updates.

Another way to write the source program is to code in octal or hexadecimal notation. An octal or hexadecimal loader program can be used to convert the octal or hexadecimal coding into the binary equivalents. This method is a little easier to read and requires less writing (4 octal or 3 hexadecimal digits as compared to 12 binary bits). However, it is still very error prone and the program logic remains inscrutable.

The most common method of writing a source program for a microcomputer is in assembly language. Assembly language provides a compromise between the symbolic notation understood by humans and the machine code understood by the microcomputer. Assembly language is the closest link to the actual machine code that still retains some speaking language characteristics.

An assembler program is required to assemble and convert the source program written in assembly language into the object program. The assembler program also provides many program development and debugging aides.

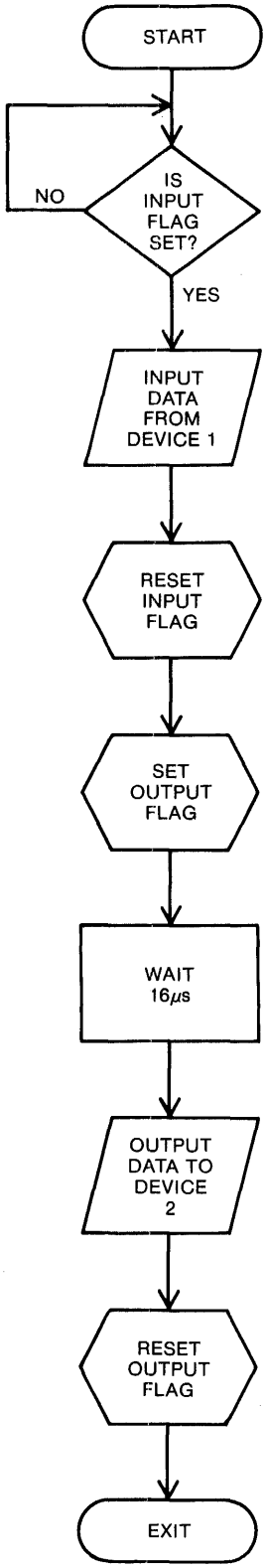
Each type of computer has its own individual assembly language and assembler. This is because the assembly language and assembler are designed to interface directly with the CPU's unique architecture and processing modes.

It is understood that in assembly language, there is a simple relationship between each line of source code and each line of object code. In higher level languages such as FORTRAN, PASCAL, COBAL, RPG, BASIC, etc., there is no such simple relationship between source code and object code. One line of code in a higher level language can accomplish the equivalent of many lines of code in an assembly language. This is known as a macro-instruction and when executed, results in a specified sequence of machine instructions.

The program responsible for converting a source program written in a higher level language into an object program is known as a compiler.

Whereas in an assembler operation, there is a direct conversion of the assembled source program into an object program, in a compiler operation, one extra conversion is required. The source code written in high level language must be converted into the equivalent assembly-type codes (macro-conversion). Then the source program is assembled and converted to object code.

Fig. 4 FLOW CHART OF PROGRAM TO MOVE DATA FROM ONE EXTERNAL DEVICE TO ANOTHER



The object code for the PIC microcomputer must be generated on another computer via a process known as cross-assembling. This process is enabled by loading a program known as a Cross-Assembler into the host computer. The Cross-Assembler is written in the language of the host computer's resident assembler or compiler. The Cross-Assembler enables the host computer to translate the PIC assembly language source instructions and provide object code formatted for PIC applications rather than object code for the host computer's internal CPU.

The PIC Cross-Assembler, PICAL, executes on any minicomputer or large scale computer having a resident editor and FORTRAN IV compiler or BASIC interpreter. PICAL enables the host computer to assemble the PIC source program and provide an object program that can execute on the PICES in-circuit emulation system.

After the source program has been loaded and assembled, a program listing may be printed out. Each line of the source program is listed exactly as coded. This includes the label, OP Code, operand(s) and comment fields. In addition, three other columns are provided: the first column is the line number; the second column is the program location (address) expressed in octal; the third column is the object code, also expressed in octal.

If there are syntax errors in any of the assembly statements or any illegal operations, the Cross-Assembler will flag the statements in which these errors are found and generate an error message. The programmer, once he analyzes the error messages has the option of correcting and re-assembling the source program, or entering simple corrections directly to the object program.

FORMAT OF PIC ASSEMBLER LISTING

```
LINE    ADDR    B1    B2                PIC MACRO ASSEMBLER VER 1.0                    PAGE    1

   1  000020                 L0INFO  EQU     .16       ;TOTAL NUMBER OF DATA BITS
   2  000310                 L0LTIM  EQU     .200      ;MAX LOOP TIME
   3  000003                 L0DELY  EQU     .3        ;INTER-PULSE DELAY
   4  000034                 L00LOW  EQU     .28       ;0 BIT PULSE COUNT
   5  000044                 L00HI   EQU     .36       ;+/- 4
   6  000054                 L01LOW  EQU     .44       ;1 BIT PULSE COUNT
   7  000064                 L01HI   EQU     .52       ;+/- 4
   8  000010                 L0END   EQU     .8        ;8 BITS IN COMMAND WORD
   9                                                   ;
  10  000011                 F0CMD   EQU     11
  11  000012                 F0CMP   EQU     12
  12  000013                 F0INF0  EQU     13
  13  000001                 F0RTCC  EQU     1
  14  000014                 F0LTIM  EQU     14
  15  000015                 F0CSAV  EQU     15
  16  000016                 F0DELY  EQU     16
  17  000004                 F0FSR   EQU     .4
  18                                                   ;
  19                                                   ;CMDREC: RECIEVE CODED COMMAND FROM IR TRANSMITTER

  20                                                   ;        DECODE AND STORE IN F0CMD
  21                                                   ;
  22  000000                 CMDREC  RES     0
  23  000000  0151                   CLRF    F0CMD     ;CLEAR COMMAND WORD
  24  000001  0152                   CLRF    F0CMP     ;AND COMMAND COMPLIMENT
  25  000002  6011                   MOVLW   F0CMD     ;COMMAND WORD IN INDEX
  26  000003  0044                   MOVWF   F0FSR
  27  000004                 L01HI   RES     0
*** DUPLICATE LABEL
  28  000004  6020                   MOVLW   L0INFO    ;16 INFORMATION BITS
  29  000005  0053                   MOVWF   F0INF0    ;
  30  000006                 CMD100  RES     0
  31  000006  0141                   CLRF    F0RTCC    ;PULSE COUNT =0
  32  000007  6310                   MOVLW   L0LTIM    ;MAX LOOP TIME
  33  000010  0054                   MOVWF   F0LTIM    ;
  34  000011  0155                   CLRF    F0CSAV    ;SAVED PULSE COUNT =0
  35  000012                 CMD101  RES     0
  36  000012  1041                   TSTF    F0RTCC    ;ANY DATA?
  37  000013  0045                   MOVWF   5
  38  000014  3103                   SKPNZ
  39  000015  5012                   GOTO    CMD101    ;NO, WAIT FOR DATA
  40  000016                 CMD102  RES     0
  41  000016  1001                   MOVF    F0RTCC,W  ;ANY MORE DATA
  42  000017  0255                   SUBWF   F0CSAV
  43  000020  0000  0000             SKNPZ
*** OPCODE ERROR
  44  000022  3103                   SKPNZ
  45  000023  5033                   GOTO    CMD104    ;NO, PROCESS INFO BIT
  46  000024  0055                   MOVWF   F0CSAV    ;YES, SAVE PULSE COUNT
  47  000025  1240                   INCF    77
*** INVALID FILE REGISTER
  48  000026  6003                   MOVLW   L0DELY    ;WAIT BEFORE CHECKING
  49  000027  0056                   MOVWF   F0DELY    ;PULSE COUNT AGAIN
  50  000030                 CMD103  RES     0
  51  000030  1356                   DECFSZ  F0DELY
```

1.5.2 HARDWARE DEVELOPMENT

During the hardware development phase, a circuit is developed that interfaces with the programmed PIC microcomputer and performs in accordance with the product specifications. During this phase, the operating voltage requirements of the PIC chip and input/output loading requirements are analyzed. The number of inputs and outputs and input/output timing requirements are also analyzed and I/O lines allocated. Interrupts, I/O flag and real-time clock counter functions are worked out and I/O specifications provided for software development. Design of the external clock circuit (RC or crystal driver) is implemented, based upon the timing requirements of the application hardware and use of the real time clock generator.

1.5.3 IN-CIRCUIT EMULATION

In-circuit emulation allows the user to integrate the hardware and software functions and debug the system. PICES II (PIC In-Circuit-Emulation System) is a low cost development tool consisting of:

- A 16 bit control processor to execute the debug facilities
- A "personality" module containing a ROMless development PIC microcomputer configured to emulate one of the processors in the PIC Family
- An optional EPROM programmer

The PICES II system enables the user to execute an application program in real time or in the trace mode. In addition, the contents of all the PIC registers can be displayed and modified. Refer to the PICES II user's manual for a detailed description of the system's capabilities.

1.5.4 FIELD DEMONSTRATION

Once the hardware and software are functioning correctly within the in-circuit emulation setup, field demonstrations can be performed using the PIC Field Demo System. The PFD modules consist of a ROMless PIC microcomputer that can emulate the entire PIC family, sockets for erasable PROMs, and a 40- or 28-lead cable for connection to the applications hardware. The E/PROMs hold the application program. This unit, when connected to the application hardware, provides field demonstration of the integrated hardware/software system.

2 ARCHITECTURE

The various members of the PIC family of microcomputers have the same basic architecture and almost identical instruction sets. Major differences are in the I/O port arrangement and in interrupt handling. Therefore the following description is of PIC functional blocks in general terms. Each PIC microcomputer will be described in terms of differences in the following sections.

2.1 PIC Basic Functional Blocks

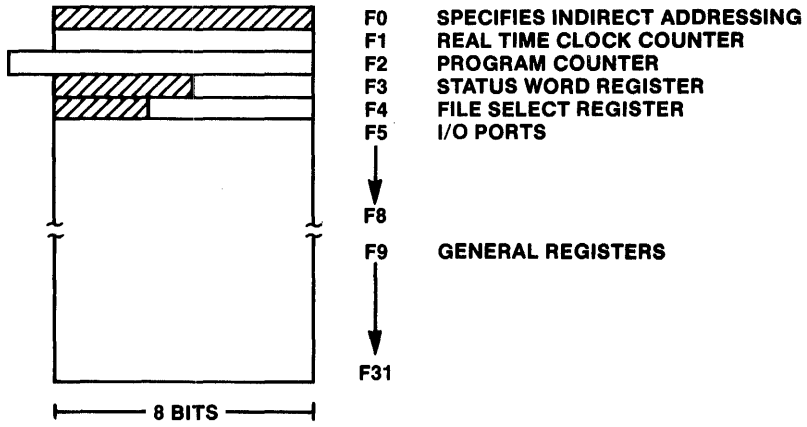
Figure 5 is a functional block diagram of a PIC microcomputer. The PIC microcomputer consists of the following functional elements:

- Instruction Decode and Control Unit
- Program Counter
- Hardware Stack
- File Select Register (for indirect addressing)
- Arithmetic Logic Unit (ALU)
- Accumulator (W register)
- Status Word Register
- Real-Time Clock Counter Register
- Program ROM
- Data RAM
- 8-Bit I/O Registers
- Interrupt Logic

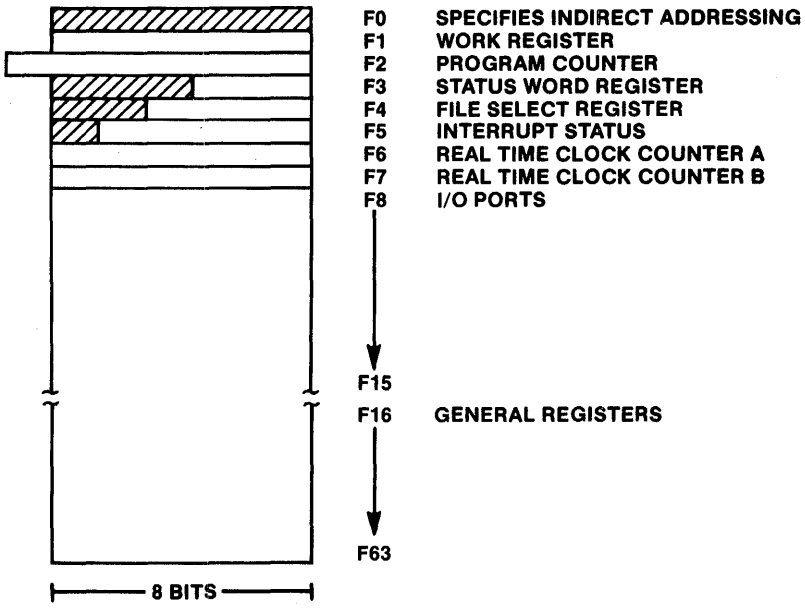
Internally, the functional elements of a PIC microcomputer are tied together by a bidirectional data bus. The transfer of data via the bus is controlled by the instruction decode and control logic which decodes the instruction to provide an address and/or control signals to each location that is to receive, transmit, and/or manipulate data transferred via the bus.

The special registers (RTCC register, PC, status word register and file select register), the four I/O registers, and the data RAM are organized as a RAM file. Each register has its own unique RAM file address.

RAM ORGANIZATION—PIC1650 SERIES



RAM ORGANIZATION—PIC1670 SERIES



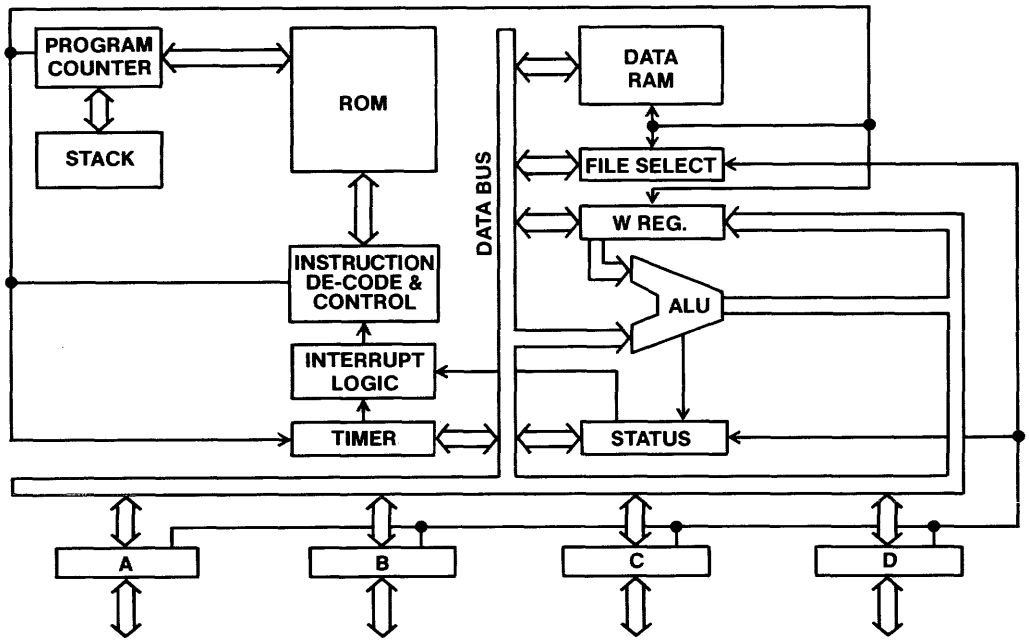


Fig. 5 PIC BLOCK DIAGRAM

File registers can be directly addressed by the instruction word, or indirectly addressed by specifying F0 when the contents of the file select register (FSR) is to be used as the file address.

The purpose of each of the functional elements of the PIC1650A is described in the following paragraphs.

2.1.1 INSTRUCTION DECODE AND CONTROL UNIT

The instruction decode and control unit receives the 12-bit instruction word from program memory, decodes it, and issues the appropriate control signals to cause the desired operations to take place. At the same time, the instruction decode and control unit, depending upon the instruction type, issues a file address, a literal OPERAND, or a program address that vectors a call or GOTO operation.

2.1.2 PROGRAM COUNTER (F2)

The program counter is an addressable register that points to the address of the next instruction to be fetched out of program memory. The PC provides for direct addressing of all memory locations.

The program counter is incremented by one under control of the instruction decode and control unit after the execution of every instruction. Exceptions are conditional and unconditional skips and branches and subroutine calls and returns.

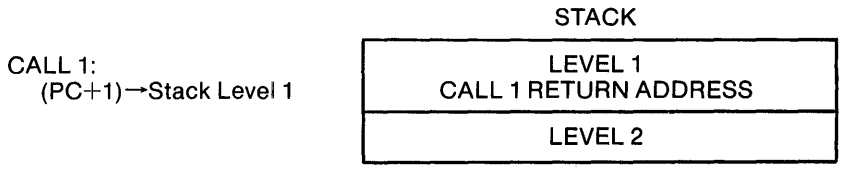
When performing a skip operation, the program counter is incremented by one, but a NOP instruction replaces the next instruction from the main program. When a GOTO operation is performed, the program counter is vectored to the specified address. When a subroutine is called, the contents of the program counter plus one (PC+1) are pushed onto the stack and the value in the program counter is vectored to the specified address. When there is a return from the subroutine, the contents of the top level of the stack are transferred to the program counter, also, the contents of the second level of the stack move to the top.

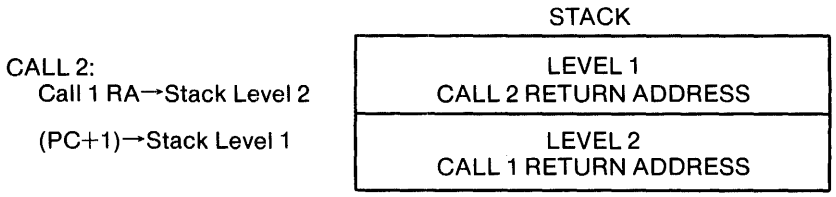
Bits 0 through 7 (but **not** bit 8) of the program counter may be read and transferred to a data location to construct a software stack pointer and stack in data memory. The program counter may be used as the destination of any operand, but bit 8 will always be zero.

2.1.3 STACK

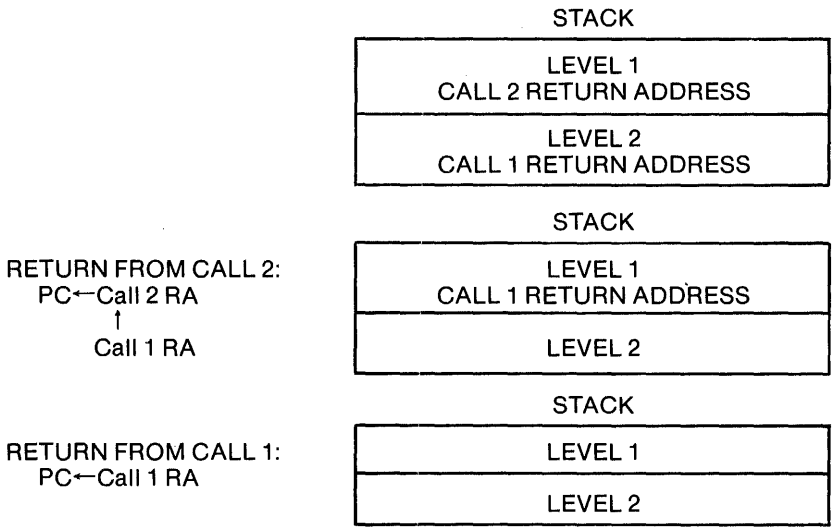
A hardware stack is provided to accommodate two return addresses. This facilitates execution of nested subroutines.

When executing a nested subroutine, the contents of the stack are as follows:





When returning from a nested subroutine, the contents of the stack are as follows:



2.1.4 FILE SELECT REGISTER (F4)

The FSR is an addressable five-bit register used to indirectly address the register file.

An address can be written into the FSR via the eight-bit internal data bus. Only the lower order of the eight-bit word is relevant.

When the indirect address mode (F0) is indicated, the contents of the register pointed to by the FSR will be accessed. For example, the expression ADDWF 0, W specifies that the contents of the W register and the contents of the register pointed to by the FSR will be added and the result will be placed in the W register.

The contents of the file select register can be stored in another location by directly addressing the FSR (F4), and moving its contents to the accumulator. From the accumulator, the contents can be moved to another register. However, the three high order bits are read as 111 if the FSR is specified in an instruction. For example:

MOVF 4, W	(F4)→W
MOVWF 23	(W)→F23

The contents of the FSR can be restored by reversing the procedure:

MOVF 23 W	(F23)→W
MOVWF 4	(W)→F4

2.1.5 ARITHMETIC LOGIC UNIT (ALU)

The ALU implements various binary arithmetic and Boolean logic operations utilizing one or two 8-bit operands. One operand is fetched from any of the file locations or is a literal in the instruction itself. The other operand (if applicable) is held in the accumulator. Operations performed by the ALU are as follows:

- ☐ Add/Subtract
- ☐ Increment/Decrement
- ☐ AND, OR, Exclusive OR
- ☐ Complement, Clear
- ☐ Rotate Left/Right, Swap Half-Bytes

By using one or a combination of these operations, the ALU performs binary addition, subtraction, multiplication, and division on 8-bit operands. When 16-bit operands are required, double-precision arithmetic operations can be implemented. BCD, mask operations, and bit and field manipulations can also be performed.

2.1.6 WORKING REGISTER (W)

The W register serves as the accumulator for the ALU. The W register holds one of the operands operated on during an arithmetic or logical operation and may store the result.

2.1.7 STATUS WORD REGISTER (F3)

The status word register is an addressable register that stores the condition of the most recent ALU operation. Bits 0 through 2 of the status register are used to store the carry, digit carry, and zero status. The bits in the status register can be set or cleared by bit level program instructions, or by the MOVW F3 instruction. Only file register operations which do not affect any status bit can be used on the status register.

7-4	3	2	1	0
Not used	OV	Z	DC	C

C (Carry): Stores the carry out of arithmetic operations and acts as a bit link in rotate operations. This bit is also set to a one during a subtract operation if the absolute value in the file register is greater than the absolute value in the W register.

DC (Digit Carry): Stores the carry out of the low order digit (4 LSB's) in an arithmetic operation. This bit is also set to a one during a subtract operation if the absolute value of the four LSB's in the file register is greater than the absolute value of the four LSB's in the W register.

Z (Zero): Set if the result of the arithmetic operation is zero.

OV (Overflow): Set if the carry out from the MSB is opposite to the carry out from MSB-1.

2.1.8 REAL-TIME CLOCK/COUNTER REGISTER

The RTCC register is an addressable eight-bit up-counter that is used to time or to count external events. The RTCC register can be preset under program control to any eight-bit binary value. The count input to the RTCC register is applied via the external $\overline{RTCC}$ pin. The counter increments on the falling edge of $\overline{RTCC}$. When it reaches 377_8, it keeps on counting through 000_8 but does not set the carry flag.

The RTCC register can be used to count up to 256 external events via the $\overline{RTCC}$ line. The program requirement may be to count a predetermined number of events, or the program requirement may be to count an undetermined number of events occurring within a particular program sequence.

If an unknown number of events is to be counted, the RTCC register will first be set to zero under program control. The counter will then increment on each event input at the $\overline{RTCC}$ pin. The contents of the RTCC register (number of events counted) are interrogated under program control.

If a count of more than 256 is required, a number of bits in a data register can be appropriated to accumulate and store the carry bits from the RTCC register. In this way, the magnitude of the event count can be increased.

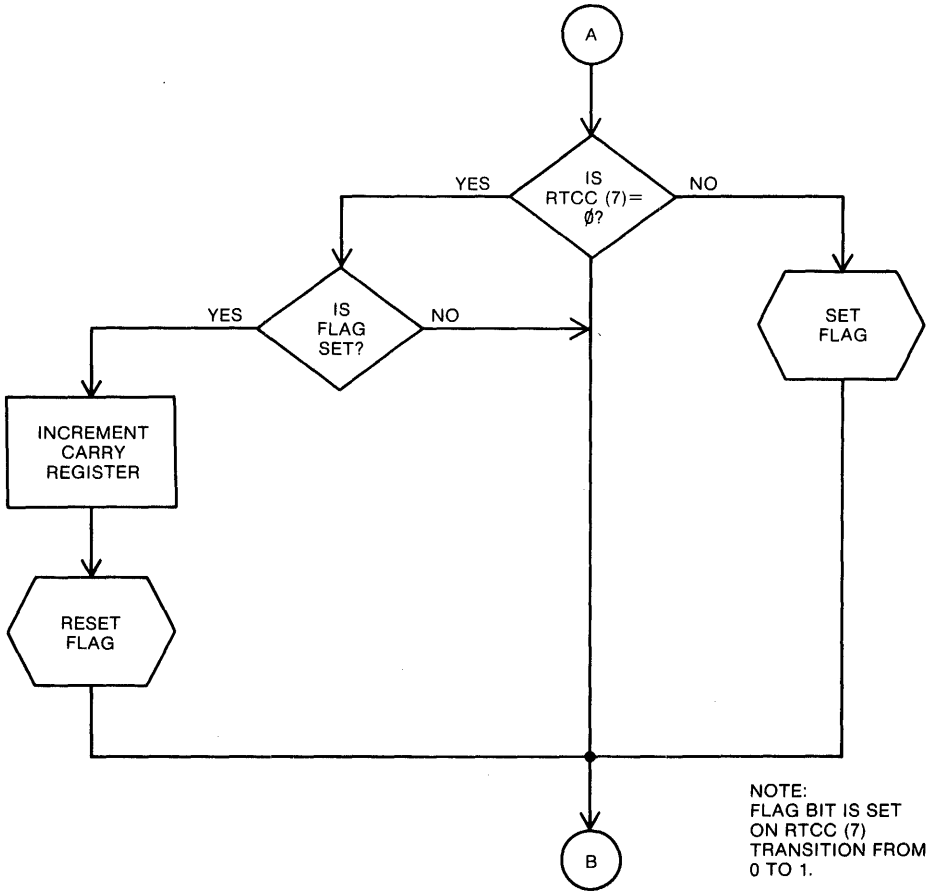
Figure 6 is a flow chart of program logic that can be used to implement this operation. Assume that the four low order bits of a data register (F23) are assigned to accumulate the carries from the RTCC register and that its high order bit is used as a flag to signal when RTCC bit 7 sets. When the RTCC register subsequently attains a full count and then resets (RTCC bit 7 resets), the carry register will be incremented and the flag bit reset.

The following is a sample program illustrating the coding required to implement the logic illustrated in Figure 6. (Refer to Section 3 for an explanation of the coding.)

	Program Steps	Description
	BTFSC 1, 7	Skip if RTCC (7) is zero
	GOTO NOT0	Jump if RTCC (7) is not zero
	BTFSS 23, 7	Skip if FLAG is set
	INCF 23	Increment Carry Register
	BCF 23, 7	Reset FLAG
	GOTO B	EXIT
NOT0:	BSF 23, 7	Set FLAG
	GOTO B	EXIT

When the RTCC register is used to count a predetermined number of events, the number of events is subtracted from zero (two's complement) and this number is preset into the counter. When the counter increments to zero, the required number of events has occurred. Similar logic to that shown in Figure 6 can be used to determine when the counter has reset on a full count.

Fig. 6 RTCC COUNT EXPANSION FLOW CHART



The RTCC register can also be used to time events or the interval between events. These events may be input via the input/output ports or may be generated by the program.

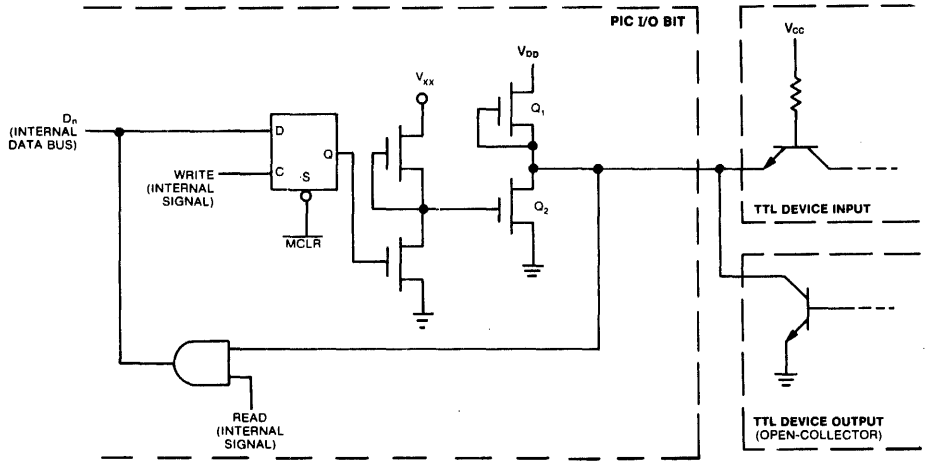
The timing clock may be a real-time clock (e.g. 60 Hz) applied to the $\overline{\text{RTCC}}$ input, the external clock generated by the PIC1650A, or any other clock applied to the $\overline{\text{RTCC}}$ input that is within the $\overline{\text{RTCC}}$ timing specifications.

2.1.9 I/O REGISTERS

The PIC has up to four 8-bit bidirectional input/output registers (A through D) providing a total of 32 bidirectional input/output lines for interfacing with external devices.

The equivalent circuit for an individual bit of an I/O port is shown in Figure 7 as it would interface with input and output TTL devices. As shown in Figure 7, data written to a port for outputting is strobed into the I/O port latch from the internal data bus by a WRITE command. This data remains unchanged until rewritten. Data applied to the port for inputting is not latched.

Fig. 7 TYPICAL INTERFACE, BIDIRECTIONAL I/O LINE



*Pull-up resistor may be deleted via a mask option.

Input data is available on the I/O line for a period of time determined by the input device. The input data is transferred to the accumulator via the internal data bus when the READ line is high.

Each I/O line is pulled up to V_{DD} through pullup transistor Q1 which provides sufficient source current for a TTL high level, yet can still be pulled down by a TTL low level. When inputting data through an I/O port, the latch must be set to a logic 1 level under program control. This turns off Q2 which allows the TTL open collector device to drive the pin, pulled up by Q1.

The bidirectional interface illustrated in Figure 7 is only one of many possible input/output configurations.

Any of the bits in an I/O register can be used as an individual dedicated input or output line. I/O lines are normally grouped together into I/O files to minimize software servicing.

An input operation is performed when an external input device has valid input data for the PIC. This input data may be available at predetermined intervals during the program or at intervals monitored by the RTCC register. At these intervals, the program will set the output latches to logical 1's and execute an input instruction that loads the input data into the W register, from where it may be transferred or manipulated.

2.1.10 PROGRAM MEMORY (ROM)

The ROM contains the customer-defined operational program. Since the instruction word is wider than 8 bits, instructions are all single word, more versatile, and usually require only one machine cycle to execute.

2.1.11 DATA MEMORY (RAM)

Data memory consists of special purpose registers and general purpose registers. Data memory can be directly addressed via the internal address bus or indirectly addressed via the FSR.

Input data may be available from more than one input device and may be asynchronous. With this type of input arrangement, the program must determine when valid input data is available and which external device is inputting before it executes an input operation. Moreover, if more than one device has input data available at the same time, priorities must be assigned to determine which set of inputs will be serviced first.

In order for each input device to signal that it has data available, an I/O register, or a portion thereof, may be utilized as a "flag" register. Each flag bit is assigned to an associated input device which, when it has data ready, causes its associated flag bit to set. The program periodically interrogates the flag bits to determine which devices have input data and then performs the necessary input operations.

Figure 8 illustrates program logic that could be utilized for this type of input operation.

Assume that there are four input devices and that bits 0 through 3 of I/O register F7 are used as flags for each of the devices. Bit 0 is associated with the highest priority device (A); bit 3 with the lowest priority device (D). When a bit is set, it means that the associated device has data for the PIC. If more than one bit is set, it means that more than one device has data available. Data will be input in the order of highest priority. Figure 8 is a flow chart of the bit interrogation logic.

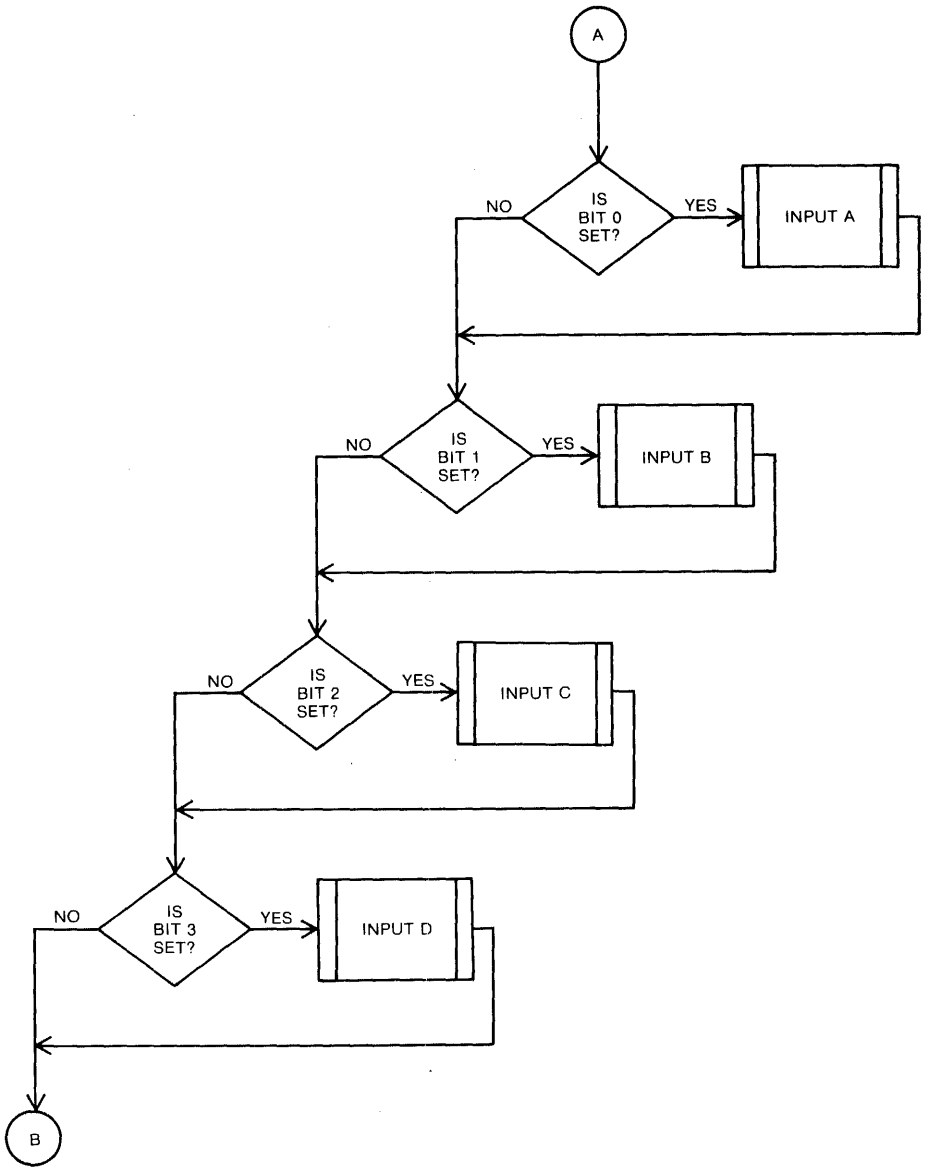
The following is a sample program illustrating the coding required to implement the logic illustrated in Figure 8. (Refer to Section 3 for an explanation of the coding.)

Program Steps	Description
BTFSZ 7, 0	Skip if bit 0 is zero
CALL INPUT A	Call INPUT A subroutine
BTFSZ 7, 1	Skip if bit 1 is zero
CALL INPUT B	Call INPUT B subroutine
BTFSZ 7, 2	Skip if bit 2 is zero
CALL INPUT C	Call INPUT C subroutine
BTFSZ 7, 3	Skip if bit 3 is zero
CALL INPUT D	Call INPUT D subroutine
GOTO B	EXIT

When a port is dedicated to output operations only, data can be written to that port at any time, and the output latch can be used for data manipulations.

When an I/O port is used for bidirectional transfer of data, caution must be exercised when performing output operations. Bit manipulations performed on output data stored in the output latch can be affected by data input by an external device at the same time the output data in the latch is accessed. Extraneous input bits having logical 0 values may be introduced. To avoid this possibility, output data can be stored in a data

Fig. 8 **INPUT FLAG INTERROGATION FLOW CHART**



register where it can be accessed for bit manipulation without being affected by input operations. When the data is ready for output, it is transferred to the output port.

NOTE: Any output line sinking more than 5mA could be read as a logic 1.

Each I/O port can be individually time-multiplexed between input and output functions under software control. For information on I/O timing refer to PIC data sheets.

2.1.12 CLOCK GENERATOR

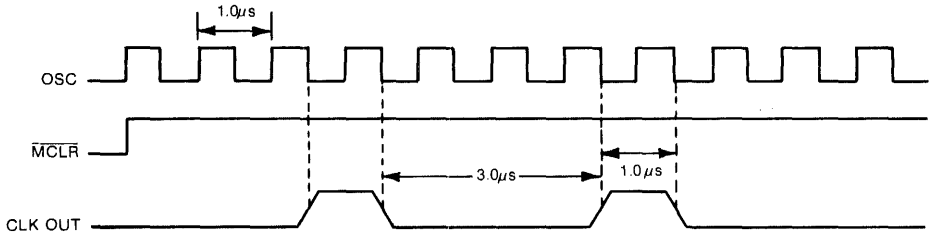
The clock generator generates the internal clocks from which the microprocessor machine cycle is derived. It also generates an external clock at the instruction cycle rate. The clock generator frequency is controlled externally for the devices which do not directly support a crystal oscillator (PIC1650A, PIC1655A). Frequency control may be established by an RC network connected to the OSC input pin, or in applications where more precise timing is required, by a buffered crystal driver.

The PIC1650A and PIC1655A clock generator divides by four the frequency measured at the OSC pin. Therefore, a 1MHz frequency at the OSC pin results in a machine cycle of 4μs (0.25MHz). The minimum machine cycle time is 4μs; the maximum is 20μs. Therefore, the frequency at the OSC pin must range between 1MHz and 200KHz.

Figure 9 is a timing diagram that illustrates the relationship between OSC, $\overline{\text{MCLR}}$, and CLK OUT, assuming a frequency at the OSC pin of 1MHz. Figure 10 illustrates resistance values required to obtain instruction execution speeds of 50 to 250KHz, where the external capacitance is 47pf and the value of R is selected within the range of 14K to 28K.

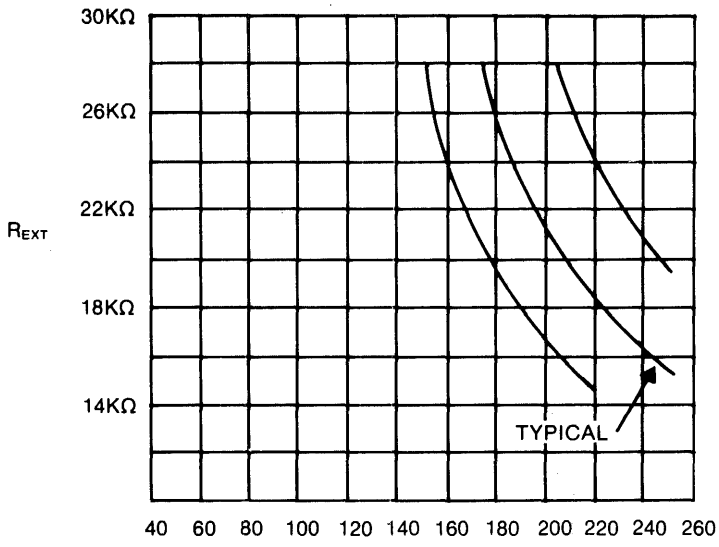
The oscillator itself consists of 7 inverters connected in a ring fashion as shown in Figure 11A. The diagram in Figure 11B describes the technique for supplying an external clock.

Fig. 9 CLOCK GENERATOR TIMING DIAGRAM



Note: PIC1650A, PIC1655A only.

Fig. 10 TYPICAL OSCILLATOR RC CHART



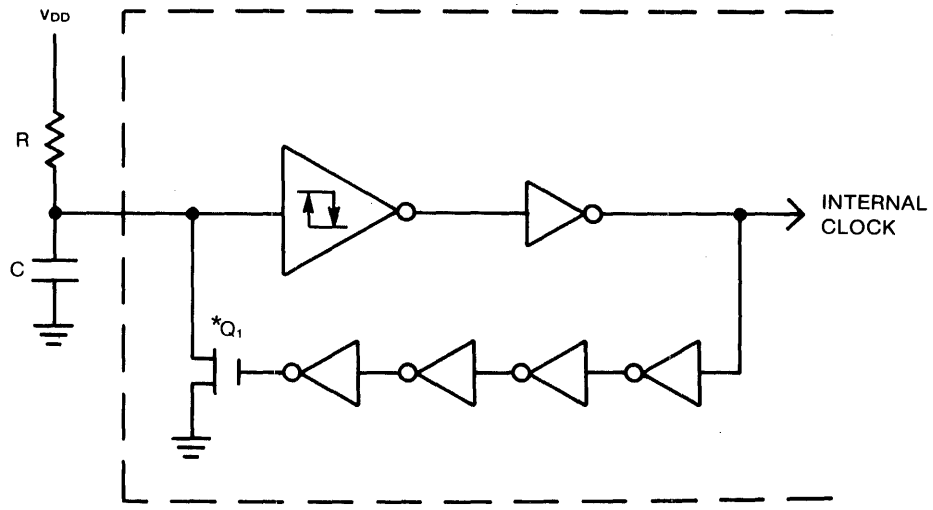
V_{DD} = 5.0V
C = 47pF
T_A = 25°C

Oscillator Frequency With Typical Unit To Unit Variance

Unit to Unit Variation at V_{DD} = 5.0V, T_A = 25°C is ±25%
Variation from V_{DD} = 4.5V −7.0V referenced to 5V is −3%, +9%
Variation from T_A = 0°C −70°C referenced to 25°C is +3%, −5%

Note: PIC1650A, PIC1655A only.

Fig. 11A **OSCILLATOR CIRCUIT**

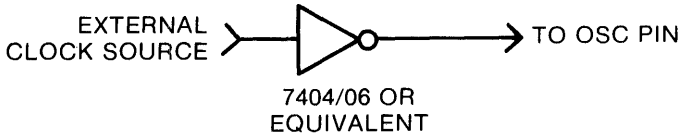


*Q_1 may be deleted via a mask option when an external clock drive is desired.

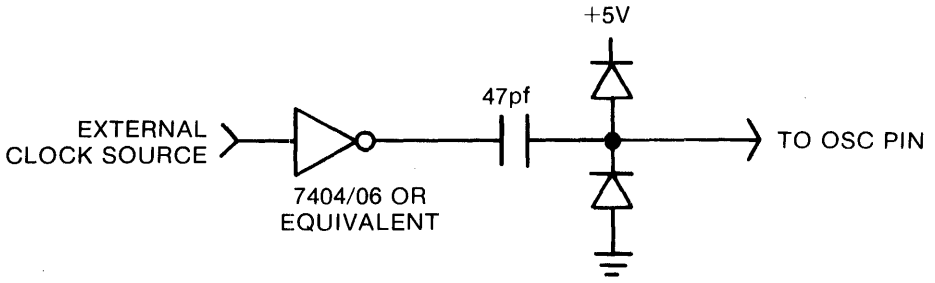
The first inverter from the OSC pin is a high gain Schmitt trigger to provide trip point control, while Q_1 in combination with the external resistor serves to form the 7th inverter in the ring.

Fig. 11B **EXTERNAL CLOCK**

When driving the oscillator directly from a buffer, it is necessary that the buffer be capable of pulling the input to a level of V_{DD} −1 Volts driving 100K ohms. When the positive threshold of the input (Schmitt trigger) is reached, Q1 turns on pulling the input to ground. The buffer must then be capable of sourcing sufficient current to keep the input above 2.0V driving 120 ohms (Q1 on resistance) during the remaining positive half cycle. During the negative half cycle the input must be driven below 0.8 Volts. The oscillator duty cycle should be between 20 and 60%.



An alternate external buffer circuit which consumes much less power is as shown.



However, when an external clock is to be used, it is recommended that the options which remove Q_1 (Fig. 11A) be specified.

2.2 PIC1650A

ROM	RAM	I/O	Interrupt	Stack (Levels)	Timer	Package	Process
512 x 12	32 x 8	32	No	2	Yes	40 Pin	NMOS

Four 8-bit I/O registers are provided. These registers (A, B, C and D) are addressable as F5 through F8.

2.3 PIC1654

ROM	RAM	I/O	Interrupt	Stack (Levels)	Timer	Package	Process
512 x 12	32 x 8	12	No	2	Yes	18 Pin	NMOS

The PIC1654 provides the same architectural features of the PIC1650A in an 18-pin package. The PIC1654 has 12 I/O lines compared to the PIC1650's 32 lines.

One 4 bit and one 8 bit bidirectional I/O registers are provided. These registers (A and B) are addressable as F5 and F6. F7 and F8 are general purpose registers.

2.4 PIC1655A

ROM	RAM	I/O	Interrupt	Stack (Levels)	Timer	Package	Process
512 x 12	32 x 8	20	No	2	Yes	28 Pin	NMOS

The PIC1655A provides the same architectural features of the PIC1650A in a 28-pin package. The major difference is that the PIC1655A has 20 I/O lines rather than the 32 I/O lines of the PIC1650A.

One 4-bit and two 8-bit I/O registers are provided. These registers (A, B, and C) are addressable as F5 through F7, respectively. Register A (F5) controls four dedicated non-latching input lines; register B (F6), which cannot be read internally, controls eight dedicated latched output lines; and register C (F7) controls eight bidirectional input/output lines. Register file F10, which in the PIC1650A was I/O register D, is an additional general purpose data register in the PIC1655A.

The PIC1655A utilizes the same instruction set as the PIC1650A.

2.5 PIC16C58

ROM	RAM	I/O	Interrupt	Stack (Levels)	Timer	Package	Process
512 x 12	32 x 8	20	No	2	Yes	28 Pin	CMOS

The PIC16C58 is the low power CMOS version of the PIC1655A. The PIC16C58 has an additional architectural feature in that all I/O lines can be put in the tri-state mode. It also has an ultra-low power standby mode, wherein the oscillator is stopped and the chip draws only leakage current while the RAM contents are retained. Refer to the PIC16C58 data sheet for complete description.

2.6 PIC1656

ROM	RAM	I/O	Interrupt	Stack (Levels)	Timer	Package	Process
512 x 12	32 x 8	20	Yes	3	Yes	28 Pin	NMOS

The PIC1656 employs the same basic architecture as the PIC1655A with the addition of an interrupt system (Fig. 12). To accommodate the interrupt logic, five status bits have been added to the status register. The interrupt logic operates in conjunction with the RT input pin, the RTCC register and the status register.

The $\overline{RT}$ pin can be used to provide a clock input for the RTCC register or it can be used as an external interrupt input. The function of this pin is controlled by the contents of the status register. When the $\overline{RT}$ pin is used as an external interrupt pin, a high-to-low transition initiates a vectored interrupt (external interrupt mode) if IE is set.

The status word also controls the count function of the RTCC register. It enables the RTCC register to increment on the internal clock (same clock as CLK OUT) or on the input at the $\overline{RT}$ pin. When the RTCC register overflows, it initiates a vectored interrupt (RTCC interrupt mode), if interrupts are enabled (RTCE set.)

2.6.1 INTERRUPT LOGIC

The interrupt logic generates an interrupt request to the control unit to initiate a vectored interrupt. One of two possible interrupt requests (external interrupt request or RTCC interrupt request) can be generated. Only one interrupt at a time can be serviced. Nested interrupts are not possible since additional interrupts are disabled by an internal latch.

The contents of the status register indicate whether any interrupts are pending. If only one interrupt is pending, it is serviced immediately providing the interrupt is enabled (i.e., IE or RTCE is set) and the processor is not already servicing another interrupt. If both external and RTCC interrupts are pending and enabled, the external interrupt has priority. If an external interrupt is input on the $\overline{RT}$ pin while another external interrupt is being serviced, a new external interrupt request will be generated to the processor which will reinterrupt immediately upon its return from the current interrupt.

CAUTION

A return from an interrupt routine must not be executed using any other instruction but RETURN. If any other instruction is executed to restore the return address to the program counter, the interrupt logic will not be enabled. This effectively prevents any other interrupts from being serviced. If the interrupt routine contains subroutines, returns from the subroutines should be made using the RETLW instruction. If the RETURN instruction is used mistakenly, additional interrupts that occur while the first interrupt routine is in process will be enabled and can corrupt the interrupt routine in process.

2.6.2 STATUS REGISTER

The Status register (F3) of the PIC1656 is provided with additional status bits that control the interrupt logic and the count function of the RTCC register. The status register is configured as follows:

7	6	5	4	3	2	1	0
CNT	RTCR	IR	RTCE	IE	Z	DC	C

■ BITS 0-2: Carry, digit carry and zero status bits. Same function as PIC1650A.

■ BIT 3: Interrupt Enable (IE) status bit. When set to a one, this bit enables the external interrupt to occur when and if the interrupt request (IR) status bit (bit 5) is also set. When reset to a zero, the external interrupt is disabled.

■ BIT 4: Real-Time Clock Enable (RTCE) status bit. When set to a one, this bit enables the real-time clock/counter interrupt to occur when and if the real-time clock interrupt request (RTCR) status bit (bit 6) is also set. When reset to a zero, the interrupt is disabled.

■ BIT 5: Interrupt Request (IR) status bit. This bit is set by a high-to-low transition on the $\overline{RT}$ pin, generating an interrupt request. If and when the interrupt enable (IE) bit (bit 3) is also set, an interrupt will occur. This causes the current PC address to be pushed onto the stack and the processor to execute the instruction at location 760_8. The IR bit is then immediately cleared. Note that the IR bit can be set regardless of the state of the IE bit, thus requesting an interrupt which can be serviced or not at the programmer's option.

■ BIT 6: Real-Time Clock/Counter Interrupt Request (RTCR) status bit. This bit is set when the RTCC register (File 1) transitions from a full count (377_8) to a zero count (000_8). If and when the RTCE bit is also set, an interrupt will occur. This causes the current PC address to be pushed onto the stack and the processor to execute the instruction at location 740_8. The RTCR bit is then immediately cleared. Note that the RTCR bit can be set regardless of the state of the RTCE bit, thus requesting an interrupt which can be serviced or not, at the programmer's option.

NOTE: Although the processor cannot be interrupted during an interrupt (i.e., until the RETFI instruction is executed), (an)other interrupt(s) can be requested (status bits 5 and/or 6 can be set). This will cause the processor to reinterrupt immediately upon its return from the current interrupt assuming the interrupt(s) is (are) enabled. (Pending external interrupts have priority over pending real-time clock/counter interrupts.)

■ Bit 7: Count Select (CNT) status bit. When the CNT bit is set to a one, the RTCC register will increment on each high-to-low transition at the RT pin. If the CNT bit is reset to a zero, the RTCC register will increment at the internal clock rate (1/16 of the frequency at the OSC pins).

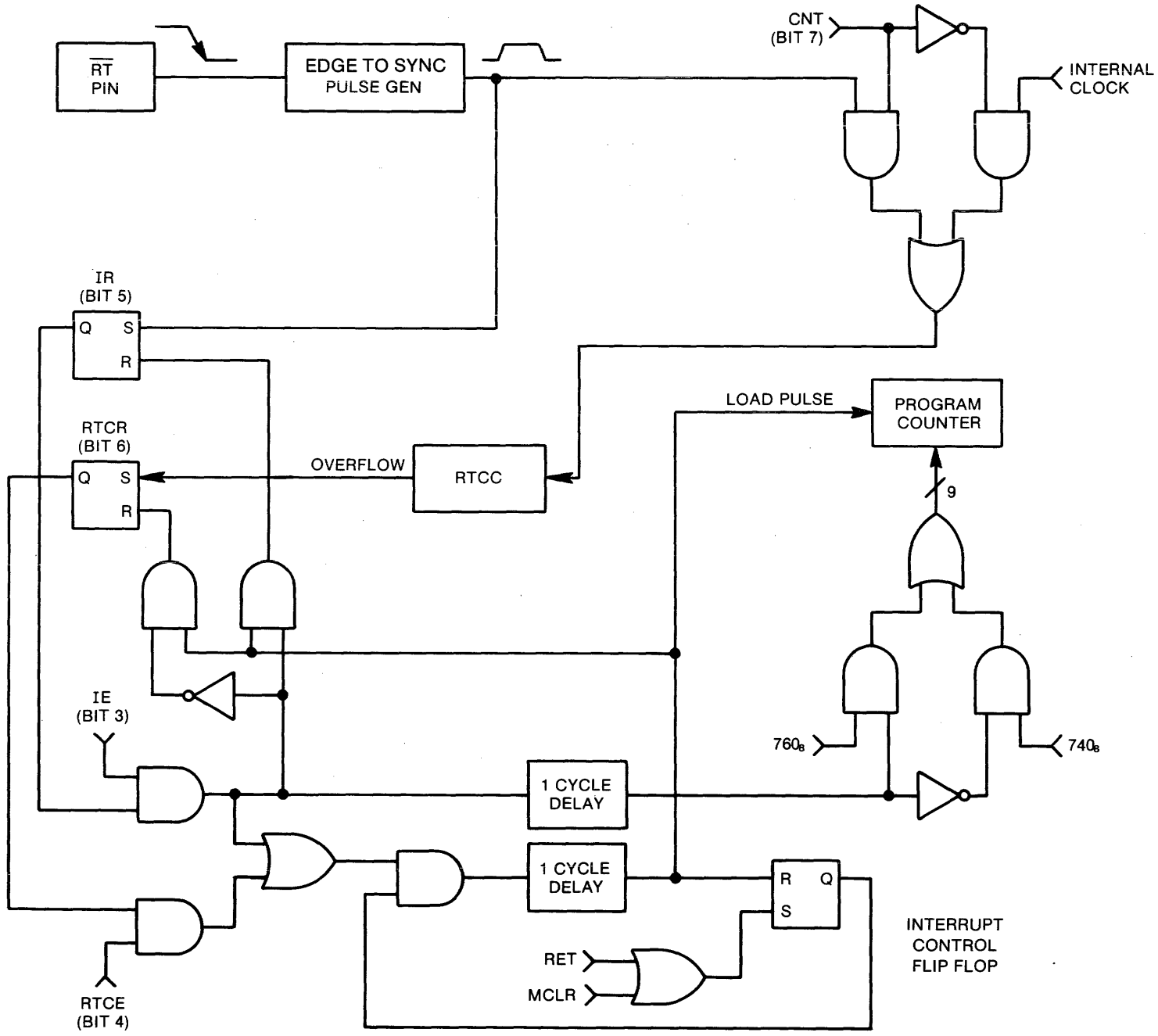


Fig. 12 INTERRUPT SYSTEM

2.6.3 STACK

A three-level stack is provided to accommodate three return addresses. One level of the stack should be reserved to store the return address of an interrupt. The other two levels provide storage for two return addresses from a nested subroutine.

NOTE: One level of the stack must always be available to accommodate an interrupt return address. When an interrupt occurs, the firmware automatically pushes the return address onto the stack. Should three subroutines be nested, the return address of the current subroutine will be destroyed. Only if the PIC1656 is not programmed for interrupts is it permissible to use all three levels of the stack.

2.6.4 RTCC REGISTER

The RTCC register (F1), in conjunction with the status register, is programmable for internal clock or $\overline{RT}$ clock operation.

Bit 7 of the status register, when set to a one, selects the $\overline{RT}$ pin as the clocking source and, when reset to a zero, selects the internal clock as the clocking source. When the RTCC register transitions from a count of 377_8 to a count of 000_8, bit 6 (RTCR) of the status register sets to a one, requesting a real-time clock interrupt. An interrupt to 740_8 is generated if RTCE (bit 4) is set.

The RTCC register can be preset and read under program control at any time. If the RTCC register is not used as a counter, it can be used as a general-purpose data register provided the $\overline{RT}$ pin is tied low and CNT is set to a one. (Note MCLR resets CNT.)

2.6.5 I/O REGISTERS (F5-F7)

The I/O interface consists of three I/O registers controlling 20 input/output lines. These registers (A, B, and C) are addressable as F5 through F7, respectively. Register A (F5) controls four dedicated non-latching input lines. Register B (F6) controls eight dedicated latched output lines, and register C (F7) controls eight bidirectional input/output lines. As with the PIC1655A, register file F10, which in the PIC1650A was I/O register D, is an additional general purpose register in the PIC1656.

2.6.6 CLOCK GENERATOR

The internal timing rate of the PIC1656 is controlled by an external control source connected across two input pins, OSC 1 and OSC 2. This may be established by an RC network (RC control) connected across the OSC 1 and OSC 2 pins or by a non-buffered external crystal connected across the OSC 1 and OSC 2 pins.

The PIC1656 clock generator divides the frequency at the OSC 1 and OSC 2 pins by 16 to derive the internal machine cycle rate. A 4MHz frequency at the OSC 1 and OSC 2 pins will result in a 4μs (0.25MHz) instruction cycle. This enables the use of a low-cost standard 3.58MHz crystal to provide a machine cycle of approximately 4μs.

2.7 PIC1670

ROM	RAM	I/O	Interrupt	Stack (Levels)	Timer	Package	Process
1024 x 13	64 x 8	32	Yes	6	Yes	40 Pin	NMOS

The PIC1670 has several distinct differences from the PIC1650 series. The 13 bit wide ROM enables the PIC1670 to directly address all 64 registers in addition to enhancing the PIC1650 series instruction set. The interrupt system (Fig. 12) is similar to the PIC1656 interrupt system (a separate interrupt status register is added).

2.7.1 INTERRUPT SYSTEM

The interrupt system of the PIC1670 is comprised of an external interrupt and a real-time clock counter interrupt. These have different interrupt vectors, enable bits and status bits. Both interrupts are controlled by the status register (F5)** shown below.

NOT USED	CNTE	A/B	CNTS	RTCIR	XIR	RTCIE	XIE
7*	6	5	4	3	2	1	0

*Bit 7 is unused and is read as zero.

**Register 5 will power up to all zeroes.

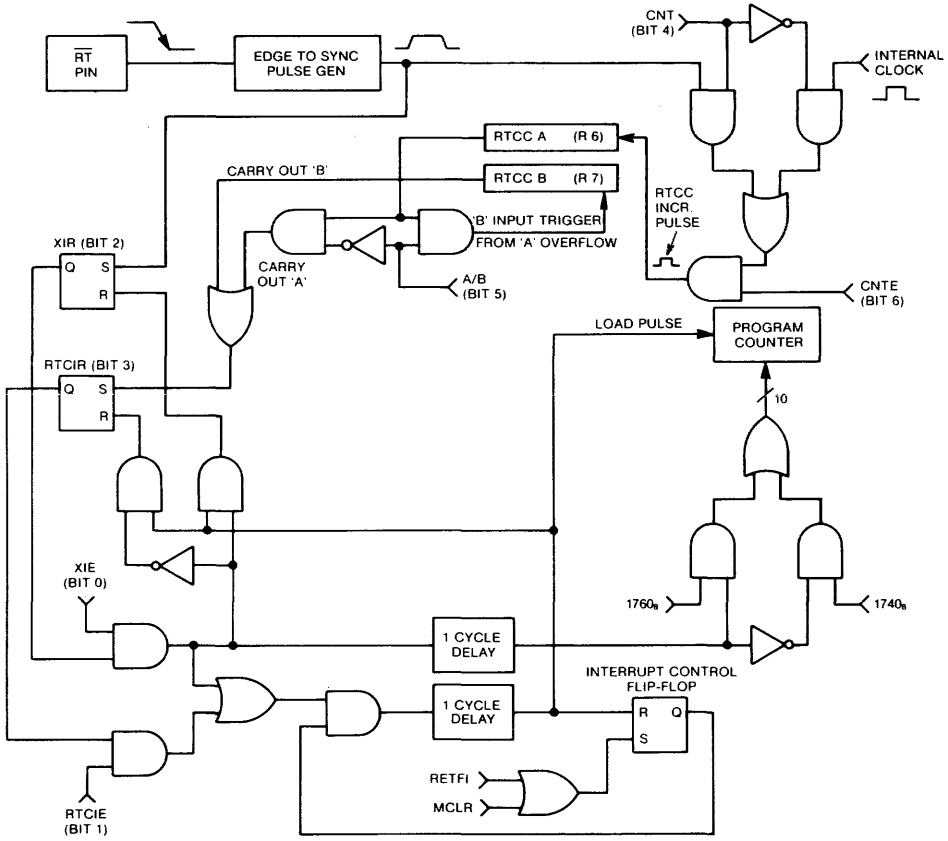
2.7.2 EXTERNAL INTERRUPT

On any high to low transition of the $\overline{RT}$ pin the external interrupt request (XIR) bit will be set. This request will be serviced if the external interrupt enable (XIE) bit is set or if it is set at a later point in the program. The latter allows the processor to store a request (without interrupting) while a critical timing routine is being executed. Once external interrupt service is initiated, the processor will clear the XIR bit, push the current program counter on to the stack and execute the instruction at location 1760_8. This program setup requires two instruction cycles and no new interrupts can be serviced until a return from interrupt (RETFI) instruction has been executed.

2.7.3 REAL-TIME CLOCK INTERRUPT

The real-time clock counter (RTCCA & RTCCB, file registers F6 and F7) have a similar mechanism of interrupt service. The RTCCA register will increment if the count enable (CNTE) bit is set. If this bit is not set the RTCCA & RTCCB will maintain their present contents and can therefore be used as general purpose RAM registers. The count source (CNTS) bit selects the clocking source for RTCCA. If CNTS is cleared to a '0', then RTCCA will use the internal instruction clock and increment at 1/8 the frequency present on the OSC pins. If CNTS is set to a '1', then RTCCA will increment on each high to low transition of the $\overline{RT}$ pin. RTCCB can only be incremented when RTCCA makes a transition from 377_8 to 0 and the A/B status bit is set. This condition links the two eight bit registers together to form one sixteen bit counter. An interrupt request under these conditions will occur when the combined registers make a transition from 177777_8 to 0. If, however, the A/B bit is not set, then RTCCA will be the only incrementing register and an interrupt request will occur when RTCCA makes a transition from 377_8 to 0. (In this setup the RTCCB register will not increment and can be used as a

INTERRUPT SYSTEM BLOCK DIAGRAM, PIC1670

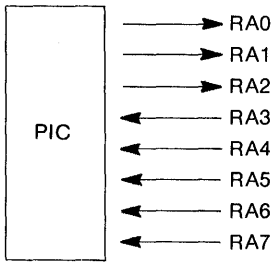


general purpose RAM register). Once a request has come from the real-time clock counter, the real-time clock interrupt request (RTCIR) bit will be set. At this point, the request can either be serviced immediately if the real-time clock interrupt enable (RTCIE) bit is set or be stored if RTCIE is not set. The latter allows the processor to store a real-time clock interrupt while a critical timing routine is being executed. Once interrupt service is initiated, the processor will clear the RTCIR bit, push the present program counter onto the stack and execute the instruction at location 1740_8. This setup requires two instruction cycles and no new interrupts can be serviced until a RETFI instruction has been executed.

The RETFI instruction (00002_8) must be used to return from any interrupt service routine if any pending interrupts are to be serviced. External interrupts have priority over RTCC driven interrupt in the event both types occur simultaneously. Interrupts cannot be nested but will be serviced sequentially. The existence of any pending interrupts can be tested via the state of the XIR (bit 2) and RTCIR (bit 3) in the status word F5.

2.7.4 INPUT/OUTPUT CAPABILITY

The PIC1670 provides four complete quasi-bidirectional input/output ports. A simplified schematic of an I/O pin is shown below. The ports occupy address locations in the register file space of the PIC1670. Thus, any instruction than can operate on a general purpose register can operate on an I/O port. Two locations in the register file space are allocated for each I/O port. Port RA0-7 is addressable as either F10 or F11. Port RB0-7 is addressable as either F12 or F13. Port RC0-7 is addressable as either F14 or F15 and Port RD0-7 is addressable as either F16 or F17. An I/O port READ on its odd-numbered location will interrogate the chip pins while an I/O port READ on its even-numbered location will interrogate the internal latch in that I/O port. This simplifies programming in cases where a portion of a single port is used for inputting only, while the remainder is used for outputting as illustrated in the following example:



Here, the low 3 bits of port RA are used as output-only, while the high 5 bits are used as input-only. During power on reset (MCLR low), the latches in the I/O ports will be set high, turning off all pull down transistors as represented by Q_2 in Figure 13. During program execution if we wish to interrogate an input pin, then, for example,

BTFSS 11,6

will test pin RA6 and skip the next instruction if that pin is set. If we wish to modify a single output, then, for example,

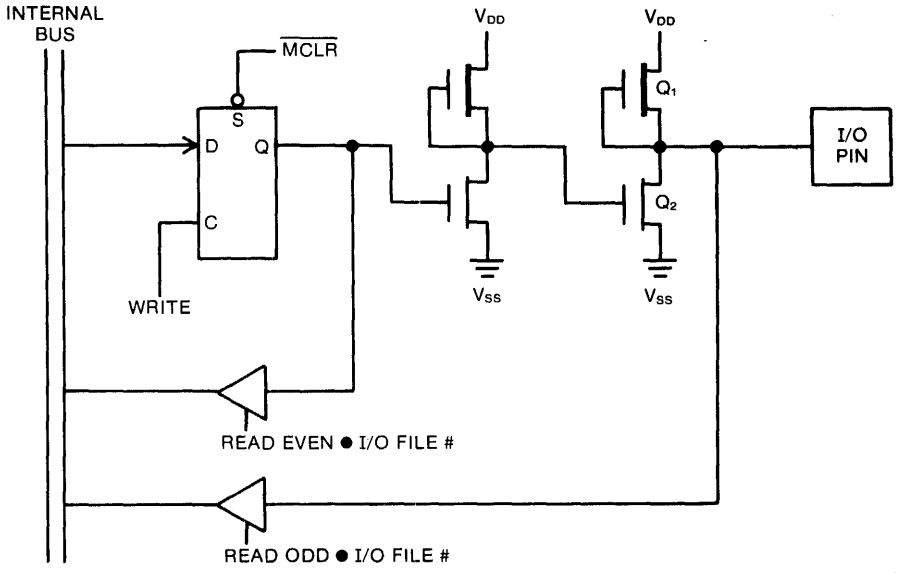
BCF 10,2

will force RA2 to zero because its internal latch will be cleared to zero. This will turn on A_2 and pull the pin to zero.

The way this instruction operates internally is the CPU reads file 10 into the A.L.U., modifies the bit and re-outputs the data to file 10. If the pins were read instead, any input which was grounded externally would cause a zero to be read on that bit. When the CPU re-outputted the data to the file, that bit would be cleared to zero, no longer useful as an input until set high again.

During program execution, the latches in bits 3-7 should remain in the high state. This will keep A_2 off, allowing external circuitry full control of pins RA3-RA7, which are being used here as input.

Fig. 13 **BIDIRECTIONAL INPUT-OUTPUT PORT**



2.8 Pin Assignments

The PIC family is supplied in dual in-line packages with the pin assignments as shown in Figs. 14-19, respectively.

Fig. 14 PIC1650A PIN ASSIGNMENTS

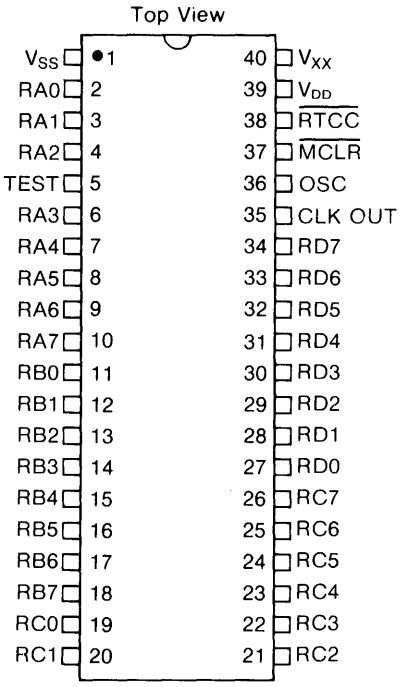


Fig. 15 PIC1654 PIN ASSIGNMENTS

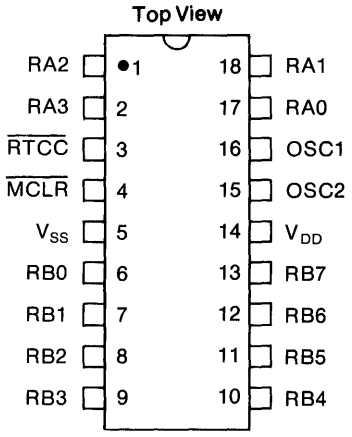


Fig. 16 PIC1655A PIN ASSIGNMENTS

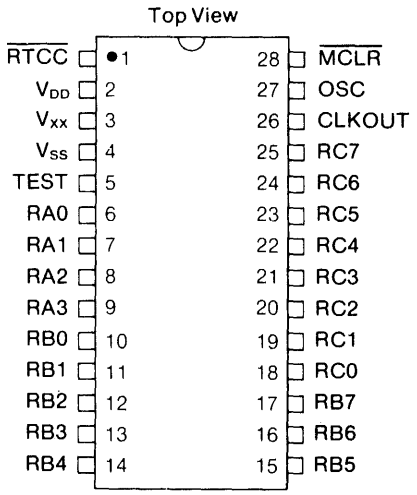


Fig. 17 PIC16C58 PIN ASSIGNMENTS

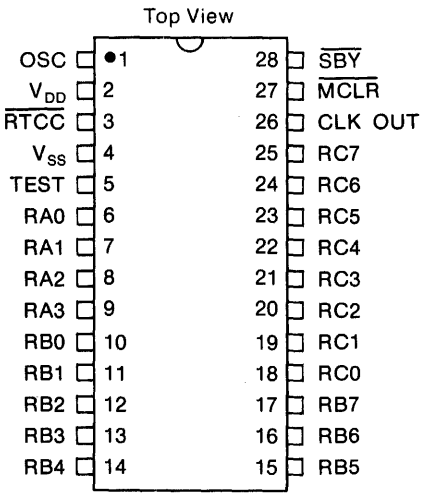


Fig. 18 PIC1656 PIN ASSIGNMENTS

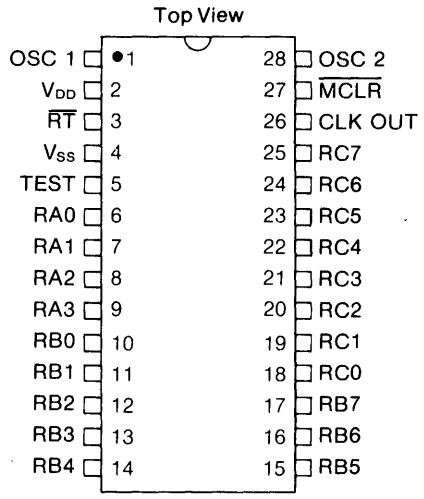
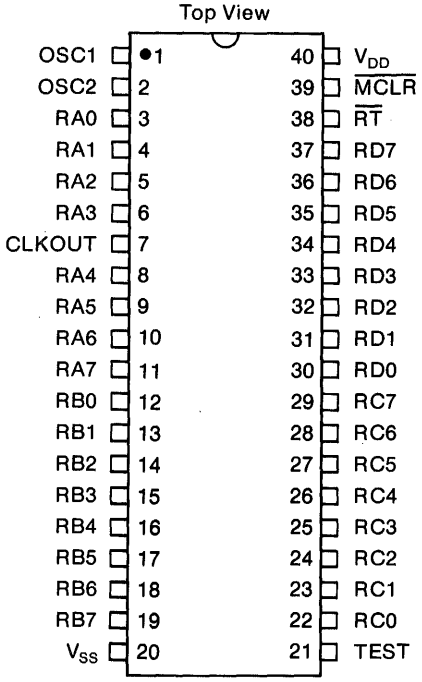


Fig. 19 PIC1670 PIN ASSIGNMENTS



3 INSTRUCTION SET

3.1 General Instruction Format

The PIC1650 series instruction set has a basic repertoire of 30 instruction words. These instructions fall into three general categories:

- General file register operations (byte-oriented)
- Bit level file register operations
- Literal and control operations.

Each instruction word consists of 12 binary bits. The instruction word, when expressed in binary, is also known as a machine code or object code. A certain number of bits in the instruction word are allocated as an operator (OP Code). An OP Code specifies the type of operation to be performed. The balance of the instruction word includes one or more operands which further specify the operation of the instruction.

In general file register operations, six bits are allocated for the OP Code. In bit level file register operations, four bits are allocated; and in control and literal operations, three or four bits are allocated for the OP Code.

The operand field can provide the following information:

- File address of the register from which data is to be obtained.
- File address of the register into which data from the W register is to be written.
- Destination (file register or W register) of the results of an operation.
- Bit number of the bit affected by a bit level file register operation.
- Instruction address to which the program counter will be vectored.
- Literal value stored in the program memory (ROM).

An example of a PIC instruction, in object code, to move a numeric literal, octal 26, to the W register is 110000010110, with the OP Code and operand as follows:

OP Code	Operand
1 1 0 0	0 0 0 1 0 1 1 0

OP Code 1100 specifies that a literal shall be placed in the W register. The operand is the binary equivalent of the literal 26_8. The complete 12-bit binary object code is used by the processor to execute the instruction.

It is normally very difficult for the programmer to read or write more than a few lines of this type of code. Therefore programs are usually written in a symbolic language that is easily understood by the programmer and is also executable by the PIC Cross Assembler.

Using symbolic notation, the OP Code is expressed as a mnemonic. The operand(s) can be expressed in octal, binary, hexadecimal, decimal, or symbolic notation. However, unless otherwise specified, all operands are considered octal.

The PIC object code instruction

1 1 0 0	0 0 0 1 0 1 1 0

just described can be expressed as:

OP Code Operand
MOVLW 26

Other examples of the same instruction with the operand expressed in different notations include:

MOVLW B00010110
where: the B preceding B00010110 specifies binary notation for Octal 26

MOVLW X16
where: the X preceding 16 specifies hexadecimal notation for Octal 26

MOVLW .22
where: the period preceding 22 specifies decimal notation for Octal 26

MOVLW SAMPLE
where: SAMPLE is the symbolic notation for Octal 26. This symbol must be defined in an appropriate place in the program so the assembler can substitute the correct binary value when the notation "SAMPLE" occurs.

The use of an Assembler enables an instruction or group of instructions to be identified by a label. In branch and call instructions, which provide the address of an instruction to be branched to as an operand, the label may be substituted for the address as the operand. This label must exactly match the label of the instruction to which a branch is specified. The Assembler will substitute the proper address in the operand field containing the label.

The instruction to branch to program location 472_8 can be expressed as:

GOTO 472
or
GOTO OVFLO, where OVFLO is the label or symbolic name for the address of the referenced instruction.

In PIC object code, this instruction would be written as:

1 0 1	1 0 0 1 1 1 0 1 0

The Assembler also provides a comments field. This field is for the convenience of the programmer in documenting his program. A typical assembly language line therefore consists of the following:

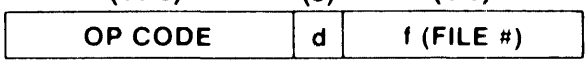
Label	OP Code	Operand	Comments

The label and comments fields are not always used.

All instructions, except for subroutine calls and conditional skips and branches, are executed during one machine cycle. The exceptions are executed in two machine cycles.

PIC1650 SERIES INSTRUCTION SET

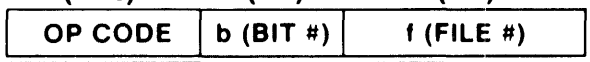
BYTE-ORIENTED FILE REGISTER OPERATIONS



For d = 0, f→W (PICAL accepts d = 0 or d = W in the mnemonic)
d = 1, f→f (If d is omitted, assembler assigns d = 1.)

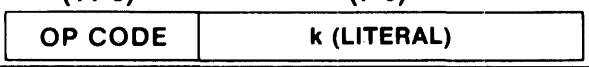
Instruction-Binary (Octal)	Name	Mnemonic, Operands		Operation	Status Affected
000 000 000 000 (0000)	No Operation	NOP	—	—	None
000 000 1ff fff (0040)	Move W to f (Note 1)	MOVWF	f	W→f	None
000 001 000 000 (0100)	Clear W	CLRW	—	0→W	Z
000 001 1ff fff (0140)	Clear f	CLRF	f	0→f	Z
000 010 dff fff (0200)	Subtract W from f	SUBWF	f, d	f - W→d [f+$\overline{W}$+1→d]	C,DC,Z
000 011 dff fff (0300)	Decrement f	DECF	f, d	f - 1→d	Z
000 100 dff fff (0400)	Inclusive OR W and f	IORWF	f, d	WVf→d	Z
000 101 dff fff (0500)	AND W and f	ANDWF	f, d	W•f→d	Z
000 110 dff fff (0600)	Exclusive OR W and f	XORWF	f, d	W⊕f→d	Z
000 111 dff fff (0700)	Add W and f	ADDWF	f, d	W+f→d	C,DC,Z
001 000 dff fff (1000)	Move f	MOVF	f, d	f→d	Z
001 001 dff fff (1100)	Complement f	COMF	f, d	$\overline{f}$→d	Z
001 010 dff fff (1200)	Increment f	INCF	f, d	f+1→d	Z
001 011 dff fff (1300)	Decrement f, Skip if Zero	DECFSZ	f, d	f - 1→d, skip if Zero	None
001 100 dff fff (1400)	Rotate Right f	RRF	f, d	f(n)→d(n-1), f(0)→C, C→d(7)	C
001 101 dff fff (1500)	Rotate Left f	RLF	f, d	f(n)→d(n+1), f(7)→C, C→d(0)	C
001 110 dff fff (1600)	Swap halves f	SWAPF	f, d	f(0-3)⇋f(4-7)→d	None
001 111 dff fff (1700)	Increment f, Skip if Zero	INCFSZ	f, d	f+1→d, skip if zero	None

BIT-ORIENTED FILE REGISTER OPERATIONS



Instruction-Binary (Octal)	Name	Mnemonic, Operands		Operation	Status Affected
010 0bb bff fff (2000)	Bit Clear f	BCF	f, b	0→f(b)	None
010 1bb bff fff (2400)	Bit Set f	BSF	f, b	1→f(b)	None
011 0bb bff fff (3000)	Bit Test f, Skip if Clear	BTFSC	f, b	Bit Test f(b): skip if clear	None
011 1bb bff fff (3400)	Bit Test f, Skip if Set	BTFSS	f, b	Bit Test f(b): skip is set	None

LITERAL AND CONTROL OPERATIONS



Instruction-Binary (Octal)	Name	Mnemonic, Operands		Operation	Status Affected
100 0kk kkk kkk (4000)	Return and place Literal in W	RETLW	k	k→W, Stack→PC	None
100 1kk kkk kkk (4400)	Call subroutine (Note 1)	CALL	k	PC+1 → Stack, k → PC	None
101 kkk kkk kkk (5000)	Go To address (k is 9 bits)	GOTO	k	k→PC	None
110 0kk kkk kkk (6000)	Move Literal to W	MOVLW	k	k→W	None
110 1kk kkk kkk (6400)	Inclusive OR Literal and W	IORLW	k	kVW→W	Z
111 0kk kkk kkk (7000)	AND Literal and W	ANDLW	k	k•W→W	Z
111 1kk kkk kkk (7400)	Exclusive OR Literal and W	XORLW	k	k⊕W→W	Z

3.2 General File Register Operations

This group of instructions is used to operate on data located in any of the file registers including the I/O registers.

Operations performed using general file register instructions include:

- ☐ Two data transfer operations
- ☐ Six arithmetic operations
- ☐ Six logical operations
- ☐ Three rotate operations.

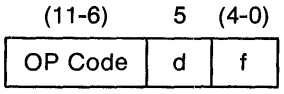
One of two different address modes (direct address or indirect address) is used in a general file register instruction. The most commonly used address mode is direct addressing.

The direct address mode is specified by any file address of one through 37_8 in the operand field. The operation called for by the OP Code will be performed on the data stored in the specified file location.

The indirect address mode is specified by a file address of zero in the operand field. The operation called for by the OP Code will be performed on the data in the file location pointed to by the five LSB's of the file select register, F4. Since the file select register must be loaded under program control, an instruction must be executed to load the FSR with an appropriate file address prior to using the indirect address mode.

For example: Assume that file address F23 has been previously loaded into the FSR. When the indirect address instruction MOVF 0,W is issued, the file select register is pointing at file register F23. The contents of file register 23 are read and transferred to the W register.

The format of the general file register instructions is as follows:



f = file register address (normally expressed in octal notation)
d = destination of result where: 0=W register
1=file register

The instruction may be expressed symbolically as:

OP Code f,d

where: f may be expressed in octal (ASSUMED), or may be expressed in binary, hexadecimal, decimal, or symbolic notation. d may be expressed as a 0 or W for the W register, or as a 1 or a blank for the file register.

Note that if no destination is specified (d operand position left blank), a default value of 1 is assumed (the file register becomes the destination).

Examples: Increment File

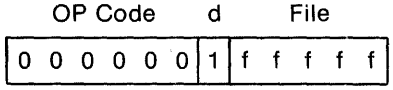
INCF 6,0
INCF 6,W } (F6)+1→W

INCF 6,1
INCF 6 } (F6)+1→F6

3.2.1 DATA TRANSFER OPERATIONS

Two move instructions are provided in the PIC instruction set. One instruction (MOVWF) moves data from the W register to a file register (W → f). The other instruction (MOVF) moves data from a file register to the W register (f → W). A variation of the MOVWF instruction (NOP) is also provided. This instruction is a do-nothing instruction that uses up a period of time equal to one machine cycle.

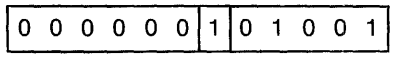
MOVWF f Move Contents of W register to File Register



(W) → f

Status Bits affected: None

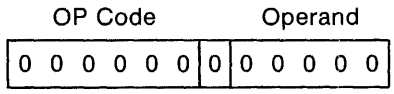
Example: MOVWF 11



(W) → F11

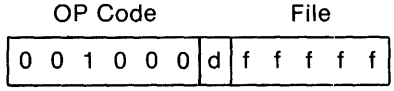
The contents of the W register are moved to file register 11_8.

NOP No Operation



Status bits affected: None

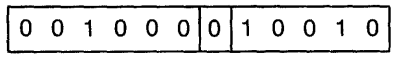
MOVF f,d Move Contents of File Register



(f) → d

Status bits affected: Zero

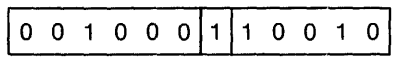
Example: MOVF 22,W



(F22) → W

The contents of file register 22_8 are moved to the W register. If the contents of the register are zero, the Zero status bit will be set.

Example: MOVF 22



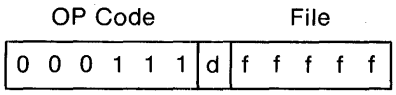
(F22) → F22

The contents of file register 22_8 are moved to the ALU and back to File 22. This instruction can be used to examine the contents of a file register since, if the contents of the register are zero, the Zero status bit will be set.

3.2.2 ARITHMETIC OPERATIONS

Six arithmetic instructions are provided in the PIC instruction set: ADD (ADDWF), Subtract (SUBWF), Increment (INCF), Increment and skip if zero (INCFSZ), Decrement (DECF), and Decrement and skip if zero (DECFSZ). The result of each operation can be placed in the W register or the file register. All arithmetic operations affect the status register. In addition to performing a subtract operation, the SUBWF instruction, when combined with interrogation of the status register, can be used to perform a compare operation. The INCFSZ and DECFSZ instructions are commonly used in loop operations.

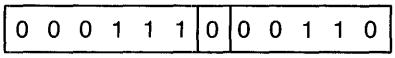
ADDWF f,d Add Contents of W Register to Contents of File Register



(f) + (W) → d

Status bits affected: Carry, Digit Carry, Zero

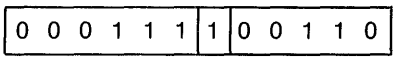
Example: ADDWF 6,W



(F6) + (W) → W

The contents of the W register are added to the contents of file register 6. The result is placed in the W register (d=0). The contents of F6 are not affected.

Example: ADDWF 6



(F6) + (W) → F6

The contents of the W register are added to the contents of file register 6. The result is placed in F6 (d=1). The contents of the W register are not affected.

Assume 242_8 in F6 and 117_8 in the W register to see the effect of the ADDWF instruction on the status bits:

```
  10100010
+ 01001111
------------
  11110001    → F6
```

STATUS

C	DC	Z
0	1	0

The Carry bit is reset, indicating no overflow (sum of 8-bit values in W register and file register $\leq$255). The Digit Carry bit is set indicating a digit overflow (sum of 4 LSB's in W register and file register>15). The Zero bit is reset, indicating that the result of the addition has not provided an 8-bit value of zero.

SUBWF f,d Subtract Contents of W Register from Contents of File Register

OP Code							File					
0	0	0	0	1	0	d	f	f	f	f	f	f

(f) − (W) → d
[in detail,
(f) + ($\overline{W}$) + 1 → d]

Status bits affected: Carry, Digit Carry, Zero

Example: SUBWF 17,W

0	0	0	0	1	0	0	0	1	1	1	1	

(F17) − (W) → W

The contents of the W register are subtracted from the contents of file register 17_8 (using two's complement addition). The result is placed in the W register (d=0). The contents of F17 are not affected.

Assume 104_8 in F17 and 50_8 in the W register to see the effect of the SUBWF instruction on the status bits:

```
 01000100
-00101000
----------
 00011100   → W
```

STATUS

C	DC	Z
1	0	0

The Carry bit is set, indicating no overflow (absolute value in W register not greater than absolute value in F17). The Digit Carry bit is reset, indicating a digit-overflow (absolute value of 4 LSB's of F17 greater than absolute value of 4 LSB's of W Register).

Assume 50_8 in F17 and 104_8 in the W register:

```
 00101000
-01000100
----------
 11100100   → W
```

STATUS

C	DC	Z
0	1	0

The Carry bit is reset, indicating an overflow (absolute value in the W register is greater than absolute value in F17). The Digit Carry bit is set, indicating no digit overflow.

Note that the result obtained when a higher absolute value is subtracted from a lower absolute value is the two's complement of the correct result. In this case 344_8 was obtained which is the two's complement of the actual difference between the two values (34_8, or 000111000). Thus, C = 0 indicates a negative result and it is in two's complement form. Assume 50_8 in F17 and 50_8 in the W register:

```
 00101000
-00101000
----------
 00000000   → W
```

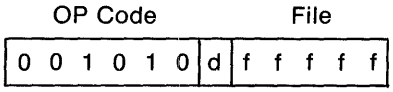
STATUS

C	DC	Z
1	1	1

The SUBWF instruction can be used to compare two values; one in the W register, the other in the file register. After the SUBWF instruction is implemented, it can be determined if the value in W is >, < or = to the value in a file register by testing the status bits as follows:

Condition	True	False
W > F	C=0	C=1
W ≤ F	C=1	C=0
W = F	Z=1	Z=0

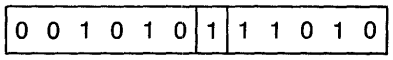
INCF f,d Increment Contents of File Register



(f) + 1 → f

Status bits affected: Zero

Example: INCF 32

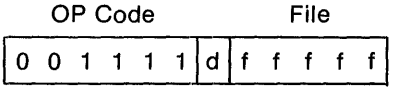


(F32) + 1 → F32

The contents of file register 32_8 are incremented. The result is placed in F32 (d=1). The contents of the W register are not affected.

Assume that the contents of F32 are 01010111 before the INCF instruction. After the INCF instruction, the contents of F32 are 01011000.

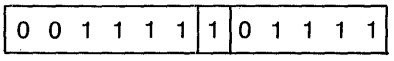
INCFSZ f,d Increment Contents of File Register, Skip If Zero



(f) + 1 → d, skip if zero

Status bits affected: None

Example: INCFSZ 17



(F17) + 1 → F17, skip if zero

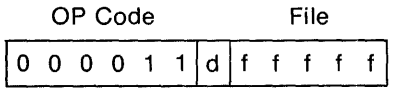
Assume that a table is to be accessed seven times to perform an update operation. The file select register F4 will be loaded with the starting address of the table. The two's complement of the number of passes to be made (loops) will be loaded into $F17_8$.

Aside from the table update, two operations will be performed during each loop: (1) The table address will be incremented; (2) The pass count will be incremented.

```
        MOVLW TABLE     ; Load starting address into W
        MOVWF 4         ; Move starting address into F4 (FSR)
        MOVLW 371       ; Two's complement (octal) of 7
        MOVWF 17        ; Move number of passes into F17
        ----------------
        ----------------
LOOP:   ADDWF 0         ; Add contents of W register to
                        ; contents of table at location
                        ; referenced by FSR.
        ----------------
        ----------------
        INCF 4          ; Increment table address.
        INCFSZ 17       ; Increment count, skip if zero.
        GOTO LOOP
        EXIT
```

At the end of the end of the seventh loop, F17 increments to zero and a skip past the GOTO LOOP instruction is made.

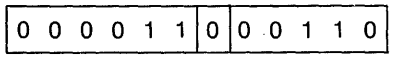
DECF f,d Decrement Contents of File Register



(f) − 1 → d

Status bits affected: Zero

Example: DECF 6,W

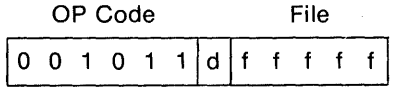


(F6) − 1 → W

The contents of file register 6 are decremented. The result is placed in the W register (d=0). The contents of F6 are not affected.

Assume that the contents of F6 are 00000001 before DECF. When the contents are decremented, the result is 00000000 and the Zero status bit is set.

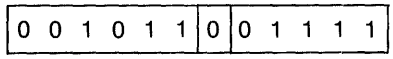
DECFSZ f,d Decrement File Register, Skip If Zero



(f) − 1 → d, skip if zero

Status bits affected: None

Example: DECFSZ 17,W



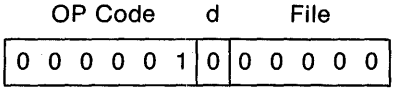
(F17) − 1 → F17, skip if zero

This instruction operates similarly to the INCFSZ instruction in the table update example except that the actual loop count is loaded into the loop register rather than the two's complement of the loop count. On the last loop count, the register decrements to zero and skips the next instruction.

3.2.3 LOGICAL OPERATIONS

Six logical instructions are provided in the PIC instruction set: Clear Contents of W register (CLRW), Clear Contents of File Register (CLRF), AND Contents of W Register and Contents of File Register (ANDWF), Inclusive OR Contents of W Register and Contents of File Register (IORWF), Exclusive OR Contents of W register and Contents of File Register (XORWF) and Complement Contents of File Register (COMF).

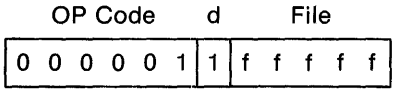
CLRW Clear Contents of W Register



0 → W

Status bits affected: Zero

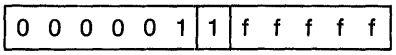
CLRF f Clear Contents of File Register



0 → f

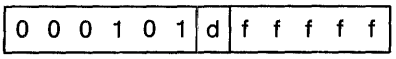
Status bits affected: Zero

Example: CLRF 12



0 → F12

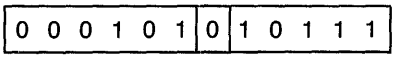
ANDWF f,d AND contents of W Register and Contents of File Register



(W) • (f) → d

Status bits affected: Zero

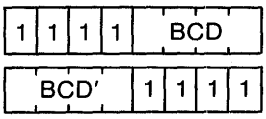
Example: ANDWF 27,W



(W) • (F27) → W

The contents of the W register are ANDed with the contents of file register 27_8. The result is placed in the W register (d=0). The contents of F27 are not affected.

Assume that it is required to pack two bytes of BCD data into one register. The high order bits in the W register and the low order bits in F27 are packed with 1's. When the two registers are ANDed:

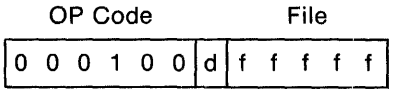


W
•
F27

The result is:

→ W

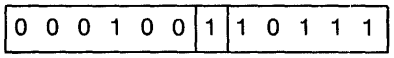
IORWF f,d Inclusive OR Contents of W Register and Contents of File Register



(W) V (f) → d

Status bits affected: Zero

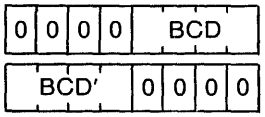
Example: IORWF 27



(W) V (F27) → F27

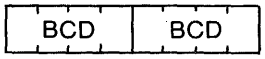
The contents of file register 27_8 are inclusive ORed with the contents of the W register. The result is placed in F27 (d=1). The contents of W are not affected.

Assume that it is required to pack two bytes of BCD data into one register. The high order bits in the W register and the low order bits in F27 are packed with 0's. When the two registers are ORed:



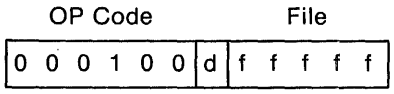
W
V
F27

The result is:



→ W

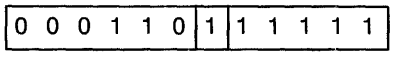
XORWF f,d Exclusive OR Contents of W Register and Contents of File Register



(W) ⊕ (f) → d

Status bits affected: Zero

Example: XORWF 37

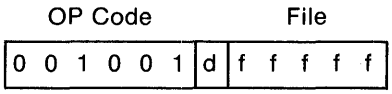


(W) ⊕ (F37) → F37

The contents of the W register are exclusive ORed with the contents of file register 37_8. The result is placed in F37 (d=1). The contents of the W register are not affected.

Assume that it is required to compare the contents of the W register with the contents of F37. If the contents are the same, the result of the Exclusive OR will be zero and the Zero status bit will be set.

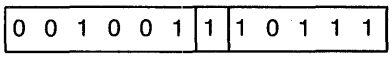
COMF f,d Complement Contents of File Register



$(\bar{f}) \rightarrow d$

Status bits affected: Zero

Example: COMF 27



$(\overline{F27}) \rightarrow F27$

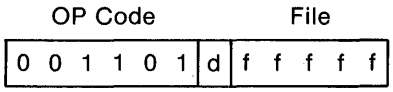
The contents of file register 27_8 are complemented. The result is placed in F27 (d=1). The contents of the W register are not affected.

Assume that the contents of F27 are 01110110 before the COMF instruction. After the COMF instruction is executed, the contents of F27 are 10001001.

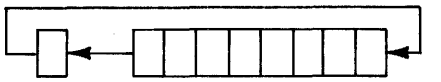
3.2.4 ROTATE OPERATIONS

Three rotate instructions are provided in the PIC instruction set. These instructions permit data in any file register to be rotated left or right. These operations are useful in a wide range of applications, including serial output operations and binary multiplication and division. A special rotate instruction allows two halves within a register to be swapped. This instruction is useful in packing and unpacking data and also aids in BCD arithmetic.

RLF f,d Rotate Contents of File Register Left Through Carry



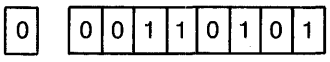
Status bits affected: Carry



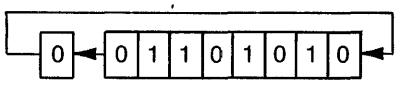
C → d(0), f(6-0) → d(7-1), f(7) →C

Example: RLF 20

Assume the value stored in file register 20_8 is to be doubled, and that the Carry bit has been reset:



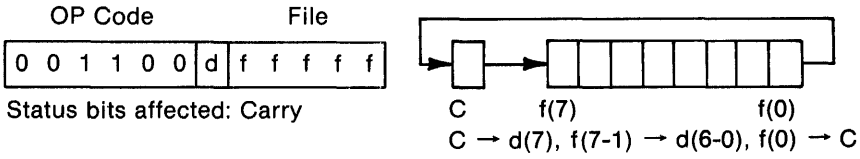
Before Rotate Left



After Rotate Left

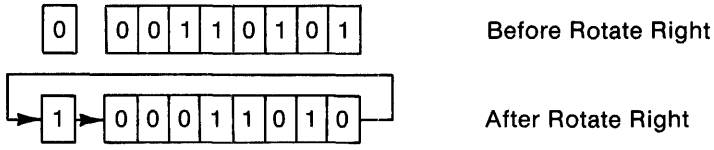
The value stored in F20 has been doubled from 65_8 to 152_8.

RRF f,d Rotate Contents of File Register Right Through Carry



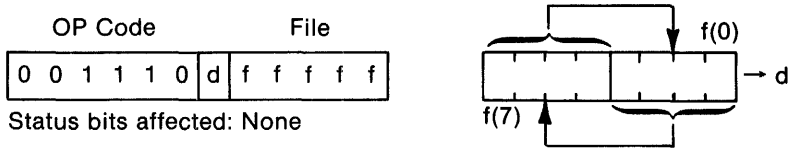
Example: RRF 20

Assume the contents of file register 20_8 are to be serially shifted out, using the Carry bit as the link:

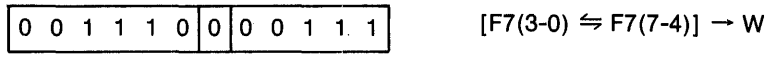


The Carry bit can be interrogated and its contents output after each rotate instruction.

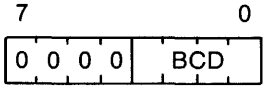
SWAPF f,d Swap halves of File Register



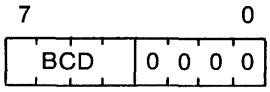
Example: SWAPF 7,W



Assume that it is required to pack two bytes of BCD data into one register. One byte of BCD data is located in F7. Another byte is located in $F10_8$. Each register contains the BCD byte in the four low order bits. Zeros are packed into the four high order bits.



The instruction SWAPF 7,W swaps the BCD byte and the zeros and places the result in the W register:



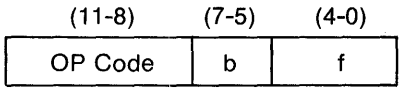
By inclusive ORing the contents of the W register and F10 using the instruction

IORWF 10 the two BCD bytes are packed into F10.

3.3 Bit Level File Register Operations

This group of instructions provides the ability to manipulate and test individual bits in any addressable register. These instructions use the same address modes (direct and indirect) as the general register instructions.

The format of the bit level file register instructions is:



f = file register address
b = bit number

The instruction may be expressed symbolically as:

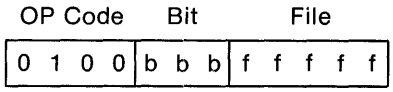
OP Code f,b

where: f and b are expressed in octal (ASSUMED), binary, hexadecimal, decimal, or symbolic notation.

3.3.1 BIT MANIPULATIONS

Two instructions are included in the PIC instruction set to manipulate individual bits in the register file. One instruction (BCF) clears a bit; the other instruction (BSF) sets a bit.

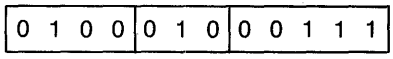
BCF f,b Clear Bit in File Register



0 → f(b)

Status bits affected: None

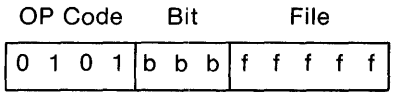
Example: BCF 7,2



0 → F7(2)

Assume that contents of F7 are 11111111 before the BCF instruction. After the BCF instruction, the contents of F7 are 11111011.

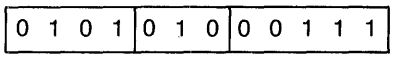
BSF f,b Set Bit in File Register



1 → f(b)

Status bits affected: None

Example: BSF 7,2



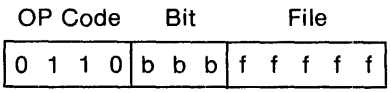
1 → F7(2)

Assume that contents of F7 are 11111011 before the BSF instruction. After the BSF instruction, the contents of F7 are 11111111.

3.3.2 CONDITIONAL SKIPS ON BIT TEST

Two instructions are provided in the PIC instruction set to test an individual bit. One instruction (BTFSC) skips the next instruction if the bit tested is clear (is a zero). The other instruction (BTFSS) skips the next instruction if the bit tested is set (is a one). These instructions are used to interrogate status and flag bits and, based upon the result of the interrogation, go to different points in the program.

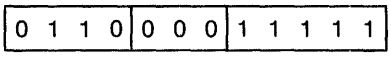
BTFSC f,b Test Bit in File Register, Skip If Clear



Test F(b), skip if clear

Status bits affected: None

Example: BTFSC 37,0



Test F37(0), skip if clear

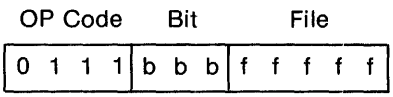
The content of bit 0 of file register 37_8 is tested. If bit 0 is a zero, the next instruction is skipped.

Assuming that bit 0 of F37 is an overflow bit, coding might be written as follows:

```
BTFSC 37,0
INCF 23
GOTO SCAN
```

If there is an overflow, F23 is incremented before going to SCAN routine. If there is no overflow, F23 is not incremented.

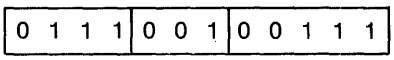
BTFSS f,b Test Bit in File Register, Skip if Set



Test F(b), skip if set

Status bits affected: None

Example: BTFSS 7,1



Test F7(1), skip if set

The contents of bit 1 of file register 7_8 is tested. If bit 1 is a one, the next instruction is skipped.

Assuming that bit 1 of F7 is an input flag bit, coding might be written as follows:

```
BTFSS 7,1
GOTO CALC
GOTO INPUT
```

If bit 1 is set, the program will jump to the INPUT routine. If bit 1 is clear, the program will jump to the CALC routine.

3.4 Literal and Control Operations

This group of instructions is used to operate on literals located in program memory or to branch to or call instructions located in program memory.

Operations performed using literal instructions are:

- ☐ Move literal to W
- ☐ Logical operations on literals

Operations performed using control instructions are:

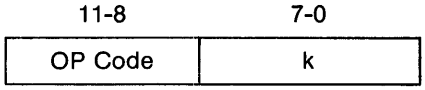
- ☐ Jump
- ☐ Calls and Returns

The literal and control instructions employ immediate addressing. The instruction word consists of an OP Code (three or four high order bits) immediately followed by an 8 or 9-bit literal (constant). This literal can be used as an operand in arithmetic and logical operations.

3.4.1 LITERAL OPERATIONS

Four literal instructions are provided in the PIC instruction set. One instruction (MOVLW) moves a literal to the W register. The other three instructions (IORLW, XORLW, and ANDLW) perform a logical operation between the literal and the contents of the W register.

The format of the literal instructions is as follows:

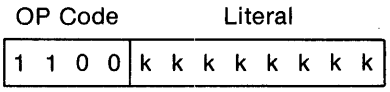


The instruction may be expressed symbolically as:

OP Code k

where: k is expressed in octal (ASSUMED), binary, hexadecimal, decimal, or symbolic notation.

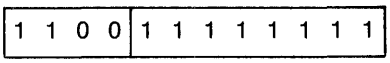
MOVLW k Move Literal k to W Register



k → W

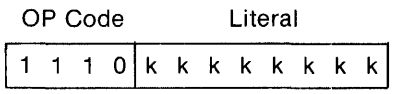
Status bits affected: None

Example: MOVLW 377



377_8 → W

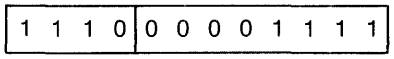
ANDLW k AND Literal k and Contents of W Register



k•(W) → W

Status bits affected: Zero

Example: ANDLW 17

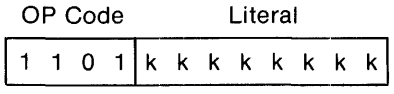


17_8• (W) → W

The four MSBs in the W register are masked by ANDing them with the zeros in the four MSBs of the literal. The four LSBs of the W register are not affected.

Assume that the contents of the W register are 01001001 before the ANDLW 17 instruction. After the ANDLW 17 instruction, the contents of the W register are 00001001.

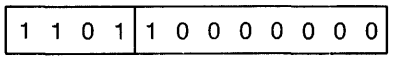
IORLW k Inclusive OR Literal k and Contents of W Register



k V (W) → W

Status bits affected: Zero

Example: IORLW 200

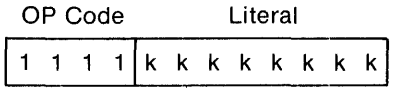


200_8 V (W) → W

Assume that it is required to change the sign bit from positive to negative during an arithmetic operation. By inclusive ORing 200_8 (1000000) with the contents of the W register, the sign bit (MSB) will be set to 1 (negative sign).

Assume that contents of W register are 01101110 before the IORLW 200 instruction. After the IORLW 200 instruction, the contents of the W register are 11101110.

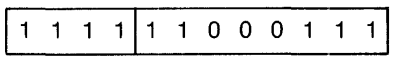
XORLW k Exclusive OR Literal k and Contents of W Register



k ⊕ (W) → W

Status bits affected: Zero

Example: XORLW 307



307_8 ⊕ (W) → W

307_8 is Exclusive ORed with the contents of the W register. If the contents are the same, the result of the Exclusive OR will be zero and the Zero status bit will be set to a one.

3.4.2 CONTROL OPERATIONS

Four control instructions are provided in the PIC instruction set for jumps, calls, and returns. One instruction (GOTO) is an unconditional jump (branch). The address of the instruction to be branched to is loaded into the program counter.

The call and return instructions are provided for calling subroutines and returning to the main program. The CALL instruction pushes the address of the location immediately following the CALL instruction (PC + 1) onto the stack before the address of the subroutine is loaded into the program counter.

Two return instructions are provided. One of these, RETURN, is a special instruction for the PIC1656 that provides for a return from interrupt. The other return instruction, RETLW, is a return from subroutine instruction. All return instructions pop the return address off the Stack and into the program counter. In addition, RETLW moves a literal that is specified by the operand into the W register, and RETURN allows any pending interrupt request to proceed.

In the PIC1656, the RETURN instruction (return from interrupt) can also be used as a return from subroutine, with the W register unaffected. This instruction must not be used instead of RETLW as a return from subroutine during an interrupt service routine since only RETURN enables further interrupts.

GOTO k Go to address k (Note that k for this instruction is 9 bits)

OP Code	Address	
1 0 1	k k k k k k k k k	k → PC

Status bits affected: None

Example: GOTO 677

1 0 1	1 1 0 1 1 1 1 1 1	677_8 → PC

CALL k Call Subroutine at Address k

OP Code	Address	
1 0 0 1	k k k k k k k k	(PC) + 1 → Stack
		k → PC

Status bits affected: None

This instruction increments the contents of the program counter by one and places the result (PC + 1) into the stack. Then the subroutine address specified in the program is placed in the program counter. The program executes at this location.

NOTE: Any instruction address up to 377_8 can be represented by an 8-bit binary number (377_8 = 11111111). Any address past 377_8 requires a ninth bit. The ninth bit of the program counter is a zero for a CALL or MOVWF F2 instruction. **THEREFORE, SUBROUTINES MUST BE LOCATED IN PROGRAM MEMORY LOCATIONS 0-377_8.** However, subroutines can be called from anywhere in the program memory since the Stack is 9 bits wide (Not a restriction in PIC1670).

Example: CALL 256

1 0 0 1	1 0 1 0 1 1 1 0

Assume program is at location 417:

417 + 1 → Stack
256 → PC

RETLW k Return and Place Literal k in W Register

OP Code	Literal
1 0 0 0	k k k k k k k k

k → W
(Stack) → PC

Status bits affected: None

This command is used at the end of a subroutine to return to the address immediately following the CALL instruction. The contents of the top level of the Stack are popped off and placed in the program counter. The literal value is placed in the W register.

RETFI Return From Interrupt (PIC1656 only)

OP Code	Operand
0 0 0 0	0 0 0 0 0 0 1 0

(Stack) → PC

Status bits affected: None

This command is used at the end of an interrupt routine to return to the address immediately following the interrupt. The contents of the top level of the Stack are popped off and placed in the program counter. The contents of the W register are not affected. Any pending interrupt is enabled.

3.5 Special Instruction Mnemonics

Frequently used operations such as conditional skips and branches on status bit test, two's complement register contents, carry and digit carry addition can all be performed using file, bit, literal and control instructions in combination with the specific operands required.

These operations can be performed using special mnemonics that are recognized by the PIC Assembler. These mnemonics do not imply that there are additional instruction words. Each of these special mnemonics calls up one or more or the PIC instructions. The Assembler inserts the proper operands required for specific locations and destinations.

Special instruction mnemonics are provided for the following operations:

- ☐ Move file to W register
- ☐ Test file
- ☐ Two's complement file register contents
- ☐ Unconditional branch
- ☐ Six status bit manipulations
- ☐ Six conditional skips on status bit test
- ☐ Six conditional branches on status bit test
- ☐ Four Carry and Digit Carry arithmetic operations.

3.5.1 MOVE FILE TO W REGISTER

A special instruction mnemonic is provided to move the contents of file register to the W register.

MOVFW f Move Contents of File Register to W

OP Code						d	File					
0	0	1	0	0	0	0	f	f	f	f	f	(f) → W

Status bits affected: Zero

Equivalent file operation: MOVF f,0

3.5.2 TEST FILE

One special instruction mnemonic is provided to test the contents of a file register for zero. This instruction moves the contents of a file register back into itself. In the process, the Zero status bit is set to a one if the contents of the file are zero.

TSTF f Test Contents of File Register

OP Code						d	File					
0	0	1	0	0	0	1	f	f	f	f	f	(f) → f

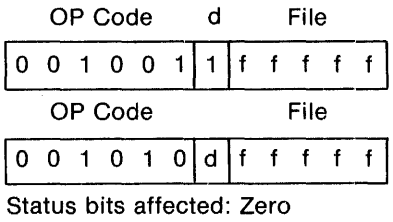
Status bits affected: Zero

Equivalent file operation: MOVF f,1

3.5.3 TWO's COMPLEMENT REGISTER CONTENTS

A special instruction mnemonic is provided to obtain the two's complement of the contents of a file register. This mnemonic calls up two instructions. The first instruction complements the contents of the addressed file register. The second instruction adds binary 1 to the least significant bit.

NEGF f,d Negate File Register Contents



$\overline{(f)} \rightarrow f$

$(f) + 1 \rightarrow d$

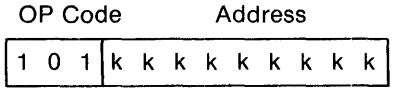
Status bits affected: Zero

Equivalent file operations: COMF f,1
INCF f,d

3.5.4 UNCONDITIONAL BRANCH

A special instruction mnemonic is provided for an unconditional branch instruction.

B k Branch to Address k (Note that k for this instruction is 9 bits)



$k \rightarrow PC$

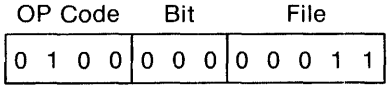
The 9-bit instruction address is placed into the program counter, causing the program to jump to that location.

Equivalent control operation: GOTO k

3.5.5 STATUS BIT MANIPULATIONS

Six special instruction mnemonics are provided to set and clear the Carry, Digit Carry, and Zero status bits.

CLRC Clear Carry



$0 \rightarrow F3(0)$

Bit 0 (Carry bit) of status register F3 is cleared to a zero.

Equivalent bit operation: BCF 3,0

SETC Set Carry

OP Code	Bit	File	
0 1 0 1	0 0 0	0 0 0 1 1	1 → F3(0)

Bit 0 (Carry bit) of status register F3 is set to a one.
Equivalent bit operation: BSF 3,0

CLRDC Clear Digit Carry

OP Code	Bit	File	
0 1 0 0	0 0 0	0 0 0 1 1	0 → F3(1)

Bit 1 (Digit Carry bit) of status register F3 is cleared to a zero.
Equivalent bit operation: BCF 3,1

SETDC Set Digit Carry

OP Code	Bit	File	
0 1 0 1	0 0 1	0 0 0 1 1	1 → F3(1)

Bit 1 (Digit Carry bit) of status register F3 is set to a one.
Equivalent bit operation: BSF 3,1

CLRZ Clear Zero

OP Code	Bit	File	
0 1 0 0	0 1 0	0 0 0 1 1	0 → F3(2)

Bit 2 (Zero bit) of status register F3 is cleared to a zero.
Equivalent bit operation: BCF 3,2

SETZ Set Zero

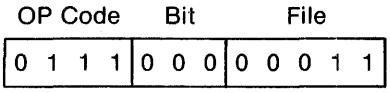
OP Code	Bit	File	
0 1 0 1	0 1 0	0 0 0 1 1	1 → F3(2)

Bit 2 (Zero bit) of status register F3 is set to a one.
Equivalent bit operation: BCF 3,2

3.5.6 CONDITIONAL SKIPS ON STATUS BIT TEST

Six special instruction mnemonics are provided for a skip operation that is conditional on the result of a status bit test.

SKPC Skip On Carry

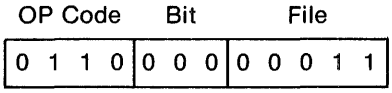


Test Carry, skip if set

Bit 0 (Carry bit) of the status register F3 is tested. If it is a one, the next instruction is skipped.

Equivalent bit operation: BTFSS 3,0

SKPNC Skip On No Carry

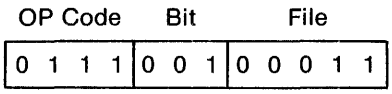


Test Carry, skip if reset

Bit 0 (Carry bit) of status register F3 is tested. If it is a zero, the next instruction is skipped.

Equivalent bit operation: BTFSC 3,0

SKPDC Skip On Digit Carry

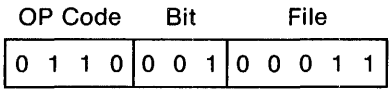


Test Digit Carry, skip if set

Bit 1 (Digit Carry bit) of status register F3 is tested. If it is a one, the next instruction is skipped.

Equivalent bit operation: BTFSS 3,1

SKPNDC Skip On No Digit Carry

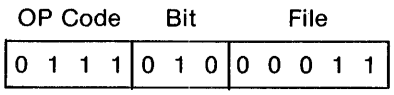


Test Digit Carry, skip if reset

Bit 1 (Digit Carry bit) of status register F3 is tested. If it is a zero, the next instruction is skipped.

Equivalent bit operation: BTFSC 3,1

SKPZ Skip On Zero

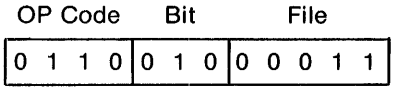


Test Zero bit, skip if set

Bit 2 (Zero bit) of status register F3 is tested. If it is a one, the next instruction is skipped.

Equivalent bit operation: BTFSS 3,2

SKPNZ Skip On No Zero



Test Zero bit, skip if reset

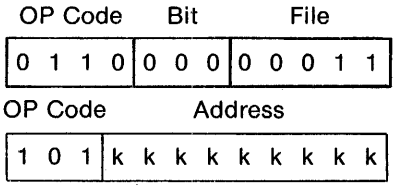
Bit 1 (Zero bit) of status register F3 is tested. If it is a zero, the next instruction is skipped.

Equivalent Bit operation: BTFSC 3,2

3.5.7 CONDITIONAL BRANCHES ON STATUS BIT TEST

Six special instruction mnemonics are provided for branch operations conditional on the result of a status bit test. Each of these mnemonics calls two instructions. The first instruction tests the status bit. If the required condition is present, the second instruction places the specified 9-bit program address in the program counter, causing a program jump to this address. If the required status condition is not present, the jump instruction (GOTO) is skipped and the program continues.

BC k Branch On Carry to Address k



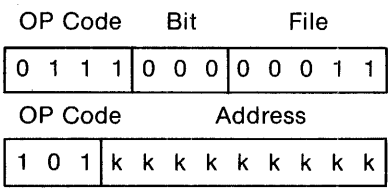
Skip if Carry is clear

k → PC

The Carry bit is tested. If it is a zero, the GOTO instruction is skipped. If it is one, the 9-bit instruction address (k) is placed in the program counter, causing the program to jump to that location.

Equivalent bit and control operations: BTFSC 3,0
GOTO k

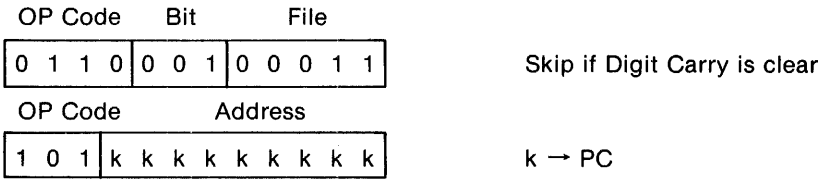
BNC k Branch On No Carry to Address k



The Carry status bit is tested. If it is a one, the GOTO instruction is skipped. If it is a zero, the 9-bit instruction address (k) is placed in the program counter, causing the program to jump to that location.

Equivalent bit and control operations: BTFSS 3,0
GOTO k

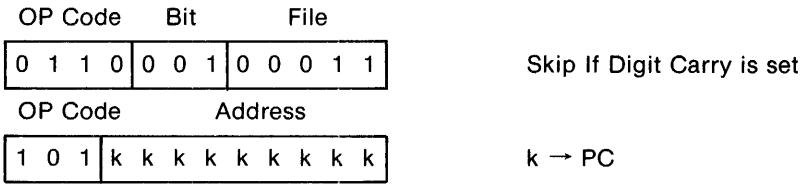
BDC k Branch On Digit Carry to Address k



The Digit Carry status bit is tested. If it is zero, the GOTO instruction is skipped. If it is a one, the 9-bit instruction address (k) is placed in the program counter, causing the program to jump to that location.

Equivalent bit and control operations: BTFSC 3,1
GOTO k

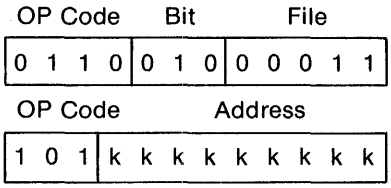
BNDC k Branch On No Digit Carry to Address k



The Digit Carry status bit is tested. If it is a one, the GOTO instruction is skipped. If it is a zero, the 9-bit instruction address (k) is placed in the program counter, causing the program to jump to that location.

Equivalent bit and control operations: BTFSS 3,1
GOTO k

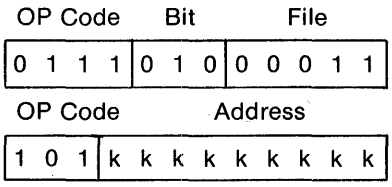
BZ k Branch On Zero to Address k



The Zero status bit is tested. If it is a zero, the GOTO instruction is skipped. If it is a one, the 9-bit instruction address (k) is placed in the program counter, causing the program to jump to that location.

Equivalent bit and control operations: BTFSC 3,2
GOTO k

BNZ k Branch On No Zero to Address



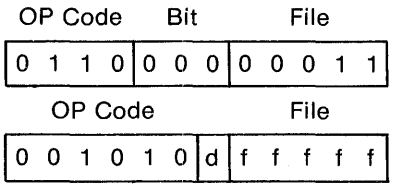
The Zero status bit is tested. If it is a one, the GOTO instruction is skipped. If it is to zero, the 9-bit instruction address (k) is placed in the program counter, causing the program to jump to that location.

Equivalent bit and control operations: BTFSS 3,2
GOTO k

3.5.8 CARRY AND DIGIT CARRY ARITHMETIC

Four special instruction mnemonics are provided to add the Carry or Digit Carry bits to a file register or to subtract the Carry or Digit Carry bits from a file register. Each of these mnemonics calls up two instructions. The first instruction tests the content of the Carry or Digit Carry bit. If the content is a one, the second instruction increments or decrements the file register. If the content is a zero, the second instruction is skipped.

ADDCF f,d Add Carry to Contents of File Register

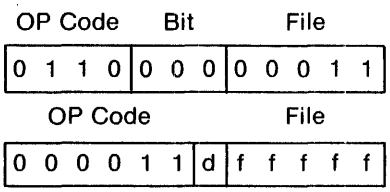


Status bits affected: Zero

The Carry status bit is tested. If it is a zero, the increment instruction is skipped. If it is a one, the file register is incremented.

Equivalent bit and file operations: BTFSC 3,0
INCF f,d

SUBCF f,d Subtract Carry From Contents of File Register

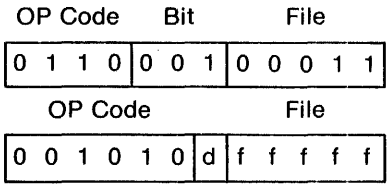


Status bits affected: Zero

The Carry status bit is tested. If it is a zero, the decrement instruction is skipped. If it is a one, the file register is decremented.

Equivalent bit and file operations: BTFSC 3,0
DECF f,d

ADDDCF f,d Add Digit Carry to Contents of File Register

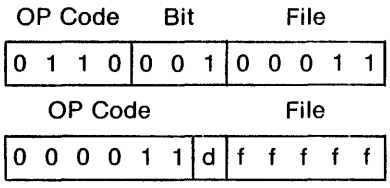


Status bits affected: Zero

The Digit Carry status bit is tested. If it is a zero, the increment instruction is skipped. If it is a one, the file register is incremented.

Equivalent bit and file operations: BTFSC 3,1
INCF f,d

SUBDCF f,d Subtract Digit Carry From Contents of File Register



Status bits affected: Zero

The Digit Carry status bit is tested. If it is a zero, the decrement instruction is skipped. If it is a one, the file register is incremented.

Equivalent bit and file operations: BTFSC 3,1
DECF f,d

SUPPLEMENTAL INSTRUCTION SET SUMMARY

The following supplemental instructions summarized below represent specific applications of the basic PIC instructions. For example, the "CLEAR CARRY" supplemental instruction is equivalent to the basic instruction BCF 3,0 ("Bit Clear, File 3, Bit 0"). These instruction mnemonics are recognized by the PIC Cross Assembler (PICAL).

Instruction-Binary (Octal)	Name	Mnemonic, Operands	Equivalent Operation(s)	Status Affected
010 000 000 011 (2003)	Clear Carry	CLRC	BCF 3, 0	—
010 100 000 011 (2403)	Set Carry	SETC	BSF 3, 0	—
010 000 100 011 (2043)	Clear Digit Carry	CLRDC	BCF 3, 1	—
010 100 100 011 (2443)	Set Digit Carry	SETDC	BSF 3, 1	—
010 001 000 011 (2103)	Clear Zero	CLRZ	BCF 3, 2	—
010 101 000 011 (2503)	Set Zero	SETZ	BSF 3, 2	—
011 100 000 011 (3403)	Skip on Carry	SKPC	BTFSS 3, 0	—
011 000 000 011 (3003)	Skip on No Carry	SKPNC	BTFSC 3, 0	—
011 100 100 011 (3443)	Skip on Digit Carry	SKPDC	BTFSS 3, 1	—
011 000 100 011 (3043)	Skip on No Digit Carry	SKPNDC	BTFSC 3, 1	—
011 101 000 011 (3503)	Skip on Zero	SKPZ	BTFSS 3, 2	—
011 001 000 011 (3103)	Skip on No Zero	SKPNZ	BTFSC 3, 2	—
001 000 1ff fff (1040)	Test File	TSTF f	MOVF f, 1	Z
001 000 0ff fff (1000)	Move File to W	MOVFW f	MOVF f, 0	Z
001 001 1ff fff (1140) 001 010 dff fff (1200)	Negate File	NEGF f,d	COMF f, 1 INCF f, d	Z
011 000 000 011 (3003) 001 010 dff fff (1200)	Add Carry to File	ADDCF f, d	BTFSC 3,0 INCF f, d	Z
011 000 000 011 (3003) 000 011 dff fff (0300)	Subtract Carry from File	SUBCF f,d	BTFSC 3,0 DECF f, d	Z
011 000 100 011 (3043) 001 010 dff fff (1200)	Add Digit Carry to File	ADDDCF f,d	BTFSG 3,1 INCF f,d	Z
011 000 100 011 (3043) 000 011 dff fff (0300)	Subtract Digit Carry from File	SUBDCF f,d	BTFSC 3,1 DECF f,d	Z
101 kkk kkk kkk (5000)	Branch	B k	GOTO k	—
011 000 000 011 (3003) 101 kkk kkk kkk (5000)	Branch on Carry	BC k	BTFSC 3,0 GOTO k	—
011 100 000 011 (3403) 101 kkk kkk kkk (5000)	Branch on No Carry	BNC k	BTFSS 3,0 GOTO k	—
011 100 100 011 (3043) 101 kkk kkk kkk (5000)	Branch on Digit Carry	BDC k	BTFSC 3,1 GOTO k	—
011 001 000 011 (3443) 101 kkk kkk kkk (5000)	Branch on No Digit Carry	BNDC k	BTFSS 3,1 GOTO k	—
011 101 000 011 (3103) 101 kkk kkk kkk (5000)	Branch on Zero	BZ k	BTFSC 3,2 GOTO k	—
011 101 000 011 (3503) 101 kkk kkk kkk (5000)	Branch on No Zero	BNZ k	BTFSS 3,2 GOTO k	—

3.6 PIC1670 Series Instruction Set

The PIC1670 series instruction set is a superset of the PIC1650 series instruction set — the software is upwardly compatible.

BYTE ORIENTED FILE REGISTER OPERATIONS

(12-7)	(6)	(5-0)
OP CODE	d	f (FILE #)

Instruction—Binary (Octal)					Name	Mnemonic,	Operands	Operation	Status Affected
0 0 0 0	0 0 0 0	0 0 0 1 0 0	(00004)	Decimal adjust W	DAW	—	(Note 1)	C
0 0 0 0	0 0 1 f f f	f f f	(00100)	Move W to file	MOVWF	f	W→f	—
0 0 0 0	1 d f f f	f f f	(00200)	Subtract W from file w/borrow	SUBBWF	f,d	f+$\overline{W}$+c →d	OV,C,DC,Z
0 0 0 0	1 0 d f f f	f f f	(00400)	Subtract W from file	SUBWF	f,d	f+$\overline{W}$ + 1→d	OV,C,DC,Z
0 0 0 0	1 1 d f f f	f f f	(00600)	Decrement file	DECF	f,d	f - 1→d	OV,C,DC,Z
0 0 0 1	0 0 d f f f	f f f	(01000)	Inclusive or W with file	IORWF	f,d	WVf→d	Z
0 0 0 1	0 1 d f f f	f f f	(01200)	And W with file	ANDWF	f,d	W•f→d	Z
0 0 0 1	1 0 d f f f	f f f	(01400)	Exclusive OR W with file	XORWF	f,d	W⊕f→d	Z
0 0 0 1	1 1 d f f f	f f f	(01600)	Add W with file	ADDWF	f,d	W+f→d	OV,C,DC,Z
0 0 1 0	0 0 d f f f	f f f	(02000)	Add W to file with carry	ADCWF	f,d	W+f+c→d	OV,C,DC,Z
0 0 1 0	0 1 d f f f	f f f	(02200)	Complement file	COMPF	f,d	$\overline{f}$→d	Z
0 0 1 0	1 0 d f f f	f f f	(02400)	Increment file	INCF	f,d	f+1→d	OV,C,DC,Z
0 0 1 0	1 1 d f f f	f f f	(02600)	Decrement file, skip if zero	DECFSZ	f,d	f - 1→ d, skip if zero	—
0 0 1 1	0 0 d f f f	f f f	(03000)	Rotate file right thru carry	RRCF	f,d	f(n)→d(n−1), c→d(7), f(0)→c	C
0 0 1 1	0 1 d f f f	f f f	(03200)	Rotate file left thru carry	RLCF	f,d	f(n)→d(n+1), c→d(0), f(7)→c	C
0 0 1 1	1 0 d f f f	f f f	(03400)	Swap upper and lower nibble of file	SWAPF	f,d	f(0-3)⇌(4-7)→d	—
0 0 1 1	1 1 d f f f	f f f	(03600)	Increment file, skip if zero	INCFSZ	f,d	f+1→d, skip if zero	—

(12-6)	(5-0)
OP CODE	f (FILE #)

Instruction—Binary (Octal)			Name	Mnemonic,	Operands	Operation	Status Affected
1 0 0 0	0 0 0 f f f	f f f	(10000)	Move file to W	MOVFW	f	f→W	Z
1 0 0 0	0 0 1 f f f	f f f	(10100)	Clear file	CLRF	f	0→f	Z
1 0 0 0	0 1 0 f f f	f f f	(10200)	Rotate file right/no carry	RRNCF	f	f(n)→d(n−1), f(0), →f(7)	—
1 0 0 0	0 1 1 f f f	f f f	(10300)	Rotate file left/no carry	RLNCF	f	f(n)→d(n+1), f(7), →f(0)	—
1 0 0 0	1 0 0 f f f	f f f	(10400)	Compare file to W, skip if F < W	CPFSLT	f	f - W, Skip if C = 0	—
1 0 0 0	1 0 1 f f f	f f f	(10500)	Compare file to W, skip if F = W	CPFSEQ	f	f - W, Skip if $\overline{Z}$ = 1	—
1 0 0 0	1 1 0 f f f	f f f	(10600)	Compare file to W, skip if F > W	CPFSGT	f	f - W, Skip if $\overline{Z}$ • C = 1	—
1 0 0 0	1 1 1 f f f	f f f	(10700)	Move file to itself	TESTF	—	f→f	Z

BIT ORIENTED FILE REGISTER OPERATIONS

(12-9)	(8-6)	(5-0)
OP CODE	b (BIT #)	f (FILE #)

Instruction—Binary (Octal)			Name	Mnemonic,	Operands	Operation	Status Affected
0 1 0 0	b b b f f f	f f f	(04000)	Bit clear file	BCF	f,b	0→f(b)	—
0 1 0 1	b b b f f f	f f f	(05000)	Bit set file	BSF	f,b	1→f(b)	—
0 1 1 0	b b b f f f	f f f	(06000)	Bit test, skip if clear	BTFSC	f,b	Bit Test f(b): skip if clear	—
0 1 1 1	b b b f f f	f f f	(07000)	Bit test, skip if set	BTFSS	f,b	Bit Test f(b): skip if set	—

LITERAL AND CONTROL OPERATIONS

(12-8)	(7-0)
OP CODE	k (LITERAL)

Instruction—Binary (Octal)			Name	Mnemonic,	Operands	Operation	Status Affected
0 0 0 0	0 0 0 0 0 0 0 0	(00000)	No Operation	NOP	—	—	—
0 0 0 0	0 0 0 0 0 0 0 1	(00001)	Halt in PIC1665	HALT	—	—	—
0 0 0 0	0 0 0 0 0 0 1 0	(00002)	Return from Interrupt	RETFI	—	Stack → PC	—
0 0 0 0	0 0 0 0 0 0 1 1	(00003)	Return from Subroutine	RETFS	—	Stack → PC	—
1 0 0 1	0 k k k k k k k	(11000)	Move Literal to W	MOVLW	k	k→W	—
1 0 0 1	1 k k k k k k k	(11400)	Add Literal to W	ADDLW	k	k+W→W	OV,C,DC,Z
1 0 1 0	0 k k k k k k k	(12000)	Inclusive OR Literal to W	IORLW	k	kVW→W	Z
1 0 1 0	1 k k k k k k k	(12400)	And Literal and W	ANDLW	k	k•W→W	Z
1 0 1 1	0 k k k k k k k	(13000)	Exclusive OR Literal and W	XORLW	k	k⊕W→W	Z
1 0 1 1	1 k k k k k k k	(13400)	Return and load literal in W	RETLW	k	k→W, Stack →PC	—

(12-10)	(9-0)
OP CODE	k (LITERAL)

Instruction—Binary (Octal)			Name	Mnemonic,	Operands	Operation	Status Affected
1 1 0 k	k k k k k k k k	(14000)	Go to address	GOTO	k	k→PC	—
1 1 1 k	k k k k k k k k	(16000)	Call Subroutine	CALL'	k	PC+1→Stack, k→PC	—

NOTE: If the lower nibble is greater than 9 or the digit carry flag (DC) is set, 06 is added to the W register.

3.6.1 ADDITIONAL INSTRUCTIONS

ADCWF f,d Add with carry

OP Code							FK						
0	0	1	0	0	0	d	f	f	f	f	f	f	(f) + (W) + C → d

Status bits affected: OV, C, DC, Z

Example: ADDWF 6 (F6) + (W) + C → F6

0	0	1	0	0	0	1	0	0	0	1	1	0	

The contents of the W register and carry flag are added to the contents of file register 6. The contents of the W register are not affected.

SUBBWF, f,d Subtract with borrow

OP Code							File						
0	0	0	0	0	1	d	f	f	f	f	f	f	(F17) + $(\overline{W})$ + C d

Status bits affected: OV, C, DC, Z

Example: SUBBWF 17, W

0	0	0	0	0	1	0	0	0	1	1	1	1	

The contents of the W register are complemented, added with the carry flag and register 17_8. The result is placed in the W register (d = 0).

ADDLW K Add literal to W Register

OP Code					Literal								
1	0	0	1	1	K	K	K	K	K	K	K	K	K + (W) W

Status bits affected: OV, C, DC, Z

Example: ADDLW 200 200 + (W) W

1	0	0	1	1	1	0	0	0	0	0	0	0	

The 8 bit literal 200_8 is added to the contents of the W register.

CPFSLT f Compare File to W, Skip is f W

OP Code							File						
1	0	0	0	1	0	0	f	f	f	f	f	f	(f) − (W), Skip if C = 0

Status bits affected: None

Example: CPSLT 27

1	0	0	0	1	0	0	0	1	0	1	1	1	(F27) − (W), Skip if C = 0

If the contents of register 27_8 are less than the contents of the W register, the next instruction is skipped.

CPFSEQ f Compare File to W, Skip if F = W

OP Code						File						
1	0	0	1	0	1	f	f	f	f	f	f	(f) − (W), Skip if Z = 1

Status bits affected: None

Example: CPSEQ 27

1	0	0	0	1	0	0	0	1	0	1	1	1	(F27) − (W), Skip if Z = 1

If the contents of register 27_8 equals the contents of the W register, the next instruction is skipped.

CPFSGT f Compare File to W, Skip if F > W

OP Code						File						
1	0	0	1	1	0	f	f	f	f	f	f	(f) − (W), Skip if $\overline{Z}\cdot C = 1$

Status bits affected: None

Example: CPFSGT 27

1	0	0	0	1	1	0	0	1	0	1	1	1	(F27) − (W), Skip if $\overline{Z}\cdot C = 1$

If the contents of register 27_8 are greater than the contents of the W register, the next instruction is skipped.

RLNCF f Rotate Contents of File Register Left

OP Code							File						
1	0	0	0	0	1	1	f	f	f	f	f	f	f(1)→d(0), F(6-0)→d(7-1)

Status bits affected: None

Example: RLNCF 20

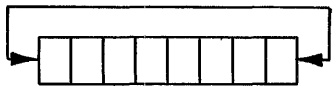
Assume the value stored in file register 20_8 is to be doubled:

0	0	1	1	0	1	0	1	Before Rotate Left
0	1	1	1	0	1	1	0	After Rotate Left

The value stored in F20 has been doubled from 65_8 to 152_8

RRNCF f Rotate Contents of File Register Right

OP Code							File					
1	0	0	0	0	1	0	f	f	f	f	f	f



Status bits affected: None

f(0)→d(7), f(7-1)→d(6-0)

Example: RRNCF 20

0	0	1	1	0	1	0	1

Before Rotate Right

0	0	0	1	1	0	1	0

After Rotate Right

TESTF f Test Contents of File Register

OP Code							File					
1	0	0	0	1	1	1	f	f	f	f	f	f

(f) → f

Status bits affected: Zero

This instruction moves the contents of a file register back into itself. In the process, the Zero status bit is set to a one if the contents of the file are zero.

RETFS Return From Subroutine

OP Code												
0	0	0	0	0	0	0	0	0	0	0	1	1

Stack → PC

Status bits affected: Zero

This command is used at the end of a subroutine to return to the address immediately following the CALL instruction. The contents of the top of the Stack are popped off and placed in the program counter. The W register is unaffected.

DAW Decimal Adjust W

OP Code												
0	0	0	0	0	0	0	0	0	0	1	0	1

This instruction adjusts the eight bit number in the W register to form two valid BCD (binary coded decimal digits, one in the lower and one in the upper nibble). (The results will only be meaningful if the number in W to be adjusted is the result of adding together two valid two digit BCD numbers.)

The adjustment obeys the following two step algorithm:

1. If the lower nibble is greater than 9 or the digit carry flag (DC) is set, 06 is added to the W register.
2. Then, if the upper nibble is greater than 9 or the carry from the original or step 1 addition is set, 60 is added to the W register. The carry bit is set if there is a carry from the original, step 1 or step 2 addition.

Example: Assume the W register contains 1011 1010 (the result of adding $65 + 55 = 120_{10}$, for instance).

C	DC	W		
0	0	1011	1010	
			0110	Add 6 to W
0	1	1100	0000	
		0110	0000	Add 60 to W
1	0	0010	000	Result (20) left in W, with C set

3.7 I/O Programming Caution

The use of the bidirectional I/O ports and the dedicated input or output ports are subject to certain rules of operation. These rules must be carefully followed in the instruction sequences written for I/O operation.

Bidirectional I/O Ports

The bidirectional ports may be used for both input and output operations. For input operations these ports are non-latching. Any input must be present until read by an input instruction. The outputs are latched and remain unchanged until the output latch is rewritten. **For use as an input port the output latch must be set in the high state.** Thus the external device inputs to the PIC circuit by forcing the latched output line to the low state or keeping the latched output high. This principle is the same whether operating on individual bits or the entire port.

Some instructions operate internally as input followed by output operations. The BCF and BSF instructions, for example, read the entire port into the CPU, execute the bit operation, and re-output the result. Caution must be used when using these instructions. As an example a BSF operation on bit 5 of F7 (Port C-PIC1650) will cause all eight bits of F7 to be read into the CPU. Then the BSF operation takes place on bit 5 and F7 is re-output to the output latches. If another bit of F7 is used as an input (say bit 0) then bit 0 must be latched high. If during the BSF instruction on bit 5 an external device is forcing bit 0 to the low state then the input/output nature of the BSF instruction will leave bit 0 latched low after execution. In this state bit 0 cannot be used as an input until it is again latched high by the programmer.

Successive Operations on Bidirectional I/O Ports

Care must be exercised if successive instructions operate on the same I/O port. The sequence of instructions should be such to allow the pin voltage to stabilize (load dependent) before the next instruction which causes that file to be read into the CPU (MOVF, BIT SET, BIT CLEAR, and BIT TEST) is executed. Otherwise, the previous state of that pin may be read into the CPU rather than the new state. This will happen if t_{pd} (See I/O Timing Diagram) is greater than $\frac{1}{4}t_{cy}$ (min). When in doubt, it is better to separate these instructions with a NOP or other instruction.

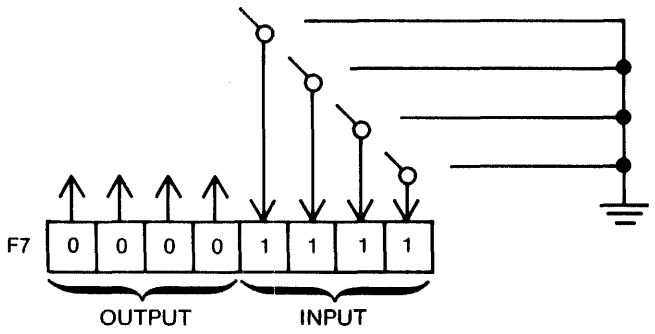
Input Only Ports

The input only port of the PIC1655A and PIC1656 consists of the four LSBs of F5 (port RA). An internal pull-up device is provided so that external pull-ups on open collector logic are unnecessary. The four MSBs of this port are always read as zeroes. Operations whose results are placed in F5 are not defined. File register instructions whose results are placed in W can be used. Note that the BTFSC and BTFSS instructions are input only operations and so can be used with F5.

Output Only Ports

The output only port contains no input circuitry and is therefore not capable of instructions requiring an input followed by output operation. The only instructions which can validly use F6 are MOVWF and CLRF.

EXAMPLE 1:



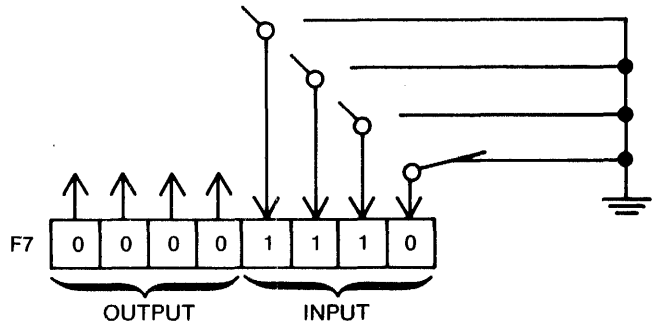
What is thought to be happening:

BSF 7,5

Read into CPU:	00001111
Set bit 5:	00101111
Write to F7:	00101111

If no inputs were low during the instruction execution, there would be no problem.

EXAMPLE 2:



What could happen:

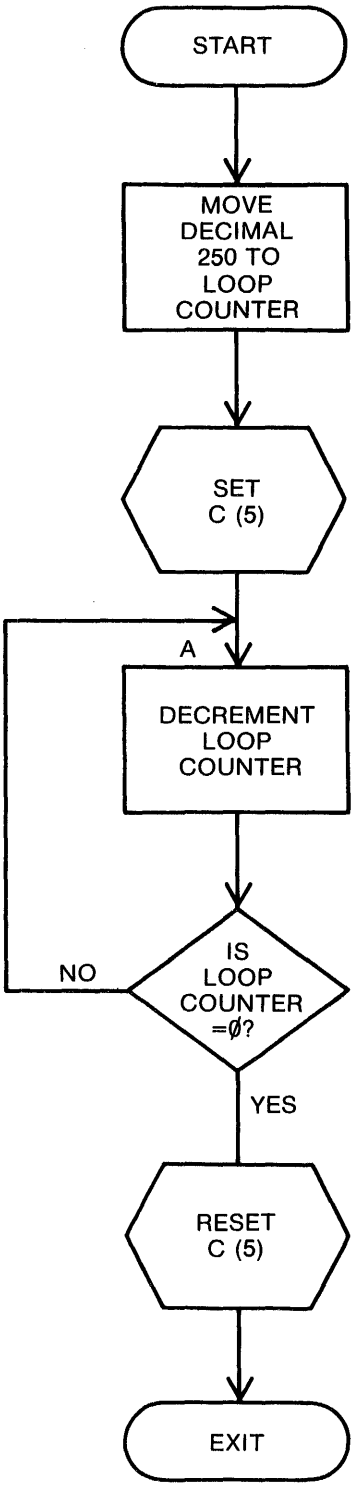
BSF 7,5

Read into CPU:	00001110
Set bit 5:	00101110
Write to F7:	00101110

In this case bit 0 is now latched low and is no longer useful as an input until set high again.

3.8 Sample Program

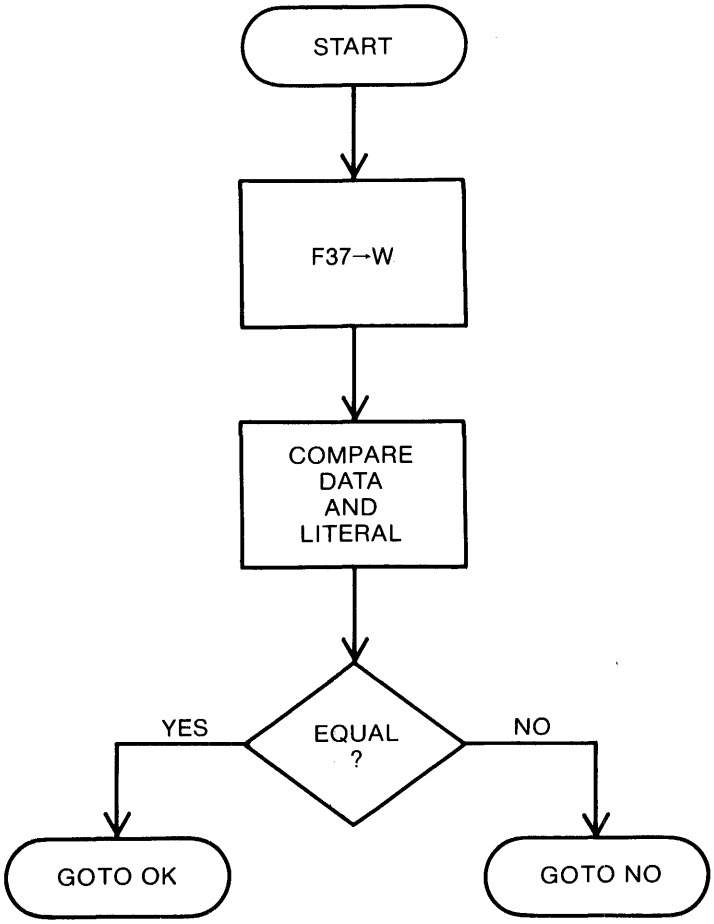
Example 1: Generate a 3ms pulse on I/O line C5 (F7, bit 5).



	Program Steps	Description
	MOVLW .250	LOAD decimal 250 into W.
	MOVWF 11	Transfer 250 to F11
	BSF 7,5	Set output file 7, bit 5 high.
A:	DECFSZ 11,1	Decrement F11, skip if zero.
	GOTO A	This GOTO instruction will cause F11 to be decremented 250 times. The decrement executes in 4μs while the GOTO takes 8μs. Therefore the loop executes in (4 + 8) μs x 250 = 3ms.
	BCF 7,5	Reset output file 7, bit 5 low.

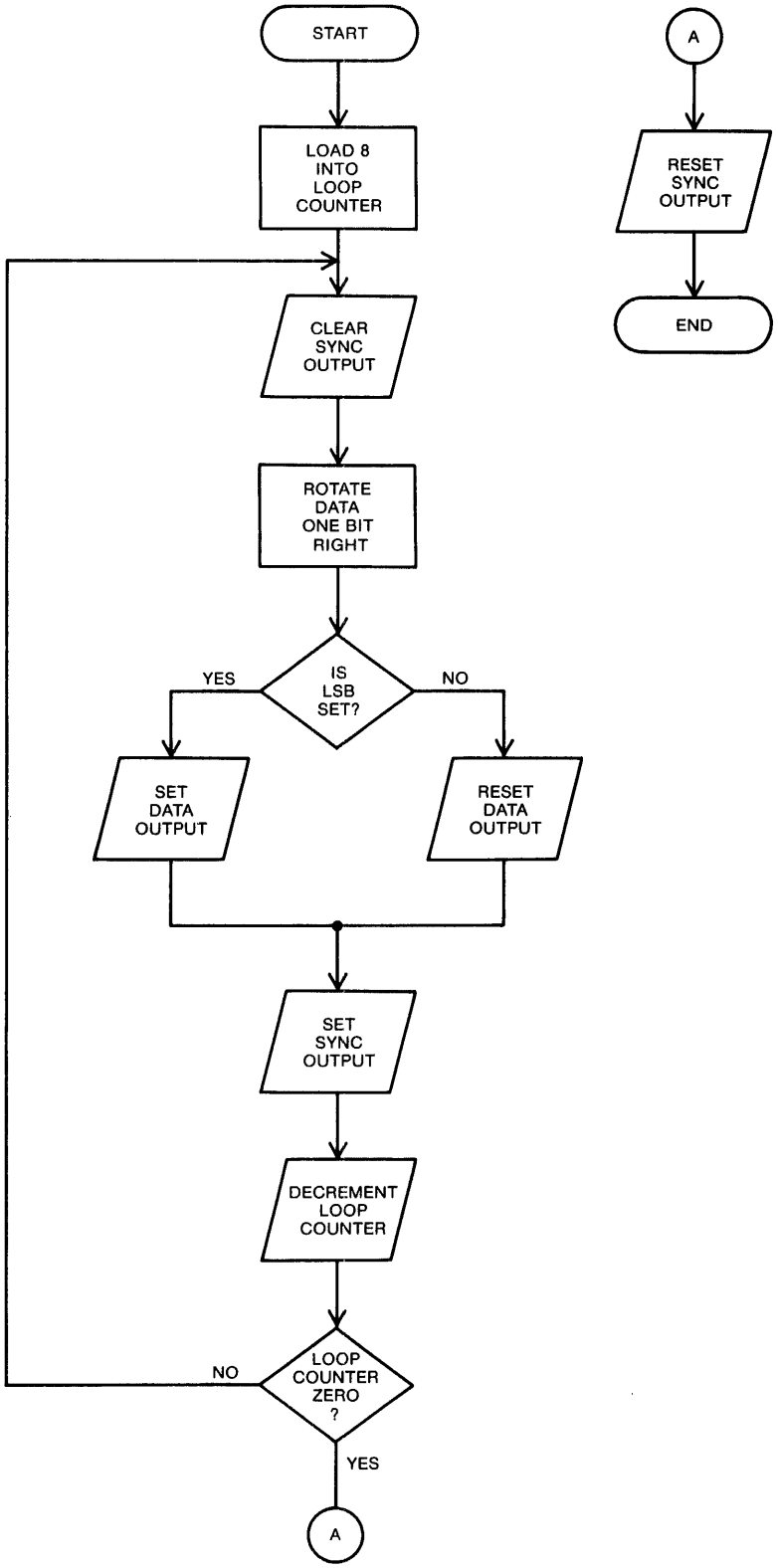
NOTE: For precise timing generation, an external crystal oscillator must be used. Otherwise the actual timing is dependent on the tolerances of the external RC components.

Example 2: Compare contents of F37 to a constant, if equal GOTO OK; if not equal GOTO NO.



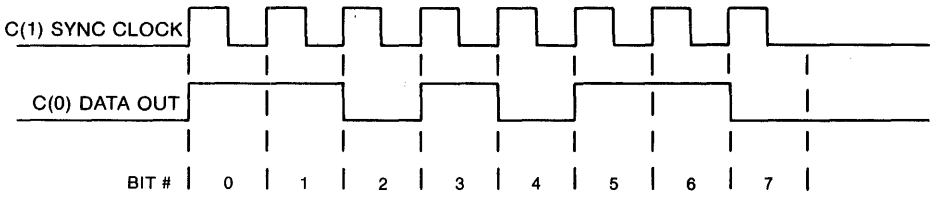
Program Steps	Description
MOVF 37,W	Move the contents of F37 to the working register W.
XORLW CONST	Exclusive OR the contents of W and the literal CONST. If they are equal, all zero bits will result in W and bit 2 in the status register (F3) will be set to a one. Although the SUBWF instruction could be used, it would also alter the Carry status bit.
BTFSS 3,2	If bit 2 in F3 is a one, skip the next step. (Bit 2 is the Zero status bit.)
GOTO NO	They are not equal.
GOTO OK	They are equal.

Example 3: Serially output the 8 bits in a file register. In this example, file register F24's contents are outputted via I/O C0 (F7, bit 0). I/O line C1 (F7, bit 1) is used to synchronize the output using the rising edge.

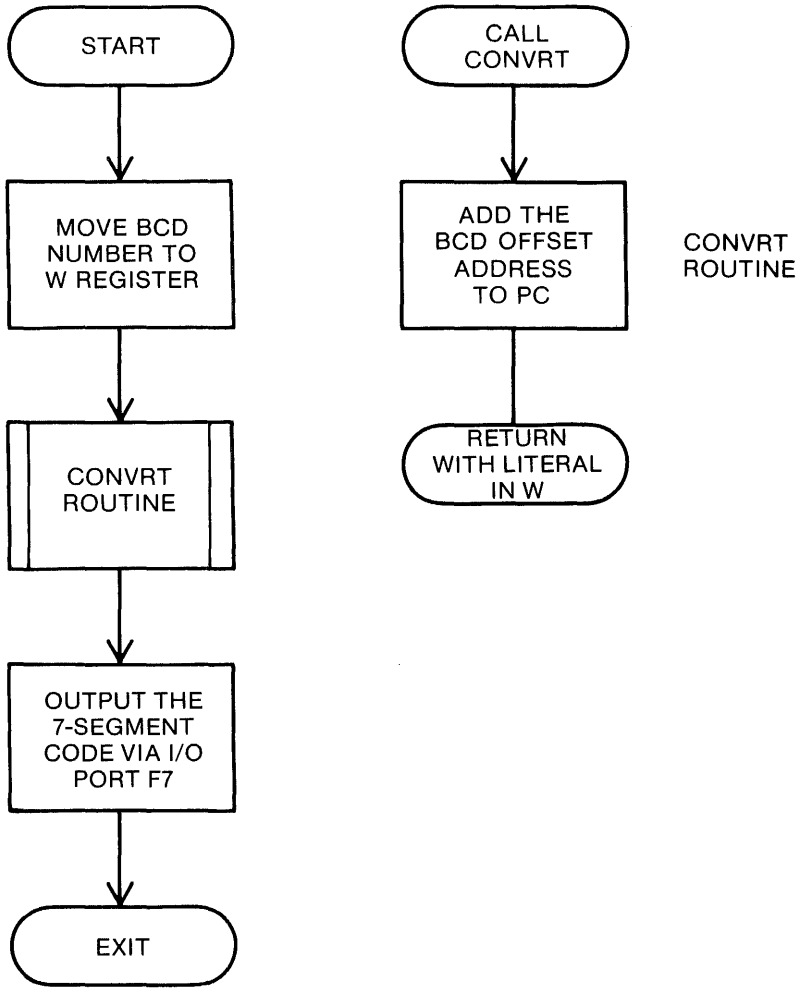


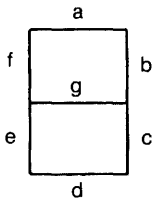
	Program	Description
	MOVLW .8	LOAD the decimal 8 into working register W.
	MOVW 11	Put decimal 8 into F11 (General Purpose register).
LOOP:	BCF 7,1	Clear the sync output.
	RRF 24,1	Rotate F24 one bit right. Bit 0 to Carry.
	BTFSS 3,0	Test Carry (F3, bit 0), skip if set to a one.
	GOTO A	Carry clear, go to A.
	BSF 7,0	Carry set, set C0; i.e., output positive signal
	GOTO B	Go to B.
A:	BCF 7,0	Carry clear, clear C0; i.e., output negative signal
B:	BSF 7,1	Raise sync line.
	DECFSZ 11	Have output all eight bits?
	GOTO LOOP	No, output next bit.
	BCF 7,1	Yes, clear sync output to a zero.
	END	

If File Register F24 contains 153 (octal), then the output will be as follows:



Example 4: Convert a BCD held in a 4 LSBs of F24 (the 4 MSBs are assumed zero) to a 7-segment code. The 7-segment code is output via I/O port F7. This program illustrates the use of a computed GOTO instruction.



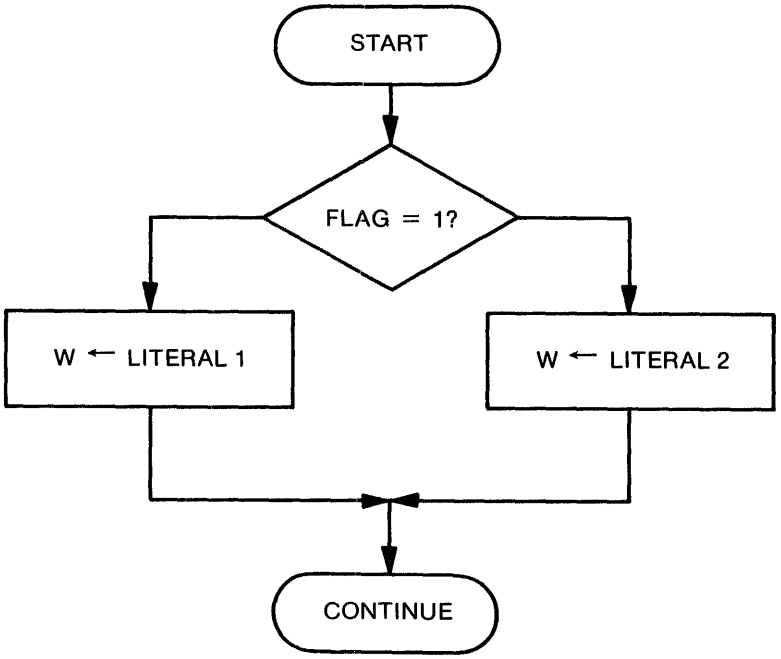


Typical 7-Segment bar position. The PIC Assembler recognizes the format B'bbbbbbbb' as an eight-bit binary data word where b is 0 or 1. The LED bar positions are thus B'0abcdefg'.

	Program Steps	Description
	MOVF 24, W	Starting address of table Move BCD number as offset into the W register.
	CALL CONVRT	Call the conversion subroutine. The program counter executes the next instruction at CONVRT.
	MOVWF 7	Output the 7-segment code via I/O port F7. The 7-segment display will now show the BCD number and this output will remain stable until F7 is set to a new value.
	END	
CONVRT:	ADDWF, PC	Add the BCD offset to the PC. This is a computed GOTO. Because the ninth bit of PC is set to zero by a ADDWF 2 the CONVRT routine must be located within 000 to 377_8.
	RETLW B'00000001'	complement of 0 in 7-segment code
	RETLW B'01001111'	complement of 1 in 7-segment code
	RETLW B'00010010'	complement of 2 in 7-segment code
	RETLW B'00000110'	complement of 3 in 7-segment code
	RETLW B'01001100'	complement of 4 in 7-segment code
	RETLW B'00100100'	complement of 5 in 7-segment code
	RETLW B'01100000'	complement of 6 in 7-segment code
	RETLW B'00001111'	complement of 7 in 7-segment code
	RETLW B'00000000'	complement of 8 in 7-segment code
	RETLW B'00001100'	complement of 9 in 7-segment code

NOTE: The RETLW instruction loads the W register with the specified literal value and returns to the instruction following the CALL instruction (MOVWF 7). The complement of the 7-segment code is used when the LED display unit is common anode (a bar is activated when the output is set low).

Example 5: Move one of two literals to W depending on the condition of a flag bit. This example illustrates a more efficient way (Method 2) of implementing the code.



Method 1

```
1.    BTFSC FLAG, BIT        ; TEST FLAT
2.    GOTO A
3.    MOVLW LITERAL 1        ; FLAG = 0
4.    GOTO CONTINUE
5.    MOVLW LITERAL 2        ; FLAG = 1
```

Method 2

```
1.    MOVLW LITERAL 1
2.    BTFSS FLAG, BIT        ; TEST FLAG
3.    MOVLW LITERAL 2        ; FLAG = 1
```

Example 6: Output the file pointed to by F37 via I/O register C (F7). Assume octal 24 in F37 and octal 100 in F24. Therefore, the following program will output 100_8 via F7.

Program Steps	Description
MOVF 37,W	Move the contents of F37 to W. After execution, W contains 24_8.
MOVWF 4	Move the contents of W to FSR (F4). After execution, F4 contains 24_8.
MOVF 0,W	Move the contents of the file pointed to by the FSR (the contents of F24) to W. Thus, W contains 100_8 after execution.
MOVWF 7	Move the contents of W to F7 where 100_8 will be latched.

Example 7: Clear files F5 to F37. This program illustrates the use of the File Select Register (F4) and the indirect addressing mode using F0.

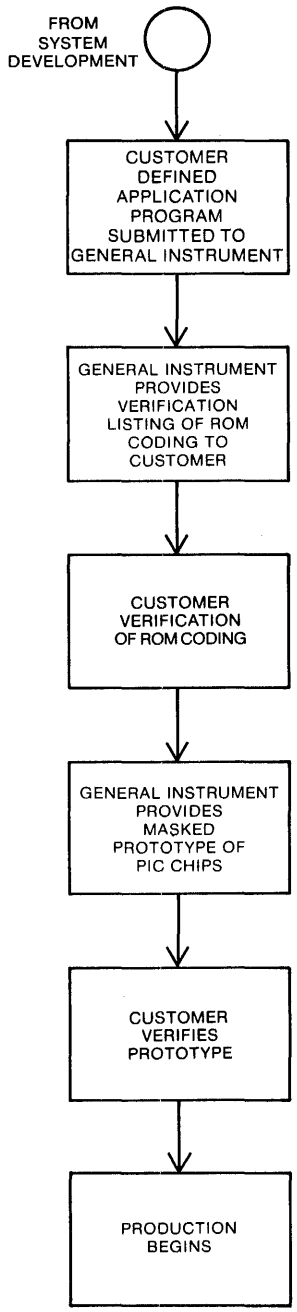
	Program Steps	Description
	MOVLW 5	Move the literal 5 to the working register W.
	MOVWF 4	Move the literal 5 to the File Select Register (F4). These two steps initialize the FSR to 5.
LOOP:	CLRF 0	Clear the contents of the file pointed to by the FSR.
	INCFSZ 4,1*	Increment the FSR. The PC counter will skip after File 37 is cleared.
	GOTO LOOP	Repeat the steps beginning at loop to clear the next file.
	END	Files F5 to F37 are cleared.

*The upper three bits of the FSR are always read as ones. When the FSR points to F37 all bits of the FSR are ones. The INCFSZ instruction reads this value into the ALU and increments it. The result of this increment equalling zero causes a skip. If the FSR is read after this operation, however, the result will be 340_8.

4 PRODUCTION CYCLE

Figure 20 is a flow chart of the production cycle. During the production cycle, the customer developed application program is verified, a prototype is masked and verified, and then production of mask programmed PIC microcomputers for the customer is initiated.

Fig. 20 PRODUCTION CYCLE



4.1 Hardware Support

Hardware support available from General Instrument includes:

- ☐ ROMless Development Microcomputer
- ☐ PICES PIC In-Circuit Emulation System
- ☐ PFD Field Demo System

The ROMless PIC microcomputers can emulate the operation of the entire PIC family. Pins are provided for connection to an external RAM or E/PROM that hold the application program.

The ROMless PIC is used as part of the PICES II In-Circuit Emulation System and the PFD Field Demo System. The ROMless PIC can also be used as part of a customer designed in-circuit emulation system.

The PICES II is an in-circuit emulation system than can be used in a stand-alone mode with a teletypewriter terminal or can be used as a peripheral in a large computer system. When the PICES II is used as a peripheral, the user's computer facility becomes a one-station total development system.

The PFD is a field demo system that demonstrates the integrated hardware/software application.

4.1.1 ROMless DEVELOPMENT PIC

■ **Description and Features.** The ROMless PIC MOS/LSI circuit array employs the same basic architecture as the PIC microcomputers except that the ROM is removed and the ROM address and data lines are brought out to pins, resulting in a 64-pin package. Basic features are:

PIC ROM can be replaced with RAM or E/PROM

HALT pin for single stepping or stopping program execution

TTL-compatible input/output lines

4.5 to 7.0V power supply operation

Same instruction set as that of PIC microcomputer being emulated.

One additional instruction, HALT (0001_8) is provided.

Note: Refer to Data Sheets for additional information.

4.1.2 PICES II-PIC IN-CIRCUIT EMULATION SYSTEM

■ **Features.**

- ☐ Complete in-circuit emulation and debug capability
- ☐ Multiple system configurations to match user requirements
- ☐ Standard serial interface for system integration
- ☐ Powerful 16-bit microprocessor for system control
- ☐ Multiple breakpoints, single step, program trace and editing capabilities
- ☐ On-board diagnostics for system hardware troubleshooting

The PICES II system is an in-circuit emulation and debug facility designed to provide the user with a complete tool for testing, troubleshooting, and modifying both the software program for the PIC circuit as well as the total system application. The PICES II is a self-contained unit which can operate in a stand-alone configuration or as a peripheral device to a host computer.

■ **Architecture.** The PICES II system contains two processors. The user processor is a ROMless PIC microcomputer with external RAM. With the RAM loaded with the user's application program, the ROMless PIC emulates the operation of the entire PIC family. A 40- or 28-pin in-circuit emulation cable attaches from the ROMless to the application system. The control processor is a CP1600 16-bit microprocessor with 12K words of program ROM and 2K words of RAM. This processor controls the functions of the PICES II including I/O interfacing, manipulation of the user processor and interpretation and execution of the PICES II command set.

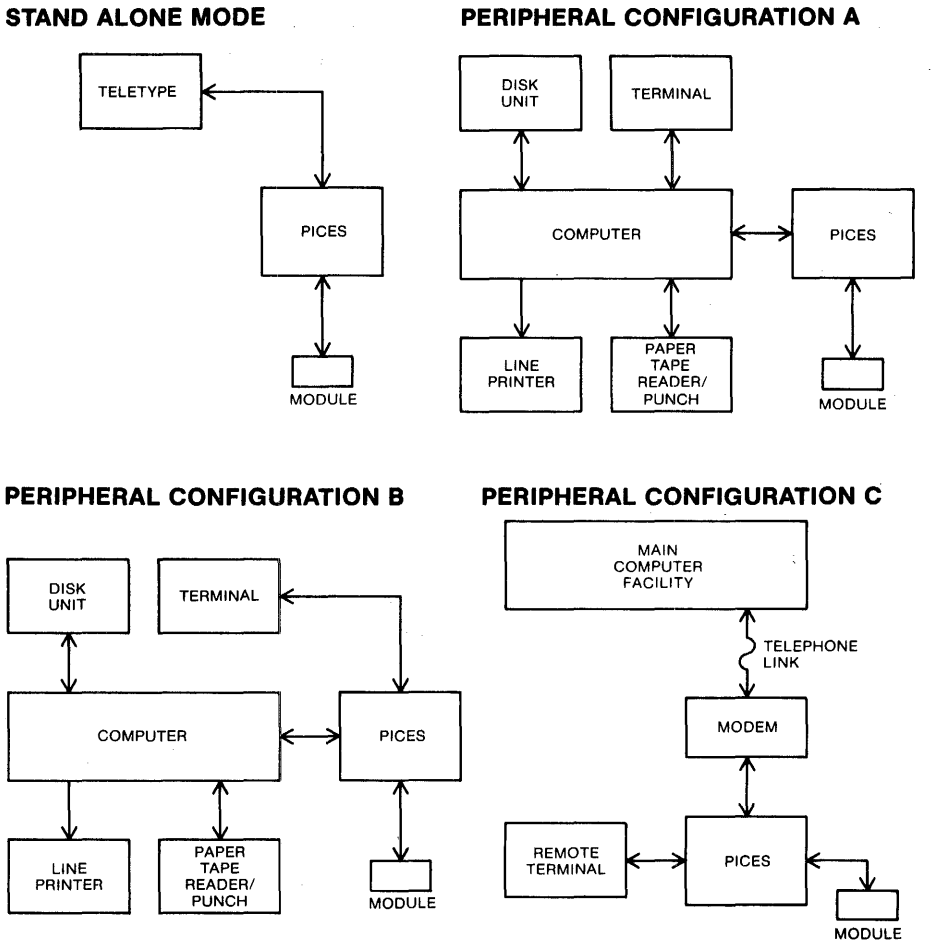
■ **Operation.** The PICES II operates in several configurations:

STAND-ALONE MODE. The PICES II is attached directly to a serial I/O device; typically a teletype. The user program is entered either using the paper tape reader/punch unit on the teletype or by manually setting each location in the PIC program memory to the desired value. Once the program memory is loaded, all PICES II emulation and debug commands can be issued on the teletype keyboard and PICES II responses are returned on the teletype printer. The serial interface can be either RS232C or current loop and the baud rate is switch selectable.

PERIPHERAL MODE. The PICES II can be configured such that the unit itself is a serial peripheral device attached to another computer system. The PICES II can be attached as an additional peripheral device or in series with the system TTY or CRT device. In this mode, the user's computer facility can become a one station total development system. The computer text editor is used to develop the PIC source code. The PIC Cross Assembler (PICAL) will translate this source code into PIC object code; the object code is then downloaded into the PICES II. All PICES II commands are entered through the system terminal. Minor modifications can be done directly on the PICES II. Major changes require re-editing the source code, re-assembling, and re-loading of the PICES II.

■ **Reference Manual.** A detailed PICES II Data Manual is available. This manual describes the installation and operation of the PICES II system. Included in the manual are explanations of the PICES II command set with examples for illustration.

Fig. 21 PICES CONFIGURATIONS



4.1.3 PFD-PIC FIELD DEMO SYSTEM

■ **Features.**

- ☐ 5 Volt, single supply, operation
- ☐ Low power —55 mA maximum
- ☐ Optional external clock
- ☐ Optional external power-on-clear
- ☐ Dimensions: 4" x 4⅜"
- ☐ Cable length: 14"

■ **Description.** The PIC Field Demo System provides the user with a compact and portable method of evaluating and demonstrating application performance before the commitment is made to ROM masking of the PIC circuit.

The PFD module contains a ROMless PIC microcomputer, sockets for two ultraviolet-erasable PROMs, an on-board oscillator and power-on clear circuitry. A cable is provided to interface the PFD to the user's system.

■ **Reference Manual.** A complete description of the PFD systems is contained in the PIC Field Demo Systems Data Manual.

Fig. 22 DEVELOPMENT SYSTEMS

TARGET MICROCOMPUTER	ROMless MICROCOMPUTER	DEVELOPMENT SYSTEM	PFD BOARD
PIC1650	PIC1664	PICES II	PFD1000
PIC1654	PIC1664	PICES II	PFD1007
PIC1655	PIC1664	PICES II	PFD1000
PIC1656	PIC1664	PICES II	PFD1010
PIC16C58	PIC16C63	PICES II	PFD2010
PIC1670	PIC1665	PICES II	PFD1020

4.2 Software Support

Software support available from General Instrument includes the PIC cross-assembler, PICAL.

4.2.1 PICAL-PIC MACROASSEMBLER

■ **Features.**

- ☐ Symbolic representation of instructions
- ☐ User defined six character symbols
- ☐ Octal, decimal, hexadecimal, ASCII, and EBCDIC literals
- ☐ Expression evaluation
- ☐ Extensive assembly directives
- ☐ Full program and sorted symbol listing
- ☐ Extensive error detection
- ☐ User-defined macro generation.

■ **Description.** PICAL is loaded into any minicomputer or large-scale computer having an editor and FORTRAN IV compiler. PICAL, written in FORTRAN IV, enables the host computer to assemble the PIC source programs and provide object programs that can execute on the PICES emulation system. The PICAL Cross-Assembler also enables the generation of user-defined macro-instructions. PICAL also generates a program listing, in which any syntax errors, illegal operations, or ROM overflow are flagged. An object program cannot be generated until all errors are corrected.

■ **Reference Manual.** A complete description of PICAL, its installation and operation is provided in the PICAL Users Manual.

5 MATH ROUTINES

5.1a Unsigned BCD Addition

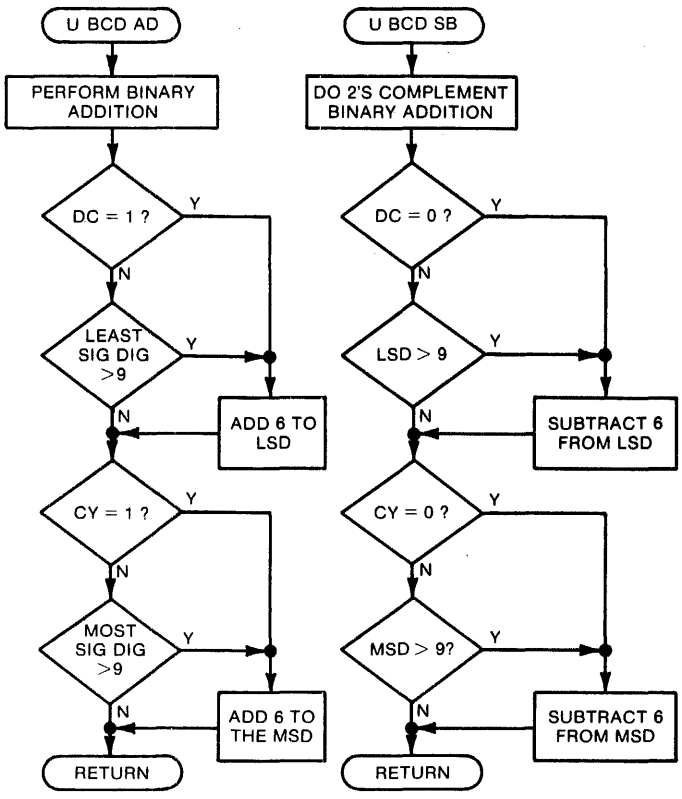
This section describes commonly used math routines. It is intended to be a guide to the programmer and engineer, who can use the routines as is or modify them appropriately for their particular application. Also, coding techniques can be learned by studying the descriptions, flowcharts and listings, and then applying them to other tasks.

Doing straight binary addition of BCD Numbers necessitates adjustment of the result for it to be interpreted as BCD digits.

This routine uses two steps to accomplish this:

1. If the least significant four bits of the result represent a number > 9, or if the DC bit is set to 1, 6 is added to the result (DC must propagate) otherwise no addition occurs.
2. After completion of Step 1—if the most significant four bits of the result represent a number > 9, or if the CY bit from the original or Step 1 addition is set to 1, 60 is added to the result (or 6 added to MSD) otherwise no addition is done.

NOTE: To extend routine to more than two digits, all additions must be performed with carry (or DC). Otherwise same rules as above apply to each digit. A carry from any of the three additions (Original, Step 1 or Step 2) constitutes an overflow to the next digit, if any.



DC = DIGIT CARRY
CY = CARRY
LSD = LEAST SIGNIFICANT DIGIT (LS 4 BITS)
MSD = MOST SIGNIFICANT DIGIT (MS 4 BITS)

```
LINE    ADDR   B1    B2      UNSIGNED BCD ADDITION                          PAGE    1

  1                            TITLE  'UNSIGNED BCD ADDITION'
  2                                                  ;PERFORMS 2 DIGIT UNSIGNED BCD ADDITION.
  3                                                  ;ROUTINE ASSUMES THE AUGEND IN F12 &
  4                                                  ;THE ADDUEND IN F11.ROUTINE RETURNS
  5                                                  ;WITH THE SUM IN F12 AND OVERFLOW
  6                                                  ;CARRY IN F11.
  7                                                  ;
  8                                                  ;
  9                                                  ;
 10  000000  1011          UBCDAD  MOVF   11,W       ;DO BINARY ADDITION
 11  000001  0752                  ADDWF  12          ;
 12  000002  0151                  CLRF   11          ;
 13  000003  1555                  RLF    15          ;SAVE CY.
 14  000004  3043                  SKPNDC             ;DC=1?
 15  000005  5014                  GOTO   ADJST1      ;YES! ADJUST LSD OF RESULT
 16  000006  6006                  MOVLW  .6          ;TEST FOR LSD>9(ADD 6-
 17  000007  0752                  ADDWF  12          ;IF DC=1 THEN LSD>9).
 18  000010  3443                  SKPDC              ;
 19  000011  0252                  SUBWF  12          ;DC=0(LSD<9) SO RESTORE RESULT.
 20  000012  2003                  CLRC               ;
 21  000013  5016                  GOTO   OVR1        ;
 22  000014  6006          ADJST1  MOVLW  6           ;ADJUST-ADD 6 TO LSD
 23  000015  0752                  ADDWF  12          ;
 24  000016  1551          OVR1    RLF    11          ;SAVE CY.
 25  000017  6140                  MOVLW  140         ;ADD 6 TO MSD.
 26  000020  0752                  ADDWF  12          ;
 27  000021  3003  5033            BC     OVR-2       ;TEST FOR MSD>9 (CY=1 AFTER ADDING 6).
 28  000023  3411                  BTFSS  11,0        ;TEST SAVED CY.
 29  000024  3015                  BTFSC  15,0        ;DITTO.
 30  000025  5030                  GOTO   OVR-5       ;CY'S=1--KEEP ADJUST.
 31  000026  0252                  SUBWF  12          ;CY'S=0--NO ADJUST.
 32  000027  5035                  GOTO   OVR         ;
 33  000030  6001                  MOVLW  1           ;SAVE OVERFLOW.
 34  000031  0051                  MOVWF  11          ;
 35  000032  5035                  GOTO   OVR         ;
 36  000033  0151                  CLRF   11          ;SAVE OVERFLOW.
 37  000034  1551                  RLF    11          ;
 38  000035  4000          OVR     RET
 39  000036                        END


 ASSEMBLER ERRORS =    0
```

5.1b Unsigned BCD Addition of 2 Digits

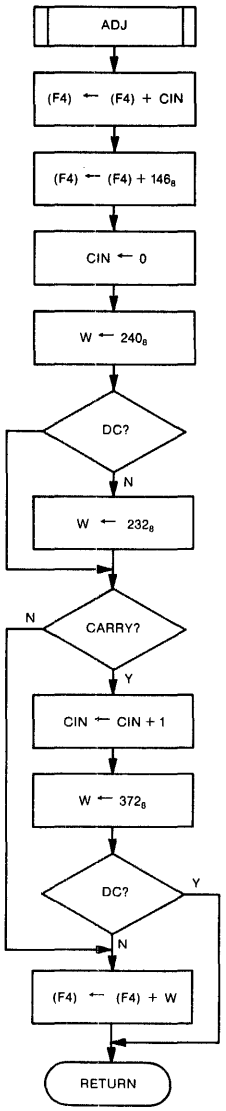
It is often necessary to add together two 8-bit registers containing 2 unsigned BCD digits in each. The following algorithm is used:

Algorithm:

1. Add augend to addend.
2. Add carry in to result of Step 1.
3. Add hexadecimal 66(146_8) to result of Step 2.
4. Add the following correction factor to the result of Step 3, with carry out (CO) set as noted:

if: C=0 and DC=0, add HEX 9A (232_8); CO=0
C=0 and DC=1, add HEX A0 (240_8); CO=0
C=1 and DC=0, add HEX FA (372_8); CO=1
C=1 and DC=1, add HEX 00 (000_8); CO=1

The flow chart and program for the above algorithm follows. Note that it is assumed that Step 1 has been done and the result is in the register pointed to by the FSR (F4) when the routine ADJ is called. Carry in is in CIN.



```
 1 ADJ     MOVF     CIN, W
 2         ADDWF    0        ;(F4) = (F4) + CARRY IN
 3         MOVLW    146
 4         ADDWF    0        ;(F4) = (F4) + 146(8)
 5         CLRF     CIN      ;CIN = 0
 6         MOVLW    240      ;DC = 1
 7         BTFSS    3, 1     ;TEST DC
 8         MOVLW    232      ;DC = 0
 9         BTFSS    3, 0     ;TEST CARRY
10         GOTO     ADJ0
11         INCF     CIN      SET CARRY IN BIT
12         MOVLW    372      ;DC = 0
13         BTFSS    3, 1     ;TEST DC
14 ADJ0    ADDWF    0        ;(F4) = (F4) + CORRECTION
                               FACTOR
15         RET
```

NOTE: Normally one would not use an entire register to store the carry in bit —a single bit of a register is all that is needed. In this case, the following changes would be made:

```
 1. ADJ    BTFSCC   f, 0     ;TEST CARRY IN BIT
 2.        INCF     0        ;ONE, ADD 1
 5.        BCF      f, 0     ;CLEAR CARRY IN BIT
11         BSF      f, 0     ;SET CARRY IN BIT
```

5.2 Unsigned BCD Subtraction

Straight binary subtraction (two's complement addition) of two 2-digit BCD numbers necessitates the adjusting of the result for it to be interpreted as a BCD number.

This is done in two steps:

1. If the least significant 4 bits of the result is > 9 or if DC is **not** set (0) then subtract 6 from the least significant 4 bits (LSD) of the result (DC propagated is added to next digit), otherwise no subtraction is done.
2. After Step 1 is complete—if the most significant 4 bits (MSD) of the result is > 9 or if CY is **not** set (0), subtract 6 from the most significant 4 bits (MSD) of the result, otherwise no subtraction is done.

NOTES: 1. To extend routine to more than two digits, same rules as above apply to each BCD digit.
2. The CY tested (in Step 2) is that obtained after two's complement addition.
3. When F11 has .9, result is −VE. Take ten's complement to get its value.

```
LINE    ADDR   B1    B2       UNSIGNED BCD SUBTRACTION                    PAGE    1

  1                             TITLE  'UNSIGNED BCD SUBTRACTION'
  2                                                  ;PERFORMS 2 DIGIT UNSIGNED BCD
  3                                                  ;SUBTRACTION. ROUTINE ASSUMES MINUEND
  4                                                  ;IN F12 & THE SUBTRAHEND IN F11. THE
  5                                                  ;ROUTINE RETURNS WITH THE DIFFERENCE
  6                                                  ;IN F12 & THE OVERFLOW CARRY (SIGN)  IN F11.
  7  000000  1011          UBCDSB  MOVF   11,W       ;DO BINARY TWO'S COMPLEMENT
  8  000001  0252                  SUBWF  12         ;SUBTRACTION.
  9  000002  0151                  CLRF   11         ;
 10  000003  1551                  RLF    11         ;SAVE CY.
 11  000004  3443                  SKPDC             ;DC=0?
 12  000005  5014                  GOTO   ADJST1     ;YES! ADJUST LSD OF RESULT.
 13  000006  3552                  BTFSS  12,3       ;NO! TEST FOR LSD>9.
 14  000007  5016                  GOTO   OVR1       ;
 15  000010  3112                  BTFSC  12,2       ;
 16  000011  5014                  GOTO   ADJST1     ;YES! ADJUST LSD OF RESULT.
 17  000012  3452                  BTFSS  12,1       ;
 18  000013  5016                  GOTO   OVR1       ;NO! GO FOR MSD
 19  000014  6006          ADJST1  MOVLW  6          ;ADJUST-SUBTRACT 6 FROM LSD
 20  000015  0252                  SUBWF  12         ;
 21  000016  3411          OVR1    BTFSS  11,0       ;CY=0?
 22  000017  5027                  GOTO   ADJST2     ;YES! ADJUST MSD OF RESULT.
 23  000020  0151                  CLRF   11         ;
 24  000021  3752                  BTFSS  12,7       ;NO! TEST FOR MSD>9.
 25  000022  5036                  GOTO   OVR        ;
 26  000023  3312                  BTFSC  12,6       ;
 27  000024  5027                  GOTO   ADJST2     ;YES! ADJUST MSD.
 28  000025  3652                  BTFSS  12,5       ;
 29  000026  5036                  GOTO   OVR        ;NO! DONE-RETURN.
 30  000027  6140          ADJST2  MOVLW  140        ;ADJUST-SUBTRACT 6 FROM
 31  000030  0252                  SUBWF  12         ;MSD OF RESULT.
 32  000031  0151                  CLRF   11         ;
 33  000032  3403                  SKPC              ;TEST CY-IF SET UNDERFLOW.
 34  000033  5036                  GOTO   OVR        ;CY=0!NO UNDERFLOW-DONE.
 35  000034  6011                  MOVLW  11         ;CY=1!UNDERFLOW SET -VE SIGN.
 36  000035  0051                  MOVWF  11         ;
 37  000036  4000          OVR     RET
 38  000037                        END


ASSEMBLER ERRORS =    0
```

5.3 Signed BCD Addition

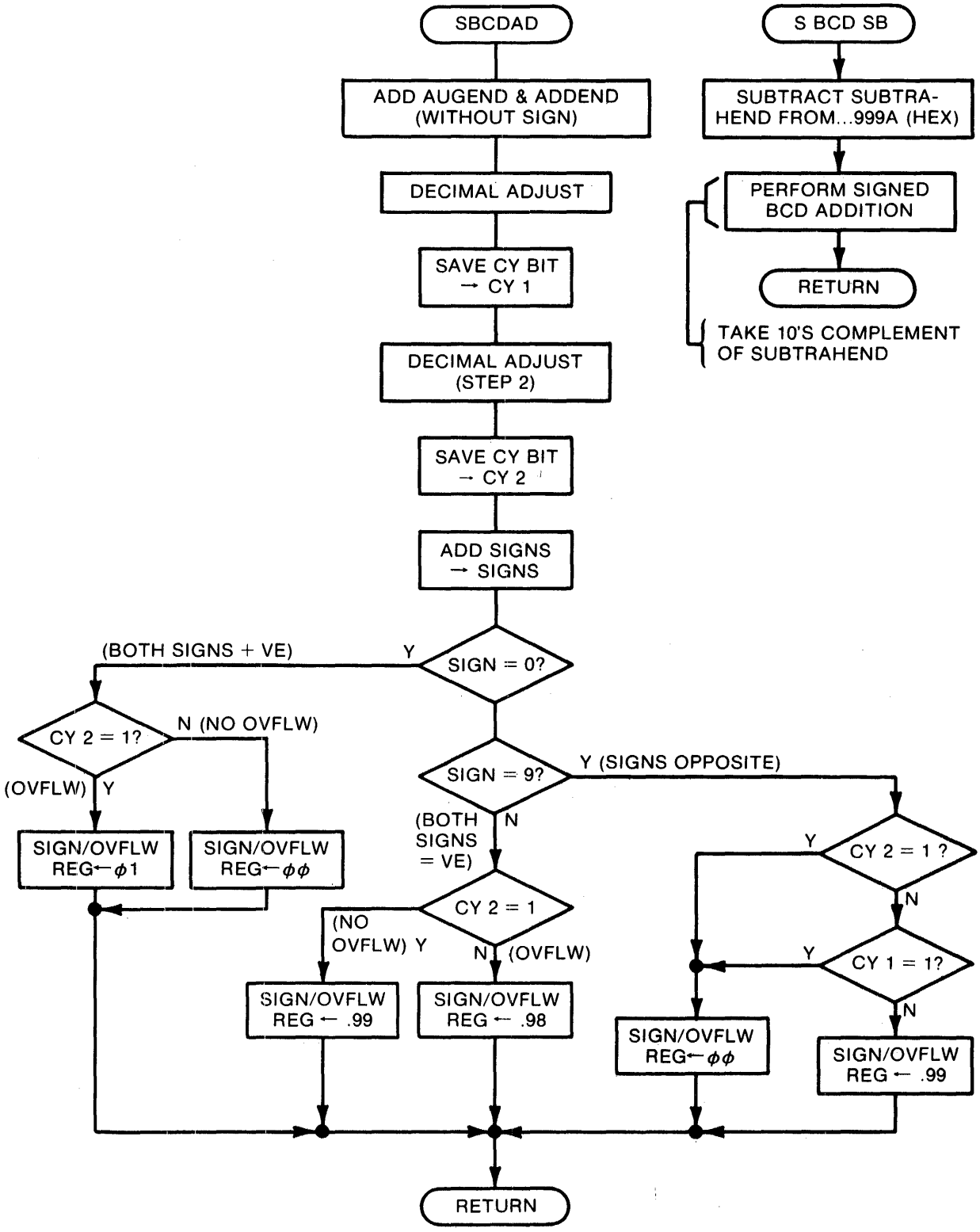
This routine performs Signed BCD Addition by performing Unsigned BCD Addition and adjusting the sign of the BCD result according to the sign of the augend and addend. The sign nibble is set to .9 for a −VE result and 0 for a positive result. The overflow nibble is set according to Table 1:

TABLE 1

Sign	Overflow	Overflow Nibble
+VE	0	0
+VE	1	1
−VE	0	.9
−VE	1	.8

The values in Table 1 are arrived at in accordance with ten's complement arithmetic.

NOTE: In ten's complement arithmetic the sign nibble is 0 for a +VE number and .9 for a −VE number.


SBCDAD
ADD AUGEND & ADDEND (WITHOUT SIGN)
DECIMAL ADJUST
SAVE CY BIT → CY 1
DECIMAL ADJUST (STEP 2)
SAVE CY BIT → CY 2
ADD SIGNS → SIGNS
SIGN = 0?
Y
(BOTH SIGNS + VE)
CY 2 = 1?
N (NO OVFLW)
(OVFLW) Y
SIGN/OVFLW REG←φ1
SIGN/OVFLW REG←φφ
SIGN = 9?
Y (SIGNS OPPOSITE)
(BOTH SIGNS = VE)
N
CY 2 = 1
(NO OVFLW) Y
N (OVFLW)
SIGN/OVFLW REG ← .99
SIGN/OVFLW REG ← .98
CY 2 = 1 ?
Y
N
CY 1 = 1?
Y
N
SIGN/OVFLW REG←φφ
SIGN/OVFLW REG ← .99
RETURN
S BCD SB
SUBTRACT SUBTRAHEND FROM...999A (HEX)
PERFORM SIGNED BCD ADDITION
RETURN
TAKE 10'S COMPLEMENT OF SUBTRAHEND

```
LINE   ADDR   B1   B2      SIGNED-BCD ADDITION                          PAGE   1

  1                          TITLE  'SIGNED-BCD ADDITION'
  2                                                  ;
  3                                                  ;
  4                                                  ;PERFORMS 2-DIGIT SIGNED-BCD ADDITION.
  5                                                  ;THE ROUTINE ASSUMES AUGEND IN F11 &
  6                                                  ;F12 (LSD OF F11 IS SIGN DIGIT), &
  7                                                  ;THE ADDEND IN F13 & F14 (LSD OF F13
  8                                                  ;IS SIGN DIGIT).THE ROUTINE RETURNS
  9                                                  ;WITH RESULT IN F13 & F14 (LSD OF F13
 10                                                  ;IS OVERFLOW DIGIT & MSD OF F13 IS
 11                                                  ;SIGN DIGIT).
 12                                                  ;
 13                                                  ;
 14                                                  ;
 15  000000  1012          SBCDAD  MOVF   12,W       ;DO BINARY ADDITION.
 16  000001  0754                  ADDWF  14          ;
 17  000002  0152                  CLRF   12          ;
 18  000003  1555                  RLF    15          ;
 19  000004  3043                  SKPNDC             ;DO BCD DECIMAL ADJUST--SEE UNSIGNED
 20  000005  5014                  GOTO   ADJST1      ;BCD ADDITION ROUTINE.
 21  000006  6006                  MOVLW  .6          ;
 22  000007  0754                  ADDWF  14          ;
 23  000010  3443                  SKPDC              ;
 24  000011  0254                  SUBWF  14          ;
 25  000012  2003                  CLRC               ;
 26  000013  5016                  GOTO   OVR1        ;
 27  000014  6006          ADJST1  MOVLW  6           ;
 28  000015  0754                  ADDWF  14          ;
 29  000016  1552          OVR1    RLF    12          ;SAVE CY.
 30  000017  6140                  MOVLW  140         ;
 31  000020  0754                  ADDWF  14          ;
 32  000021  3003  5032            BC     OVR-1       ;
 33  000023  3412                  BTFSS  12,0        ;
 34  000024  3015                  BTFSC  15,0        ;
 35  000025  5031                  GOTO   OVR-2       ;
 36  000026  0254                  SUBWF  14          ;
 37  000027  2003                  CLRC               ;
 38  000030  5032                  GOTO   OVR-1       ;
 39  000031  2403                  SETC               ;
 40  000032  1552                  RLF    12          ;
 41  000033  1011          OVR     MOVF   11,W       ;ADD SIGNS.
 42  000034  0753                  ADDWF  13          ;
 43  000035  3103                  SKPNZ              ;RESULT=0?
 44  000036  5064                  GOTO   BPOS        ;YES!-- BOTH SIGNS +VE.
 45  000037  1013                  MOVF   13,W        ;NO! THEN RESULT=9?
 46  000040  7411                  XORLW  11          ;
 47  000041  3103                  SKPNZ              ;
 48  000042  5053                  GOTO   OPPST       ;YES!-- SIGNS OPPOSITE-NO OVERFLOW.
```

```
LINE    ADDR   B1    B2       SIGNED-BCD ADDITION                          PAGE    2

 49  000043  3412            BTFSS  12,0        ;NO!-- BOTH SIGNS -VE.
 50  000044  5050            GOTO   OVFLW       ;TEST SAVED CY.CY=0-OVERFLOW.
 51  000045  6231            MOVLW  231         ;CY=1-NO OVERFLOW.
 52  000046  0053            MOVWF  13          ;SET SIGN -VE.
 53  000047  5073            GOTO   FIN         ;
 54  000050  6230       OVFLW  MOVLW  230       ;OVERFLOW-SET SIGN -VE &
 55  000051  0053            MOVWF  13          ;OVERFLOW DIGIT =1.
 56  000052  5073            GOTO   FIN         ;
 57  000053  3012       OPPST  BTFSC  12,0      ;TEST SAVED CY.
 58  000054  5062            GOTO   POS         ;CY=1!
 59  000055  3052            BTFSC  12,1        ;TEST CY FROM 1ST ADJUST.
 60  000056  5062            GOTO   POS         ;
 61  000057  6231            MOVLW  231         ;SET SIGN -VE.
 62  000060  0053            MOVWF  13          ;
 63  000061  5073            GOTO   FIN         ;
 64  000062  0153       POS    CLRF   13        ; SET SIGN +VE.
 65  000063  5073            GOTO   FIN         ;
 66  000064  3012       BPOS   BTFSC  12,0      ;TEST SAVED CY.
 67  000065  5071            GOTO   OVFLW1      ;CY=1! OVERFLOW.
 68  000066  6000            MOVLW  00          ;CY=0! NO OVERFLOW-SET SIGN
 69  000067  0053            MOVWF  13          ;+VE & OVERFLOW DGT 0.
 70  000070  5073            GOTO   FIN         ;
 71  000071  6001       OVFLW1 MOVLW  1         ;SET SIGN +VE & OVER-
 72  000072  0053            MOVWF  13          ;FLOW DGT 1.
 73  000073  4000       FIN    RET             ;FINISHED-RETURN.
 74  000074                 END


ASSEMBLER ERRORS =    0
```

5.4 Signed BCD Subtraction

This routine performs signed BCD subtraction by taking ten's complement of the subtrahend and adding the minuend with signed BCD addition.

The routine takes ten's complement of the subtrahend by subtracting the least significant digit (of the subtrahend) from ten and subtracting each of the other digits (including the sign digit) from nine.

```
LINE    ADDR   B1    B2       SIGNED-BCD SUBTRACTION                          PAGE    1

  1                              TITLE  'SIGNED-BCD SUBTRACTION'
  2                                                      ;
  3                                                      ;
  4                                                      ;PERFORMS 2-DIGIT SIGNED-BCD SUBTRACTION.
  5                                                      ;THE ROUTINE ASSUMES AUGEND IN F11 &
  6                                                      ;F12 (LSD OF F11 IS SIGN DIGIT), &
  7                                                      ;THE ADDEND IN F13 & F14 (LSD OF F13
  8                                                      ;IS SIGN DIGIT).THE ROUTINE RETURNS
  9                                                      ;WITH RESULT IN F13 & F14 (LSD OF F13
 10                                                      ;IS OVERFLOW DIGIT & MSD OF F13 IS
 11                                                      ;SIGN DIGIT).SUBTRACTION IS DONE BY
 12                                                      ;TAKING THE TEN'S COMPLEMENT OF THE
 13                                                      ;SUBTRAHEND & THEN DOING SIGNED BCD
 14                                                      ;ADDITION.
 15                                                      ;
 16                                                      ;
 17                                                      ;
 18                                                      ;
 19  000000  6011         SBCDSB  MOVLW  11              ;TAKE TEN'S COMPLEMENT OF THE SUBTRAHEND.
 20  000001  0055                 MOVWF  15              ;
 21  000002  6232                 MOVLW  232             ;
 22  000003  0056                 MOVWF  16              ;
 23  000004  1013                 MOVF   13,W            ;THIS IS DONE BY SUBTRACTING THE LSD
 24  000005  0215                 SUBWF  15,W            ;FROM .10 & EACH OF THE MORE SIGNIFICANT
 25  000006  0053                 MOVWF  13              ;DIGITS FROM .9.
 26  000007  1014                 MOVF   14,W            ;
 27  000010  0216                 SUBWF  16,W            ;
 28  000011  0054                 MOVWF  14              ;
 29  000012  1012                 MOVF   12,W            ;DO BINARY ADDITION.
 30  000013  0754                 ADDWF  14              ;
 31  000014  0152                 CLRF   12              ;
 32  000015  1555                 RLF    15              ;SAVE CY.
 33  000016  3043                 SKPNDC                 ;DO BCD DECIMAL ADJUST--SEE UNSIGNED
 34  000017  5026                 GOTO   ADJST1          ;BCD ADDITION ROUTINE.
 35  000020  6006                 MOVLW  .6              ;
 36  000021  0754                 ADDWF  14              ;
 37  000022  3443                 SKPDC                  ;
 38  000023  0254                 SUBWF  14              ;
 39  000024  2003                 CLRC                   ;
 40  000025  5030                 GOTO   OVR1            ;
 41  000026  6006         ADJST1  MOVLW  6               ;
 42  000027  0754                 ADDWF  14              ;
 43  000030  1552         OVR1    RLF    12              ;SAVE CY.
 44  000031  6140                 MOVLW  140             ;
 45  000032  0754                 ADDWF  14              ;
 46  000033  3003  5044           BC     OVR-1           ;
 47  000035  3412                 BTFSS  12,0            ;
 48  000036  3015                 BTFSC  15,0            ;
```

```
LINE   ADDR   B1   B2      SIGNED-BCD SUBTRACTION                    PAGE   2

 49  000037  5043            GOTO   OVR-2        ;
 50  000040  0254            SUBWF  14           ;
 51  000041  2003            CLRC                ;
 52  000042  5044            GOTO   OVR-1        ;
 53  000043  2403            SETC                ;
 54  000044  1552            RLF    12           ;
 55  000045  1011       OVR  MOVF   11,W         ;ADD SIGNS.
 56  000046  0753            ADDWF  13           ;
 57  000047  3103            SKPNZ               ;RESULT=0?
 58  000050  5076            GOTO   BPOS         ;YES!-- BOTH SIGNS +VE.
 59  000051  1013            MOVF   13,W         ;NO! THEN RESULT=9?
 60  000052  7411            XORLW  11           ;
 61  000053  3103            SKPNZ               ;
 62  000054  5065            GOTO   OPPST        ;YES!-- SIGNS OPPOSITE-NO OVERFLOW.
 63  000055  3412            BTFSS  12,0         ;NO!-- BOTH SIGNS -VE.
 64  000056  5062            GOTO   OVFLW        ;TEST SAVED CY.CY=0-OVERFLOW.
 65  000057  6231            MOVLW  231          ;CY=1-NO OVERFLOW.
 66  000060  0053            MOVWF  13           ;SET SIGN -VE.
 67  000061  5105            GOTO   FIN          ;
 68  000062  6230      OVFLW MOVLW  230          ;OVERFLOW-SET SIGN -VE &
 69  000063  0053            MOVWF  13           ;OVERFLOW DIGIT =1.
 70  000064  5105            GOTO   FIN          ;
 71  000065  3012      OPPST BTFSC  12,0         ;TEST SAVED CY.
 72  000066  5074            GOTO   POS          ;CY=1!
 73  000067  3052            BTFSC  12,1         ;TEST CY AFTER 1ST ADJUST.
 74  000070  5074            GOTO   POS          ;
 75  000071  6231            MOVLW  231          ;SET SIGN -VE.
 76  000072  0053            MOVWF  13           ;
 77  000073  5105            GOTO   FIN          ;
 78  000074  0153       POS  CLRF   13           ; SET SIGN +VE.
 79  000075  5105            GOTO   FIN          ;
 80  000076  3012      BPOS  BTFSC  12,0         ;TEST SAVED CY.
 81  000077  5103            GOTO   OVFLW1       ;CY=1! OVERFLOW.
 82  000100  6000            MOVLW  00           ;CY=0! NO OVERFLOW-SET SIGN
 83  000101  0053            MOVWF  13           ;+VE & OVERFLOW DGT 0.
 84  000102  5105            GOTO   FIN          ;
 85  000103  6001      OVFLW1 MOVLW  1            ;SET SIGN +VE & OVER-
 86  000104  0053            MOVWF  13           ;FLOW DGT 1.
 87  000105  4000       FIN  RET                 ;FINISHED-RETURN.
 88  000106               END


ASSEMBLER ERRORS =    0
```

5.5 Two Digit BCD Multiply

Program Name: BCDM2D

Objective: This routine yields a 4 BCD digit product when two 2 BCD digit numbers are input.

Input Data:
1. 2 digit BCD multiplier in register A
2. 2 digit BCD multiplicand in register B

Output Data: 4 digit product in registers A, B

Approach: The algorithm used to compute the product of two 2 digit numbers is as follows:

if $A = A_1,A_2$ and $B = B_1,B_2$
where A_1,A_2,B_1,B_2 are single BCD digits
$A,B = A_2 \bullet B_2 + 10 * A_1B_2 + 10 * A_2B_1 + 100 * A_1B_1$

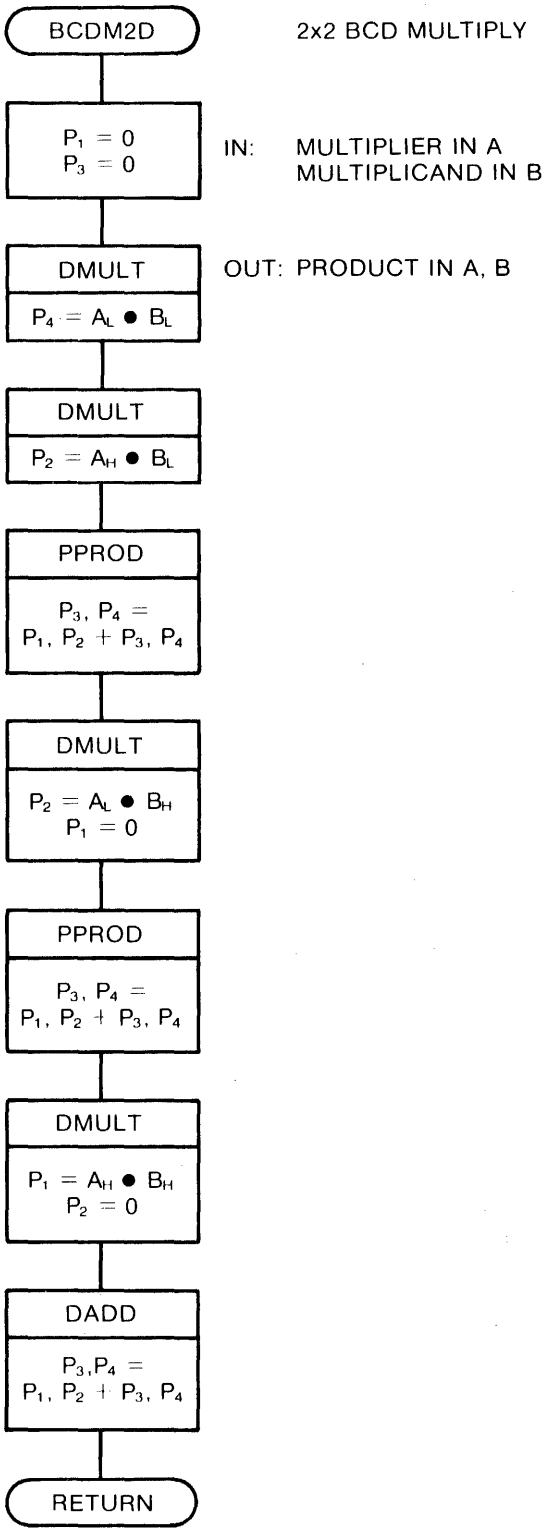
The single digit multiply is accomplished via repeated addition.

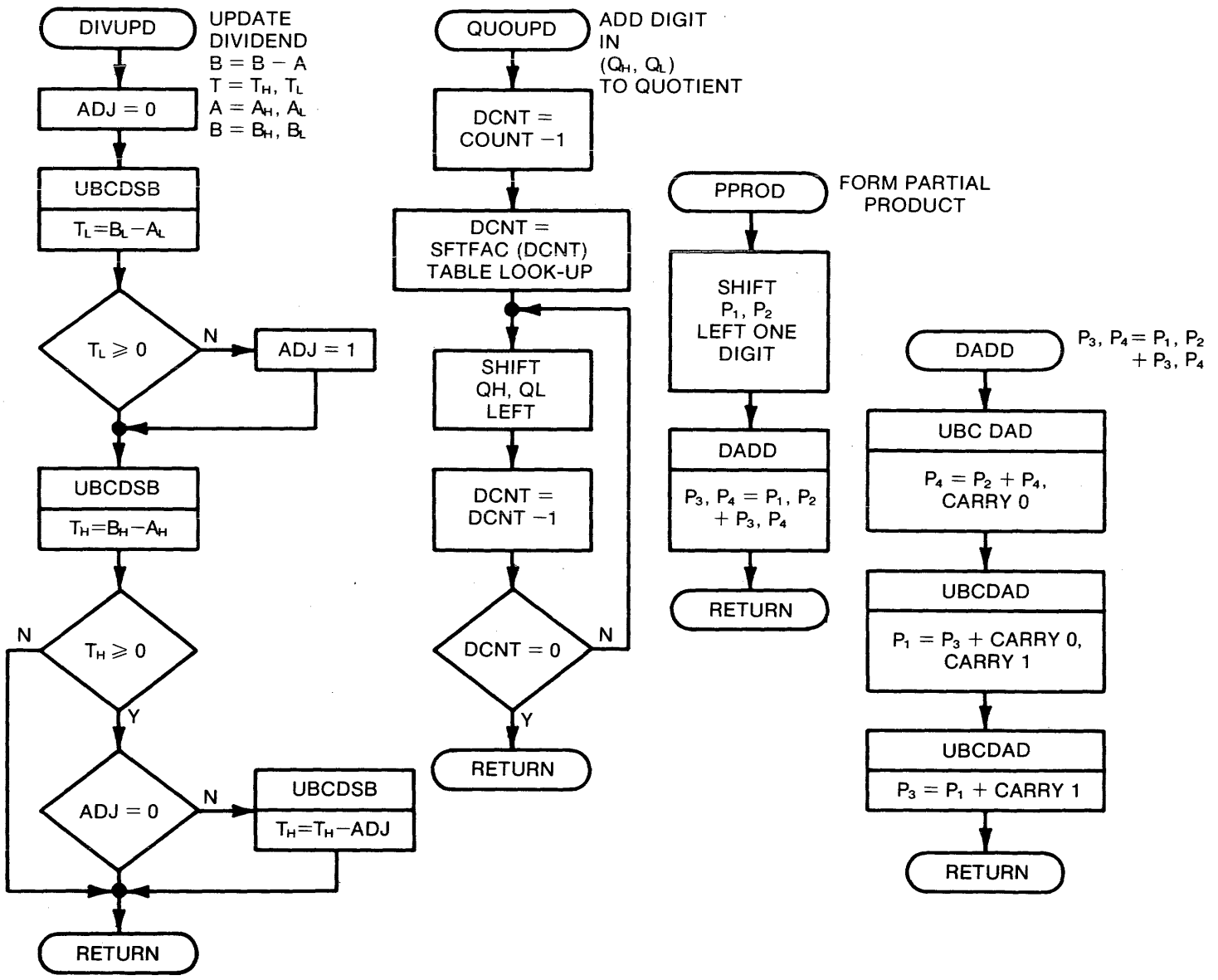
This routine may be used to multiply two 4 digit BCD numbers by using the same algorithm above but calling BCDM2D instead of the single digit multiply routine as follows:

$A = A_1A_2A_3A_4 \qquad B = B_1B_2B_3B_4$
$A,B = A_3A_4 * B_3B_4 + 100(A_1A_2 * B_3B_4) + 100\ (A_3A_4 * B_1B_2) + 10000(A_1A_2 * B_1B_2)$

The multiply by powers of 10 is accomplished by shifting left one BCD digit (4 bits) for each power of 10.

NOTE: This routine uses 2 levels of subroutine nesting, so it can only be called from the main line program in a PIC1656. For use in a PIC1650A or PIC1655A, either do not use this routine as a subroutine, or modify it to use only one level of subroutine nesting.


BCDM2D
2x2 BCD MULTIPLY
P1 = 0
P3 = 0
IN: MULTIPLIER IN A
MULTIPLICAND IN B
DMULT
P4 = AL • BL
OUT: PRODUCT IN A, B
DMULT
P2 = AH • BL
PPROD
P3, P4 =
P1, P2 + P3, P4
DMULT
P2 = AL • BH
P1 = 0
PPROD
P3, P4 =
P1, P2 + P3, P4
DMULT
P1 = AH • BH
P2 = 0
DADD
P3,P4 =
P1, P2 + P3, P4
RETURN


DIVUPD
UPDATE DIVIDEND
B = B − A
T = TH, TL
A = AH, AL
B = BH, BL
ADJ = 0
UBCDSB
TL=BL−AL
TL ⩾ 0
N
ADJ = 1
UBCDSB
TH=BH−AH
TH ⩾ 0
N
Y
ADJ = 0
N
UBCDSB
TH=TH−ADJ
RETURN
QUOUPD
ADD DIGIT IN (QH, QL) TO QUOTIENT
DCNT = COUNT −1
DCNT = SFTFAC (DCNT) TABLE LOOK-UP
SHIFT QH, QL LEFT
DCNT = DCNT −1
DCNT = 0
N
Y
RETURN
PPROD
FORM PARTIAL PRODUCT
SHIFT P1, P2 LEFT ONE DIGIT
DADD
P3, P4 = P1, P2 + P3, P4
RETURN
DADD
P3, P4 = P1, P2 + P3, P4
UBC DAD
P4 = P2 + P4, CARRY 0
UBCDAD
P1 = P3 + CARRY 0, CARRY 1
UBCDAD
P3 = P1 + CARRY 1
RETURN

5.6 Four Digit BCD Divide

Program Name: BCDD4D

Objective: This routine yields a 4 digit BCD quotient when two 4 digit BCD numbers are input.

Input Data:
1. 4 digit BCD divisor in registers A_H, A_L
2. 4 digit BCD dividend in registers B_H, B_L

Output Data:
1. 4 digit quotient in registers A_H, A_L
2. Remainder in registers B_H, B_L

Approach: The algorithm used to compute the quotient is similar to that used in long division.

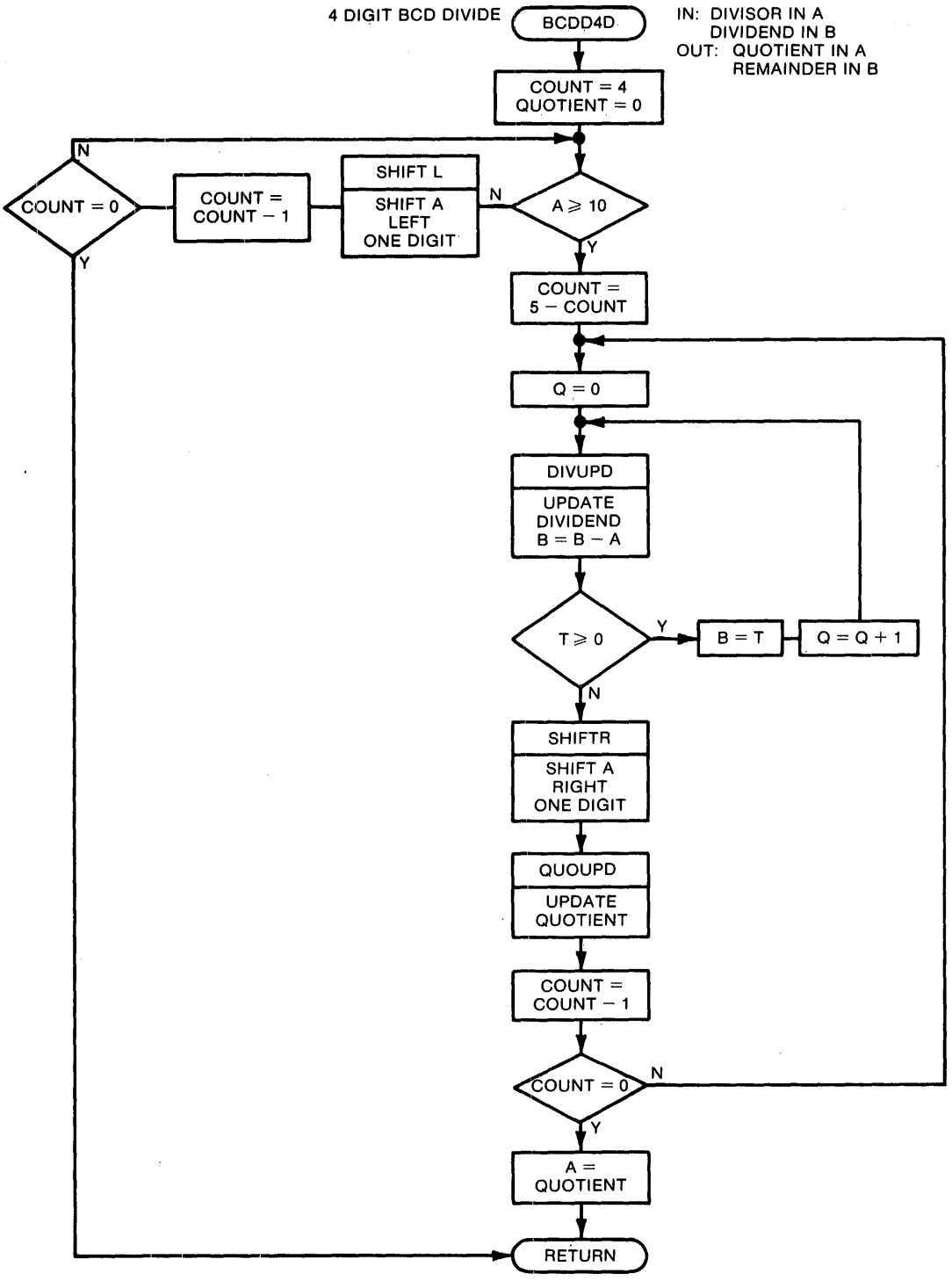
The divisor is first normalized such that the most significant digit (MSD) is in the 1000 place digit position.

The normalized divisor is then repeatedly subtracted from the dividend until the result is negative. The number of times that the divisor is subtracted is the decimal digit that is stored in the quotient. The dividend is restored to the value it had before the negative result, the divisor is shifted right one digit, and the above process is repeated. This process continues until the entire quotient is computed. An example is shown below.

DIVISOR = 25
DIVIDEND = 625

1. Divisor normalized to 2500
2. Digit count = 4 − no. of shifts necessary to normalize divisor = 2

```
          025
2500 ) 0625
       -2500     DC = 2, COUNT = 0      PLACE 0 IN POSITION 2
       -1875
        0625
        0250     DC = 1, COUNT = 0
        0375
        0250     DC = 1, COUNT = 1
        0125
        0250     DC = 1, COUNT = 2      PLACE 2 IN POSITION 1
        -125
        0125
        0025     DC = 0, COUNT = 0
        0100
        0025     DC = 0, COUNT = 1
        0075
        0025     DC = 0, COUNT = 2
        0050
        0025     DC = 0, COUNT = 3
        0025
        0025     DC = 0, COUNT = 4
        0000
        0025     DC = 0, COUNT = 5      PLACE 5 IN POSITION 0
         -25
REMAINDER
```


4 DIGIT BCD DIVIDE
BCDD4D
IN: DIVISOR IN A
DIVIDEND IN B
OUT: QUOTIENT IN A
REMAINDER IN B
COUNT = 4
QUOTIENT = 0
N
COUNT = 0
Y
COUNT =
COUNT − 1
SHIFT L
SHIFT A
LEFT
ONE DIGIT
N
A ⩾ 10
Y
COUNT =
5 − COUNT
Q = 0
DIVUPD
UPDATE
DIVIDEND
B = B − A
T ⩾ 0
Y
B = T
Q = Q + 1
N
SHIFTR
SHIFT A
RIGHT
ONE DIGIT
QUOUPD
UPDATE
QUOTIENT
COUNT =
COUNT − 1
COUNT = 0
N
Y
A =
QUOTIENT
RETURN

```
LINE    ADDR   B1    B2       BCD OPERATIONS                                    PAGE    1

  1                                TITLE  'BCD OPERATIONS'
  2                                                        ;
  3                                                        ; FILE DEFINITIONS
  4                                                        ;
  5  000004                 FSR     EQU    4              ;FILE SELECT REGISTER
  6  000011                 TEMPH   EQU    11             ;TEMPORARY HIGH
  7  000012                 TEMPL   EQU    12             ;TEMPORARY LOW
  8  000013                 A       EQU    13             ;INPUT MULTIPLICAND/OUTPUT HI PRODUCT
  9  000014                 B       EQU    14             ;INPUT MULTIPLIER/OUTPUT LO PRODUCT
 10  000015                 P1      EQU    15             ;FIRST PARTIAL PRODUCT
 11  000016                 P2      EQU    16             ;SECOND  "       "
 12  000017                 P3      EQU    17             ;THIRD   "       "
 13  000020                 P4      EQU    20             ;FOURTH  "       "
 14  000021                 COUNT   EQU    21             ;MULTIPLY COUNTER
 15  000022                 MULTC   EQU    22             ;MULTIPLY MULTIPLIER
 16                                                       ;
 17  000000
 18  000004                 MTEN    EQU    4              ;SHIFT 10
 19  000360                 UBMSK   EQU    X'F0'          ;UPPER DIGIT MASK
 20  000017                 LBMSK   EQU    X'0F'          ;LOWER DIGIT MASK
 21                                                       ;
 22                                                       ; DIVIDE DEFINITIONS
 23                                                       ;
 24  000013                 AH      EQU    A              ;HI DIVISOR/HI QUOTIENT
 25  000014                 AL      EQU    B              ;LO DIVISOR/LO QUOTIENT
 26  000015                 BH      EQU    P1             ;HI DIVIDEND
 27  000016                 BL      EQU    P2             ;LO DIVIDEND/REMAINDER
 28  000017                 QH      EQU    P3             ;HI PARTIAL QUOTIENT
 29  000020                 QL      EQU    P4             ;LO PARTIAL QUOTIENT
 30  000022                 QUOTH   EQU    MULTC          ;HI QUOTIENT(TEMP)
 31  000023                 QUOTL   EQU    23             ;LO QUOTIENT(TEMP)
 32  000011                 SIGN    EQU    TEMPH          ;SIGN INDICATOR
 33  000011                 DCNT    EQU    TEMPH          ;SHIFT COUNTER
 34  000024                 TH      EQU    24             ;TEMP DIVIDEND
 35  000025                 TL      EQU    25             ; "     "
 36  000026                 ADJ     EQU    26             ;CARRY ADJUST FOR SUBTRACTION
 37                                                       ;
 38                                                       ;
 40                                                       ;
 41                                                       ; SHIFT FACTOR FOR BCD DIGITS
 42                                                       ;
 43  000000  0742          SFTFAC  ADDWF  2
 44  000001  0000                  NOP
 45  000002  4004                  RETLW  .4
 46  000003  4010                  RETLW  .8
 47  000004  4014                  RETLW  .12
```

```
 49 000005
 50                                                    ;
 51                                                    ; 4 DIGIT DIVIDE: (BH,BL)/(AH,AL) ---> (AH,AL),BL
 52                                                    ;
 53 000005 0162          BCDD4D  CLRF   QUOTH
 54 000006 0163                  CLRF   QUOTL
 55 000007 6004                  MOVLW  4
 56 000010 0061                  MOVWF  COUNT
 57 000011 6012          NORM    MOVLW  .10
 58 000012 0213                  SUBWF  AH,W
 59 000013 3003                  SKPNC                 ;IF AH IS >10
 60 000014 5021                  B      DIV            ;DIVISOR(A) IS NORMALIZED,DO DIVIDE
 61 000015 4507                  CALL   SHIFTL         ;ELSE SHIFT ONE DIGIT LEFT
 62 000016 1361                  DECFSZ COUNT          ;KEEP TRACK OF SHIFTS
 63 000017 5011                  B      NORM
 64 000020 5050                  B      EXIT           ;DIVISOR=0, EXIT
 65                                                    ;
 66 000021 6006          DIV     MOVLW  6
 67 000022 0261                  SUBWF  COUNT          ;USE COUNT FOR QUOTIENT DIGIT COUNT
 68 000023 1161                  COMF   COUNT
 69 000024 0160          DLOOP   CLRF   QL
 70 000025 0157                  CLRF   QH
 71 000026 4451          NLOOP   CALL   DIVUPD         ;SUBTRACT DIVISOR FROM PARTIAL DIVIDEND
 72 000027 1051                  TSTF   SIGN
 73 000030 3503                  SKPZ                  ;IF NEGATIVE, ADD Q TO QUOTIENT
 74 000031 5040                  B      SAVEQ
 75 000032 1024                  MOVF   TH,W
 76 000033 0055                  MOVWF  BH             ;UPDATE DIVIDEND
 77 000034 1025                  MOVF   TL,W
 78 000035 0056                  MOVWF  BL
 79 000036 1260                  INCF   QL
 80 000037 5026                  B      NLOOP
 81 000040 4517          SAVEQ   CALL   SHIFTR         ;SHIFT DIVISOR RIGHT ONE DIGIT
 82 000041 4527                  CALL   QUOUPD         ;UPDATE QUOTIENT
 83 000042 1361          CHKCNT  DECFSZ COUNT          ;UPDATE COUNT
 84 000043 5024                  B      DLOOP
 85 000044 1022                  MOVF   QUOTH,W        ;IF COUNT=0,SAVE QUOTIENT,EXIT
 86 000045 0053                  MOVWF  AH
 87 000046 1023                  MOVF   QUOTL,W
 88 000047 0054                  MOVWF  AL
 89 000050 4000          EXIT    RET
 91                                                    ;
 92                                                    ; SUBTRACT DIVISOR FROM DIVIDEND
 93                                                    ; (BH,BL)-(AH,AL) ---> SIGN,TH,TL
 94                                                    ;
 95 000051 0166          DIVUPD  CLRF   ADJ
 96 000052 1016                  MOVF   BL,W
 97 000053 0052                  MOVWF  TEMPL
 98 000054 1014                  MOVF   AL,W
 99 000055 0051                  MOVWF  TEMPH
100 000056 4716                  CALL   UBCDSB         ;TEMPL-TEMPH
```

```
101  000057  1012            MOVF   TEMPL,W
102  000060  0065            MOVWF  TL           ;RESULT
103  000061  1051            TSTF   TEMPH        ;
104  000062  3503            SKPZ                ;IF RESULT (-),
105  000063  1266            INCF   ADJ          ;ADJ=1
106  000064  1015            MOVF   BH,W
107  000065  0052            MOVWF  TEMPL
108  000066  1013            MOVF   AH,W
109  000067  0051            MOVWF  TEMPH        ;
110  000070  4716            CALL   UBCDSB       ;TEMPL-TEMPH
111  000071  1051            TSTF   TEMPH
112  000072  3503            SKPZ
113  000073  5106            B      EXIT1        ;IF RESULT NEGATIVE,EXIT
114  000074  1066            TSTF   ADJ
115  000075  3103            SKPNZ               ;
116  000076  5104            B      SAVE         ;ADJ=0,NO ADJUSTMENT
117  000077  1015            MOVF   BH,W
118  000100  0052            MOVWF  TEMPL
119  000101  1026            MOVF   ADJ,W
120  000102  0051            MOVWF  TEMPH
121  000103  4716            CALL   UBCDSB       ;ADJUST HI RESULT
122  000104  1012     SAVE   MOVF   TEMPL,W
123  000105  0064            MOVWF  TH           ;HI RESULT
124  000106  4000     EXIT1  RET
126                                              ;
127                                              ; SHIFT AH,AL LEFT ONE BCD DIGIT
128                                              ;
129  000107  6004     SHIFTL MOVLW  4
130  000110  0051            MOVWF  DCNT
131  000111  2003     SLLOOP CLRC
132  000112  1554            RLF    AL
133  000113  1553            RLF    AH
134  000114  1351            DECFSZ DCNT
135  000115  5111            B      SLLOOP
136  000116  4000            RET
137                                              ;
138                                              ; SHIFT AH,AL RITE ONE DIGIT
139                                              ;
140  000117  6004     SHIFTR MOVLW  4
141  000120  0051            MOVWF  DCNT
142  000121  2003     SRLOOP CLRC
143  000122  1453            RRF    AH
144  000123  1454            RRF    AL
145  000124  1351            DECFSZ DCNT
146  000125  5121            B      SRLOOP
147  000126  4000            RET
148                                              ;
149                                              ; UPDATE QUOTIENT BY SHIFTING Q AND ADDING IT TO Q
                                                  -UOTIENT
150                                              ;
```

```
151  000127  1021        QUOUPD  MOVF   COUNT,W
152  000130  0051                MOVWF  DCNT
153  000131  1351                DECFSZ DCNT
154  000132  5134                GOTO   FNDSFT        ;IF COUNT=/=1,  SHIFT
155  000133  5144                GOTO   ORQUOT        ;NO SHIFT
156  000134  1011        FNDSFT  MOVF   DCNT,W
157  000135  4400                CALL   SFTFAC
158  000136  0051                MOVWF  DCNT          ;GET SHIFT FACTOR
159  000137  2003        SQLOOP  CLRC
160  000140  1560                RLF    QL
161  000141  1557                RLF    QH
162  000142  1351                DECFSZ DCNT
163  000143  5137                B      SQLOOP
164  000144  1017        ORQUOT  MOVF   QH,W
165  000145  0462                IORWF  QUOTH         ;ADD TO QUOTIENT
166  000146  1020                MOVF   QL,W
167  000147  0463                IORWF  QUOTL
168  000150  4000                RET
170                                                   ;
171                                                   ; DOUBLE DIGIT BCD MULTIPLY
172                                                   ;
173                                                   ; INPUT: 2 DIGIT MULTIPLICAND IN REGISTER A
174                                                   ;        2 DIGIT MULTIPLIER IN REGISTER B
175                                                   ;
176                                                   ; OUTPUT: 4 DIGIT PRODUCT IN A,B
177                                                   ;
178  000151  0157        BCDM2D  CLRF   P3            ;CLEAR PARTIAL PRODUCT 3
179  000152  0155                CLRF   P1            ; AND PARTIAL PRODUCT 1
180  000153  6017                MOVLW  LBMSK
181  000154  0513                ANDWF  A,W           ;AL
182  000155  0060                MOVWF  P4
183  000156  6017                MOVLW  LBMSK
184  000157  0514                ANDWF  B,W           ;BL
185  000160  0062                MOVWF  MULTC
186  000161  6020                MOVLW  P4
187  000162  0044                MOVWF  FSR           ;FSR=P4
188  000163  4664                CALL   DMULT         ;AL*BL
189  000164  6360                MOVLW  UBMSK
190  000165  0513                ANDWF  A,W
191  000166  0056                MOVWF  P2            ;AH
192  000167  1656                SWAPF  P2
193  000170  6016                MOVLW  P2
194  000171  0044                MOVWF  FSR           ;FSR=P2
195  000172  4664                CALL   DMULT         ;AH*BL
196  000173  6004                MOVLW  NTEN
197  000174  0061                MOVWF  COUNT         ;SHIFT P1,P2 BY TEN
198                                                   ;FORM PARTIAL PRODUCT IN P3,P4
199  000175  2003        PPRD1   CLRC
200  000176  1556                RLF    P2
201  000177  1555                RLF    P1
202  000200  1361                DECFSZ COUNT
```

```
203  000201  5175          B       PPRD1
204  000202  4644          CALL    DADD
205                                        ;
206  000203  0155          CLRF    P1
207  000204  6360          MOVLW   UBMSK
208  000205  0514          ANDWF   B,W          ;BH
209  000206  0062          MOVWF   MULTC
210  000207  1662          SWAPF   MULTC
211  000210  6017          MOVLW   LBMSK
212  000211  0513          ANDWF   A,W          ;AL
213  000212  0056          MOVWF   P2
214  000213  6016          MOVLW   P2
215  000214  0044          MOVWF   FSR          ;FSR=P2
216  000215  4664          CALL    DMULT        ;AL*BH
217  000216  6004          MOVLW   NTEN
218  000217  0061          MOVWF   COUNT        ;SHIFT P1,P2 BY TEN
219                                        ;FORM PARTIAL PRODUCT IN P3,P4
220  000220  2003  PPRD2   CLRC
221  000221  1556          RLF     P2
222  000222  1555          RLF     P1
223  000223  1361          DECFSZ  COUNT
224  000224  5220          B       PPRD2
225  000225  4644          CALL    DADD
226                                        ;
227  000226  0156          CLRF    P2
228  000227  6360          MOVLW   UBMSK
229  000230  0513          ANDWF   A,W          ;AH
230  000231  0055          MOVWF   P1
231  000232  1655          SWAPF   P1           ;PRODUCT IN P1 TO SHIFT BY 100
232  000233  6015          MOVLW   P1
233  000234  0044          MOVWF   FSR          ;FSR=P1
234  000235  4664          CALL    DMULT        ;AH*BH
235  000236  4644          CALL    DADD         ;ADD P1,P2 TO P3,P4 FOR FINAL PRODUCT
236  000237  1017          MOVF    P3,W
237  000240  0053          MOVWF   A            ;FINAL HIGH PRODUCT
238  000241  1020          MOVF    P4,W
239  000242  0054          MOVWF   B            ;FINAL LO PRODUCT
240  000243  4000          RET
242                                        ;
243                                        ; ADD P1,P2 TO P3,P4 AND STORE RESULT IN P3,P4
244                                        ;
245  000244  1016  DADD    MOVF    P2,W
246  000245  0051          MOVWF   TEMPH        ;P2 IN TEMPH FOR UBCDAD
247  000246  1020          MOVF    P4,W
248  000247  0052          MOVWF   TEMPL        ;P4 INU TEMPL FOR UBCDAD
249  000250  4716          CALL    UBCDAD       ;P2+P4
250  000251  1012          MOVF    TEMPL,W
251  000252  0060          MOVWF   P4           ;LO RESULT IN P4
252  000253  1017          MOVF    P3,W
253  000254  0052          MOVWF   TEMPL
254  000255  4716          CALL    UBCDAD       ;P2+P3+CARRY 0
```

```
255  000256  1015          MOVF   P1,W
256  000257  0051          MOVWF  TEMPH
257  000260  4716          CALL   UBCDAD       ;P1+P3+CARRY0+CARRY1
258  000261  1012          MOVF   TEMPL,W
259  000262  0057          MOVWF  P3           ;HI RESULT IN P3
260  000263  4000          RET
262                                            ;
263                                            ; SINGLE BCD DIGIT MULTIPLY
264                                            ; INPUT- MULTC: MULTIPLIER
265                                            ;        FSR: MULTIPICAND/PRODUCT
266                                            ;
267  000264  1000    DMULT MOVF   0,W
268  000265  3103          SKPNZ
269  000266  5315          B      EXITM        ;FRS=FSR,EXIT
270  000267  1300          DECFSZ 0,W
271  000270  5273          B      CHKM
272  000271  1022          MOVF   MULTC,W
273  000272  5314          B      STRF         ;FSR=1,FSR=MULTC
274  000273  1022    CHKM  MOVF   MULTC,W
275  000274  3503          SKPZ
276  000275  5300          B      CONT
277  000276  0040          MOVWF  0
278  000277  5315          B      EXITM        ;MULTC=0,FSR=FSR,EXIT
279  000300  0061    CONT  MOVWF  COUNT
280  000301  1361          DECFSZ COUNT        ;COUNT=MULTC-1
281  000302  5304          B      MUL
282  000303  5315          B      EXITM        ;MULTC=1,FSR=FSR,EXIT
283  000304  1000    MUL   MOVF   0,W
284  000305  0052          MOVWF  TEMPL
285  000306  1000    LOOP  MOVF   0,W
286  000307  0051          MOVWF  TEMPH
287  000310  4716          CALL   UBCDAD       ;ADD MULIPLICAND TO ITSELF
288  000311  1361          DECFSZ COUNT        ;MULTIPLIER TIMES
289  000312  5306          B      LOOP
290  000313  1012          MOVF   TEMPL,W      ;LO RESULT IS FINAL
291  000314  0040    STRF  MOVWF  0            ;ADD CARRY TO UPPER DIGIT
292  000315  4000    EXITM RET
```

5.7a Binary To BCD Conversion Method 1

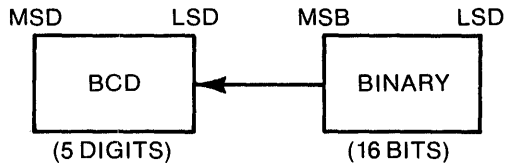
Program Name: BINTOB

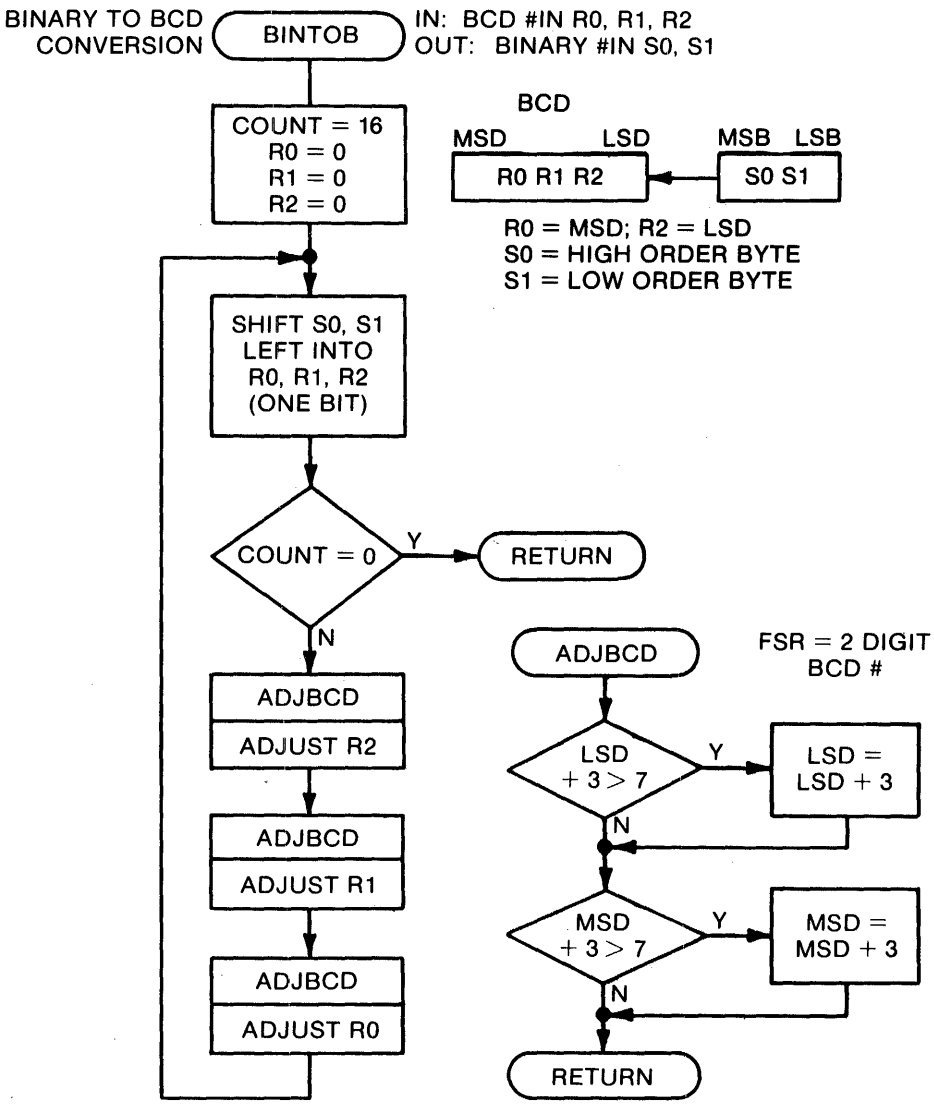
Objective: This routine converts a 16 bit binary number to a 5 digit BCD number.

Input Data: The 16 bit binary number is input in registers S0, S1 with S0 containing the high order byte.

Output Data: The 5 digit BCD number is output in registers R0, R1, R2 with R0 containing the MSD in its rightmost nibble.

Approach: A very simple algorithm is used to accomplish the conversion. The binary number is shifted left one bit into the BCD number. If 16 shifts were performed, the program exits. Otherwise, each BCD digit is checked for a value greater than 4. If this is the case, 3 is added to the digit. The above process is then repeated.




BINARY TO BCD CONVERSION
BINTOB
IN: BCD #IN R0, R1, R2
OUT: BINARY #IN S0, S1
BCD
MSD LSD
MSB LSB
R0 R1 R2
S0 S1
R0 = MSD; R2 = LSD
S0 = HIGH ORDER BYTE
S1 = LOW ORDER BYTE
COUNT = 16
R0 = 0
R1 = 0
R2 = 0
SHIFT S0, S1
LEFT INTO
R0, R1, R2
(ONE BIT)
COUNT = 0
Y
RETURN
N
ADJBCD
ADJUST R2
ADJBCD
ADJUST R1
ADJBCD
ADJUST R0
ADJBCD
FSR = 2 DIGIT
BCD #
LSD
+ 3 > 7
Y
LSD =
LSD + 3
N
MSD
+ 3 > 7
Y
MSD =
MSD + 3
N
RETURN

```
LINE   ADDR   B1   B2                PIC MACRO ASSEMBLER VER 1.0              PAGE   3

 48                                                        ;
 49                                                        ; BINARY TO BCD CONVERSION
 50                                                        ; INPUT 16 BIT BINARY NUMBER IN S0,S1
 51                                                        ; OUTPUT 5 DIGIT BCD NUMBER IN R0,R1,R2
 52                                                        ;
 53  000036  6020          BINTOB  MOVLW  .16
 54  000037  0056                  MOVWF  COUNT
 55  000040  0151                  CLRF   R0            ;CLEAR BCD NO.
 56  000041  0152                  CLRF   R1
 57  000042  0153                  CLRF   R2
 58  000043  1555          LOOPC   RLF    S1            ;SHIFT BINARY INTO BCD NO.
 59  000044  1554                  RLF    S0
 60  000045  1553                  RLF    R2
 61  000046  1552                  RLF    R1
 62  000047  1551                  RLF    R0
 63  000050  1356                  DECFSZ COUNT
 64  000051  5053                  B      ADJDEC
 65  000052  4000                  RET                  ;EXIT IF 16 SHIFTS
 66                                                        ;
 67  000053  6013          ADJDEC  MOVLW  R2
 68  000054  0044                  MOVWF  FSR
 69  000055  4465                  CALL   ADJBCD        ;ADJUST R2
 70  000056  6012                  MOVLW  R1
 71  000057  0044                  MOVWF  FSR
 72  000060  4465                  CALL   ADJBCD        ;ADJUST R1
 73  000061  6011                  MOVLW  R0
 74  000062  0044                  MOVWF  FSR
 75  000063  4465                  CALL   ADJBCD        ;ADJUST R0
 76  000064  5043                  B      LOOPC
 77                                                        ;
 78  000065  6003          ADJBCD  MOVLW  X'03'
 79  000066  0700                  ADDWF  0,W           ;ADD 3 TO LSD
 80  000067  0057                  MOVWF  TEMP
 81  000070  3157                  BTFSC  TEMP,3        ;IF RESULT > 7
 82  000071  0040                  MOVWF  0             ;SAVE INTO LSD
 83  000072  6060                  MOVLW  X'30'
 84  000073  0700                  ADDWF  0,W           ;ADD 3 TO MSD
 85  000074  0057                  MOVWF  TEMP
 86  000075  3357                  BTFSC  TEMP,7        ;IF RESULT > 7
 87  000076  0040                  MOVWF  0             ;SAVE INTO MSD
 88  000077  4000                  RET
 89  000100                        END


ASSEMBLER ERRORS =    0
```

5.7b Binary To BCD Conversion (2 digits) Method II

This routine converts the 8 bit binary number in the W register to a 2 digit BCD number, which is then converted to drive 2 7-segment LED displays from ports 6 and 7.

STEMP 1 is the temporary register which will contain the least significant digit on conversion.

STEMP 2 is the temporary register which will contain the most significant digit on conversion.

DIG 1 is the least significant digit output port.

DIG 2 is the most significant digit output port.

```
OUTPUT      CLRF    STEMP2
            MOVWF   STEMP1
GTENTH      MOVLW    10              ; sub .10 from STEMP1
            SUBWF   STEMP1,W
            SKPC                    ; Positive
            GOTO    GSEGL
            MOVWF   STEMP1          ; Yes
            INCF    STEMP2
            GOTO    GTENTH
GSEGL       MOVLW   TBSTRL
            ADDWF   STEMP1,W
            CALL    CONVRT
AA          MOVWF   DIG1
GSEG2       MOVLW   TBSTRL
            ADDWF   STEMP2,W
            CALL    CONVRT
CC          MOVWF   DIG2
            RET
CONVRT      MOVWF   2
TBSTRL      RETLW   300             ; Decimal to 7 seg. conversion table
            RETLW   371
            RETLW   244
            RETLW   260
            RETLW   231
            RETLW   222
            RETLW   202
            RETLW   230
            RETLW   200
            RETLW   230
```

The algorithm used here is to count the number of times 10 (ten) can be subtracted from the binary number before a negative result is obtained. The count then becomes the 10's digit. The units digit is the remainder before the last subtraction.

5.8 BCD To Binary Conversion

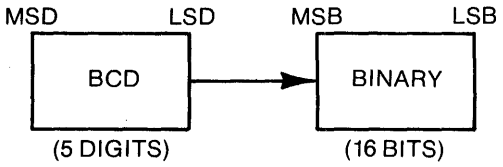
Program Name: BCDTOB

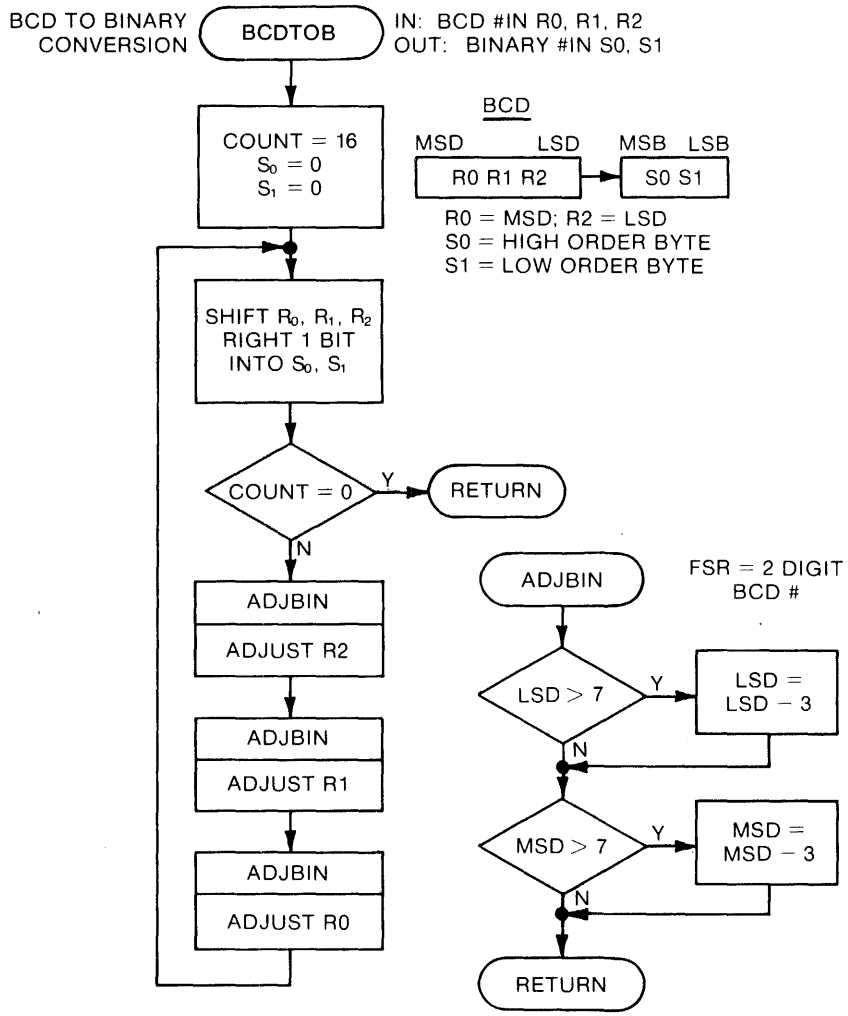
Objective: This routine converts a 5 digit BCD number to a 16 bit binary number.

Input Data: The 5 digit BCD number is input in registers R0, R1, R2 with R0 containing the MSD in its rightmost nibble.

Output Data: The 16 bit binary number is output in registers S0, S1 with S0 containing the high order byte.

Approach: The program uses a very simple algorithm to accomplish the conversion. The BCD number is shifted right one bit into the binary number. If 16 shifts were performed, the program exits. Otherwise, each BCD digit is checked for a value greater than 7. If this is the case, 3 is subtracted from the digit. The above process is then repeated.




BCD TO BINARY CONVERSION
BCDTOB
IN: BCD #IN R0, R1, R2
OUT: BINARY #IN S0, S1
BCD
MSD LSD
R0 R1 R2
MSB LSB
S0 S1
R0 = MSD; R2 = LSD
S0 = HIGH ORDER BYTE
S1 = LOW ORDER BYTE
COUNT = 16
S0 = 0
S1 = 0
SHIFT R0, R1, R2 RIGHT 1 BIT INTO S0, S1
COUNT = 0
Y
RETURN
N
ADJBIN
ADJUST R2
ADJBIN
ADJUST R1
ADJBIN
ADJUST R0
ADJBIN
FSR = 2 DIGIT BCD #
LSD > 7
Y
LSD = LSD − 3
N
MSD > 7
Y
MSD = MSD − 3
N
RETURN

```
 10                                                ;
 11                                                ; BCD TO BINARY CONVERSION
 12                                                ; 5 DIGIT BCD NUMBER INPUT IN R0,R1,R2
 13                                                ; 16 BIT BINARY NUMBER OUTPUT IN S0,S1
 14                                                ;
 15 000000 6020          BCDTOB  MOVLW  .16
 16 000001 0056                  MOVWF  COUNT
 17 000002 0154                  CLRF   S0         ;CLEAR BINARY NO.
 18 000003 0155                  CLRF   S1
 19 000004 2003          LOOPD   CLRC
 20 000005 1451                  RRF    R0         ;SHIFT BCD INTO BINARY NO.
 21 000006 1452                  RRF    R1
 22 000007 1453                  RRF    R2
 23 000010 1454                  RRF    S0
 24 000011 1455                  RRF    S1
 25 000012 1356                  DECFSZ COUNT
 26 000013 5015                  B      ADJOCT
 27 000014 4000                  RET               ;EXIT IF 16 SHIFTS
 28                                                ;
 29 000015 6013          ADJOCT  MOVLW  R2
 30 000016 0044                  MOVWF  FSR
 31 000017 4427                  CALL   ADJBIN     ;ADJUST R2
 32 000020 6012                  MOVLW  R1
 33 000021 0044                  MOVWF  FSR
 34 000022 4427                  CALL   ADJBIN     ;ADJUST R1
 35 000023 6011                  MOVLW  R0
 36 000024 0044                  MOVWF  FSR
 37 000025 4427                  CALL   ADJBIN     ;ADJUST R0
 38 000026 5004                  B      LOOPD
 39                                                ;
 40 000027 6003          ADJBIN  MOVLW  X'03'
 41 000030 3140                  BTFSC  0,3        ;IF >7
 42 000031 0240                  SUBWF  0          ;SUBTRACT 3 FROM LSD
 43 000032 6060                  MOVLW  X'30'
 44 000033 3340                  BTFSC  0,7        ;IF >7
 45 000034 0240                  SUBWF  0          ;SUBTRACT 3 FROM MSD
 46 000035 4000                  RET
```

5.9 Double Precision Signed Integer Math Package

The following is the program listing for a double precision signed integer math package, which does addition, subtraction, multiplication and division.

```
LINE    ADDR   B1    B2      MATHS                                          PAGE    1

  1                            TITLE  'MATHS'
  2                                                ; DOUBLE PRECISION SIGNED INTEGER MATH PACKAGE
  3                                                ;
  4                                                ; DEFINE THE FOLLOWING SYMBOLS:
  5                                                ;
  6                                                ; ACCA  BEGINNING OF 2 REGISTER FILE FOR FIRST OPE
                                                    -RAND
  7                                                ; ACCB  BEGINNING OF 2 REGISTER FILE FOR SECOND OP
                                                    -ERAND
  8                                                ; ACCC  2 REGISTER FILE FOR MPY/DIV
  9                                                ; ACCD   ''         ''
 10                                                ; MATORG       ORIGIN FOR LOAD OF PACKAGE
 11                                                ; TEMP  TEMPORARY SCRATCH REGISTER
 12                                                ; SIGN  TEMPORARY SCRATCH REGISTER
 13                                                ;
 14                                                ; USAGE:
 15                                                ;     LOAD ACCA AND ACCB WITH THEIR RESPECTIVE
 16                                                ; CONTENTS, CALL THE SUBROUTINE, AND OBTAIN RESULT
                                                    -
 17                                                ; IN ACCB.  ACCA IS HIGH 8 BITS, ACCA+1 IS LOW 8 B
                                                    -ITS.
 18
 19
 20  000000
 21  000000              MATORG  EQU    0
 22  000011              TEMP    EQU    11
 23  000012              ACCA    EQU    12
 24  000014              ACCB    EQU    14
 25  000016              ACCC    EQU    16
 26  000020              ACCD    EQU    20
 27  000022              SIGN    EQU    22
 28  000000
 29  000000
 30                              ORG    MATORG
 31  000000
 32                                                ; *** SUB ***
 33                                                ; ACCB - ACCA --> ACCB
 34  000000
 35  000000  4565        MSUB    CALL   NEGA
 36  000001
 37                                                ;------> IMPORTANT <------
 38                                                ; MADD MUST FOLLOW...
 39                                                ;------> IMPORTANT <------
 40  000001
 41                                                ;*** ADD ***
 42                                                ; ACCA+ACCB --> ACCB
 43  000001
 44  000001  1013        MADD    MOVF   ACCA+1,W
 45  000002  0755                ADDWF  ACCB+1
 46  000003  3003                BTFSC  3,0         ; ADD IN CARRY
 47  000004  1254                INCF   ACCB
```

```
48  000005  1012            MOVF   ACCA,W
49  000006  0754            ADDWF  ACCB
50  000007  4000            RET
51  000010
52                                          ;*** SHIFT RIGHT, ARITHMETIC ***
53                                          ; SHIFT ACCB RIGHT ONE PLACE
54                                          ; SIGN OF OPERAND IS PRESERVED (OPTIONAL)
55  000010
56  000010  2003        MASR1  CLRC
57  000011  3354               BTFSC  ACCB,7       ; ***OPTIONAL FOR SIGN
58  000012  2403               SETC                ; ***SET CARRY IF < 0
59  000013  1454               RRF    ACCB
60  000014  1455               RRF    ACCB+1
61  000015  4000               RET
62  000016
63                                          ;*** SHIFT RIGHT, ARITHMETIC, MULTIPLE PLACES
64                                          ; SHIFT ACCB RIGHT THE NUMBER OF PLACES IN W
65                                          ; CALLS MASR1
66  000016
67  000016  0051        MASR   MOVWF  TEMP          ; SAVE COUNT
68  000017  4410        MRLOOP CALL   MASR1
69  000020  1351               DECFSZ TEMP
70  000021  5017               GOTO   MRLOOP
71  000022  4000               RET
72  000023
73                                          ;*** SHIFT LEFT, ARITHMETIC ***
74                                          ; SHIFT ACCB LEFT ONE PLACE
75                                          ; SIGN OF OPERAND IS PRESERVED (OPTIONAL)
76  000023
77  000023  2003        MASL1  CLRC
78  000024  1555               RLF    ACCB+1
79  000025  1554               RLF    ACCB
80  000026  2354               BCF    ACCB,7       ; ***OPTIONAL FOR SIGN
81  000027  3003               SKPNC               ; ***
82  000030  2754               BSF    ACCB,7       ; ***CARRY... SET SIGN
83  000031  4000               RET
84  000032
85                                          ;*** SHIFT LEFT, ARITHMETIC, MULTIPLE PLACES ***
86                                          ; SHIFT ACCB LEFT THE NUMBER OF PLACES IN W
87                                          ; CALLS MASL1
88  000032
89  000032  0051        MASL   MOVWF  TEMP          ; SAVE COUNT
90  000033  4423        MLOOP  CALL   MASL1
91  000034  1351               DECFSZ TEMP
92  000035  5033               GOTO   MLOOP
93  000036  4000               RET
94  000037
95                                          ;*** INC ***
96                                          ; ACCB+1 --> ACCB
97  000037
98  000037  1255        MINC   INCF   ACCB+1
```

```
 99 000040 3103          SKPNZ
100 000041 1254          INCF   ACCB
101 000042 4000          RET
102 000043
103                                          ;*** DEC ***
104                                          ; ACCB-1 --> ACCB
105 000043
106 000043 1055     MDEC TSTF   ACCB+1
107 000044 3103          SKPNZ
108 000045 0354          DECF   ACCB
109 000046 0355          DECF   ACCB+1
110 000047 4000          RET
111 000050
112                                          ;*** MPY ***
113                                          ; A*B --> (B,C) , HIGH ORDER B, LOW C
114 000050
115 000050 4551     MPY  CALL   PSIGN
116 000051 4502          CALL   SETUP
117 000052 1460     MPLOOP RRF  ACCD         ; ROTATE D RIGHT
118 000053 1461          RRF    ACCD+1
119 000054 3003          SKPNC               ; NEED TO 'ADD' ??
120 000055 4401          CALL   MADD         ; ADD A TO B
121 000056 1454          RRF    ACCB         ; ROTATE (B,C) RIGHT
122 000057 1455          RRF    ACCB+1
123 000060 1456          RRF    ACCC
124 000061 1457          RRF    ACCC+1
125 000062 1351          DECFSZ TEMP         ; LOOP TILL  DONE
126 000063 5052          GOTO   MPLOOP
127 000064 3762          BTFSS  SIGN,7
128 000065 4000          RET
129 000066 1157          COMF   ACCC+1       ; RESTORE THE SIGN
130 000067 1257          INCF   ACCC+1
131 000070 3103          SKPNZ
132 000071 0356          DECF   ACCC
133 000072 1156          COMF   ACCC
134 000073 3103          SKPNZ
135 000074 0355     NEGB DECF   ACCB+1       ;*** NEGB ***
136 000075 1155          COMF   ACCB+1       ; A NICE WAY TO WORK THE
137 000076 3103          SKPNZ               ; ROUTINE IN.....
138 000077 0354          DECF   ACCB
139 000100 1154          COMF   ACCB
140 000101 4000          RET
141 000102
142 000102
143 000102 6020     SETUP MOVLW .16          ; 16 PLACE SHIFT
144 000103 0051          MOVWF  TEMP
145 000104 1014          MOVF   ACCB,W       ; MOVE B TO D
146 000105 0060          MOVWF  ACCD
147 000106 1015          MOVF   ACCB+1,W
148 000107 0061          MOVWF  ACCD+1
149 000110 0154          CLRF   ACCB         ; CLEAR B
```

```
150  000111  0155            CLRF   ACCB+1
151  000112  4000            RET
152  000113
153                                          ;*** DIV ***
154                                          ; ACCB/ACCA --> ACCB, REMAINDER IN ACCC
155  000113
156  000113  4551       DIV  CALL   PSIGN
157  000114  4502            CALL   SETUP
158  000115  0156            CLRF   ACCC
159  000116  0157            CLRF   ACCC+1
160  000117  1561       DLOOP RLF   ACCD+1       ; ROTATE (C,D) LEFT
161  000120  1560            RLF    ACCD
162  000121  1557            RLF    ACCC+1
163  000122  1556            RLF    ACCC
164  000123  1012            MOVF   ACCA,W       ; CHECK IF A > C
165  000124  0216            SUBWF  ACCC,W
166  000125  3503            SKPZ
167  000126  5131            GOTO   NOCHK
168  000127  1013            MOVF   ACCA+1,W     ; HIGH'S EQUAL...CHECK LOWS
169  000130  0217            SUBWF  ACCC+1,W
170  000131  3403       NOCHK SKPC
171  000132  5142            GOTO   NOGO         ; A>C, SHIFT CLEAR CARRY
172  000133  1013            MOVF   ACCA+1,W     ; C-A --> C
173  000134  0257            SUBWF  ACCC+1
174  000135  3403            BTFSS  3,0
175  000136  0356            DECF   ACCC
176  000137  1012            MOVF   ACCA,W
177  000140  0256            SUBWF  ACCC
178  000141  2403            SETC                ; SHIFT IN A ONE
179  000142  1555       NOGO  RLF   ACCB+1       ; SHIFT B LEFT
180  000143  1554            RLF    ACCB
181  000144  1351            DECFSZ TEMP         ; LOOP TILL DONE
182  000145  5117            GOTO   DLOOP
183  000146  3762            BTFSS  SIGN,7       ; FIX SIGN, IF NEG.
184  000147  4000            RET
185  000150  5074            GOTO   NEGB
186  000151
187  000151  1012       PSIGN MOVF  ACCA,W       ; PREPARE SIGN
188  000152  0614            XORWF  ACCB,W
189  000153  0062            MOVWF  SIGN
190  000154  3754            BTFSS  ACCB,7
191  000155  5163            GOTO   TRYA
192  000156  1155            COMF   ACCB+1       ; NEGB.... CANT CALL SUBR
193  000157  1255            INCF   ACCB+1
194  000160  3103            SKPNZ
195  000161  0354            DECF   ACCB
196  000162  1154            COMF   ACCB
197  000163  3752       TRYA  BTFSS  ACCA,7
198  000164  4000            RET
199  000165
200                                          ;------> IMPORTANT <------
```

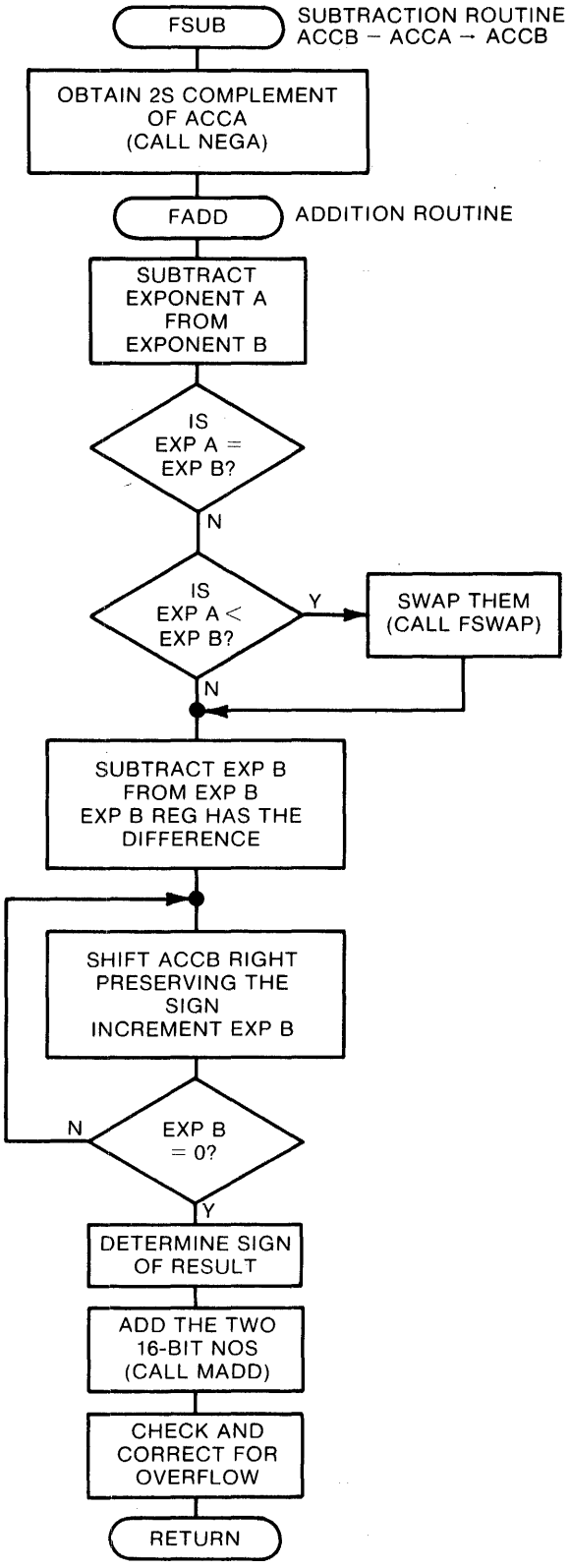
```
LINE    ADDR   B1    B2       MATHS                                          PAGE    5

 201                                          ; NEGA MUST FOLLOW...
 202                                          ;------> IMPORTANT <------
 203  000165
 204                                          ;*** NEGA ***
 205                                          ; (-ACCA) --> ACCA
 206  000165
 207  000165  1153          NEGA    COMF   ACCA+1
 208  000166  1253                 INCF   ACCA+1
 209  000167  3103                 SKPNZ
 210  000170  0352                 DECF   ACCA
 211  000171  1152                 COMF   ACCA
 212  000172  4000                 RET
 213  000173
 214  000173                       END


 ASSEMBLER ERRORS =    0
```

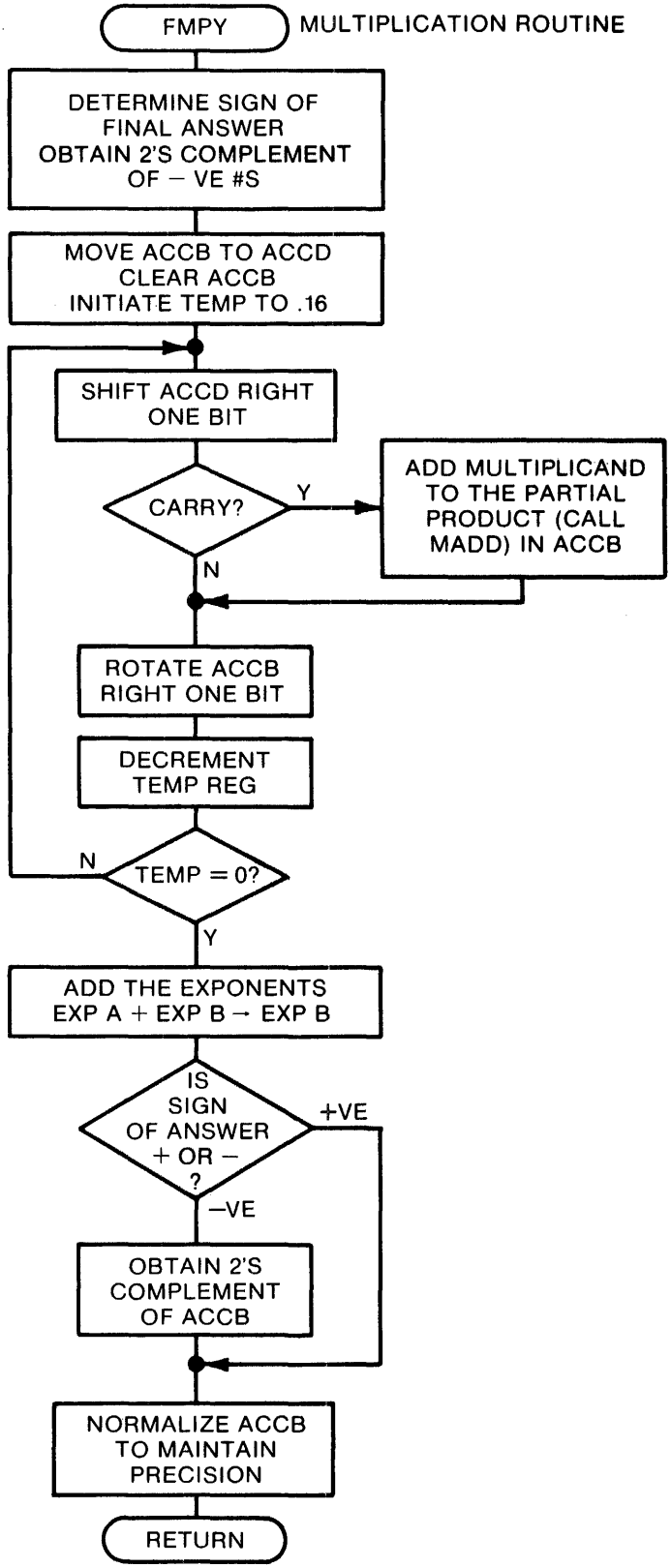
5.10 Floating-Point Double Precision Math Package

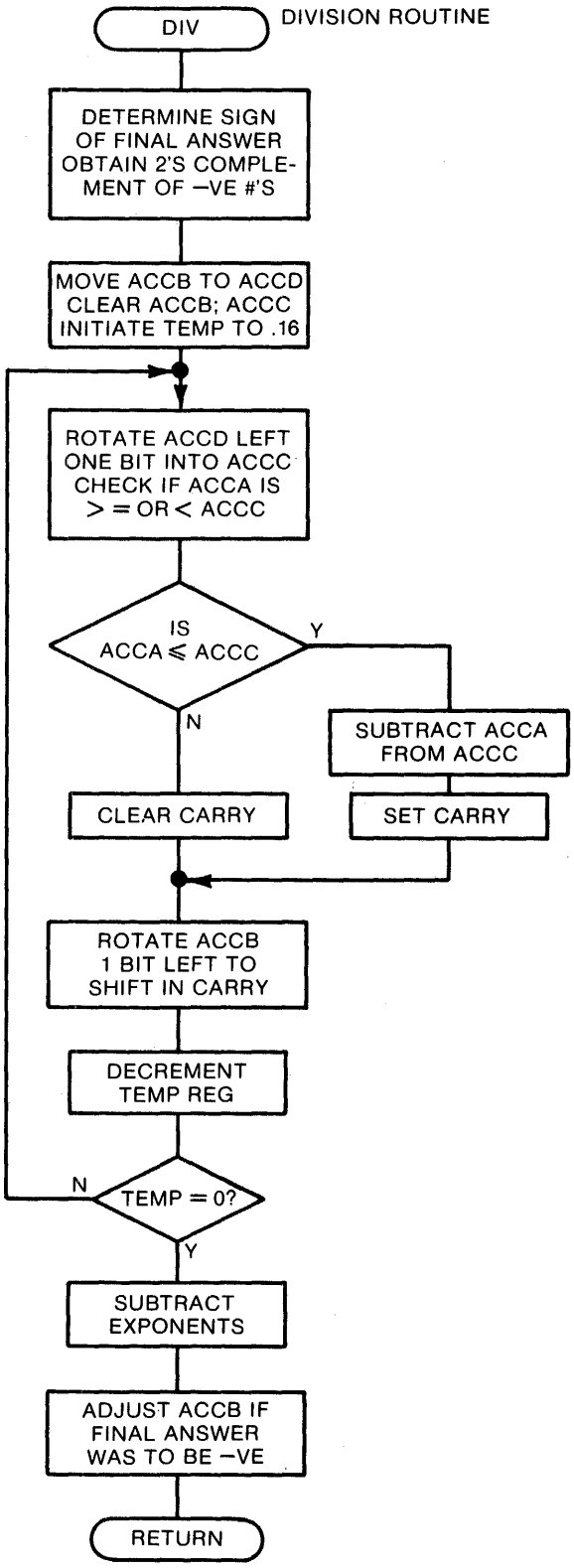
Addition, Subtraction, Multiplication and Division routines for a floating-point double precision calculations are given below. Detailed flowcharts given below describe the algorithms used. It may be observed that the powerful instruction of the PIC reduce the entire package to only 152 lines of code leaving enough space for most application programs. It is recommended that the normalize routine be called as often as possible in order to maintain the precision of the calculations. Also, since many subroutines are nested, they should be called only from the mainline.

ACCA is the beginning of 3 register accumulator
ACCB is the beginning of 3 register accumulator
ACCC 2 register file for MPY/DIV
ACCD 2 register file for MPY/DIV
TEMP Temporary Scratch Register
SIGN Temporary Scratch Register

To use the math package, load ACCA and ACCB with their respective contents, call the subroutines and obtain result in ACCB. ACCA is high 8 bits, ACCA + .1 is low 8 bits.


FSUB
SUBTRACTION ROUTINE
ACCB − ACCA → ACCB
OBTAIN 2S COMPLEMENT OF ACCA (CALL NEGA)
FADD
ADDITION ROUTINE
SUBTRACT EXPONENT A FROM EXPONENT B
IS EXP A = EXP B?
N
IS EXP A < EXP B?
Y
SWAP THEM (CALL FSWAP)
N
SUBTRACT EXP B FROM EXP B EXP B REG HAS THE DIFFERENCE
SHIFT ACCB RIGHT PRESERVING THE SIGN INCREMENT EXP B
N
EXP B = 0?
Y
DETERMINE SIGN OF RESULT
ADD THE TWO 16-BIT NOS (CALL MADD)
CHECK AND CORRECT FOR OVERFLOW
RETURN


FMPY
MULTIPLICATION ROUTINE
DETERMINE SIGN OF FINAL ANSWER OBTAIN 2'S COMPLEMENT OF – VE #S
MOVE ACCB TO ACCD CLEAR ACCB INITIATE TEMP TO .16
SHIFT ACCD RIGHT ONE BIT
CARRY?
Y
ADD MULTIPLICAND TO THE PARTIAL PRODUCT (CALL MADD) IN ACCB
N
ROTATE ACCB RIGHT ONE BIT
DECREMENT TEMP REG
N
TEMP = 0?
Y
ADD THE EXPONENTS EXP A + EXP B → EXP B
IS SIGN OF ANSWER + OR – ?
+VE
–VE
OBTAIN 2'S COMPLEMENT OF ACCB
NORMALIZE ACCB TO MAINTAIN PRECISION
RETURN


DIV
DIVISION ROUTINE
DETERMINE SIGN OF FINAL ANSWER OBTAIN 2'S COMPLE-MENT OF −VE #'S
MOVE ACCB TO ACCD CLEAR ACCB; ACCC INITIATE TEMP TO .16
ROTATE ACCD LEFT ONE BIT INTO ACCC CHECK IF ACCA IS > = OR < ACCC
IS ACCA ⩽ ACCC
Y
N
SUBTRACT ACCA FROM ACCC
CLEAR CARRY
SET CARRY
ROTATE ACCB 1 BIT LEFT TO SHIFT IN CARRY
DECREMENT TEMP REG
N
TEMP = 0?
Y
SUBTRACT EXPONENTS
ADJUST ACCB IF FINAL ANSWER WAS TO BE −VE
RETURN

```
LINE    ADDR   B1    B2      MATHF                                          PAGE    1

  1                            TITLE  'MATHF'
  2                                                ; DOUBLE PRECISION FLOATING POINT MATH PACKAGE
  3                                                ;
  4                                                ; DEFINE THE FOLLOWING SYMBOLS:
  5                                                ;
  6                                                ; ACCA  BEGINNING OF 3 REGISTER ACCUMULATOR
  7                                                ; ACCB     "     "      "        "
  8                                                ; ACCC  2 REGISTER FILE FOR MPY/DIV
  9                                                ; ACCD   "         "
 10                                                ; MATORG       ORIGIN FOR LOAD OF PACKAGE
 11                                                ; TEMP  TEMPORARY SCRATCH REGISTER
 12                                                ; SIGN  TEMPORARY SCRATCH REGISTER
 13                                                ;
 14                                                ; USAGE:
 15                                                ;      LOAD ACCA AND ACCB WITH THEIR RESPECTIVE
 16                                                ; CONTENTS, CALL THE SUBROUTINE, AND OBTAIN RESULT
                                                    -
 17                                                : IN ACCB.  ACCA IS HIGH 8 BITS, ACCA+1 IS LOW 8 B
                                                    -ITS.
 18                                                ;
 19                                                ; NOTE:  MANY SUBROUTINES ARE NESTED, SO DO NOT CA
                                                    -LL
 20                                                ; ANY OF THE ROUTINES OTHER THAN FROM THE MAINLINE
                                                    -.
 21                                                ;
 22  000000
 23  000000            MATORG  EQU    0
 24  000017            TEMP    EQU    17
 25  000011            ACCA    EQU    11
 26  000013            EXPA    EQU    13
 27  000014            ACCB    EQU    14
 28  000016            EXPB    EQU    16
 29  000020            ACCC    EQU    20
 30  000022            ACCD    EQU    22
 31  000024            SIGN    EQU    24
 32  000000
 33  000000
 34                            ORG    MATORG
 35  000000
 36                                                ; *** SUB ***
 37                                                ; ACCB - ACCA --> ACCB
 38  000000
 39  000000  4565      FSUB    CALL   NEGA
 40  000001
 41                                                ;------> IMPORTANT <------
 42                                                ; FADD MUST FOLLOW...
 43                                                ;------> IMPORTANT <------
 44  000001
 45                                                ;*** ADD ***
 46                                                ; ACCA+ACCB --> ACCB
 47  000001
```

```
 48 000001 1013          FADD    MOVF   EXPA,W        ; SCALE MANTISSAS
 49 000002 0216                  SUBWF  EXPB,W        ; FIND GREATER EXPONENT
 50 000003 3103                  SKPNZ
 51 000004 5016                  GOTO   PADD          ; EXPONENTS EQUAL...ADD
 52 000005 3003                  SKPNC
 53 000006 4606                  CALL   FSWAP         ; B > A , SWAP 'EM
 54 000007 1013                  MOVF   EXPA,W        ; COUNT FOR SHIFT RIGHT
 55 000010 0256                  SUBWF  EXPB
 56 000011 4437          SCLOOP  CALL   MASR1
 57 000012 1756                  INCFSZ EXPB
 58 000013 5011                  GOTO   SCLOOP
 59 000014 1013                  MOVF   EXPA,W
 60 000015 0056                  MOVWF  EXPB
 61 000016 1011          PADD    MOVF   ACCA,W        ; FIND SIGN OF RESULT
 62 000017 0414                  IORWF  ACCB,W        ; FOR OVERFLOW CHECK
 63 000020 0064                  MOVWF  SIGN
 64 000021 4430                  CALL   MADD
 65 000022 3764                  BTFSS  SIGN,7        ; CHECK FOR OVERFLOW
 66 000023 3754                  BTFSS  ACCB,7
 67 000024 4000                  RET
 68 000025 2003                  CLRC
 69 000026 1256                  INCF   EXPB          ; WE OVERFLOWED...
 70 000027 5042                  GOTO   ASRHCK        ; SCALE TO RIGHT
 71 000030
 72 000030 1012          MADD    MOVF   ACCA+1,W
 73 000031 0755                  ADDWF  ACCB+1
 74 000032 3003                  BTFSC  3,0           ; ADD IN CARRY
 75 000033 1254                  INCF   ACCB
 76 000034 1011                  MOVF   ACCA,W
 77 000035 0754                  ADDWF  ACCB
 78 000036 4000                  RET
 79 000037
 80                                                   ;*** SHIFT RIGHT, ARITHMETIC ***
 81                                                   ; SHIFT ACCB RIGHT ONE PLACE
 82                                                   ; SIGN OF OPERAND IS PRESERVED (OPTIONAL)
 83 000037
 84 000037 2003          MASR1   CLRC
 85 000040 3354                  BTFSC  ACCB,7        ; ***OPTIONAL FOR SIGN
 86 000041 2403                  SETC                 ; ***SET CARRY IF < 0
 87 000042 1454          ASRHCK  RRF    ACCB
 88 000043 1455                  RRF    ACCB+1
 89 000044 4000                  RET
 90 000045
 91                                                   ;*** SHIFT LEFT, ARITHMETIC ***
 92                                                   ; SHIFT ACCB LEFT ONE PLACE
 93                                                   ; SIGN OF OPERAND IS PRESERVED (OPTIONAL)
 94 000045
 95 000045 2003          MASL1   CLRC
 96 000046 1555                  RLF    ACCB+1
 97 000047 1554                  RLF    ACCB
```

```
 98 000050 2354          BCF   ACCB,7      ; ***OPTIONAL FOR SIGN
 99 000051 3003          SKPNC              ; ***
100 000052 2754          BSF   ACCB,7      ; ***CARRY... SET SIGN
101 000053 4000          RET
102 000054
103                                        ;*** MPY ***
104                                        ; ACCA*ACCB --> ACCB
105 000054
106 000054 4551     FMPY CALL  PSIGN
107 000055 4501          CALL  SETUP
108 000056 1462     MPLOOP RRF  ACCD       ; ROTATE D RIGHT
109 000057 1463          RRF   ACCD+1
110 000060 3003          SKPNC             ; NEED TO 'ADD' ??
111 000061 4430          CALL  MADD        ; ADD A TO B
112 000062 1454          RRF   ACCB        ; ROTATE B RIGHT
113 000063 1455          RRF   ACCB+1
114 000064 1357          DECFSZ TEMP       ; LOOP TILL  DONE
115 000065 5056          GOTO  MPLOOP
116 000066 1013          MOVF  EXPA,W      ; ADD EXPONENTS
117 000067 0756          ADDWF EXPB
118 000070 1256          INCF  EXPB
119 000071 3764     FINUP BTFSS SIGN,7
120 000072 5173          GOTO  NORM
121 000073 0355     NEGB DECF  ACCB+1      ;*** NEGB ***
122 000074 1155          COMF  ACCB+1      ; A NICE WAY TO WORK THE
123 000075 3103          SKPNZ             ; ROUTINE IN......
124 000076 0354          DECF  ACCB
125 000077 1154          COMF  ACCB
126 000100 5173          GOTO  NORM
127 000101
128 000101
129 000101 6020     SETUP MOVLW .16        ; 16 PLACE SHIFT
130 000102 0057          MOVWF TEMP
131 000103 1014          MOVF  ACCB,W      ; MOVE B TO D
132 000104 0062          MOVWF ACCD
133 000105 1015          MOVF  ACCB+1,W
134 000106 0063          MOVWF ACCD+1
135 000107 0154          CLRF  ACCB        ; CLEAR B
136 000110 0155          CLRF  ACCB+1
137 000111 4000          RET
138 000112
139                                        ;*** DIV ***
140                                        ; ACCB/ACCA --> ACCB, REMAINDER IN ACCC
141 000112
142 000112 4551     DIV  CALL  PSIGN
143 000113 4501          CALL  SETUP
144 000114 0160          CLRF  ACCC
145 000115 0161          CLRF  ACCC+1
146 000116 1563     DLOOP RLF  ACCD+1      ; ROTATE (C,D) LEFT
147 000117 1562          RLF   ACCD
148 000120 1561          RLF   ACCC+1
```

```
 149 000121 1560              RLF   ACCC
 150 000122 1011              MOVF  ACCA,W        ; CHECK IF A > C
 151 000123 0220              SUBWF ACCC,W
 152 000124 3503              SKPZ
 153 000125 5130              GOTO  NOCHK
 154 000126 1012              MOVF  ACCA+1,W      ; HIGH'S EQUAL...CHECK LOWS
 155 000127 0221              SUBWF ACCC+1,W
 156 000130 3403        NOCHK SKPC
 157 000131 5141              GOTO  NOGO          ; A>C, SHIFT CLEAR CARRY
 158 000132 1012              MOVF  ACCA+1,W      ; C-A --> C
 159 000133 0261              SUBWF ACCC+1
 160 000134 3403              BTFSS 3,0
 161 000135 0360              DECF  ACCC
 162 000136 1011              MOVF  ACCA,W
 163 000137 0260              SUBWF ACCC
 164 000140 2403              SETC                ; SHIFT IN A ONE
 165 000141 1555        NOGO  RLF   ACCB+1        ; SHIFT B LEFT
 166 000142 1554              RLF   ACCB
 167 000143 1357              DECFSZ TEMP         ; LOOP TILL DONE
 168 000144 5116              GOTO  DLOOP
 169 000145 6361              MOVLW -.15          ; SUBTRACT EXPONENTS
 170 000146 0713              ADDWF EXPA,W
 171 000147 0256              SUBWF EXPB
 172 000150 5071              GOTO  FINUP
 173 000151
 174 000151 1011        PSIGN MOVF  ACCA,W        ; PREPARE SIGN
 175 000152 0614              XORWF ACCB,W
 176 000153 0064              MOVWF SIGN
 177 000154 3754              BTFSS ACCB,7
 178 000155 5163              GOTO  TRYA
 179 000156 1155              COMF  ACCB+1        ; NEGB.... CANT CALL SUBR
 180 000157 1255              INCF  ACCB+1
 181 000160 3103              SKPNZ
 182 000161 0354              DECF  ACCB
 183 000162 1154              COMF  ACCB
 184 000163 3751        TRYA  BTFSS ACCA,7
 185 000164 4000              RET
 186 000165
 187                                              ;------> IMPORTANT <------
 188                                              ; NEGA MUST FOLLOW...
 189                                              ;------> IMPORTANT <------
 190 000165
 191                                              ;*** NEGA ***
 192                                              ; (-ACCA) --> ACCA
 193 000165
 194 000165 1152        NEGA  COMF  ACCA+1
 195 000166 1252              INCF  ACCA+1
 196 000167 3103              SKPNZ
 197 000170 0351              DECF  ACCA
 198 000171 1151              COMF  ACCA
 199 000172 4000              RET
```

```
200  000173
201                                          ;*** NORMALIZE ***
202                                          ; NORMALIZES ACCB FOR USE IN FLOATING POINT CALCUL
                                              -ATIONS
203                                          ; ----> IT IS RECOMENDED THAT ONE CALLS THIS ROUTI
                                              -NE
204                                          ;       FREQUENTLY SO AS NOT TO ALLOW LOSS OF PREC
                                              -ISION
205  000173
206  000173  1054          NORM    TSTF    ACCB
207  000174  3503                  SKPZ
208  000175  5201                  GOTO    CNORM
209  000176  1055                  TSTF    ACCB+1
210  000177  3103                  SKPNZ
211  000200  4000                  RET
212  000201  3314          CNORM   BTFSC   ACCB,6
213  000202  4000                  RET
214  000203  4445                  CALL    MASL1
215  000204  0356                  DECF    EXPB
216  000205  5201                  GOTO    CNORM
217  000206
218                                          ;*** FSWAP ***
219                                          ; (ACCA,EXPA) <---> (ACCB,EXPB)
220  000206
221  000206  1011          FSWAP   MOVF    ACCA,W
222  000207  0057                  MOVWF   TEMP
223  000210  1014                  MOVF    ACCB,W
224  000211  0051                  MOVWF   ACCA
225  000212  1017                  MOVF    TEMP,W
226  000213  0054                  MOVWF   ACCB
227  000214  1012                  MOVF    ACCA+1,W
228  000215  0057                  MOVWF   TEMP
229  000216  1015                  MOVF    ACCB+1,W
230  000217  0052                  MOVWF   ACCA+1
231  000220  1017                  MOVF    TEMP,W
232  000221  0055                  MOVWF   ACCB+1
233  000222  1013                  MOVF    EXPA,W
234  000223  0057                  MOVWF   TEMP
235  000224  1016                  MOVF    EXPB,W
236  000225  0053                  MOVWF   EXPA
237  000226  1017                  MOVF    TEMP,W
238  000227  0056                  MOVWF   EXPB
239  000230  4000                  RET
240  000231
```

5.11 Square Root Algorithm Using Newton's Method

Abstract: Newton's method is used in this program to find the square root of a number represented by two 8-bit registers as its mantissa and one 8-bit register as its exponent.

Description: The algorithm uses subroutines of the double precision floating point math package and is intended to be used only as part of the main program for processors with 2-level stacks (PIC1650 and PIC1655).

NEWTON'S METHOD

If N = Number and x = Square Root of N;
Then

$$x^2 - N = 0 = f(x)$$

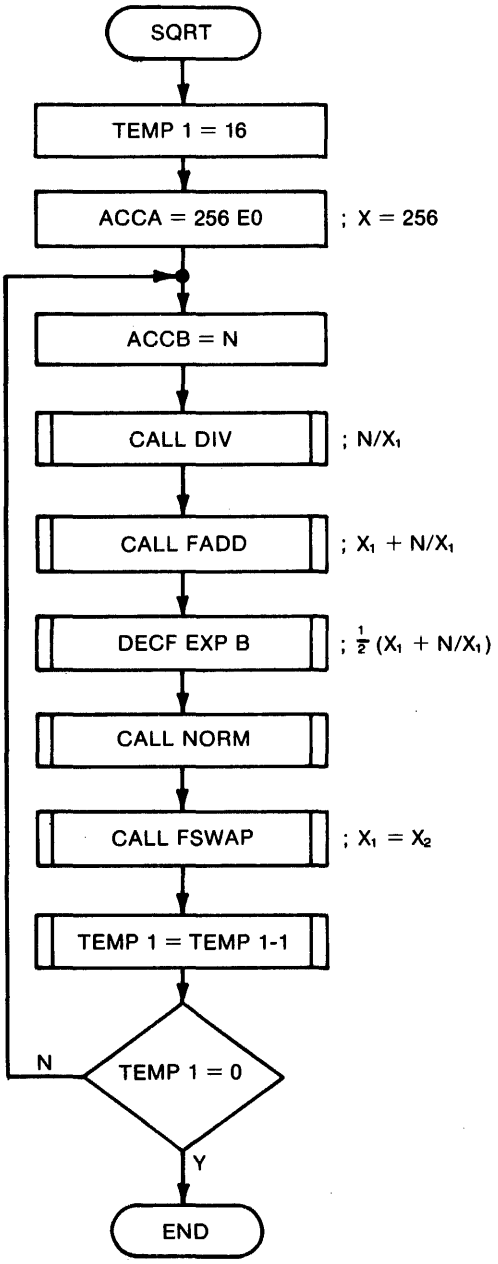
$$\tan\theta = f^1(x_1) = \frac{f(x_1)}{x_1 - x_2}$$

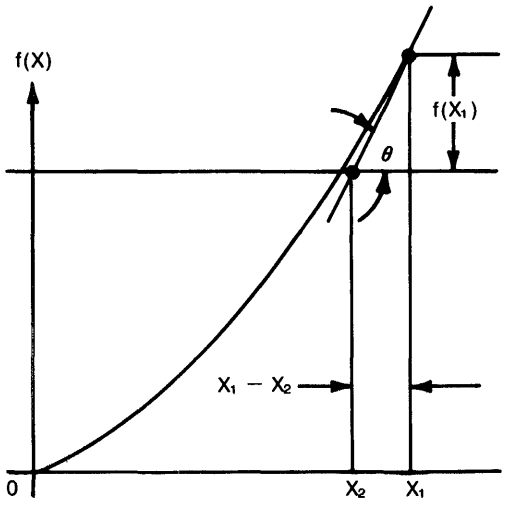
$$x_2 = x_1 - \frac{f(x_1)}{f^1(x_1)}$$

$$= x_1 - \frac{x_1^2 - N}{2x_1}$$

$$= \frac{2x_1^2 - x_1^2 + N}{2x_1} = \frac{x_1^2 + N}{2x_1}$$

$$= \tfrac{1}{2}(x_1 + \frac{N}{x_1}),$$ where x_1 = old value
x_2 = new value


SQRT
TEMP 1 = 16
ACCA = 256 E0
; X = 256
ACCB = N
CALL DIV
; N/X1
CALL FADD
; X1 + N/X1
DECF EXP B
; ½ (X1 + N/X1)
CALL NORM
CALL FSWAP
; X1 = X2
TEMP 1 = TEMP 1-1
N
TEMP 1 = 0
Y
END


f(X)
f(X1)
θ
X1 − X2
0
X2
X1

Subroutines:

```
FSUB  :  ACCB − ACCA→ACCB
FADD  :  ACCA + ACCB→ACCB
FMPY  :  ACCA*ACCB→ACCB
DIV   :  ACCB/ACCA→ACCB
```

Registers:

```
MATORG  EQU  0
TEMP    EQU 17
ACCA    EQU 11
EXPA    EQU 13
ACCB    EQU 14
EXPB    EQU 16
ACCC    EQU 20
ACCD    EQU 22
SIGN    EQU 24
```

PROGRAM

Registers: TEMP1=25 N=26 EXPN = 30

```
TEMP1   EQU   25

SQRT    MOVLW .16      ; TEMP1 = 16
        MOVWF TEMP1
        CLRF  ACCA     ; x1 = 256EO=ACCA
        CLRF  ACCA+1
        CLRF  EXPA
        INCF  ACCA
        BSF   EXPA,7
NEWTON  MOVF  N,W      ; N = ACCB
        MOVWF ACCB
        MOVF  N + 1,W
        MOVWF ACCB + 1
        MOVF  EXPN,W
        MOVWF EXPB

        CALL  DIV      ; N/x1

        CALL  FADD     ; x1 + N/x1

        DECF  EXPB     ; ½(x1 + N/x1)

        CALL  NORM     ; NORMALIZE RESULT

        CALL  FSWAP    ; ACCA↔ACCB

        DECFSZ TEMP1   ; DO16 ITERATIONS

        GO TO NEWTON

        END            ; ACCA = √N
```

6 MISCELLANEOUS ROUTINES

6.1 Keyboard Scan Program, Reads And Debounces 16 Keys And Stores Key Closures In Two Files

The display is blanked at the start of the keyboard SCAN program to prevent corruption of the display when reading the keys. After completion, the display SCAN program should be run in order to restore the display.

The SCAN file is initialized to all ones (377) and the carry bit cleared. The GETKEY subroutine rotates the SCAN file left once, which moves the carry into bit 0. The key column is enabled by the transfer of SCAN to SCNOUT and the four keys are read by File 5. A key closure will be read as a low and the complement will be stored in a temporary (TEMP) file.

The lower 4 bits (nibble) of TEMP is swapped with the upper 4 bits and the GETKEY subroutine is called. Eight keys are now positioned in TEMP and compared with the key information in Debounce Reg 1 (DEBNS1).

If the results of the XOR instruction is zero, the same key closures exist and the key data is stored in KEYREG1. If the result is not zero, key closures have not stabilized and the key data is stored in DEBNS1.

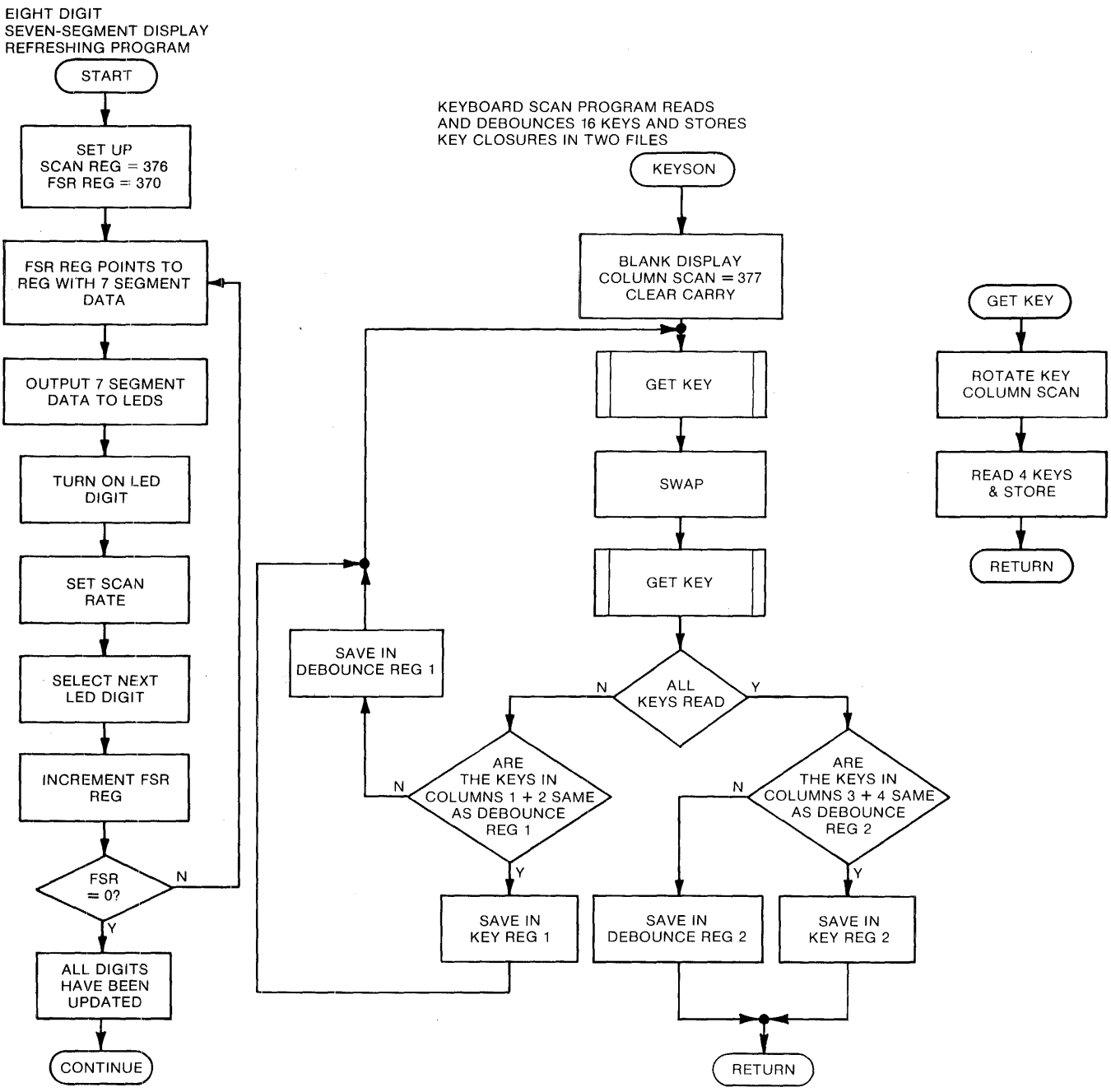
The program is then repeated for the last two columns with the results stored in DEBNS2 or KEYREG2.

6.2 Eight Digit Seven-Segment Display Refreshing Program

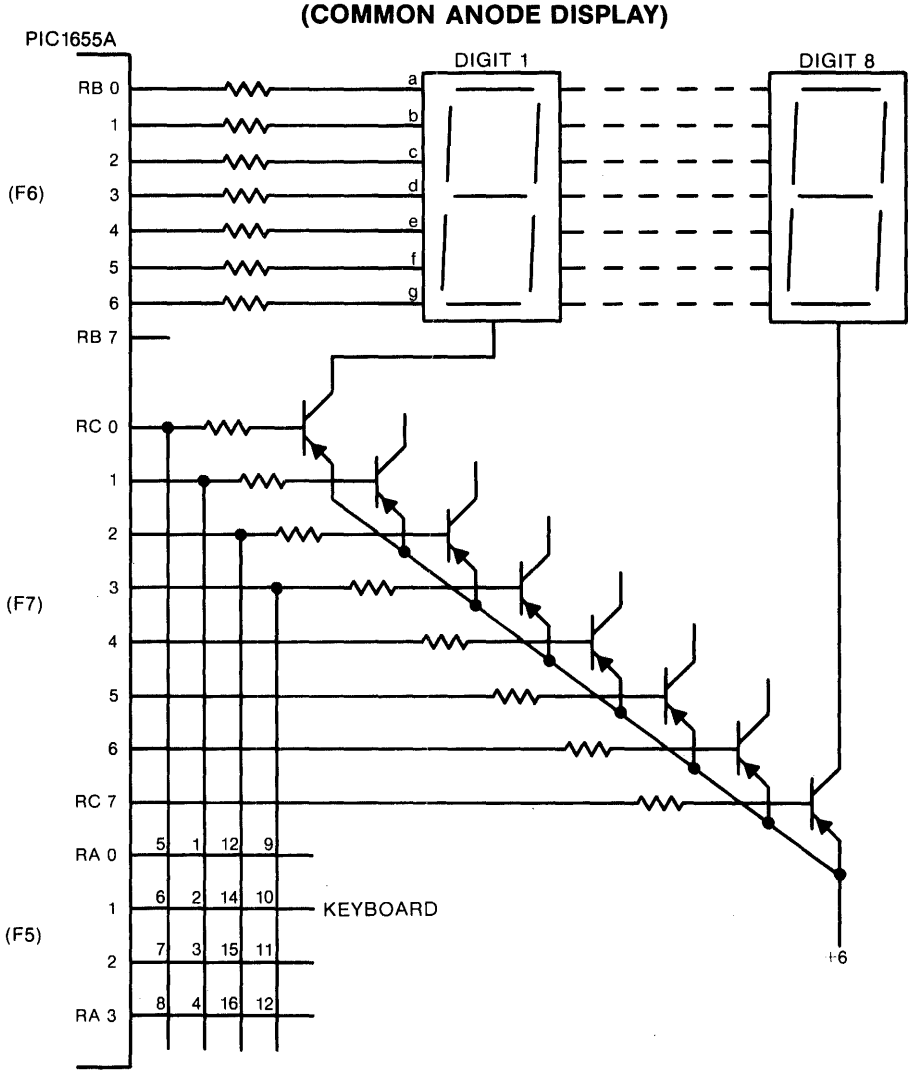
At the start of the program File 10 (SCAN) is initialized to 376 (bit 0 low) and the FSR REGISTER is initialized to 30. Data (in 7-segment code format) has been stored in Files 30 through 37 by an external conversion program.

The FSR REGISTER is addressed indirectly by the MOVF O,W instruction and the contents of File 30 is transferred to F6 (DATOUT). Next the SCAN File contents are transferred to the SCNOUT File which in turn enables the first digit. DIGIT1 information will now be displayed.

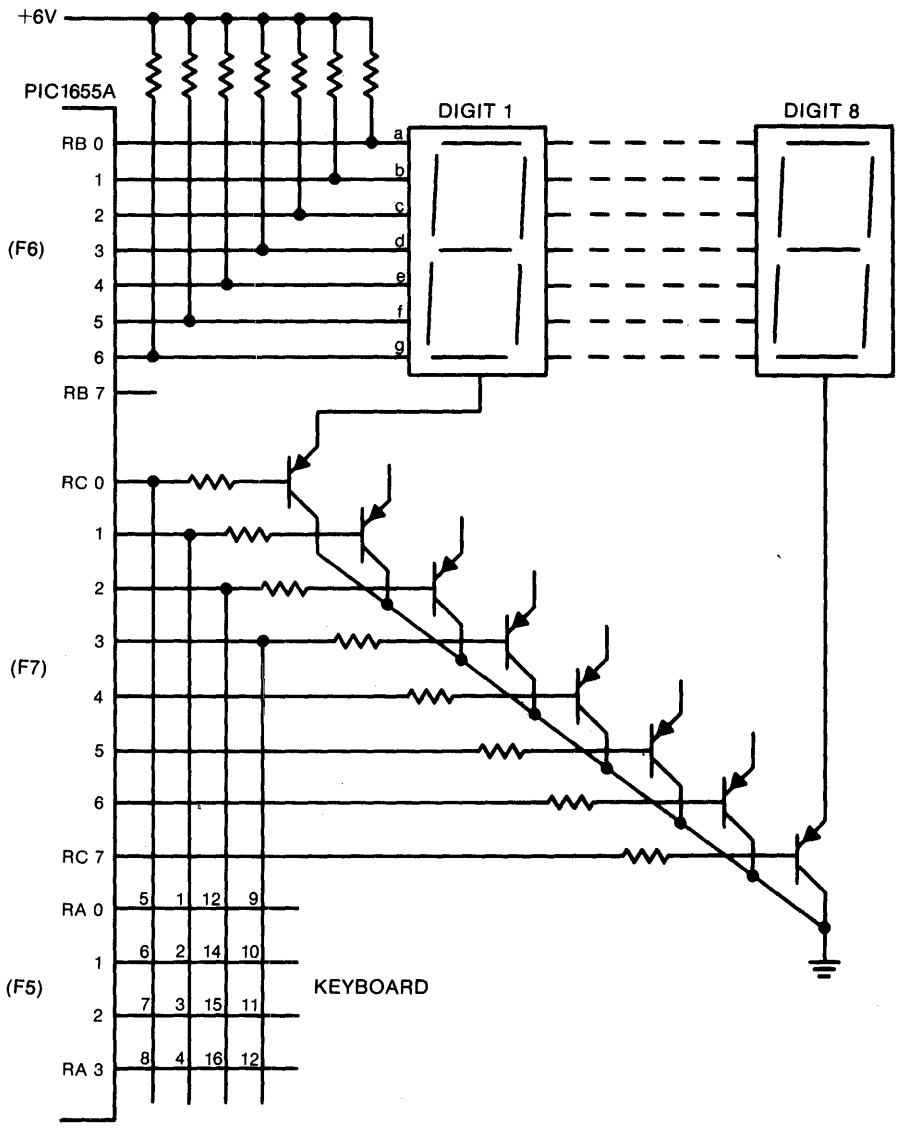
Before the next loop through the program, the SCAN File is rotated left once and the FSR REGISTER is incremented. Now the program will display the information for DIGIT2. This continues until the FSR REGISTER contains a zero at which time all eight digits have been scanned. A delay loop is added to the program to control the refresh rate of the display, but in most cases the total program delay can be set to eliminate this loop. To prevent the display from flickering, set the refresh rate at 250-500Hz.


EIGHT DIGIT
SEVEN-SEGMENT DISPLAY
REFRESHING PROGRAM
START
SET UP
SCAN REG = 376
FSR REG = 370
FSR REG POINTS TO
REG WITH 7 SEGMENT
DATA
OUTPUT 7 SEGMENT
DATA TO LEDS
TURN ON LED
DIGIT
SET SCAN
RATE
SELECT NEXT
LED DIGIT
INCREMENT FSR
REG
FSR
= 0?
N
Y
ALL DIGITS
HAVE BEEN
UPDATED
CONTINUE
KEYBOARD SCAN PROGRAM READS
AND DEBOUNCES 16 KEYS AND STORES
KEY CLOSURES IN TWO FILES
KEYSON
BLANK DISPLAY
COLUMN SCAN = 377
CLEAR CARRY
GET KEY
SWAP
GET KEY
ALL
KEYS READ
N
Y
ARE
THE KEYS IN
COLUMNS 1 + 2 SAME
AS DEBOUNCE
REG 1
N
Y
SAVE IN
DEBOUNCE REG 1
SAVE IN
KEY REG 1
ARE
THE KEYS IN
COLUMNS 3 + 4 SAME
AS DEBOUNCE
REG 2
N
Y
SAVE IN
DEBOUNCE REG 2
SAVE IN
KEY REG 2
RETURN
GET KEY
ROTATE KEY
COLUMN SCAN
READ 4 KEYS
& STORE
RETURN

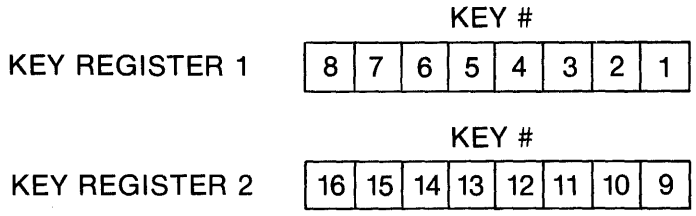
HARDWARE CONFIGURATION
(COMMON ANODE DISPLAY)



HARDWARE CONFIGURATION
(COMMON CATHODE DISPLAY)



```
KEY     = F5
TEMP    = F11
DEBNS1  = F12
KEYREG1 = F13
DEBNS2  = F14
KEYREG2 = F15
```




```
GETKEY  RLF     SCAN      ; This subroutine rotates file left.
                          ; First rotate will move the carry
        MOVF    SCAN,W    ; bit into Bit 0 of SCAN.
        MOVWF   SCNOUT    ; SCAN transferred to SCNOUT.
        COMF    KEY,W     ; Read complement of key into W
        ANDLW   17        ; Zero upper 4 bits of W
        IORWF   TEMP      ; Store in Temp. File.
        RETLW   377       ; RETURN



KEYSCN  MOVLW   377       ; START
        *       DATOUT    ; Blank Display
        MOVWF   SCAN      ; Sets up SCAN and CARRY BIT
        BCF     3,0       : for rotating a zero through the file
CTSCN   CLRF    TEMP      ; TEMP = 0
        CALL    GETKEY
        SWAPF   TEMP      ; Swap Key Data from lower nibble to
                          ; upper nibble.
        CALL    GETKEY
        MOVF    TEMP,W    ; Temp contains key info for 2 columns
        BTFSC   SCAN,1    ; Test if scan has read columns 1 and 2
        GOTO    LSTKEY    ; Last Key read. End SCAN.
        XORWF   DEBNS1    ; Compare new key data with previous
                          ; key data.
        BTFSC   3,2       ; Skip on no zero
        MOVWF   KYREG1    ; If same, save in Key Reg1
        MOVWF   DEBNS1    ; If different update debounce Reg1.
        GOTO    CTSCN     ; Scan last two columns.
LSTKEY  XORWF   DEBNS2
        BTFSC   3,2       ; Same as above debounce
        MOVWF   KEYREG2
        MOVWF   DEBNS2
        RETLW   377       ; RETURN TO MAIN PROGRAM
```

*For common anode display use "MOVWF", for common cathode use "CLRF"

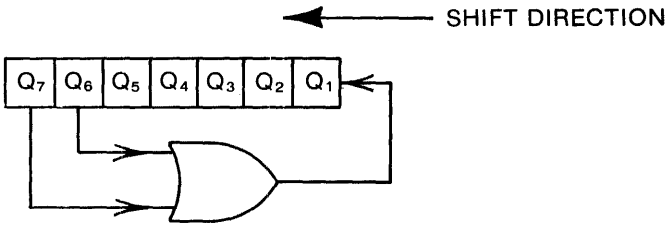
```
DATOUT = F6          OUTPUT REGISTER FOR SEGMENTS
SCNOUT = F7          OUTPUT REGISTER FOR DIGITS
SCAN   = F10         SELECTS ONE OF EIGHT DIGITS
DISDLY = F20         SETS SPEED OF SCAN
DIGIT1 = F30
DIGIT2 = F31
DIGIT3 = F32         STORES 7-SEGMENT DATA FOR
DIGIT4 = F33         EACH DIGIT. UPDATED BY AN
DIGIT5 = F34         EXTERNAL CONVERSION ROUTINE.
DIGIT6 = F35
DIGIT7 = F36
DIGIT8 = F37

STSCN   MOVLW   376     ; CONFIGURES SCAN REGISTER WITH LSB
        MOVWF   SCAN    ; SET TO "0"
        MOVLW   30      ; FSR REGISTER POINTS TO FIRST DIGIT,
        MOVWF   4       ; BUT WILL BE READ AS 370.
CTSCN   *       0,W     ; MOVES CONTENTS OF THE REGISTER POINTED
        MOVWF   DATOUT  ; TO BY FSR TO THE SEGMENTS.
        MOVF    SCAN,W  ; SELECTS DIGIT THAT WILL
        MOVWF   SCNOUT  ; DISPLAY ABOVE DATA
DLYLP   DECFSZ  DISDLY  ; PROGRAM DELAY LOOP
        GOTO    DLYLP
        MOVLW   100     ; DETERMINES SPEED OF SCAN
        MOVWF   DISDLY
        BSF3,0          ; CARRY BIT SET TO PREVENT A
                        ; ZERO ROTATED INTO THE LSB OF SCAN.
        RLF     SCAN    ; ZERO ROTATED LEFT TO NEXT BIT
        INCFSZ  4,F     ; FSR POINTS TO NEXT DIGIT
        GOTO    CTSCN   ; CONTINUE SCAN UNTIL ALL
                        ; DIGITS HAVE BEEN REFRESHED
          •
          •            ; OTHER PROGRAMS
          •
        GOTO    STSCN   ; START SCAN REFRESH
```

*For common anode display use "COMF", for common cathode display use "MOVF".

6.3 Pseudo Random Number Generator

This polynomial generator is typically used to generate white noise for sounds such as "bang", "screech", "breathing", as well as for "random" sequence generation. The seed number in the generator, if necessary, can be randomized by external events such as contact closures. This permits, for example, games to start randomly and continue pseudo randomly according to the output of the polynomial generator.

The algorithm used to generate a pseudo random number sequence uses a shift register and a feedback loop in the following fashion:



7 BIT SHIFT REGISTER

The feedback connections vary for different length shift registers. The chart below gives the connections for shift registers from 4 to 16 bits.

N	$S = 2^N - 1$
4	$Q_3 \oplus Q_4$
5	$Q_3 \oplus Q_5$
6	$Q_5 \oplus Q_6$
7	$Q_6 \oplus Q_7$
8	$Q_2 \oplus Q_3 \oplus Q_4 \oplus Q_8$
9	$Q_5 \oplus Q_9$
10	$Q_7 \oplus Q_{10}$
11	$Q_9 \oplus Q_{11}$
12	$Q_2 \oplus Q_{10} \oplus Q_{11} \oplus Q_{12}$
13	$Q_1 \oplus Q_{11} \oplus Q_{12} \oplus Q_{13}$
14	$Q_2 \oplus Q_{12} \oplus Q_{13} \oplus Q_{14}$
15	$Q_{14} \oplus Q_{15}$
16	$Q_4 \oplus Q_{13} \oplus Q_{15} \oplus Q_{16}$

The two routines given here are 7 and 16 bits which generate pseudo random numbers of non-repeating length of 127 and 65535. In either case, there is one singularity "all zeroes" that must be avoided during initialization.

6.3.1 7 BIT PSEUDO RANDOM NUMBER GENERATOR

The 7 bit routine aligns bits Q_6 and Q_7 in registers SEED and W. Then the registers are exclusive-ored and the unwanted bits are masked out leaving register W in the following state:

0	$Q_7 \oplus Q_6$	0	0	0	0	0	0

If register W equals zero, then the carry bit is cleared, otherwise it is set (carry bit gets the value of $Q_6 + Q_7$). Then the carry is shifted into bit 0 of the register SEED.

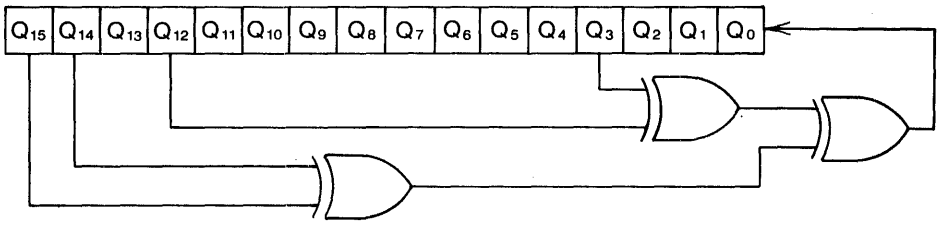
SEED holds random number
—upon initialization set SEED to 1, avoid lockup

```
RAND7   RLF     SEED,W
        XORWF   SEED,W    ; exclusive or bits Q6 & Q7
        ANDLW   100       ; mask out other bits
        SETC              ; set carry
        SKPNZ             ; if Q6 ⊕ Q7 equal 0, clear carry
        CLRC              ; else clear carry
        RLF     SEED      ; shift left
        RETLW   0
```

Routine takes 36 μsec including CALL

6.3.2 16 BIT PSEUDO RANDOM NUMBER GENERATOR

The 16 bit routine aligns the proper bits (Q_{15}, Q_{14}, Q_{12}, Q_3) and performs an exclusive or. Bit 7 of register WORK holds the result of the exclusive or's of the proper bits.



RANDH is the MSB's of the random number
RANDL is the LSB's of the random number
WORK is the temporary register

```
RAND16  MOVFW   RANDH
        MOVWF   WORK
        RLF     WORK
        XORWF   WORK,W    ; exclusive or Q15 & Q14
        RLF     WORK
        RLF     WORK
        XORWF   WORK      ; exclusive or with Q12
        SWAPF   RANDL,W
        XORWF   WORK      ; exclusive or with Q3
        RLF     RANDL
        RLF     RANDH     ; shift left
        BSF     RANDL,O
        BTFSS   WORK,7    ; if the result of the exclusive or's
        BCF     RANDL,O   ; is O, clear RANDL bit O
        RETLW   O         ; else set RANDL bit O
```

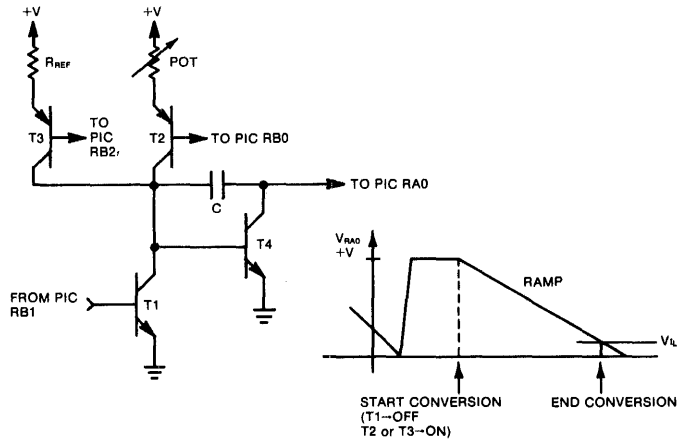
Routine takes 68 μsec including CALL

6.4 Potentiometer A/D Conversion Routine

This routine shows how a potentiometer setting can be sampled by a very simple A/D conversion which utilizes the RC time constant concept. In the normal state transistor T1 is ON and transistor T2 is OFF. In order to start the conversion, transistor T1 is turned OFF and transistor T2 is turned ON. Simultaneously, the program then loops in a count routine waiting for the input (RA0) to go low. The count obtained reflects the setting of the pot—the greater the count, the greater is the resistance. There is a maximum value of 255 for the count since only one register is incremented in the count loop.

For a more precise measurement, the ratio of the count for the potentiometer to the count for a known resistor R should be used. In this case, the subroutine should be called a second time with transistor T3 turned ON to obtain a reading for R.

```
ADCONV    MOVLW    374      ; Turns T1 OFF and T2 ON
          MOVWF      6      ; Start Conversion

LOOP      BTFSS    5,0      ; Count Loop
          GOTO     EXIT
          INCFSZ   TEMP     ; Count is in Temp Register
          GOTO     LOOP
          MOVLW    377      ; Overflow
          MOVWF    TEMP

EXIT      MOVLW    377      ; Turns T1 ON and T2 OFF
          MOVWF      6
          RET
```



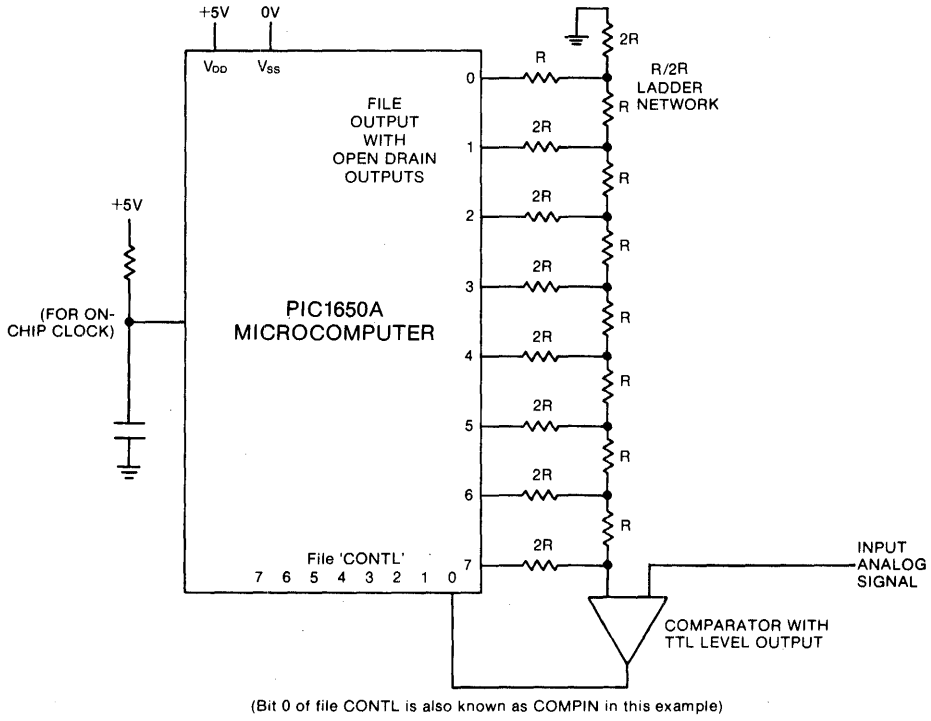
6.5 Analog To Digital Conversion

In this example an analog signal (whose value is to be digitized) is compared with the analog output of a ladder network. The output from the comparator goes to the PIC microcomputer, and the input to the ladder network comes from the chip. (Refer to diagram on page 61.)

The subroutine shown in this example can be called from anywhere in the PIC program by the statement:

CALL ATOD

and about 300μs later the file OUTPUT will contain the digital value of the analog signal, which can then be used as necessary further in the PIC program.



(Bit 0 of file CONTL is also known as COMPIN in this example)

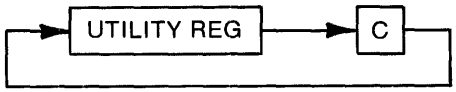
6.5.1 HOW THE PROGRAM WORKS

The flow diagram for the conversion shown should be followed through in conjunction with the program to properly understand how the conversion works.

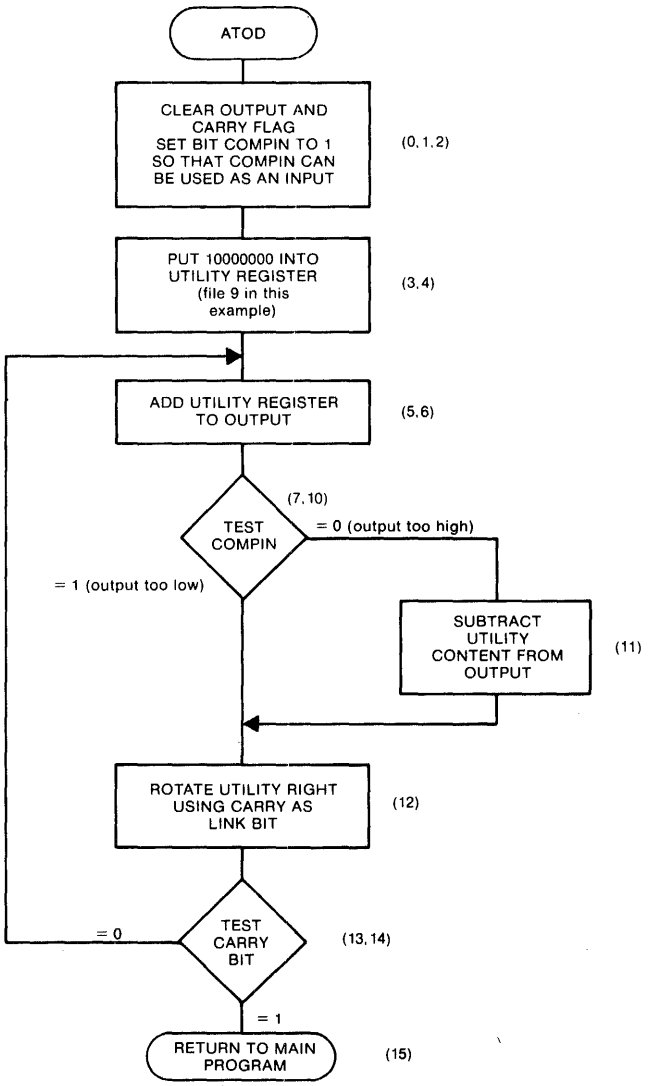
NOTES:

1) At 0, COMPIN is set because we wish to use COMPIN as an input. Since the 32 I/O lines of the PIC are both inputs and outputs it is possible to obtain a wire-AND at each pin of the output with the current input.
2) Note how the carry is cleared by BCF STATUS, CARRY. This is a bit clear instruction of bit 0 in file 3, which in fact is the carry flag.
3) Note how the PIC assembler accepts literals in decimal (.9), binary (B'10000000') or octal (by default). Hexadecimal (x 'F9') and character ('Q') are also supported.
4) The subtraction at 11 is done by exclusive OR (XORWF OUTPUT) instead of subtraction (SUBWF OUTPUT) since the latter would set the carry bit and necessitate an extra clear carry instruction.

5) At 12 the utility register is rotated to the right through the carry bit:



6) The utility register always keeps just one bit set, all others clear. The bit corresponds to the bit on OUTPUT that is currently being worked on. When the bit drops out of the right of the utility register into the carry bit the conversion is complete and the subroutine terminates, restoring control to the main program (step 15).



(THE NUMBERS IN BRACKETS SHOW THE ADDRESSES OF THE INSTRUCTIONS WHICH PERFORM THIS STEP.)

```
INE    ADDR    B1     B2       ATOD

  1                                    TITLE  'ATOD'
  2
  3  ;8 BIT A TO D SUCCESSIVE APPROXIMATION
  4  ;ROUTINE FOR PIC MICROCOMPUTER
  5  ;
  6  ;OUTPUT TO LADDER NETWORK IS FILE 'OUTPUT'
  7  ;INPUT FROM COMPARATOR IS
  8  ;BIT 0(CALLED 'COMPIN')OF FILE CONTL.
  9  ;COMPARATOR GIVES LOGIC 1 IF CURRENT
 10  ;VALUE OF 'OUTPUT' IS TOO LOW
 11  ;UTILRG IS BIT COUNT CONTROL REGISTER.
 12  ;EXECUTION TIME IS 304 US
 13  000003                  STATUS  =      3
 14  000000                  CARRY   =      0
 15  000011                  UTILRG  =      .9
 16  000005                  OUTPUT  =      5
 17  000006                  CONTL   =      6
 18  000000                  COMPIN  =      0
 19
 20  000000  2406            ATOD    BSF    CONTL,COMPIN
 21  000001  0145                    CLRF   OUTPUT
 22  000002  2003                    BCF    STATUS,CARRY
 23  000003  6200                    MOVLW  B'10000000'
 24  000004  0051                    MOVWF  UTILRG
 25  000005  1011            CYCLE   MOVF   UTILRG,W
 26  000006  0745                    ADDWF  OUTPUT
 27  000007  3006                    BTFSC  CONTL,COMPIN
 28  000010  5012                    GOTO   ENDTST
 29  000011  0645                    XORWF  OUTPUT
 30  000012  1451            ENDTST  RRF    UTILRG
 31  000013  3403                    BTFSS  STATUS,CARRY
 32  000014  5005                    GOTO   CYCLE
 33  000015  4000                    RET
 34  000016                          END
```

6.5.2 CONCLUSION

This example brings out several important and unique features of the PIC1650A microcomputer.

1) **Hardware stack:** When the CALL is made to a subroutine, the return address is stored in a hardware stack.
2) **Bit set, clear, and test:** Any bit of any file, (even an output file as in this example) can be set, cleared or tested. NO use of literals for bit manipulation is needed as in other microprocessors claiming bit handling capability and as a result time and ROM space is saved.
3) **Outputs are just like other files:** No distinction is made between a file connected to the outside world such as OUTPUT, and internal ones as UTILRG. This simplifies the instruction set resulting in less ROM space per instruction (always 12 bits only ie: one word). There are 4 output files, meaning 32 I/O lines (files 5, 6, 7 & 8).
4) **Special purpose registers are just like other files:** In this example the file status (file 3) is used. This contains carry, zero, and other flags. Likewise the real-time clock (file 1), the program counter (file 2), the indirect file pointer (file 4) and the register pointed to (file 0), are all treated (with one exception) as normal files. This again cuts down ROM space and program execution time.

Other important aspects of the single chip PIC microcomputer not shown by this example are the real-time-clock and the fact that there are a total of no less than 31 separate registers. There are 512 words of 12 bit ROM on the chip and since no instruction takes more than one word, this is similar to 1K words of 8 bit ROM on machines with earlier architecture. The whole is contained in a 40 lead dual in line package, (PIC1650A) and a version with off-chip ROM/PROM/RAM (PIC1664B).

6.6 Time Delay Routine

Many applications require precision timing as in, for example, sound generators, loop timing compensator, phase angle control, etc. Two routines are included, one for minimum size and 12μs resolution. The other for the maximum resolution of 4μs. Both the 12μs and 4μs resolution delay routines are called with the variable number of 12μs or 4μs intervals (assuming an instruction cycle time of 4 μsec) in the W register. There is a fixed delay associated with calling this routine and returning from the mainline that should be accounted for when determining the total delay.

a. 4μs Resolution Delay

```
                ; 4μs resolution time delay (1 instruction time)

VTL     MOVWF   TEMP
        CLRC
        RRF     TEMP
        SKPNC                   ; ADD 4μs
        GOTO    VTL1            ; Yes
VTL1    CLRC
        RRF     TEMP
        SKPC                    ; ADD 8μs
        GOTO    VTL3            ; No
        GOTO    VTL2            ; Yes
VTL2    NOP
VTL3    DECFSZ  TEMP
        GOTO    VTL2
        RET
```

b. 12μs Resolution Delay

```
DELAY   MOVWF   TEMP
        DECFSZ  TEMP
        GOTO    DELAY + 1
        RET
```

6.7 A Digital Clock Subroutine Using The PIC Microcomputer

Additional sales appeal can often be added to consumer and industrial products by adding a digital clock as a feature. This application note describes PIC routines needed to form a digital clock.

6.7.1 THEORY

The three basic methods of keeping accurate time using microcomputer methods are as follows:

Accurate Oscillator, fixed length instruction loop — This method allows the part to act as its own time keeping device by executing a certain number of machine cycles in a given amount of time. It requires no additional time inputs, but does require a crystal controlled oscillator.

Inaccurate Oscillator, variable instruction time—This method allows the programmer to construct routines without the need for careful fixed-loop time writing. It only requires that the program wait for a zero crossing or RTCC pin input before proceeding with the complete program loop.

Inaccurate Oscillator, fixed length instruction loop—This method provides for maintaining an instruction loop whose length is dependent upon an external time keeping signal (ordinarily 60 cycles). It provides for accurate time keeping and the "freezing" of current oscillator settings in case wall power should vanish, requiring the use of battery back-up.

6.7.2 TIME COUNTING

In all methods of clock programming, it is most convenient to keep the current (and target) times in BCD format for display. The following routine is provided to facilitate the time BCD manipulation:

Its features include:

Constant loop length for accurate timing
Add from 1 to 59 to time in BCD
Performs "time" decimal adjust
Rolls over at midnight
Keeps time in 24 hour clock for alarm purposes
Allows use with more than one clock

```
;;;;;;;;;
; ROUTINE TO TIME ADD TWO FOUR DIGIT BCD NUMBERS
;;;;;;;;
; ADDEND AND RESULT IN TWO FILES POINTED
; TO BY F4
;
; ADDER IN FILE 'PARM1' (TWO BCD DIGITS MAX)
; LOWER TWO BCD DIGITS IN EVEN LOCATION,
; UPPER TWO BCD DIGITS IN ODD LOCATION FOLLOWING
;
; LIMIT OF ADD = 59 BCD
;
; RETURNS WITH Z BIT ON IF LAST ADD CAUSED HIGH ORDER BCD TO
; GO TO BCD 25 (AROUND MIDNIGHT).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
TIMADD MOVLW 246
 ADDWF PARM1,W
 CLRF PARM1
 ADDWF 0 ;ADD TO LOW ORDER DIGITS
 BTFSC 3,0 ;SEE IF CARRY SET
 INCF PARM1
 MOVLW 132
 BTFSC 3,0
 GOTO ADD6
 BTFSC 3,1
 MOVLW 140
 GOTO ADD5 ;CAN BE REMOVED- IN FOR TIME PAD ONLY
ADD5 NOP ;CAN BE REMOVED- IN FOR TIME PAD ONLY
ADD5 NOP
ADD2 ADDWF 0
ADD3 BTFSS 4,0 ;SEE IF SECOND SET OF DIGITS
 GOTO ADD4
 INCF 4
 GOTO TIMADD
ADD4 MOVLW X'25'
 XORWF 0,W
 RET
ADD6 MOVLW 372
 BTFSC 3,1
 MOVLW 0
 GOTO ADD2
```

6.7.3 USE IN PROGRAM

The routine would be used in the following manner:

```
START   INITIALIZE TIME
              *
              *
LOOP     GET DIGIT 1-4
              *
              *
         GET SEGMENTS
         PUT OUT
              *
              *
         CONSTANT LENGTH
        PROGRAM OR LOOP
              *
              *
          LAST OF 4 DIGITS? ** NO **> TO LOOP
              *
              *
       TOO EARLY FOR 60 CYCLE **> LOOP CONSTANT INCREMENT
              *
              *
       TOO LATE FOR 60 CYCLE **> LOOP CONSTANT DECREMENT
              *
              *
        ADD TO SECONDS COUNT
          (TIMADD)
              *
              *
         LESS THAN SECOND **> TO LOOP
              *
              *
         ADD TO MINUTES/HOURS
           (TIMADD)
              *
              *
            TO LOOP
```

6.7.4 USE OF TIMADD AS TIME SET

The routine can also be used to increment the present time in the set mode. Note that when time is being set, constant loop length is maintained.

```
MOVWF 4 ;SET IN FSR FOR TIMADD
MOVLW 5 ;INCREMENT TIME BY MINUTES IF ALL CONDITIONS MET
BTFSS FLAGS,INSET ;CHECK IF IN SET MODE (SWITCH DEPRESSED)
MOVLW 0 ;NOT IN SET MODE
BTFSS FLAGS,TICK ;SEE IF ONE SECOND UP FOR TICK
MOVLW 0 ;ALREADY TICKED THIS SECOND
BCF FLAGS,TICK ;SET TO SERVICED
CALL TIMADD ;ADD ZERO OR 5 FOR CONSTANT TIME
```

7 APPLICATION NOTES

This section contains a variety of application notes which illustrate the versatility and performance capability of PIC microcomputers.

7.1 Serial Data Transmission with a PIC Microcomputer

INTRODUCTION

Serial data transmission is becoming more common in microcomputer applications. Even though the PIC does not contain a serial I/O port, the PIC can transmit serial data via an I/O line under software control. This application note describes the software techniques involved.

There will be two main tasks:

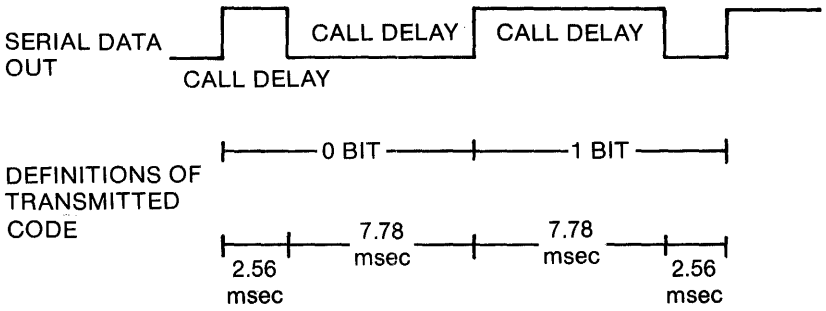
a) Control of the main application

b) Transmission of serial data

Since the timing of both tasks may be critical, the processor cannot suspend its control functions while transmitting a message — the processor must do both tasks "simultaneously." This can be accomplished by incorporating the control functions into a subroutine which is called by the transmit routine.

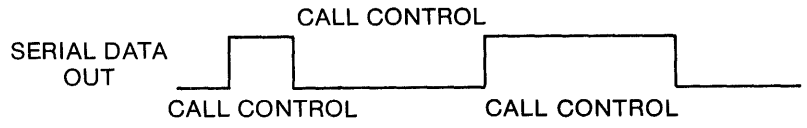
Usually, a delay subroutine is used to create the bit time:

Fig. 23



However, if the control section were made a subroutine, it could be called in place of the delay subroutine.

Fig. 24



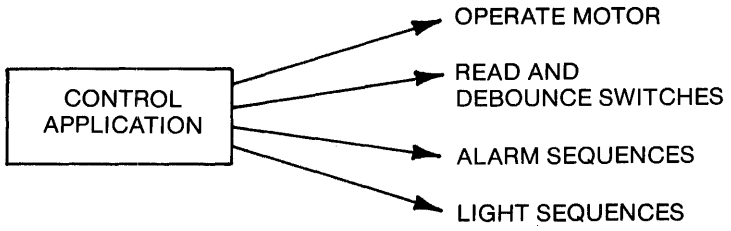
To use the control subroutine as an accurate delay, every path must be of equal time and padded to (in this example) 2.56 msecs.

In Figure 24, the control subroutine is called once to create the 2.56 msec delay and it is called three (3) consecutive times to create the 7.78 msec delay.

This technique was used in a PIC-controlled garage door opener. The PIC had to operate the motor, detect heat, carbon monoxide and intrusion, and indicate the garage status by various light and sound patterns. In addition, the PIC had to transmit a ten bit word (five bits address, five bits status) to a receiver in the home. The transmitted code was of the format shown in Figure 23.

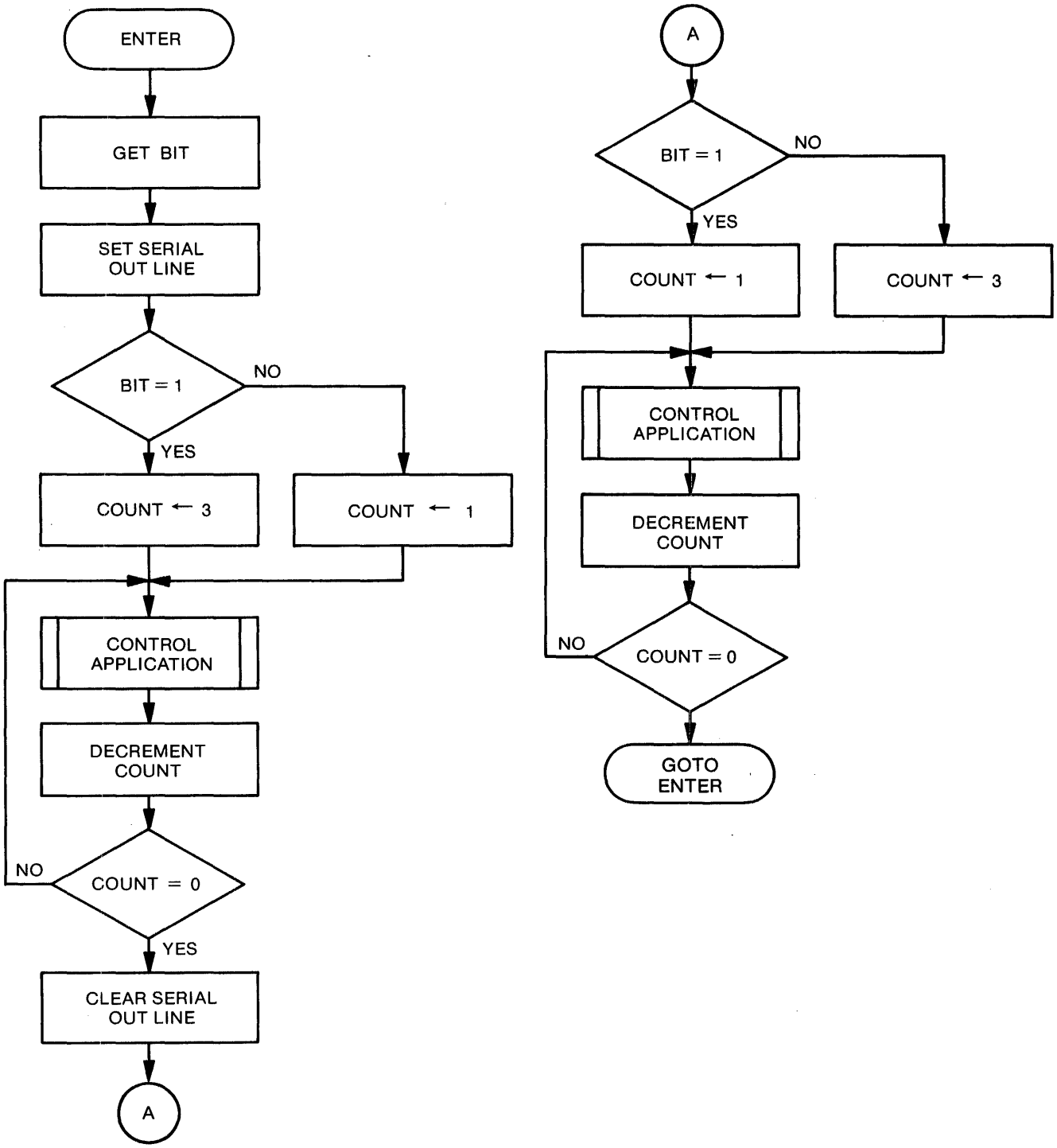
All the control functions were organized into a subroutine.

Fig. 25



The control subroutine was padded to 2.56 msec to create an accurate software timer. The general flowchart is shown in Figure 24. This scheme can be used in most applications that require serial data transmission including:

- ☐ Keyboard encoders
- ☐ Alarm systems
- ☐ UAR/T
- ☐ Systems using remote control


ENTER
GET BIT
SET SERIAL OUT LINE
BIT = 1
NO
YES
COUNT ← 3
COUNT ← 1
CONTROL APPLICATION
DECREMENT COUNT
NO
COUNT = 0
YES
CLEAR SERIAL OUT LINE
A
A
BIT = 1
NO
YES
COUNT ← 1
COUNT ← 3
CONTROL APPLICATION
DECREMENT COUNT
NO
COUNT = 0
GOTO ENTER

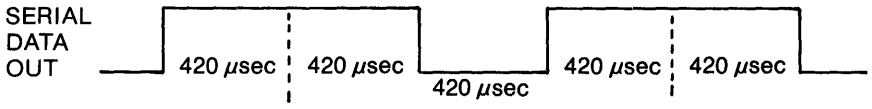
Another way of creating the correct bit time is by utilizing the Real Time Clock Counter (RTCC). By connecting the clockout output to the RTCC input, the RTCC register will increment every four microseconds (PIC1655A) independent of program execution. Thus, by presetting the RTCC register and testing for zero and wide range of bit times can be generated.

The value with which the RTCC is preset determines the interval to be timed, e.g., RTCC preset to 151_{10}.

The interval would be 256 − 151 x 4 μsec.

420 μsec = 1 bit @ 2400 Baud

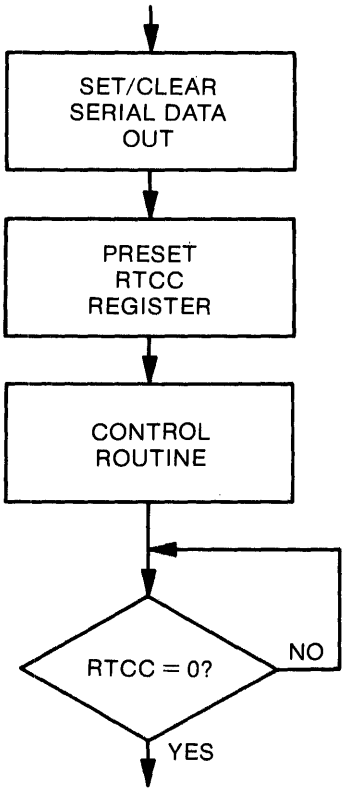
This technique was used in a PIC-controlled keyboard encoder. The PIC had to read and debounce keys plus perform the UAR/T transmit function.



In this example the control routine read, debound and stored key status.

Since the RTCC register will time the interval all paths through the control routine need not be of equal length. However, the longest path must be less than the bit time.

Flow diagram of program steps:



After the RTCC rolls over to zero, one bit has been transmitted.

In the previous flow diagram, the wait loop can be expressed by the following PIC code:

```
WAIT   BTFSC   RTCC, 7   NOTE: RTCC increments at end of
       GOTO    WAIT             instruction cycle
```

This is a twelve microsecond loop which checks for RTCC rollover. It is assumed that the RTCC register will have a value of at least 128_{10}. The loop waits for the most significant bit to change from one to zero.

The wait loop produces twelve microsecond accuracy, if four microsecond accuracy is required, the following instructions must be added:

```
WAIT   INCF    RTCC, W   One Cycle
       BTFSC   RTCC, 7   One/Two Cycles
       GOTO    WAIT      Two Cycles
       ADDWF   PC        Two Cycles
       NOP               One Cycle
       NOP               One Cycle
       NOP               One Cycle
```

The wait loop is now sixteen microseconds, however by adding the error from the loop to the program counter the appropriate number of NOP's will be executed to normalize the loop. This loop is exited seven cycles after the RTCC rolls over. When presetting the RTCC register, subtract seven from the computed value. This routine generates the accuracy needed for higher baud rates.

CONCLUSION

This application note has shown simple techniques for implementing serial communications under software control. Additional techniques using the interrupt system in the PIC1656 and PIC1670 will be covered in a future application note.

7.2 PIC Microcomputer as a Keyboard Encoder

INTRODUCTION

This application note describes the use of a PIC1650A microcomputer as a capacitive keyboard encoder. In the example explained, 128 keys are scanned sequentially. Upon detection of a key closure, encoding of the key position and outputting of the appropriate code is performed.

Depending upon I/O needs and the number of keys to be encoded, the software routines described may also be used with a PIC1655A or PIC1656 microcomputer.

CIRCUIT DESCRIPTION

Figure 26 is the keyboard encoder schematic. Ports RA and RB (X0-X15) selectively scan each column of the keyboard matrix. Only one of these scan lines will be high at any time. The CD4051 is an eight channel analog multiplexer, which is controlled by the I/O pins YA, YB, and YC. Row selection is obtained through control of this device. The output of the multiplexer will therefore correspond to that produced by the key at the junction of the column being scanned and the row selected.

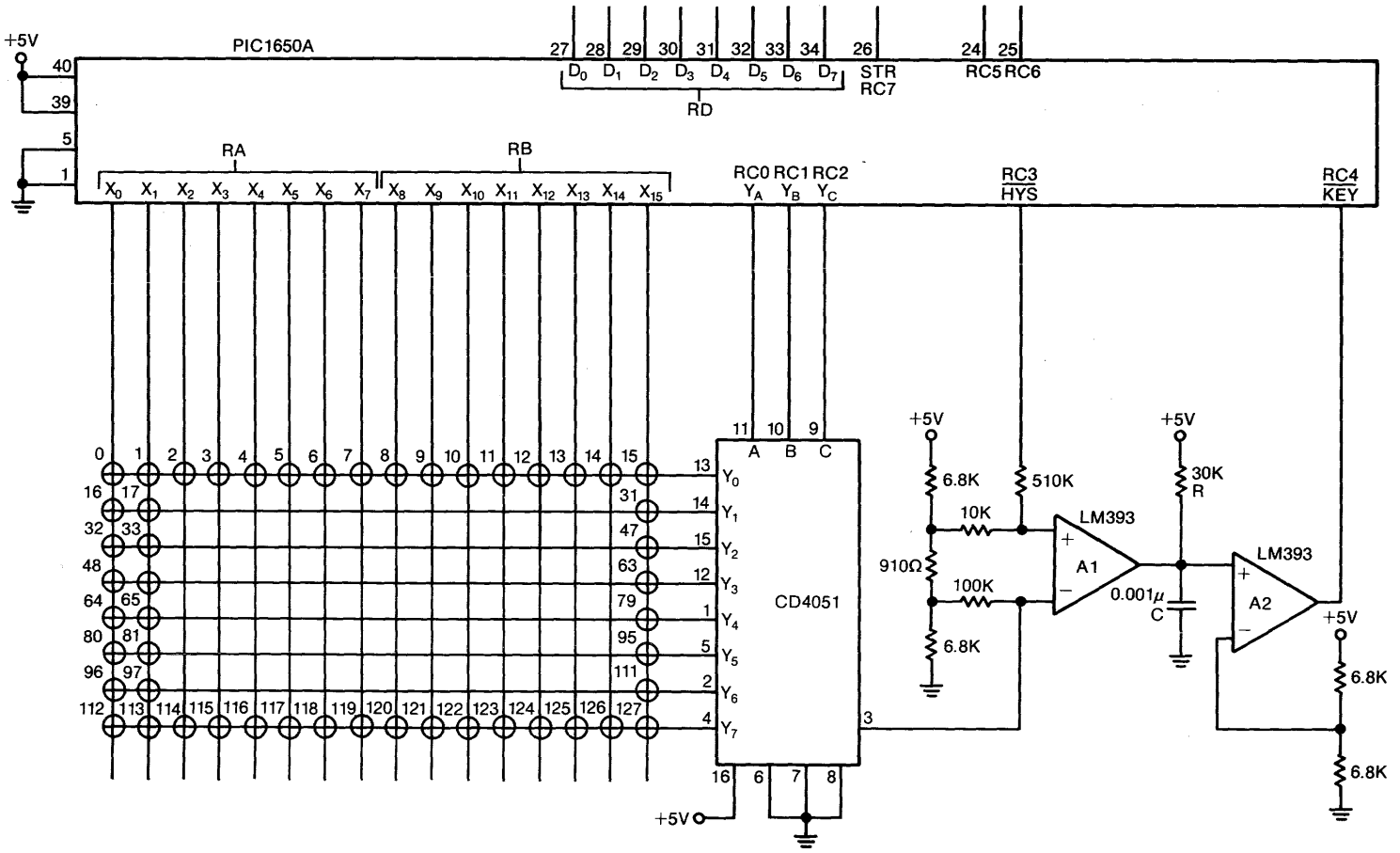


Fig. 26 PIC1650A KEYBOARD ENCODER

The detector circuit transforms this analog signal into a usable level at the $\overline{KEY}$ input of the microcomputer. The first stage of the detector circuit is a comparator formed by A1. The voltage reference for the comparator is established at the positive input by the resistor network. This reference is approximately 2.5 volts when $\overline{HYS}$ is high and slightly lower when $\overline{HYS}$ is low. Keys that have already been detected as being down are scanned with the lower threshold to provide hysteresis which prevents "teasing" of the keys. When the key being scanned is down, the scan line is capacitively coupled to the output, producing a positive going spike followed by a negative going spike. During the time that the magnitude of the positive spike is greater than the reference voltage the output will go low.

The second stage of the detector circuit serves as a one-shot to provide a pulse of approximately 21μs at the KEY input of the microcomputer when a key is down. The positive input of A2 will go low when a key is detected as being down. This is for a very short time however, so the RC network is used as a time delay to lengthen the low pulse at the output of A2. The procedure used to calculate the time delay is shown below.

$$V = V_O\ (1-e^{-t/RC})$$

The voltage at which the output will switch is $0.5V_O$, since the voltage divider network on the negative input establishes a reference of $0.5V_O$. The equation then becomes:

$$0.5V_O = V_O\ (1-e^{-t/RC})$$
$$0.5 = (1-e^{-t/RC})$$
$$0.5 = e^{-t/RC}$$
$$\ln 0.5 = -t/RC$$
$$t = -RC \ln 0.5$$
$$t = .693\ RC$$

With R = 30KΩ, and C = .001μf, t = 20.8μs.

This delay provides the needed time to sense the key status once the key has been scanned.

SOFTWARE DESCRIPTION

Figure 27 shows the internal register assignments used in the PIC1650A. F5 through F10 correspond to the I/O ports provided by the PIC1650A. SCANR is a scan register which is used to control the column in the key matrix to be scanned. WR is a working register used to store temporary data. Registers F20 through F37 provide the storage area required to record the status of each key. Each bit in the memory matrix corresponds to a key position. A bit equal to a one represents a key that is down, with a zero representing a key that is up. The numbers shown in the memory matrix registers correspond to the keys in the keyboard matrix. However these numbers do not remain in the same bit position within the register. During the scan routine, the data is rotated in a left to right fashion. When key 0 is scanned, the data in F20 is rotated right. Thus, the carry bit represents the status of key 0. Knowing this status and the result of the scan determines if key 0 has changed. The carry bit is moved into bit seven of the F20 to retain the status of key 0. Key 1 will

be scanned next, with F21 being rotated as F20 was. Keys 2 through 15 will then follow in the scan sequence, with F22 through F37 containing the respective data of each key.

The end of the memory matrix is reached after register F37, so the pointer, which is F4, is reset to point at F20 and the Y-select lines change to scan the next row. The next row contains keys 16 through 31. The same scanning routine is used now since the key status data for these keys is in bit 0 of each memory register.

Figure 28A is a flowchart showing the major steps in the scan routine. A listing of the program then follows with a detailed flowchart (Fig. 28B) of each command step being shown last.

It should be noted that upon power up, F5 and F6 should be cleared, along with the Y-select lines and all the internal registers of the PIC1650A. Key 0 will then be the first key scanned since YA, YB, and YC are all low, and the memory matrix data will indicate all keys being up as the initial condition. After the power-up routine is executed, the scan routine is executed beginning at the ENT label.

SUMMARY

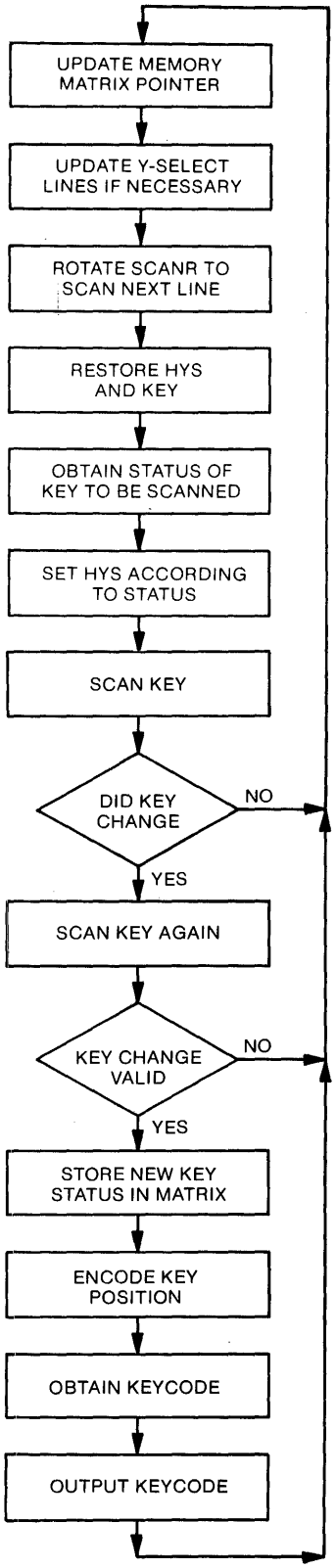
An understanding of the hardware and software described previously will give the user a basis upon which a complete keyboard encoder can be designed. The user will need to consider the number of keys to be scanned, the technology of the keys being used, the input-output configuration required, the coding requirements, and any other features desired in designing the PIC series microcomputer into a keyboard encoder system.

Fig. 27 PIC1650A KEYBOARD ENCODER REGISTER ASSIGNMENTS

	7	6	5	4	3	2	1	0	
F4									RTC
F5	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	(RA)
F6	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	X_9	X_8	(RB)
F7	STR			KEY	HYS	Y_C	Y_B	Y_A	IOR (RC)
F10 (.8)	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	DATA (RD)
F11 (.9)									SCANR
F12 (.10)									WR
F13 (.11)									
F14 (.12)									
F15 (.13)									
F16 (.14)									
F17 (.15)									
F20 (.16)	112	96	80	64	48	32	16	0	KEY MATRIX DATA
F21 (.17)	113	97	81	65	49	33	17	1	

	7	6	5	4	3	2	1	0
F22 (.18)	114	98	82	66	50	34	18	2
F23 (.19)	115	99	83	67	51	35	19	3
F24 (.20)	116	100	84	68	52	36	20	4
F25 (.21)	117	101	85	69	53	37	21	5
F26 (.22)	118	102	86	70	54	38	22	6
F27 (.23)	119	103	87	71	55	39	23	7
F30 (.24)	120	104	88	72	56	40	24	8
F31 (.25)	121	105	89	73	57	41	25	9
F32 (.26)	122	106	90	74	58	42	26	10
F33 (.27)	123	107	91	75	59	43	27	11
F34 (.28)	124	108	92	76	60	44	28	12
F35 (.29)	125	109	93	77	61	45	29	13
F36 (.30)	126	110	94	78	62	46	30	14
F37 (.31)	127	111	95	79	63	47	31	15

Fig. 28A BLOCK FLOWCHART



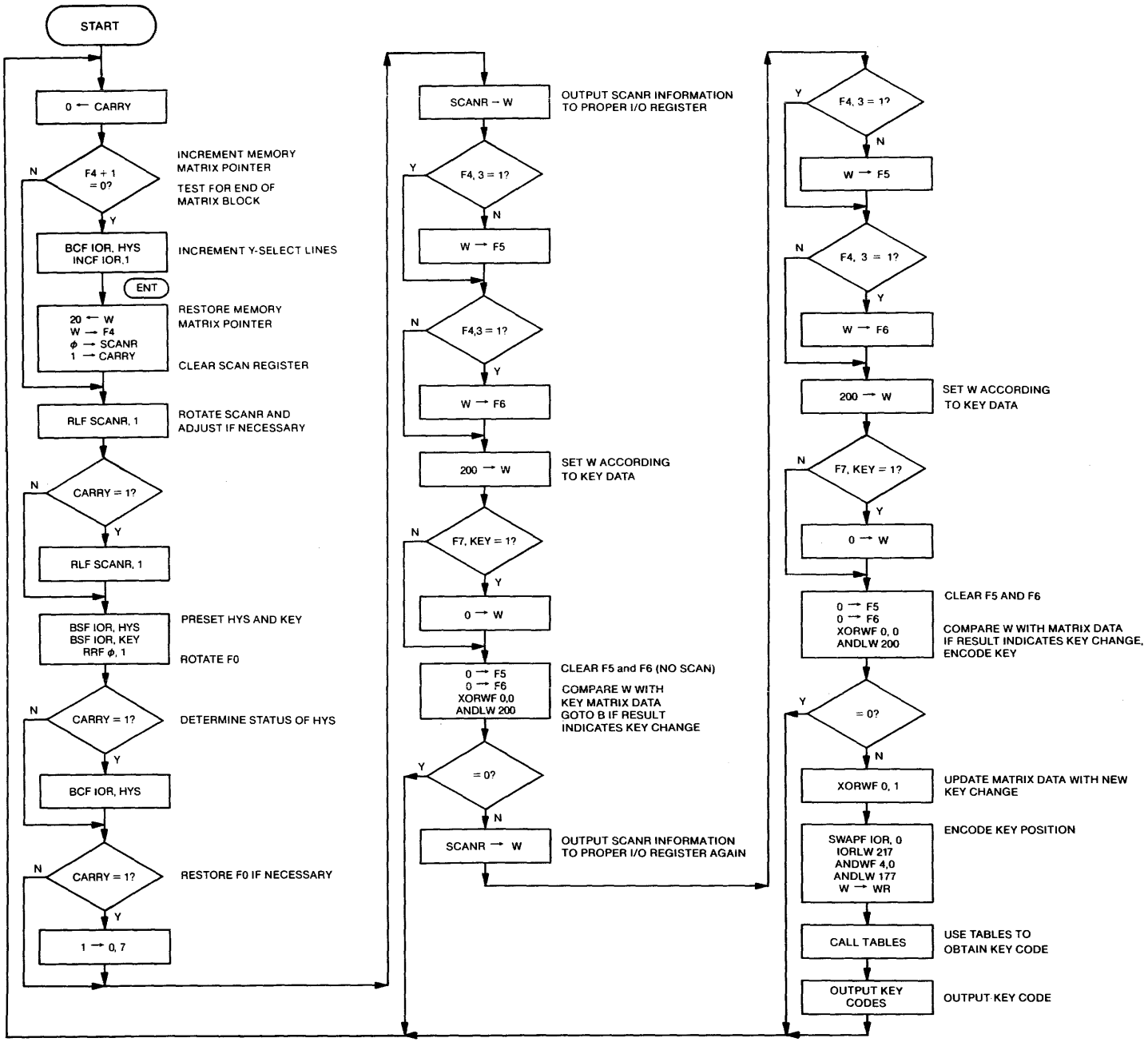


Fig. 28B DETAILED FLOWCHART

```
                                TITLE   'KEYBOARD ENCODER'
                                                        ;
                                                        ;PERFORMS SCANNING AND ENCODING OF
                                                        ;A 128 KEY CAPACITIVE KEYBOARD
                                                        ;
                                                        ;REGISTER ASSIGNMENTS
000007          IOR     =       7
000011          SCANR   =       11
000012          WR      =       12
                                                        ;
                                                        ;BIT ASSIGNMENTS
000003          HYS     =       3
000004          KEY     =       4
                                                        ;
000000  2003    A       BCF     3,0             ;CLEAR CARRY
000001  1744            INCFSZ  4,1             ;INCR POINTER (F4)
000002  5011            GOTO    SC2             ;POINTER NOT AT END
000003  2147            BCF     IOR,HYS         ;CLEAR FOR OVERFLOW
000004  1247            INCF    IOR,1           ;INCR Y-SELECT LINES
000005  6020    ENT     MOVLW   20
000006  0044            MOVWF   4               ;MATRIX POINTER SET TO BEGINNING
000007  0151            CLRF    SCANR           ;CLEAR SCAN REGISTER
000010  2403            BSF     3,0             ;SET CARRY
000011  1551    SC2     RLF     SCANR,1         ;ROTATE SCAN REGISTER
000012  3003            BTFSC   3,0             ;TEST FOR CARRY OUT
000013  1551            RLF     SCANR,1         ;ROTATE CARRY BACK IN
000014  2547            BSF     IOR,HYS         ;PRESET HYS
000015  2607            BSF     IOR,KEY         ;PRESET KEY
000016  1440            RRF     0,1             ;ROTATE MATRIX REGISTER
000017  3003            BTFSC   3,0
000020  2147            BCF     IOR,HYS         ;HYS->0 IF SCANNING KEY THAT IS DOWN
000021  3003            BTFSC   3,0
000022  2740            BSF     0,7             ;ADJUST MATRIX REGISTER IF NECESSARY
000023  1011            MOVF    SCANR,0         ;SCANR ->W
000024  3544            BTFSS   4,3
000025  0045            MOVWF   5               ;W->F5 IF F4 < 30
000026  3144            BTFSC   4,3
000027  0046            MOVWF   6               ;W->F6 IF F4 > OR = 30
000030  6200            MOVLW   200             ;SET W ACCORDING TO KEY
000031  3207            BTFSC   IOR,KEY
000032  6000            MOVLW   0
000033  0145            CLRF    5               ;CLEAR F5
000034  0146            CLRF    6               ;CLEAR F6
000035  0600            XORWF   0,0             ;COMPARE W WITH KEY MATRIX DATA
000036  7200            ANDLW   200             ;LOOK AT BIT 7 ONLY
000037  3103            BTFSC   3,2             ;DID KEY CHANGE?
000040  5000            GOTO    A               ;NO, SCAN NEXT KEY
000041  1011            MOVF    SCANR,0         ;YES, SCAN AGAIN
000042  3544            BTFSS   4,3
000043  0045            MOVWF   5               ;W-> F5 IF F4 < 30
000044  3144            BTFSC   4,3
000045  0046            MOVWF   6               ;W-> F6 IF F4 > OR = 30
000046  6200            MOVLW   200             ;SET W ACCORDING TO KEY
000047  3207            BTFSC   IOR,KEY
```

```
 55  000050  6000                MOVLW  0
 56  000051  0145                CLRF   5              #CLEAR F5
 57  000052  0146                CLRF   6              #CLEAR F6
 58  000053  0600                XORWF  0,0            #COMPARE W WITH MATRIX DATA
 59  000054  7200                ANDLW  200            #LOOK AT BIT 7
 60  000055  3103                BTFSC  3,2            #KEY STILL CHANGED?
 61  000056  5000                GOTO   A              #NO
 62  000057  0640                XORWF  0,1            #YES, CHANGE KEY MATRIX DATA
 63  000060  1607                SWAPF  IOR,0          #ENCODE KEY POSITION
 64  000061  6617                IORLW  217            #W= 1 YC YB YA 1 1 1 1
 65  000062  0504                ANDWF  4,0            #W= 1 YC YB YA (4 LSB OF F4)
 66  000063  7177                ANDLW  177            #BIT 7 ->0
 67  000064  0052                MOVWF  WR             #KEY POSITION IN WR
 68                                                    #TABLES CAN NOW BE CALLED TO GET CODES
 69                                                    #FOR THE KEY NUMBER CONTAINED IN WR.
 70                                                    #THE CODE CAN THEN BE OUTPUT IN THE
 71                                                    #FORMAT REQUIRED. EXECUTION THEN RETURNS
 72                                                    #TO LABEL A TO CONTINUE SCANNING.
 73  000065                      END

 ASSEMBLER ERRORS =     0

                              SYMBOL TABLE

 A       000000     ENT     000005     HYS     000003     IOR     000007
 KEY     000004     SC2     000011     SCANR   000011     WR      000012
 KEY
EOF:88
0:>
```

7.3 Sound Generation Using a PIC Microcomputer

This application note describes a circuit (Figure 29) using the PIC1655A to produce the following sounds commonly used for electronic toys and games.

- ☐ Machine gun and Ricochet
- ☐ European Siren
- ☐ Phasor
- ☐ Racing Car Engine — Rev Up/Down
- ☐ Car Tire Screech
- ☐ Car Crash
- ☐ Mortar Shell Whistle and Explosion
- ☐ Tune 1 — Charge
  Tune 2 — Snake Charmer's Song

Each sound is created by one or more of the following techniques:

1. Pulse train of fixed frequency.
2. Periodic increase/decrease of frequency.
3. Superimposing an exponential decay (or ramp) envelope of 1 sec or 2 sec time constants on the sound.
4. Beating (mixing) together two frequencies.
5. White noise generation — Random Pulse Output.

Exponential Decay Generator

Channels 1 and 2 (Figure 29) each have an envelope generator circuit (1 sec and 2 sec RC time constant, respectively) at the base of the switching transistor. On Channel 1, a low on RC7 discharges the capacitor and the transistor switches on. A high on RC7 activates the RC circuit and the capacitor charges up exponentially to 6V. This appears as an exponential decay at the collector of the transistor. If the pulse train is fed to the emitter, it appears at the collector with the decay superimposed on it. See Figure 30.

Machine Gun and Ricochet (Created on Channel 1)

A random number (created by routine RANGEN) between 1 and 15 gives the number of shots per burst of machine gun fire. Each shot is produced by outputting random pulses (white noise) of a width of 28μs for about 7ms. These are superimposed by a decay of 1 sec. Each shot is separated by a delay of 40ms.

Each burst of machine gun fire is followed by a ricochet. This sound is created by superimposing a decay envelope of 1 sec over a pulse train (50% duty cycle) whose frequency is decreased slowly in 80Hz steps (every 15 cycles) from 3KHz to 1KHz.

Fig. 29 SCHEMATIC

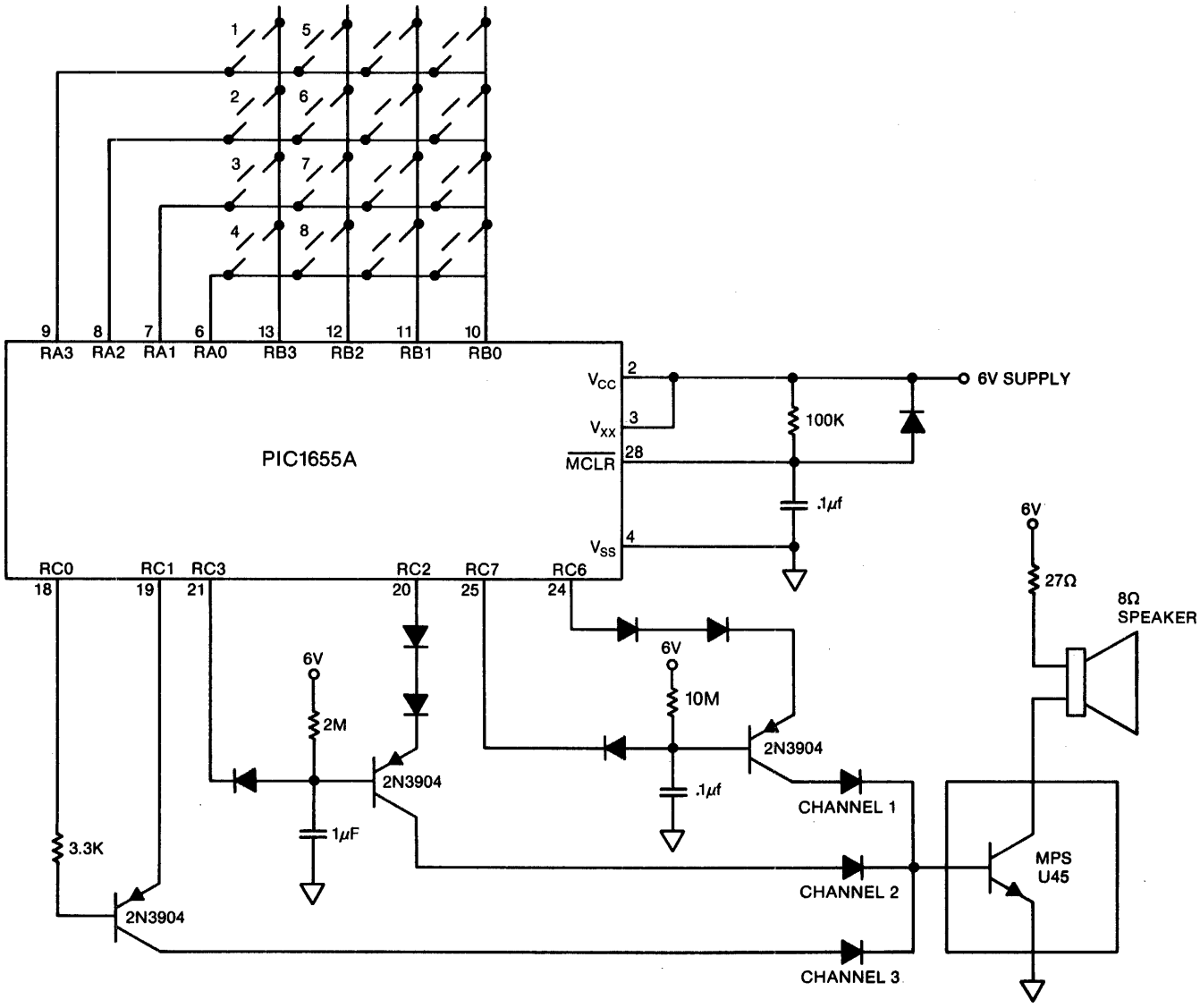
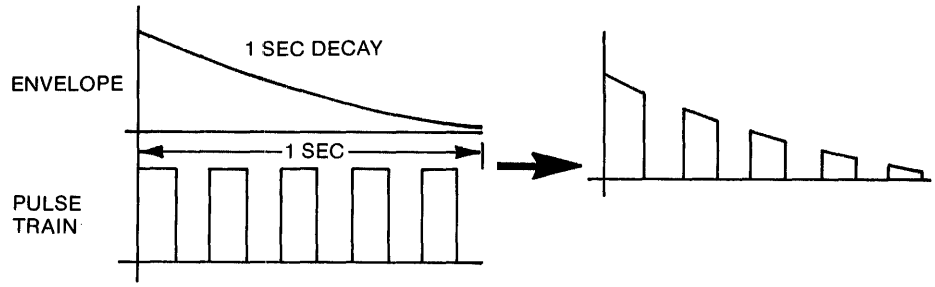


Fig. 30 EFFECT OF ENVELOPE ON PULSE TRAIN



European Siren (Created on Channel 1)
The siren is made up of two components. The higher frequency part at 500Hz and the low frequency component of 300Hz. Both components are created by pulse trains of fixed frequency. Starting with the high frequency sound, the effect of the siren is obtained by switching back and forth between the two (high and low frequency) sounds. Duration of the high frequency sound before switching is 256ms, while that of the low frequency is about 400ms.

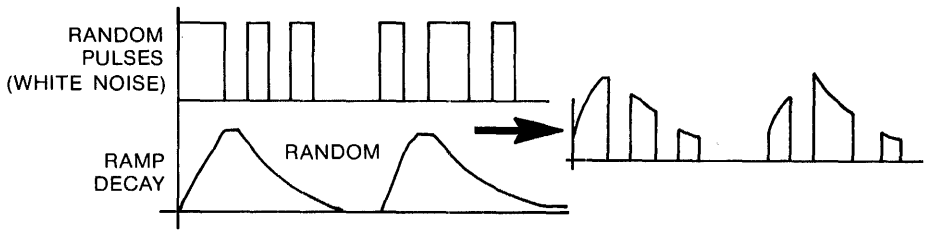
Phasor (Channel 3)
The phasor finds an application as the sound of a "phasor" gun in space war games. Starting with a frequency of about 1KHz, the frequency is decreased (in steps of 40Hz every 1/2 cycle) down to 200Hz. This is repeated for a burst of phasor fire.

Racing Car Engine — Rev Up/Down (Channel 3)
The engine sound is produced by beating (mixing) two low frequency pulse trains together. This is simulated in software by having a fixed frequency variable duty cycle output.

Starting with a frequency of 70Hz and 15% duty cycle, the duty cycle is increased in steps of 14%, until it reaches about 100% (7 cycles). The frequency is then switched to 80Hz and duty cycle is then decreased in 14% steps to 0. This is then switched back to 70Hz and the procedure repeated. The effect is to have a beat every 7th cycle (at frequencies of 10 and 11Hz). To rev up, the higher (80Hz) beating frequency is increased in ½Hz steps up to 300Hz. To rev down, frequency is decreased in ½Hz steps, back down to 80Hz

Car Tire Screech (Channel 1)
The effect of a tire screech is produced by superimposing an exponential ramp followed by a decay upon a white noise output. (See Figure 31.) Each ramp and decay is separated by a random delay.

Fig. 31 EFFECT OF RAMP AND DECAY ON WHITE NOISE



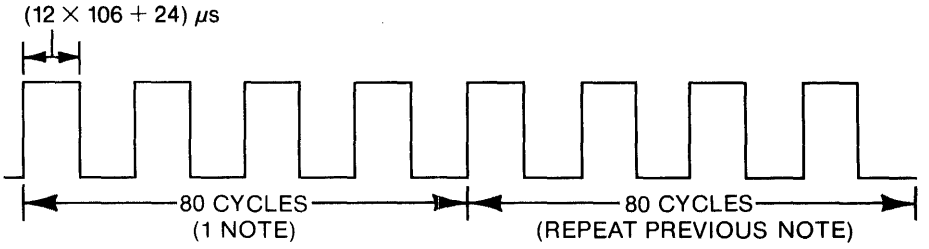
Car Crash (Channel 2)
Superimposing a 2 second exponential decay upon a white noise output creates this sound.

Shell Whistle and Explosion (Channel 3)
The method of creating a whistle sound is similar to that for the phasor except that all software loops must be equal length. Starting with a frequency of about 4KHz, the frequency of the pulse train is decreased in 150Hz steps (every 32 cycles) down to about 900Hz. When the frequency is at its minimum (≃900Hz), the crash routine is called to simulate an explosion.

Tunes — "Charge" and "Snake Charmer's Song" (Channel 3)
Each tune is a collection of notes. Each note is of fixed frequency and duration. This information is coded into a 8 bit word called "note data." Each note has a duration of 80 cycles. The most significant bit of note data gives the number of times the note must be repeated for that part of the tune. The 7 least significant bits gives the frequency.

e.g. Note data — $352_8 = 11101010$
MSB — 1: The note must be repeated once, i.e. double note.
The frequency is determined by 152_8 or 106 decimal.
It is a 12μs loop: $T = 2 \times [(12 \times 106) + 24]\ \mu s$
$= 2592\mu s$
$f = 385Hz.$

Fig. 32 A "NOTE DATA" OUTPUT — 352_8



The note data is in the form of a table. The software fetches each note data in turn, decodes it and outputs as shown in Figure 32.

A computer assembly listing of the routines used follows (Figure 32A).

Fig. 32A NOTE DATA OUTPUT: SOUND DEMO PROGRAM — SOUND EFFECTS

```
 1                                TITLE  'SOUND EFFECTS'
 2                                LIST   P=1655,E
 3                                                    ;PIC-SOUND DEMO PROGRAM
 4                                                    ;9-2-80
 5  000000
 6  000005                IN      EQU    5
 7  000006                OUT     EQU    6
 8  000007                IO      EQU    7
 9  000011                SL      EQU    11
10  000012                SH      EQU    12
11  000013                SFREQ   EQU    13
12  000014                TEMP    EQU    14
13  000015                OUTBUF  EQU    15
14  000016                INBUF   EQU    16
15  000017                TEMP2   EQU    17
16  000025                SWITCH  EQU    25
17  000026                OFFSET  EQU    26
18  000027                FREQ    EQU    27
19  000030                WAY     EQU    30
20  000031                TONE    EQU    31
21  000032                HOLDN   EQU    32
22  000033                TEMPH   EQU    33
23  000034                WORK    EQU    34
24  000035                WORK1   EQU    35
25  000036                POINT   EQU    36
26                                                   ;
27                                                   ;
28                                                   ;
29  000000 01551          RUMBLE  RLF    SL
30  000001 01552                  RLF    SH
31  000002 01015                  MOVF   OUTBUF,W
32  000003 00047                  MOVWF  IO
33  000004 04000                  RET
34  000005
35  000005
36                                                   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
37                                                   ;                                          ;
38                                                   ; SIXTEEN BIT WHITE NOISE GENERATOR
39                                                   ;                                          ;
40                                                   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
41  000005
42  000005 01011          RANGEN  MOVF   SL,W           ; XOR BITS 2,15
43  000006 00054                  MOVWF  TEMP
44  000007 01654                  SWAPF  TEMP
45  000010 01554                  RLF    TEMP
46  000011 01012                  MOVF   SH,W
47  000012 00654                  XORWF  TEMP
48  000013 01554                  RLF    TEMP
49  000014 01551                  RLF    SL
50  000015 01552                  RLF    SH
51  000016 04000                  RET
52                                                   ;
53                                                   ;
54                                                   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
55                                                   ;                                          ;
56                                                   ;20USEC RESOLUTION DELAY ROUTINE           ;
57                                                   ;                                          ;
58                                                   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
59                                                   ;
60  000017 05020          DELAY   GOTO   DELAY+1
61  000020 01354                  DECFSZ TEMP
62  000021 05017                  GOTO   DELAY
63  000022 01357                  DECFSZ TEMP2
64  000023 05017                  GOTO   DELAY
65  000024 04000                  RET
66                                                   ;
67                                                   ;
68                                                   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
69                                                   ;                                          ;
70                                                   ;4USEC RESOLUTION DELAY ROUTINE            ;
71                                                   ;                                          ;
72                                                   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
73                                                   ;
74  000025 00057          DEL4    MOVWF  TEMP2
75  000026 02003                  CLRC
76  000027 01457                  RRF    TEMP2
77  000030 03003                  SKPNC
```

```
 78  000031 05032                GOTO   VTL1
 79  000032 02003        VTL1    CLRC
 80  000033 01457                RRF    TEMP2
 81  000034 03403                SKPC
 82  000035 05040                GOTO   VTL3
 83  000036 05037                GOTO   VTL2
 84  000037 00000        VTL2    NOP
 85  000040 01357        VTL3    DECFSZ TEMP2
 86  000041 05037                GOTO   VTL2
 87  000042 04000                RET
 88                                                ;
 89                                                ;
 90                                                ;#########################################
 91                                                ;                                       #
 92                                                ;CHANNEL SWITCHING-ENVELOPE DECAY       #
 93                                                ;                                      #
 94                                                ;#########################################
 95                                                ;
 96  000043 06377        DECAY   MOVLW  377
 97  000044 00054                MOVWF  TEMP
 98  000045 06100                MOVLW  100
 99  000046 00057                MOVWF  TEMP2
100  000047 04417                CALL   DELAY
101  000050 04000                RET
102                                               ;
103                                               ;
104                                               ;#######################################
105                                               ;                                      #
106                                               ;'WILD CHARGE' TUNE-NOTE DATA          #
107                                               ;                                      #
108                                               ;#######################################
109                                               ;
110  000051 00742        PLAY    ADDWF  2
111  000052 04074                RETLW  74
112  000053 04107                RETLW  107
113  000054 04274                RETLW  274
114  000055 04107                RETLW  107
115  000056 04131                RETLW  131
116                                               ;
117                                               ;#######################################
118                                               ;                                      #
119                                               ;'SNAKE CHARMERS' TUNE-NOTE DATA       #
120                                               ;                                      #
121                                               ;#######################################
122                                               ;
123  000057 00742        PLAY1   ADDWF  2
124  000060 04167                RETLW  167
125  000061 04144                RETLW  144
126  000062 04152                RETLW  152
127  000063 04120                RETLW  120
128  000064 04144                RETLW  144
129  000065 04152                RETLW  152
130  000066 04167                RETLW  167
131  000067 04367                RETLW  367
132  000070 04352                RETLW  352
133  000071 04344                RETLW  344
134  000072 04152                RETLW  152
135                                               ;
136                                               ;
137                                               ;
138                                               ;
139  000073 06377        KEYPAD  MOVLW  377
140  000074 00046                MOVWF  OUT
141  000075 06004                MOVLW  4
142  000076 00054                MOVWF  TEMP
143  000077 06367                MOVLW  367
144  000100 00057                MOVWF  TEMP2
145  000101 01017        KEY1    MOVF   TEMP2,W
146  000102 00046                MOVWF  OUT
147  000103 01005                MOVF   IN,W
148  000104 07017                ANDLW  17
149  000105 07417                XORLW  17
150  000106 03103                SKPNZ
151  000107 05143                GOTO   ROTAT
152  000110 00056                MOVWF  INBUF
153  000111 03156                BTFSC  INBUF,3
154  000112 06122                MOVLW  SNDTBL-1
155  000113 03116                BTFSC  INBUF,2
156  000114 06126                MOVLW  .4+SNDTBL-1
157  000115 03056                BTFSC  INBUF,1
158  000116 06132                MOVLW  .8+SNDTBL-1
```

```
 159  000117 03016                    BTFSC   INBUF,0
 160  000120 06136                    MOVLW   .12+SNDTBL-1
 161  000121 00714                    ADDWF   TEMP,W
 162  000122 00042                    MOVWF   2
 163
 164  000123 05073            SNDTBL  GOTO    KEYPAD
 165  000124 05711                    GOTO    TUNE1
 166  000125 05500                    GOTO    TSCRCH
 167  000126 05150                    GOTO    MCGN
 168  000127 05073                    GOTO    KEYPAD
 169  000130 05073                    GOTO    KEYPAD
 170  000131 05527                    GOTO    CRASH
 171  000132 05262                    GOTO    SIREN
 172  000133 05073                    GOTO    KEYPAD
 173  000134 05073                    GOTO    KEYPAD
 174  000135 05610                    GOTO    WISTLE
 175  000136 05563                    GOTO    PHASOR
 176  000137 05073                    GOTO    KEYPAD
 177  000140 05073                    GOTO    KEYPAD
 178  000141 05634                    GOTO    TUNE

*TB*TOES AND GIMAIL ARE NOW AVAILABLE FOR USE....
 179  000142 05373                    GOTO    AUTGR
 180                                                     ;
 181  000143 02403            ROTAT   SETC
 182  000144 01457                    RRF     TEMP2
 183  000145 01354                    DECFSZ  TEMP
 184  000146 05101                    GOTO    KEY1
 185  000147 05073                    GOTO    KEYPAD
 186                                                     ;
 187                                                     ;
 188                                                     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 189                                                     ;                                       ;
 190                                                     ;MACHINE GUN AND RICOCHET               ;
 191                                                     ;                                       ;
 192                                                     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 193                                                     ;
 194  000150 06231            MCGN    MOVLW   231
 195  000151 00047                    MOVWF   IO              ; TURN OFF ALL SOUND CHANNELS
 196  000152 00055                    MOVWF   OUTBUF
 197  000153 04443                    CALL    DECAY
 198  000154 04405            MCGN2   CALL    RANGEN          ; F23 GETS RANDOM # 1-15:NUMBER
 199  000155 01014                    MOVF    TEMP,W          ; OF SHOTS IN 1 M/C GUN BURST.
 200  000156 06401                    IORLW   1
 201  000157 07017                    ANDLW   17
 202  000160 00063                    MOVWF   23
 203  000161 06377            MCGN1   MOVLW   377
 204  000162 00072                    MOVWF   32
 205  000163 00074                    MOVWF   34
 206  000164 06001                    MOVLW   1
 207  000165 00076                    MOVWF   36
 208  000166 02355                    BCF     OUTBUF,7
 209  000167 04405            GUN     CALL    RANGEN          ; CREATE 1 SHOT OF M/C GUN FIRE.
 210  000170 03354                    BTFSC   TEMP,7          ; RANDOM PULSES (WHITE NOISE).
 211  000171 05175                    GOTO    GUN1
 212  000172 02003                    CLRC
 213  000173 02315                    BCF     OUTBUF,6
 214  000174 05177                    GOTO    GUN2
 215  000175 02403            GUN1    SETC
 216  000176 02715                    BSF     OUTBUF,6
 217  000177 04400            GUN2    CALL    RUMBLE
 218  000200 06002                    MOVLW   2
 219  000201 00054                    MOVWF   TEMP
 220  000202 01354            GUN3    DECFSZ  TEMP
 221  000203 05202                    GOTO    GUN3
 222  000204 02755                    BSF     OUTBUF,7
 223  000205 01374                    DECFSZ  34
 224  000206 05167                    GOTO    GUN
 225  000207 01376                    DECFSZ  36
 226  000210 05167                    GOTO    GUN
 227  000211 00372            DLY     DECF    32
 228  000212 03103                    SKPNZ
 229  000213 05222                    GOTO    DLYDN
 230  000214 06005                    MOVLW   5               ; 1 M/C GUN SHOT OVER.
 231  000215 00054                    MOVWF   TEMP
 232  000216 06001                    MOVLW   1
 233  000217 00057                    MOVWF   TEMP2
 234  000220 04417                    CALL    DELAY
 235  000221 05211                    GOTO    DLY
 236  000222 01363            DLYDN   DECFSZ  23
 237  000223 05161                    GOTO    MCGN1           ; REPEAT SHOT FOR A BURST OF FIRE.
```

```
238  000224 02355                BCF    OUTBUF,7        # START RICOCHET.
239  000225 01015                MOVF   OUTBUF,W
240  000226 00047                MOVWF  IO
241  000227 06030          INRICO MOVLW  30
242  000230 00077                MOVWF  37
243  000231 00065                MOVWF  25              # START ENVELOPE DECAY (1SEC).
244  000232 02755                BSF    OUTBUF,7
245  000233 01015                MOVF   OUTBUF,W
246  000234 00047                MOVWF  IO
247  000235 06100          RICO1 MOVLW  100             # OUTPUT PULSE TRAIN.
248  000236 00655                XORWF  OUTBUF
249  000237 01015                MOVF   OUTBUF,W
250  000240 00047                MOVWF  IO
251  000241 01377          RICO2 DECFSZ 37
252  000242 05253                GOTO   NOP1
253  000243 06030                MOVLW  30
254  000244 00077                MOVWF  37
255  000245 01265                INCF   25
256  000246 06150                MOVLW  150
257  000247 00625                XORWF  25,W
258  000250 03503                BTFSS  3,2
259  000251 05257                GOTO   VTL
260  000252 05073                GOTO   KEYPAD
261  000253 05254          NOP1  GOTO   NOP1+1
262  000254 05255                GOTO   NOP1+2
263  000255 05256                GOTO   NOP1+3
264  000256 00000                NOP
265  000257 01025          VTL   MOVF   25,W
266  000260 04425                CALL   DEL4
267  000261 05235                GOTO   RICO1
268                                                    #
269                                                    #
270                                                    ######################################
271                                                    #                                    #
272                                                    #EUROPEAN SIREN                       #
273                                                    #                                    #
274                                                    ######################################
275                                                    #
276  000262 06131          SIREN MOVLW  131
277  000263 00047                MOVWF  IO
278  000264 06377                MOVLW  377
279  000265 00074                MOVWF  34
280  000266 06001                MOVLW  1
281  000267 00075                MOVWF  35
282  000270 06331                MOVLW  331
283  000271 00055                MOVWF  OUTBUF
284  000272 06200          SIREN1 MOVLW  200            # HI FREQUENCY PART OF SIREN.
285  000273 00053                MOVWF  SFREQ
286  000274 02025                BCF    SWITCH,0
287  000275 05301                GOTO   SCONT
288  000276 06350          PHSR1 MOVLW  350             # LO FREQUENCY PART OF SIREN.
289  000277 00053                MOVWF  SFREQ
290  000300 02425                BSF    SWITCH,0
291  000301 06300          SCONT MOVLW  300
292  000302 00655                XORWF  OUTBUF
293  000303 01015                MOVF   OUTBUF,W
294  000304 00047                MOVWF  IO
295  000305 01013                MOVF   SFREQ,W
296  000306 00054                MOVWF  TEMP
297  000307 01454                RRF    TEMP
298  000310 01354          SLOOP DECFSZ TEMP
299  000311 05310                GOTO   SLOOP
300  000312 01030                MOVF   WAY,W
301  000313 00753                ADDWF  SFREQ
302  000314 06375                MOVLW  375
303  000315 00213                SUBWF  SFREQ,W
304  000316 07450                XORLW  50
305  000317 03403                SKPC
306  000320 03103                SKPNZ
307  000321 05366                GOTO   NEGWAY
308  000322 01374                DECFSZ 34            # REPEAT CURRENT SOUND (HI OR LO)
309  000323 05355                GOTO   RPT           # TILL COUNTER HAS COUNTED DOWN
310  000324 01375                DECFSZ 35            # TO ZERO.
311  000325 05355                GOTO   RPT
312  000326 06003                MOVLW  3
313  000327 00076                MOVWF  36
314  000330 06377                MOVLW  377
315  000331 00077                MOVWF  37
316  000332 05333          A     GOTO   A+1           # 48USEC DELAY.
317  000333 05334                GOTO   A+2
318  000334 05335                GOTO   A+3
```

```
319  000335 05336                    GOTO   A+4
320  000336 05337                    GOTO   A+5
321  000337 05340                    GOTO   A+6
322  000340 01377                    DECFSZ 37
323  000341 05332                    GOTO   A
324  000342 01376                    DECFSZ 36
325  000343 05332                    GOTO   A
326  000344 06377          CHNGE     MOVLW  377            ; SWITCH SOUND JUST MADE (HI
327  000345 00074                    MOVWF  34             ; FREQ. TO LO FREQ. OR VICE
328  000346 06001                    MOVLW  1              ; VERSA).
329  000347 00075                    MOVWF  35
330  000350 03425                    BTFSS  SWITCH,0
331  000351 05276                    GOTO   PHSR1
332  000352 06002                    MOVLW  2
333  000353 00075                    MOVWF  35
334  000354 05272                    GOTO   SIREN1
335  000355 03025          RPT       BTFSC  SWITCH,0
336  000356 05276                    GOTO   PHSR1
337  000357 06367                    MOVLW  367
338  000360 00046                    MOVWF  6
339  000361 01005                    MOVF   5,W
340  000362 00056                    MOVWF  16
341  000363 03116                    BTFSC  16,2
342  000364 05073                    GOTO   KEYPAD
343  000365 05272                    GOTO   SIREN1
344  000366 01170          NEGWAY    COMF   WAY
345  000367 01270                    INCF   WAY
346  000370 03425                    BTFSS  SWITCH,0
347  000371 05276                    GOTO   PHSR1
348  000372 05301                    GOTO   SCONT
349                                                    ;
350                                                    ;
351                                                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
352                                                    ;                              ;
353                                                    ;AUTOMOBILE ENGINE REVV UP/DN  ;
354                                                    ;                              ;
355                                                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
356                                                    ;
357  000373 06001          AUTGR     MOVLW  1
358  000374 00073                    MOVWF  33
359  000375 06031                    MOVLW  31
360  000376 00047                    MOVWF  IO
361  000377 00070                    MOVWF  30
362  000400 06225                    MOVLW  225
363  000401 00060                    MOVWF  20
364  000402 00061                    MOVWF  21
365  000403 06200                    MOVLW  200
366  000404 00062                    MOVWF  22
367  000405 00063                    MOVWF  23
368  000406 06377                    MOVLW  377
369  000407 00071                    MOVWF  31
370  000410 06003                    MOVLW  3
371  000411 00072                    MOVWF  32
372  000412 01360          ENG       DECFSZ 20             ; LO FREQUENCY.
373  000413 05467                    GOTO   PAD
374  000414 01021                    MOVF   21,W
375  000415 00060                    MOVWF  20
376  000416 02470                    BSF    30,1
377  000417 01362          ENG1      DECFSZ 22             ; HI FREQUENCY. THE 2 SOUNDS ARE
378  000420 05470                    GOTO   PAD1           ; MIXED TOGETHER TO CREATE BEATS.
379  000421 01023                    MOVF   23,W
380  000422 00062                    MOVWF  22
381  000423 02470                    BSF    30,1
382  000424 01030          ENG2      MOVF   30,W
383  000425 00647                    XORWF  IO
384  000426 00170                    CLRF   30
385  000427 01371                    DECFSZ 31
386  000430 05471                    GOTO   PAD2
387  000431 01372                    DECFSZ 32
388  000432 05472                    GOTO   PAD2+1
389  000433 06003                    MOVLW  3
390  000434 00072                    MOVWF  32
391  000435 03433                    BTFSS  33,0           ; REVV UP/DN FLAG SET?
392  000436 05460                    GOTO   REVDN          ; NO!-REVV DOWN.
393  000437 00363          REVUP     DECF   23             ; YES!-REVV UP.
394  000440 06030                    MOVLW  30             ; DECREMENT COUNTER(INC. FREQ.).
395  000441 00623                    XORWF  23,W
396  000442 03503                    SKPZ                  ; REVVED UP TO MAX FREQUENCY?
397  000443 05446                    GOTO   KEYPR          ; NO! CHECK IF KEY PRESSED.
398  000444 06031                    MOVLW  31             ; YES! MANTAIN MAX FREQ TILL KEY
399  000445 00063                    MOVWF  23             ; RELEASED FOR REVV DOWN.
400  000446 06367          KEYPR     MOVLW  367            ; LOOK FOR KEY RELEASE.
```

```
401  000447 00046                MOVWF   OUT
402  000450 01005                MOVF    IN,W
403  000451 00056                MOVWF   INBUF
404  000452 03416                BTFSS   INBUF,0
405  000453 05456                GOTO    KEYPR1
406  000454 02033                BCF     33,0            ; KEY RELEASED-REVVDOWN.
407  000455 05412                GOTO    ENG
408  000456 02433          KEYPR1 BSF    33,0            ; KEY STILL PRESSED-REVV UP
409  000457 05412                GOTO    ENG             ; OR CONTINUE AT MAX FREQUENCY.
410                                                      ;
411  000460 01263          REVDN  INCF   23              ; INCREMENT COUNTER (DECREMENT
412  000461 06200                MOVLW   200             ; FREQUENCY).
413  000462 00623                XORWF   23,W
414  000463 03503                SKPZ                    ; FREQ REACHED ORIGINAL (MIN) VALUE?
415  000464 05446                GOTO    KEYPR           ; NO!-CHECK FOR KEY CLOSURE.
416  000465 00147                CLRF    IO              ; YES!-SOUND OVER.
417  000466 05073                GOTO    KEYPAD
418                                                      ;
419  000467 05417          PAD    GOTO    ENG1            ; TIME PADS TO EQUALIZE PROG LOOPS.
420  000470 05424          PAD1   GOTO    ENG2
421  000471 05472          PAD2   GOTO    PAD2+1
422  000472 05473                GOTO    PAD2+2
423  000473 05474                GOTO    PAD2+3
424  000474 05475                GOTO    PAD2+4
425  000475 05476                GOTO    PAD2+5
426  000476 05477                GOTO    PAD2+6
427  000477 05412                GOTO    ENG
428                                                      ;
429                                                      ;
430                                                      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
431                                                      ;                                 ;
432                                                      ;TIRE SCREECH SOUND               ;
433                                                      ;                                 ;
434                                                      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
435                                                      ;
436  000500 06121          TSCRCH MOVLW  121
437  000501 00055                MOVWF   OUTBUF
438  000502 00047                MOVWF   IO
439  000503 06200          SCRCH1 MOVLW  200
440  000504 00655                XORWF   OUTBUF
441  000505 04405                CALL    RANGEN          ; OUTPUT RANDOM PULSES-WHITE NOISE.
442  000506 02003                CLRC
443  000507 03754                BTFSS   TEMP,7
444  000510 02403                SETC
445  000511 04400                CALL    RUMBLE
446  000512 01011                MOVF    SL,W
447  000513 07017                ANDLW   17
448  000514 06440                IORLW   40
449  000515 00054          WAIT   MOVWF   TEMP            ; RANDOM DELAY.
450  000516 01354          WALOOP DECFSZ TEMP
451  000517 05516                GOTO    WALOOP
452  000520 06373                MOVLW   373             ; CHECK IF KEY PRESSED?
453  000521 00046                MOVWF   6
454  000522 01005                MOVF    5,W
455  000523 00056                MOVWF   16
456  000524 03556                BTFSS   16,3
457  000525 05503                GOTO    SCRCH1          ; YES! CONTINUE SOUND.
458  000526 05073                GOTO    KEYPAD          ; NO! SOUND DONE.
459                                                      ;
460                                                      ;
461                                                      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
462                                                      ;                               ;
463                                                      ;CAR CRASH/EXPLOSION SOUND      ;
464                                                      ;                               ;
465                                                      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
466                                                      ;
467  000527 06225          CRASH  MOVLW  225
468  000530 00055                MOVWF   OUTBUF
469  000531 00047                MOVWF   IO
470  000532 04443                CALL    DECAY
471  000533 06377                MOVLW   377
472  000534 00076                MOVWF   36
473  000535 06155                MOVLW   155
474  000536 00077                MOVWF   37
475  000537 02155                BCF     OUTBUF,3        ; DISCHARGE ENVELOPE GEN. CAP. FOR FAST  R
476  000540 04405          CCONT  CALL    RANGEN          ; OUTPUT RANDOM PULSES-WHITE NOISE.
477  000541 03354                BTFSC   TEMP,7
478  000542 05546                GOTO    DOIT
479  000543 02003                CLRC
480  000544 02115                BCF     OUTBUF,2
481  000545 05550                GOTO    SOFF
482  000546 02403          DOIT   SETC
```

```
483  000547 02515                BSF    OUTBUF,2
484  000550 04400          SOFF  CALL   RUMBLE
485  000551 06002                MOVLW  2
486  000552 00054                MOVWF  TEMP
487  000553 01354          WLOOP DECFSZ TEMP          ; 24USEC DELAY.
488  000554 05553                GOTO   WLOOP
489  000555 02555                BSF    OUTBUF,3    ; START 10SEC DECAY ON SOUND OUTPUT.
490  000556 01376                DECFSZ 36
491  000557 05540                GOTO   CCONT
492  000560 01377                DECFSZ 37
493  000561 05540                GOTO   CCONT
494  000562 05073                GOTO   KEYPAD      ; DECAY OVER-SOUND DONE.
495                                                 ;
496                                                 ;
497                                                 ;#####################################
498                                                 ;                                   ;
499                                                 ;PHASOR GUN SOUND                   ;
500                                                 ;                                   ;
501                                                 ;#####################################
502                                                 ;
503  000563 06252          PHASOR MOVLW  252
504  000564 00047                MOVWF  7
505  000565 06030          L1    MOVLW  30          ; INITIAL FREQUENCY.
506  000566 00053                MOVWF  SFREQ       ;
507  000567 06002          L2    MOVLW  2           ; DURATION BEFORE DECR. FREQ.(# OF PULSES)
508  000570 00074                MOVWF  34
509  000571 06003          L3    MOVLW  3
510  000572 00647                XORWF  7           ; OUTPUT PULSES OF FREQUENCY->SFREQ.
511  000573 06001                MOVLW  1
512  000574 00057                MOVWF  TEMP2
513  000575 01013                MOVF   SFREQ,W
514  000576 00054                MOVWF  TEMP
515  000577 04417                CALL   DELAY
516  000600 01374                DECFSZ 34          ; DURATION FOR CURRENT FREQUENCY OVER?
517  000601 05571                GOTO   L3          ; NO!-CONTINUE AT SAME FREQUENCY.
518  000602 01253                INCF   SFREQ       ; YES!-INCREMENT COUNTER (DECREMENT FREQUE
519  000603 06200                MOVLW  200         ;
520  000604 00613                XORWF  SFREQ,W     ;
521  000605 03503                SKPZ               ; FREQUENCY REACHED MIN?
522  000606 05567                GOTO   L2          ; NO!-CONTINUE SOUND.
523  000607 05073                GOTO   KEYPAD      ; YES!-SOUND DONE.
524                                                 ;
525                                                 ;
526                                                 ;#################################
527                                                 ;                                 ;
528                                                 ;MORTAR WHISTLE & EXPLOSION       ;
529                                                 ;                                 ;
530                                                 ;#################################
531                                                 ;
532  000610 06252          WISTLE MOVLW  252
533  000611 00047                MOVWF  IO
534  000612 06030                MOVLW  30
535  000613 00053                MOVWF  SFREQ
536  000614 06100          LL1   MOVLW  100
537  000615 00074                MOVWF  34
538  000616 01013          LL2   MOVF   SFREQ,W
539  000617 04425                CALL   DEL4
540  000620 06003                MOVLW  3           ; OUTPUT  PULSES OF FREQUENCY->SFREQ.
541  000621 00647                XORWF  7
542  000622 01374                DECFSZ 34          ; DURATION FOR FREQUENCY OVER?
543  000623 05630                GOTO   LL3         ; NO! CONTINUE AT SAME FREQUENCY.
544  000624 01253                INCF   SFREQ       ; YES! INCREMENT COUNTER (DECREMENT FREQUE
545  000625 03753                BTFSS  SFREQ,7     ; FREQUENCY REACHED MINIMUM?
546  000626 05614                GOTO   LL1         ; NO! CONTINUE.
547  000627 05527                GOTO   CRASH       ; YES! SOUND DONE.
548  000630 00000          LL3   NOP                ; TIME PADS FOR EQUAL LOOP LENGTHS.
549  000631 05632                GOTO   LL3+2
550  000632 05633                GOTO   LL3+3
551  000633 05616                GOTO   LL2
552                                                 ;
553                                                 ;
554                                                 ;#######################################
555                                                 ;                                     ;
556                                                 ;TUNES-'WILD CHARGE' & 'SNAKE CHARM    ;
557                                                 ;-ERS SONG'                           ;
558                                                 ;                                     ;
559                                                 ;#######################################
560                                                 ;
561  000634 06252          TUNE  MOVLW  252         ; 'WILD CHARGE' TUNE.
562  000635 00047                MOVWF  IO
563  000636 00065                MOVWF  25
```

```
564  000637 02025                BCF    25,0          # CLEAR FLAG FOR 'WILD CHARGE' TUNE.
565  000640 06005                MOVLW  5
566  000641 00076                MOVWF  POINT         # POINTER TO TABLE FOR 'WILD CHRG' NOTE DA
567  000642 00376          STLOOP DECF   POINT
568  000643 01036                MOVF   POINT,W
569  000644 03025                BTFSC  25,0          # WHICH TUNE?
570  000645 05650                GOTO   U             #
571  000646 04451                CALL   PLAY          # GET NOTE DATA FROM TABLE FOR WC TUNE
572  000647 05651                GOTO   U1
573  000650 04457          U     CALL   PLAY1         # GET NOTE DATA FROM TABLE FOR SCS TUNE
574  000651 00072          U1    MOVWF  32
575  000652 01572                RLF    32            # DECODE NOTE DATA.
576  000653 03465                BTFSS  25,1          # CHECK IF LAST NOTE.
577  000654 05657                GOTO   J1            # LAST NOTE ALREADY DECODED- SKIP.
578  000655 00177                CLRF   37
579  000656 01577                RLF    37
580  000657 02003          J1    CLRC
581  000660 01472                RRF    32            # F32 HAS NOTE FREQUENCY.
582  000661 01277                INCF   37            # F37 HAS NOTE DURATION.
583  000662 06240          NN1   MOVLW  240
584  000663 00067                MOVWF  27
585  000664 01032          NN2   MOVF   32,W
586  000665 00066                MOVWF  26
587  000666 06001                MOVLW  1
588  000667 00647                XORWF  IO
589  000670 01366          NN3   DECFSZ 26
590  000671 05670                GOTO   NN3
591  000672 01367                DECFSZ 27
592  000673 05664                GOTO   NN2
593  000674 01377                DECFSZ 37
594  000675 05662                GOTO   NN1
595  000676 01076          SNDN  TSTF   POINT         # POINTER AT LAST NOTE?
596  000677 03103                SKPNZ
597  000700 05073                GOTO   KEYPAD        # YES! TUNE DONE.
598  000701 01036                MOVF   POINT,W
599  000702 07401                XORLW  1             # POINTER AT SECOND LAST NOTE?
600  000703 03503                SKPZ
601  000704 05642                GOTO   STLOOP        # NO! OUTPUT NEXT NOTE.
602  000705 06003                MOVLW  3             # YES! LAST NOTE DURATION 3.
603  000706 00077                MOVWF  37
604  000707 02065                BCF    25,1          # CLEAR LAST NOTE FLAG.
605  000710 05642                GOTO   STLOOP        # OUTPUT LAST NOTE.
606                                                   #
607  000711 06252          TUNE1 MOVLW  252           # 'SNAKE CHARMERS SONG'
608  000712 00047                MOVWF  IO
609  000713 00065                MOVWF  25
610  000714 02425                BSF    25,0          # SET FLAG FOR 'SNAKE CHARMER' TUNE.
611  000715 06013                MOVLW  .11           # SET POINTER TO TABLE FOR 'SNKE CHMR' NOT
612  000716 05641                GOTO   STLOOP-1
613                                                   #
614                                                   #
615                              ORG    777
616  000777 05073                GOTO   KEYPAD
617                                                   #
618                                                   #
619  001000                      END

 ASSEMBLER ERRORS =     0
```

```
                        SYMBOL TABLE

A       000332   AUTGR   000373   CCONT   000540   CHNGE   000344
CRASH   000527   DECAY   000043   DEL4    000025   DELAY   000017
DLY     000211   DLYDN   000222   DOIT    000546   ENG     000412
ENG1    000417   ENG2    000424   FREQ    000027   GUN     000167
GUN1    000175   GUN2    000177   GUN3    000202   HOLDN   000032
IN      000005   INBUF   000016   INRICO  000227   IO      000007
J1      000657   KEY1    000101   KEYPAD  000073   KEYPR   000446
KEYPR1  000456   L1      000565   L2      000567   L3      000571
LL1     000614   LL2     000616   LL3     000630   MCGN    000150
MCGN1   000161   MCGN2   000154   NEGWAY  000366   NN1     000662
NN2     000664   NN3     000670   NOP1    000253   OFFSET  000026
OUT     000006   OUTBUF  000015   PAD     000467   PAD1    000470
PAD2    000471   PHASOR  000563   PHSR1   000276   PLAY    000051
PLAY1   000057   POINT   000036   RANGEN  000005   REVDN   000460
REVUP   000437   RICO1   000235   RICO2   000241   ROTAT   000143
RPT     000355   RUMBLE  000000   SCONT   000301   SCRCH1  000503
SFREQ   000013   SH      000012   SIREN   000262   SIREN1  000272
SL      000011   SLOOP   000310   SNDN    000676   SNDTBL  000123
SOFF    000550   STLOOP  000642   SWITCH  000025   TEMP    000014
TEMP2   000017   TEMPH   000033   TONE    000031   TSCRCH  000500
TUNE    000634   TUNE1   000711   U       000650   U1      000651
VTL     000257   VTL1    000032   VTL2    000037   VTL3    000040
WAIT    000515   WALOOP  000516   WAY     000030   WISTLE  000610
WLOOP   000553   WORK    000034   WORK1   000035

EOF:676
0:>
```

7.4 Frequency Locked Loop Tuning with a PIC Microcomputer

INTRODUCTION

Tuning of AM/FM radios and televisions has evolved in the past ten years from manually varying inductances and capacitors to injecting a precise DC voltage on a varactor controlled tuner. Although the mechanics of tuning has changed, the theory of varying the RF mixer oscillator frequency remains the same.

The varactor tuner offers the advantage over conventional tuners by eliminating mechanical slugs, contacts and ganged condensers from the system and replacing them with a voltage controlled oscillator, specifically a varactor oscillator. This improves system reliability by removing the mechanical devices and gives system flexibility by a variety of ways to control the oscillator.

Up until now, most varactor tuners were either controlled in an open loop configuration (via a DC voltage generated from a D/A conversion) or with a closed loop Phase Locked Loop (PLL) circuitry. The open loop system does not compensate for frequency drift in a receiver system caused by components and by temperature changes. The closed loop PLL system has the disadvantages of being inherently noisy due to continuous voltage corrections (usually at a 2.5KHz rate) and costly utilizing many components.

General Instrument has devised a way of using its standard PIC series microcomputer as a controller for the varactor tuner in a "Frequency Locked Loop" (FLL) configuration. Due to the unique architecture and characteristics of the PIC, it performs the function of a frequency comparator and adjusts the DC control voltage out of a charge pump chip to the varactor tuner to maintain the desired frequency. The PIC, being a programmable microcomputer, is not only capable of performing FLL tuning, but can also do other tasks to further reduce system costs. These additional tasks include keyboard decoding, direct LED drive, band switching, remote control decoding, On/Off control, audio amplifier muting, volume control and storage of favorite stations in external memory. The FLL program can be included with various other program options which customize the system features to the manufacturer's needs and requirements.

THEORY OF OPERATION

The FLL program designates two I/O pins as outputs to drive a charge pump. The charge pump output is filtered and delivers a DC control voltage to the varactor-controlled local oscillator in the tuner whose frequency will vary according to the control voltage.

To close the control loop, the local oscillator frequency is divided down to a suitable comparison frequency by a prescaler, and in input to the PIC microcomputer through the Real Time Clock Counter (RTCC) pin.

Inside the PIC, the frequency on the RTCC pin is measured and compared to the desired frequency generated by the microcomputer program. The outputs to the charge pump adjust the DC control voltage up or down until the local oscillator's frequency matches the desired frequency.

The PIC microcomputer continuously checks for frequency drift and makes corrections as necessary to hold a station locked in until another station frequency is selected.

The basic concept of FLL can be used in various types of RF tuned receivers.

The additional features that can be programmed into a television receiver are:

1. Favorite (local) channel storage
2. Favorite channel scan up and down
3. Direct channel entry
4. Automatic volume mute during channel change
5. Remote and local On/Off control
6. Remote and local volume control
7. Remote and local channel selection

HARDWARE REQUIREMENTS OF FLL

As shown in Figure 33, the total hardware requirements consist of a prescaler, an optional non-volatile ROM (ER2055), a PIC1650A microcomputer, and a charge pump (CT2017). The memory, microcomputer and charge pump are all integrated circuits available from General Instrument.

Figure 34 shows a typical hardware comparison of FLL to PLL systems. A considerable savings can be seen here.

SOFTWARE OPERATION

In its most basic form, the PIC operates as a counter with a gate time of 128 msecs. It obtains a count of the local oscillator frequency and compares it with an expected count for the particular station. If the actual count is different from the expected count, it charges or discharges the voltage on a capacitor which corrects the frequency error of the varactor tuned local oscillator.

In order to respond quickly to a station change requested from a local or remote keyboard, this 128 msec loop is broken down into 4 loops consisting of 16 msec, another 16 msec, 32 msec and 64 msec; ie, ⅛th, ⅛th, ¼th, ½ and full counts will be obtained at the end of the above loops.

For instance, in the first 16 msec loop, the count obtained is multiplied by 8 and then compared with the expected count. If there is a significant error, it will be corrected right away. If there is no significant error, the PIC will accumulate pulses for another 16 msec and add it to the previous count. This total count of 32 msecs is then multiplied by 4 and again compared with the expected count, and so on. If there is no error at the end of 128 msec the process will start over again.

The program scans the keyboard every time it makes a correction, as well as at the end of the 128 msec loop. The maximum time the keyboard is sensed is 128 msecs and the minimum time is 16 msecs.

The display whether static or directly driven is updated every time a station change is made. The remote control input is looked at during the count loop. When a valid "start" code is received, the program leaves the count loop and receives the rest of the code, decodes it and takes action. It then returns to the tuning control loop.

The PIC also does the band switching, on/off control, volume control, muting and memory control functions.

CONCLUSION

In concluding the Frequency Locked Loop configuration, the PIC offers an economical tuning controller which can be used in TV receivers, cablevision converters and video recorder front ends. It offers quality performance with a minimum number of parts and a low control system cost.

Fig. 33 BASIC FLL BLOCK DIAGRAM

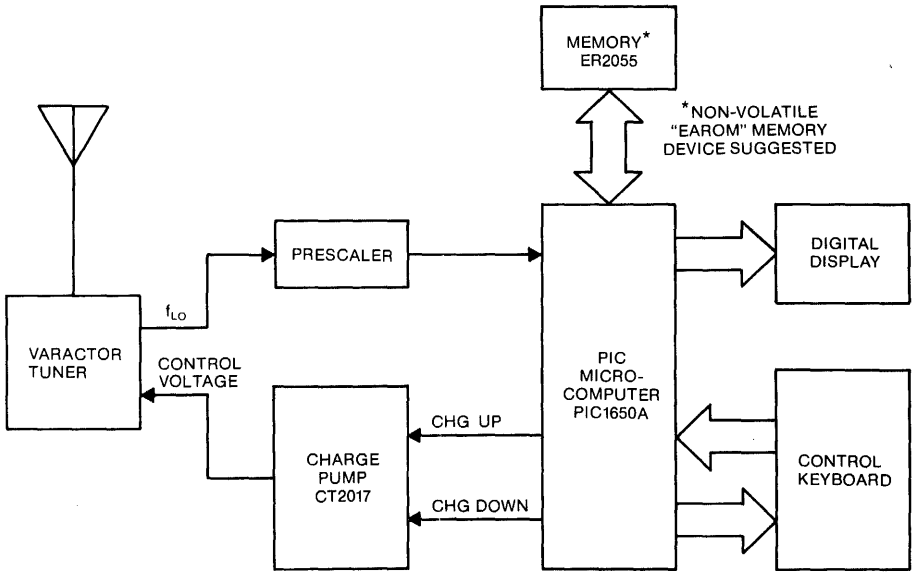


Fig. 34 TYPICAL HARDWARE COMPARISON—PHASE LOCKED LOOP VS FREQUENCY LOCKED LOOP

	PLL SYSTEM	FLL SYSTEM
MICROCOMPUTER	1	1
MEMORY	1	1
FREQUENCY SYNTHESIZER	1	NONE
CRYSTALS	2	1
TRANSISTORS	26	8
CAPACITORS	57	26
RESISTORS	104	35
DISPLAY DRIVER	2	NONE
KEYBOARD MULTIPLEXER	1 or 2	NONE
TRIMMER RESISTORS	1	NONE
TRIMMER CAPACITORS	1	NONE
EXTRA TTL		

GENERAL FLL TUNING FLOWCHART

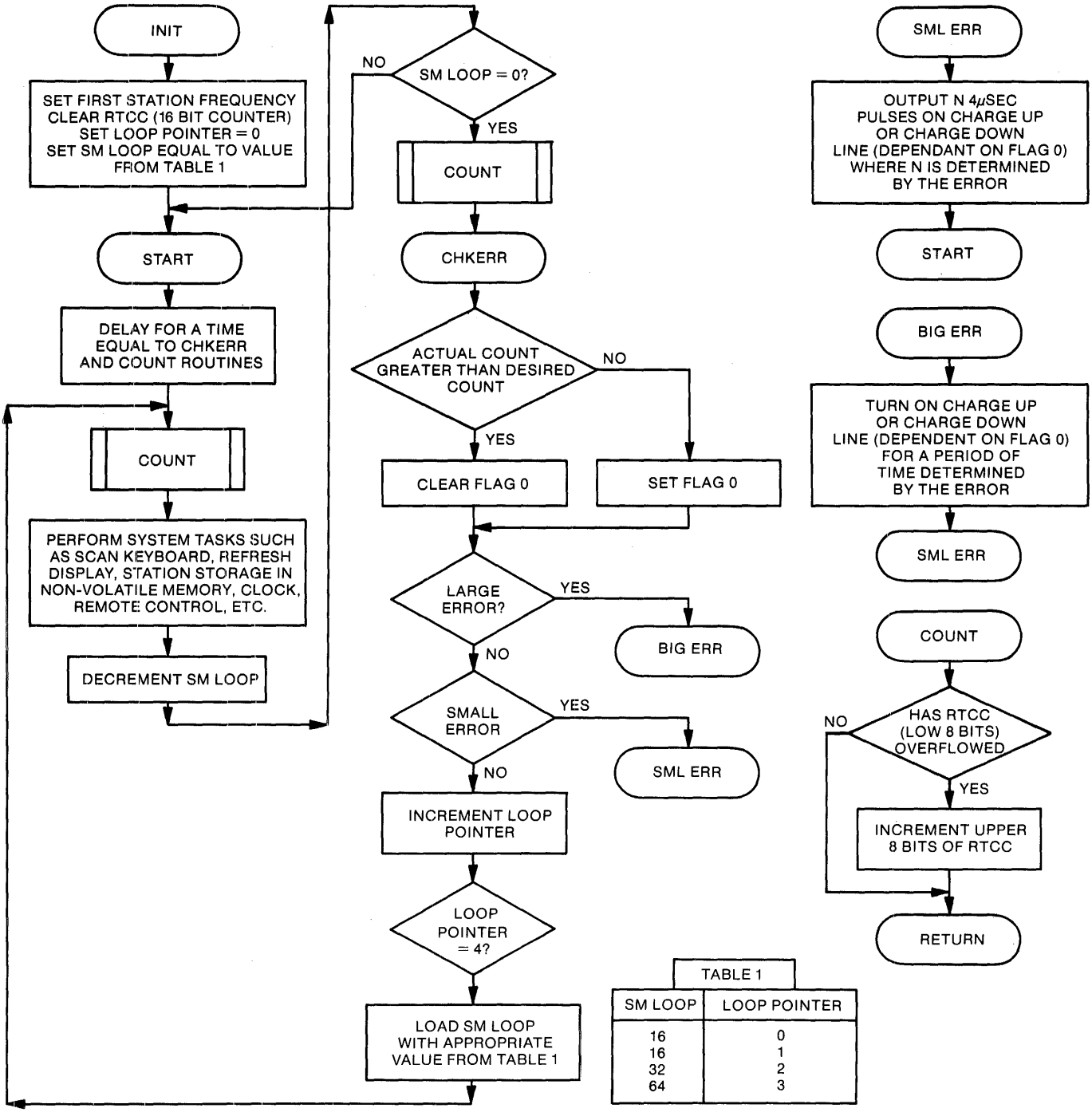
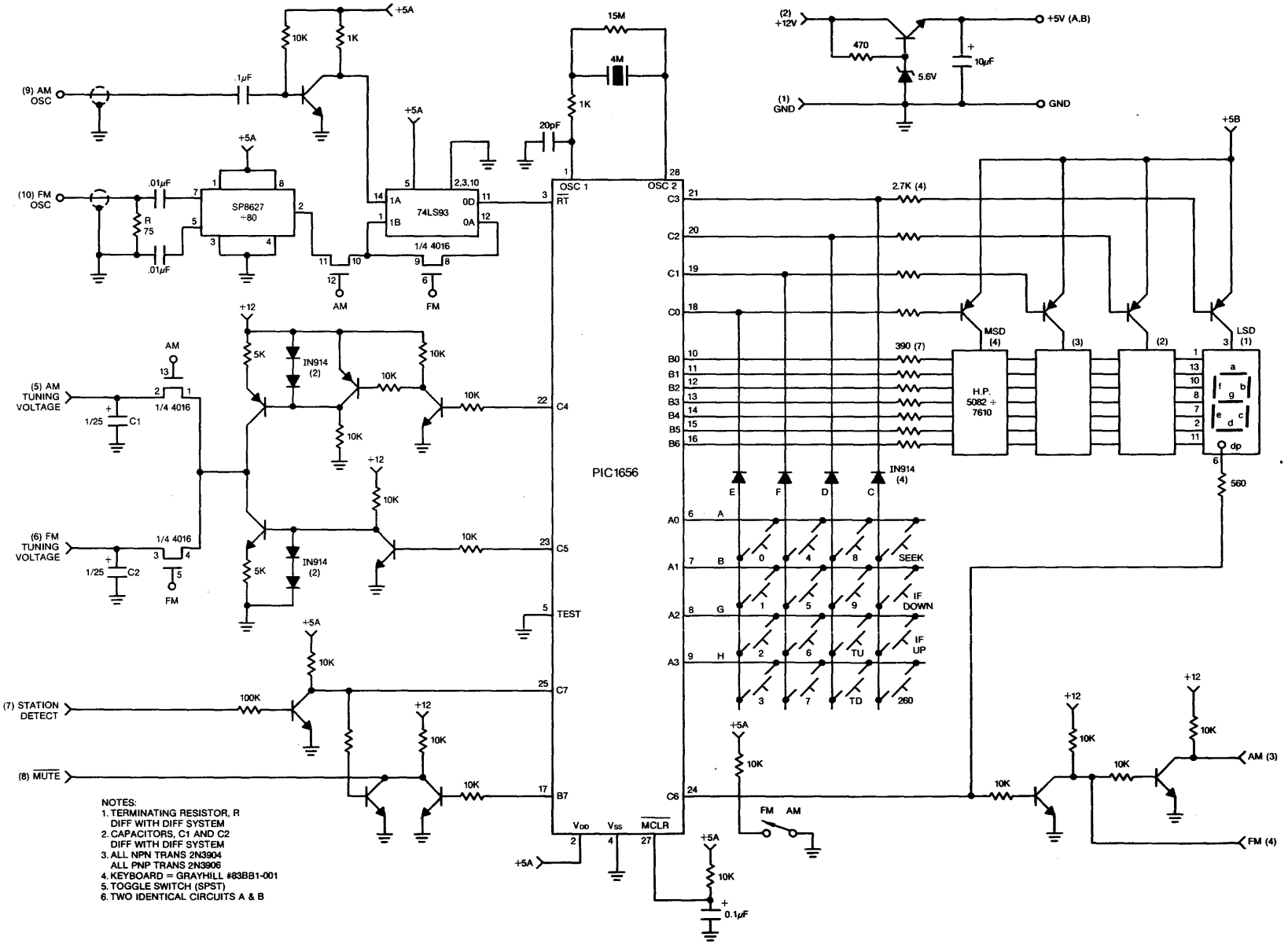


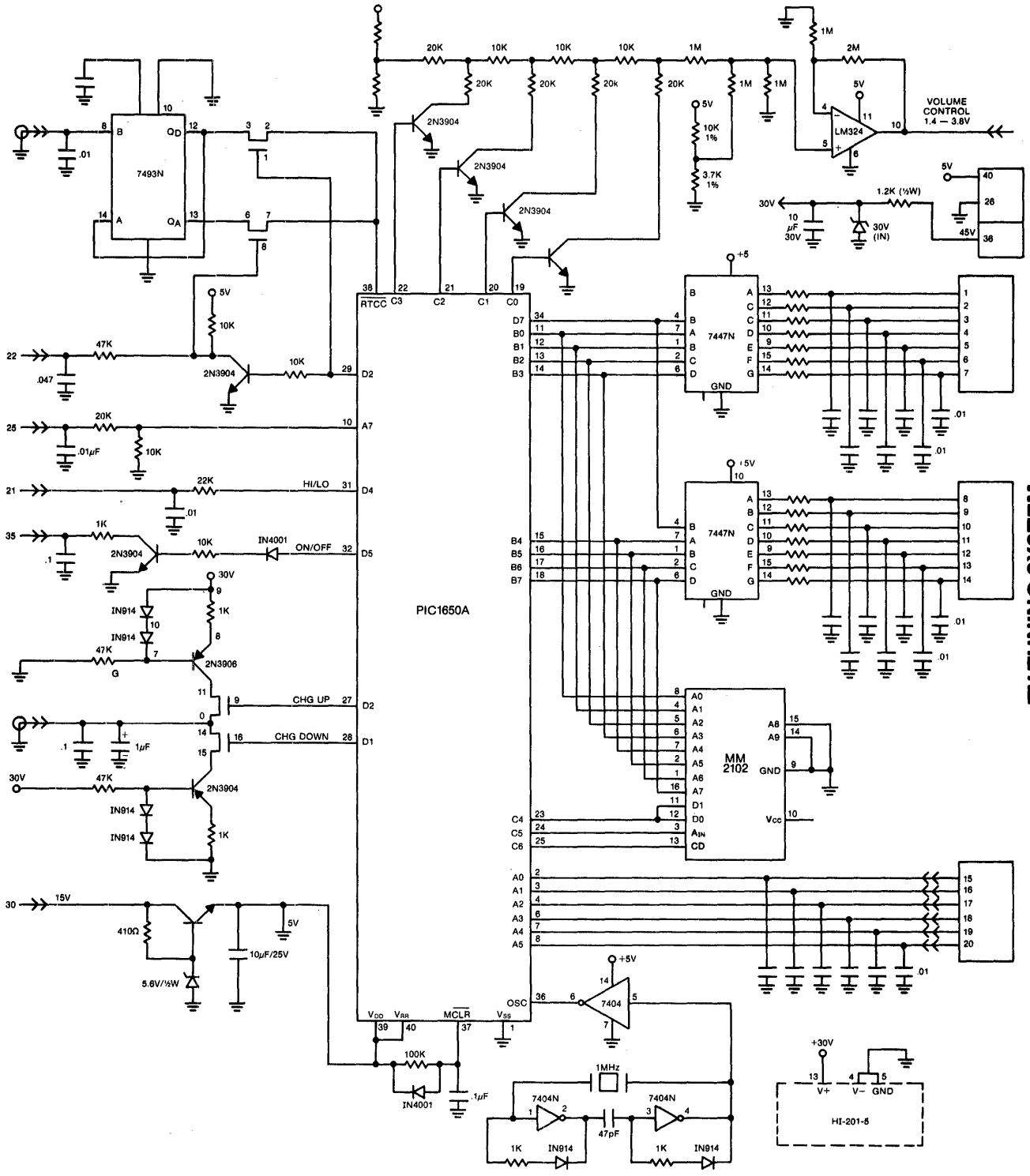
TABLE 1

SM LOOP	LOOP POINTER
16	0
16	1
32	2
64	3



NOTES:
1. TERMINATING RESISTOR, R DIFF WITH DIFF SYSTEM
2. CAPACITORS, C1 AND C2 DIFF WITH DIFF SYSTEM
3. ALL NPN TRANS 2N3904 ALL PNP TRANS 2N3906
4. KEYBOARD = GRAYHILL #83BB1-001
5. TOGGLE SWITCH (SPST)
6. TWO IDENTICAL CIRCUITS A & B

RADIO TUNING SYSTEM


TV TUNING SYSTEM
PIC1650A
7493N
7447N
MM 2102
LM324
2N3904
2N3906
IN914
IN4001
7404
7404N
HI-201-5
VOLUME CONTROL 1.4 — 3.8V
CHG UP
CHG DOWN
HI/LO
ON/OFF
1MHz
+30V
GND

7.5 PIC Microcomputers in Subscriber End Equipment

INTRODUCTION

Single chip microcomputers have become the standard circuit module of the 1980's. In this paper, General Instrument PIC series microcomputers will be reviewed and will be shown where and how they can be used to provide cost effective solutions in the design of telecommunications systems.

PIC Series Microcomputers

The PIC series of microcomputers are MOS/LSI circuit arrays containing a central processing unit, RAM, I/O and customer defined ROM on a single chip.

The power and versatility of the design combined with the low cost afforded by mass production and the use of proven technology has made this LSI family among the best selling 8 bit microcomputers.

The architecture of PIC microcomputers is register oriented optimized to perform control oriented tasks.

Internally, PIC microcomputers contain 5 functional blocks connected by a single 8 bit bidirectional bus:

1. Register files divided into two functional groups: Special Registers and General Purpose Registers. The Special Purpose Registers include:
   - ☐ Real Time Clock/Counters
   - ☐ Status Registers
   - ☐ Program Counter
   - ☐ I/O Registers
   - ☐ File Select Registers (Used to indirectly address any register.)

   Any bit, nibble or byte in the register files can be tested or modified under program control.
2. Arithmetic logic unit and working register (W) that provide full complement of arithmetic and logic operations.
3. Program ROM containing the user defined application program, supported by an instruction decoder and instruction register.
4. Multilevel stack used for subroutine and interrupt nesting.
5. Interrupt logic allowing external and real time clock counter vectored interrupts.

In addition, a PLA and on chip oscillator are used to provide instruction decoding and generation of timing and control signals.

Overlapping of the fetch and execution cycles, or pipelining, permits PIC to execute each of its instructions in a single clock cycle.

The instruction set of PIC microcomputers is compact, but very powerful. Each one of the instructions is contained in a 12 bit (13-bit PIC1670) wide single line of ROM. This width permits complete operands that can address all PIC file registers and there is no need for a second trailing line of code (very often required to complete the operand in other microcomputers such as 8048 or 3870), which takes ROM space and increases the execution time.

Microcomputer Controlled Voice Switched Speakerphone and Repertory Dialer

The speakerphone is an instrument that offers hands free telephony by means of replacing the usual telephone handset with separate loudspeaker and microphone. In order to compensate for the loss introduced by moving the handset away from the user's head, gain is inserted in the transmitting and receiving channels. This gain, however, is limited by a problem known as "singing." A signal from the microphone reaches the loudspeaker traveling through the transmitter channel, the sidetone path and the receiving channel. From the loudspeaker, this signal comes back to the microphone through the acoustic coupling of the room thus creating a closed loop (Figure 35). If the total gain within the loop is greater than or equal to 0dB, oscillation ("singing") will occur. Another unpleasant effect is caused by the acoustic coupling of the microphone and the loudspeaker in the form of an "echo" noticed at the end of the distant party. Standard telephones are usually held close to the user's head and are not affected by the acoustic properties of the room and the ambient noise level, on the contrary the performance of the speakerphone is severely limited by them.

The common solution for these limitations is to allow transmission in only one direction or voice switching. Figure 36 shows a block diagram of a voice switched speakerphone. A microphone preamplifier and a power amplifier provide the desired gains within the transmit and the receive channels respectively. A hybrid network interfaces the speakerphone to the telephone line. Two variable attenuators are incorporated, one in the transmit and one in the receive channel. The decision making unit within the speakerphone is the control unit. It compares the signal levels in the transmit and receive channels and by acting on the variable attenuators, decides the transmission direction. Obviously, the quality of the transmission through a speakerphone is a function of the intelligence of its control circuit. There are only a few high quality speakerphones available presently and all of them use highly complex analog type control circuits. Some ingenious circuits have been designed in order to minimize such problems as false switching due to high ambient noise levels, clipping due to finite switching time, etc.

With the cost of computing and control power steadily decreasing, it becomes feasible to incorporate a single chip microcomputer in the control circuit of a speakerphone. Figure 37 shows a block diagram of a microcomputer controlled speakerphone. Its building blocks can be defined as follows:

■ **Digitally Programmable Transmit and Receive Attenuators.** The loss of the attenuators is controlled by a digital binary word. For example, five bit word can provide dynamic range of 0 to 31dB at a 1dB increment. The advantages offered by these types of attenuators are: ease in generation of the loss-time curves; maintenance of constant gain within the speakerphone loop by inverse tracking of the transmit and the receive attenuators' losses; and implementation of automatic gain control.

■ **Level Sensing Circuit.** Its role is to monitor the voltage levels at the inputs of the transmit and receive channels and to convert them in a digital binary form for use by the microcomputer.

■ **Microcomputer.** It provides the necessary intelligence to the control circuit of the speakerphone. Inputs from the level sensing circuit are taken by the microcomputer and are used as a base for generating outputs to the programmable attenuators. The amount of intelligence packed within the microcomputer depends on the algorithm used by the designer and is no longer a function of the circuit complexity.

The presence of a microcomputer in a speakerphone gives the designer an opportunity to add to it repertory dialing capabilities. Figure 38 shows a block diagram of a repertory dialer in addition to a speakerphone. Some of the features that such an addition can provide are as follows:

- ☐ Display showing the number being dialed
- ☐ DTMF or pulse dialing
- ☐ Non volatile repertory storage by using EAROM (General Instrument ER3400)
- ☐ Digital clock and interval timer
- ☐ Automatic redial of busy numbers

Microcomputer Based Multiline Telephone Instruments for Use in Electronic Key Telephone Systems

A Key Telephone System (KTS) is an arrangement of multiline telephone station apparatus and associated equipment which allows a user to selectively answer, originate, or hold calls over a specific central office, PABX or other line facilities.

Key Telephone Systems on the market, until recently, have enjoyed a high degree of industry standardization, whereby, many subsystems such as instruments and line cards have been interchangeable, regardless of the origin of manufacture. During the 1970's a number of new Key Telephone Systems using electronic and digital techniques were introduced. These systems are of a design unique to each manufacturer, thus digressing from the principle of standardization. The use of advanced electronic and digital technology made possible the introduction of proprietary instruments with multiline capability utilizing drastically reduced cabling, thus overcoming the inherent disadvantages of the old Electromechanical Key Telephone Systems which require many wire pairs to interconnect each instrument. In addition, the Electronic Key Systems offer many features previously provided only by PBXs.

Figure 39 shows a block diagram of an Electronic Key System. A common control unit interfaces a number of electronic multiline instruments to the central office, PABX, or other line facilities. Three wire pairs connect each instrument to the common control unit. One of the wire pairs provides power to the instrument and the other two are used as serial data and voice links. The common control unit scans the instrument through the serial data links interrogating them about the status of their keys and hook switches and supplying appropriate sets

with the new status of their lamp fields and ringers. The electronic multiline instrument provides the user with a standard talking path, a nonlocking key field used to access individual lines or features, a lamp field indicating the status of the line or feature select keys and an electronic tone ringer. Obviously, complex logic circuitry is required within the electronic multiline instrument in order to perform those functions. A cost-effective solution in this case can be provided by a single chip microcomputer.

Figure 40 shows a single chip microcomputer based multiline telephone instrument. Standard 500 type speech network terminates the voice wire pair. A power amplifier/loudspeaker is added to enable paging and receive only conferences. Data transceivers interface the instrument to the serial data link, thus providing data communication over a single wire pair. Power to the instrument is supplied over a separate wire pair. The microcomputer is the main logic component of the instrument. The software contained in its program memory performs the following functions:

- **LED Lamp Field Control.** Part of the data memory of the microcomputer holds the status of the lamp field with binary 0 and 1 indicating off/on condition for each separate lamp respectively. This information is supplied to the lamp field through the microcomputer I/O periodically, thus saving power and improving the brightness of the LED's.

- **Key Field Scan and Encoding.** The key field of the instrument is arranged in a form of matrix and directly interfaces with the I/O of the microcomputer. Periodic scan of the key field detects key closures and enables key debouncing and encoding. The encoded version of each key closure is stored in a temporary location in the data memory of the microcomputer.

- **Serial Data Communication.** Asynchronous serial data communication enables the multiline electronic instrument to communicate with the common control unit. The common control unit periodically sends commands to the instrument instructing it to change the status of the lamp field, initiate ringing, or connect/disconnect the receive only power amplifier/loudspeaker to the voice wire pair. The instrument then responds by transmitting the encoded version of any key closure that has occurred and the status of the hook switch. Two single bit microcomputer I/O ports are used as receive and transmit ports. Timing, decoding, and encoding of the serial data is performed by the microcomputer. Any command after being received and decoded is acted upon by changing the contents of the microcomputer's data memory allocated for lamp field status, ring generation, or by performing other specified tasks.

- **Ring Generation.** A piezoelectric transducer can be used as a ringer. In such case the microcomputer controls the volume, pitch, and interruption rate of the ringer.

- **Hook Switch Sense and Power Amplifier/Loudspeaker Actuation.** Two single bit microcomputer I/O ports are dedicated to sense the status of the hook switch (up/down) and actuate the receive only power amplifier (on/off).

Some hardware external to the microcomputer is required in order to achieve the functions described above but will not be discussed here.

Conclusion

Single chip microcomputers are versatile parts and their widespread use in telecommunication systems is imminent. The intention of this paper was to review briefly one of the popular microcomputer families and show a few of its many possible applications.

Fig. 35

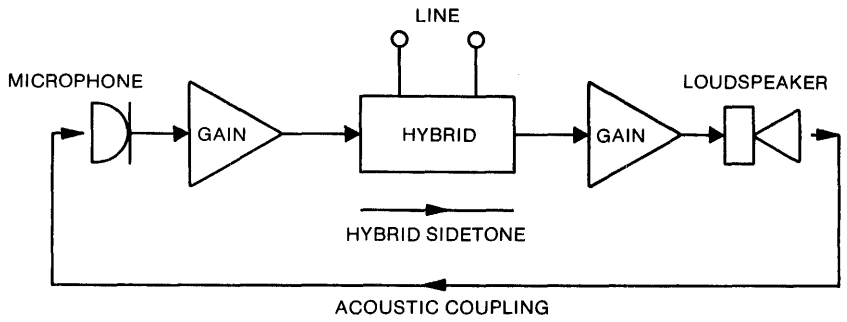


Fig. 36

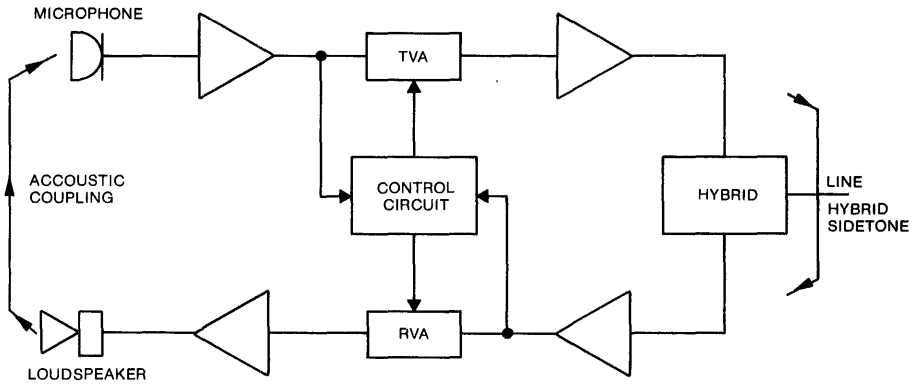


Fig. 37

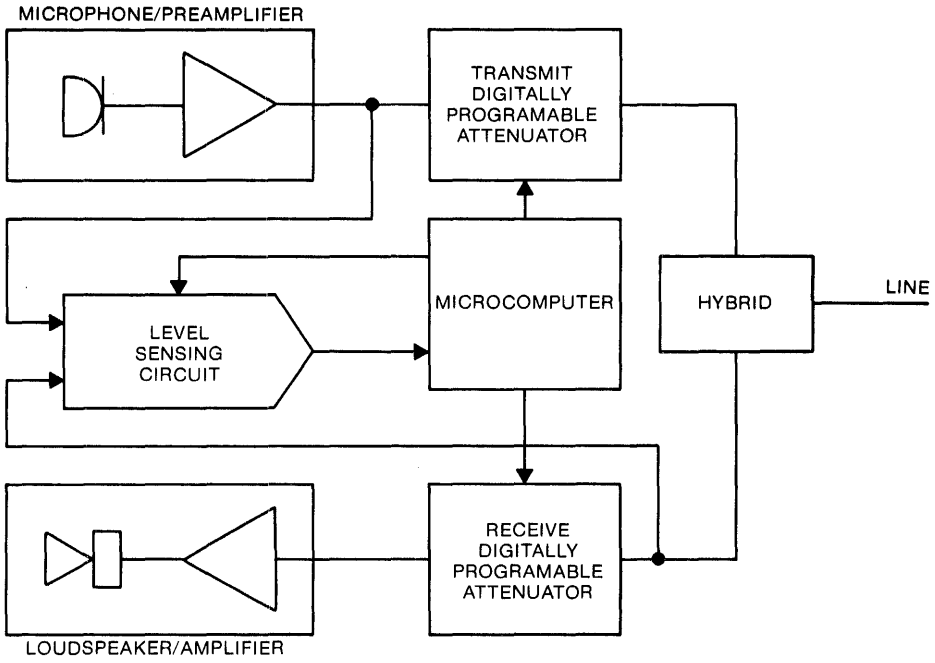


Fig. 38

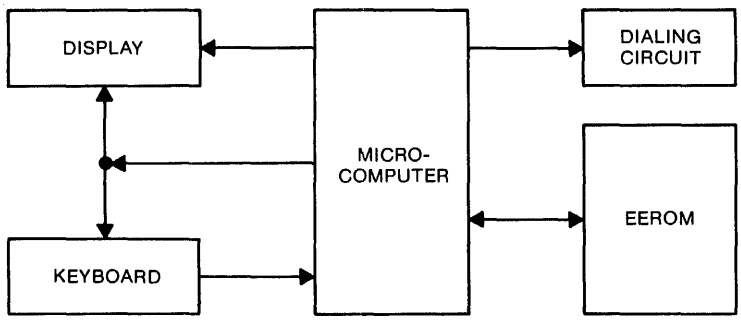

DISPLAY
MICRO-
COMPUTER
DIALING
CIRCUIT
EEROM
KEYBOARD

Fig. 39

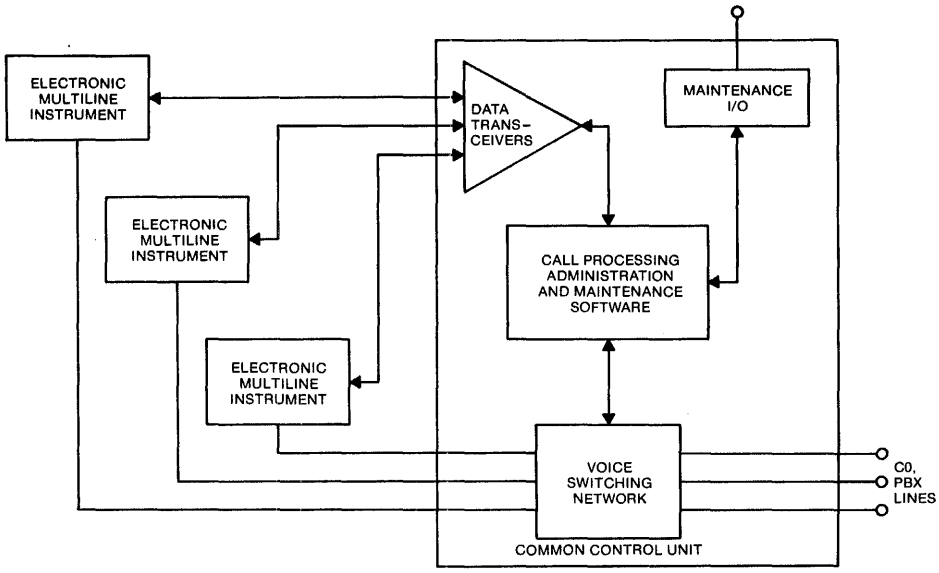
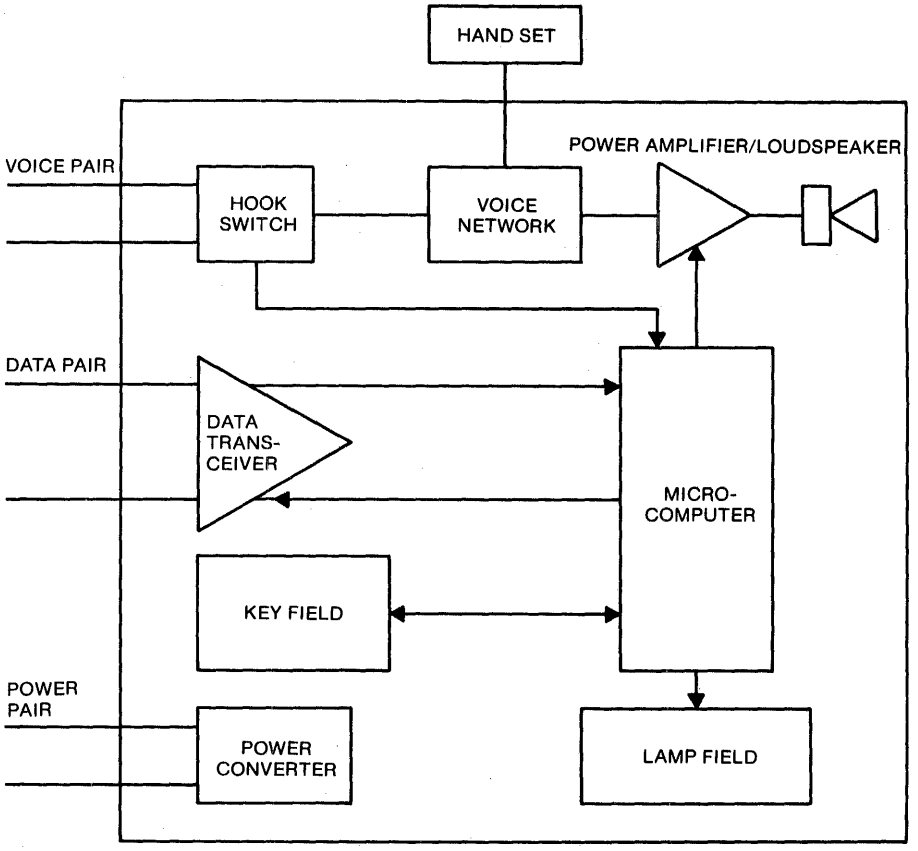

ELECTRONIC
MULTILINE
INSTRUMENT
DATA
TRANS-
CEIVERS
MAINTENANCE
I/O
ELECTRONIC
MULTILINE
INSTRUMENT
CALL PROCESSING
ADMINISTRATION
AND MAINTENANCE
SOFTWARE
ELECTRONIC
MULTILINE
INSTRUMENT
VOICE
SWITCHING
NETWORK
CO,
PBX
LINES
COMMON CONTROL UNIT

Fig. 40


HAND SET
POWER AMPLIFIER/LOUDSPEAKER
VOICE PAIR
HOOK SWITCH
VOICE NETWORK
DATA PAIR
DATA TRANS-CEIVER
MICRO-COMPUTER
KEY FIELD
POWER PAIR
POWER CONVERTER
LAMP FIELD

7.6 PIC Microcomputer-Based Control Smoothes Universal Motor Performance

Universal motors, so-called because they can run on either an alternating or a direct current, are widely used in vacuum cleaners, blenders, power tools, sewing machines, and other consumer appliances that need to operate at varying speeds. These motors supply high horsepower relative to their weight and size, easy speed control, high starting torque, and economical operation. But they also demand high starting current, generate a lot of noise, overheat at low speed, and suffer from inherently poor speed regulation as well as poor efficiency when the load is variable.

A microprocessor-based closed-loop motor controller (Figure 41) reduces or eliminates these disadvantages. Being less costly and more reliable than a closed loop built with discrete devices, it is practical for a great many more consumer applications. It is also a cost-effective means of adding several desirable operating features.

For instance, the input speed of a power tool may now be set through a digital keypad or potentiometer. (In the latter case, the microcomputer converts the analog input into digital form before setting tool speed.) Moreover, microprocessor-controlled automatic current limiting enhances the reliability and life of the universal motor, replacing the passive components that generally keep its starting and overload currents to levels that are safe for its brushes, on-off switch, and owner's housewiring. In addition, such current limiting protects the motor from overheating.

Open Versus Closed Loop

With a constant voltage input, the load that a universal motor must move determines its speed. But as Figure 42 shows, the speed-torque curve that describes this open-loop relationship (solid black line) is highly nonlinear, and it remains just as nonlinear throughout any change in driving current used to shift it (dashed black line) and thus alter motor speed. Moreover, full torque is not available at lower speeds in any case.

The operating curve for a motor with closed-loop speed control is entirely different. Now the speed remains almost constant under a variable load (nearly horizontal solid colored line) so long as the peak load does not exceed the available torque.

It is worth noting at this point that a universal motor with a closed-loop control and a variable load draws less current as a function of torque (colored dotted and dashed line) than does one without such a control (black dotted and dashed line). This not only saves power but also reduces the amount of audible noise because, when a motor uses less current, it is slower and therefore less noisy—and what is more, interferes less with its user's television reception.

A microcomputer-based implementation of such a closed loop requires only a few external components, including a speed pickup, a triac, and a power supply (see Figure 41). It assumes ac, not dc, operation of the universal motor.

A typical speed pickup might consist of a 20-pole magnetic disk and a Hall-effect sensor. Such an arrangement would feed back 10 pulses per

motor revolution to the microcomputer, since a high-resolution input is necessary if the loop is to have refined control over its output to the triac.

Triac Triggering

The loop triggers the triac at varying times after the ac reference signal's zero crossing. This variable firing angle in turn varies the power delivered to the motor by setting the average current fed to the series windings. Typically the triac is rated at 6 to 15 amperes and drives a motor of 0.5 to 2 horsepower.

The user's input to the loop may be made through a keypad and display, incorporated in it with the addition of a few extra components as shown in the figure. This keypad can be scanned and the display multiplexed at up to a 250KHz rate by the microcomputer—a more-than-adequate rate for consumer applications.

In operation, the microprocessor continually compares the speed set by the user with the speed measured by the Hall-effect pickup and then adjusts the power delivered to the motor to minimize any error in performance.

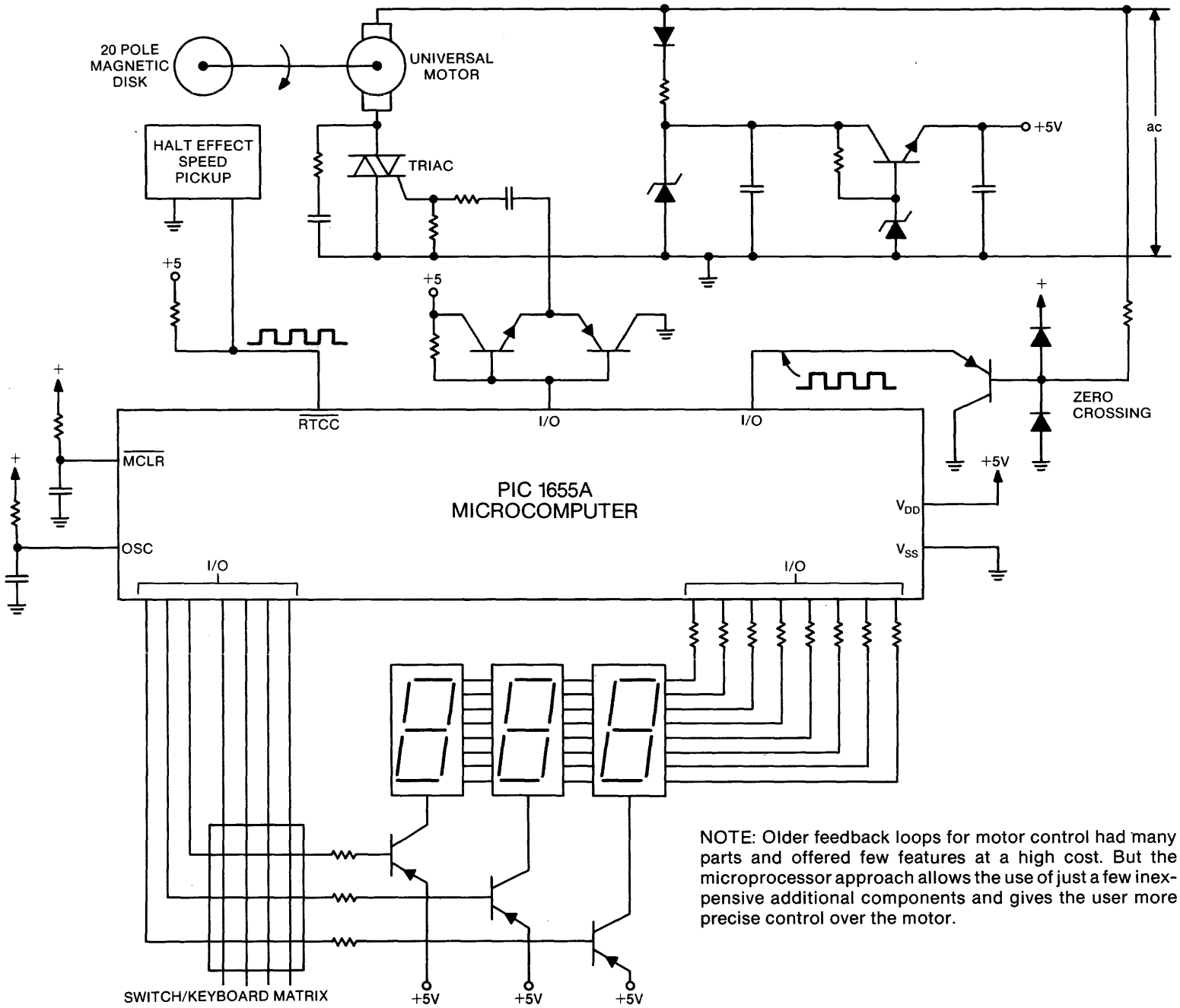
For instance, in a blender application, the desired motor speed and run time would be entered by the user, and the microcomputer would then send the triac the pulses appropriate for applying a steadily rising current to the motor until it reached the desired speed. In larger appliances, of course, this "soft" start would limit the typically very large initial surge currents of the universal motor, thus safeguarding switches and wiring.

Moreover, current limiting of the universal motor is readily achieved by limiting the firing angle of the drive triac as a function of the maximum speed desired. In essence, the maximum allowable number of pulses from the speed pickup in a given period of time is made to determine the maximum firing angle.

The operating characteristic of the motor is then modified to follow the solid vertical colored line of Figure 42 in an overload condition. (It is to be noted that * on the colored dotted and dashed current curve corresponds to this limit.)

This principle can be extended to protect the motor from overheating when it is being forced by heavy loading to run at low speed. A simple timer incorporated into the control loop just rolls back the current to a safe limit after a predetermined time (indicated by the colored dotted line in Figure 42).

In sum, then, the operation of the universal motor is limited to the horizontal solid colored line of Figure 42 for various loads until the overload condition is reached. Then its speed drops while a constant current is maintained along the vertical line. In this condition, the motor is overheating, and after a period of time predetermined by the microcomputer, the current rollback feature moves the load line back to the dotted line in the figure. When the load is reduced, the operating point will move up the dotted line to the horizontal one and into the normal region.



NOTE: Older feedback loops for motor control had many parts and offered few features at a high cost. But the microprocessor approach allows the use of just a few inexpensive additional components and gives the user more precise control over the motor.

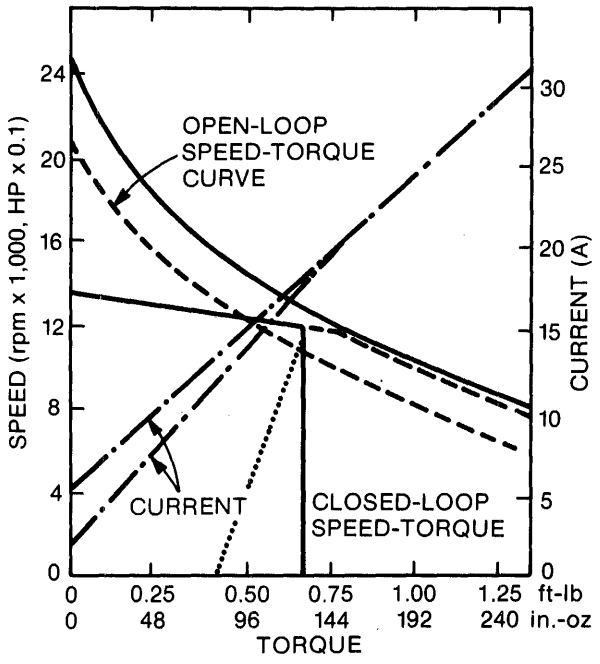
Fig. 41 CLOSE THE LOOP

Firing Angle Control

Universal motor torque is a nonlinear function of firing angle and speed (Figure 43a). In order to linearize it, so that a speed variation produces a corresponding change in torque, the deviation of the actual from the set speed—the speed error—must be mapped into the phase angle, which can then be used to adjust matters.

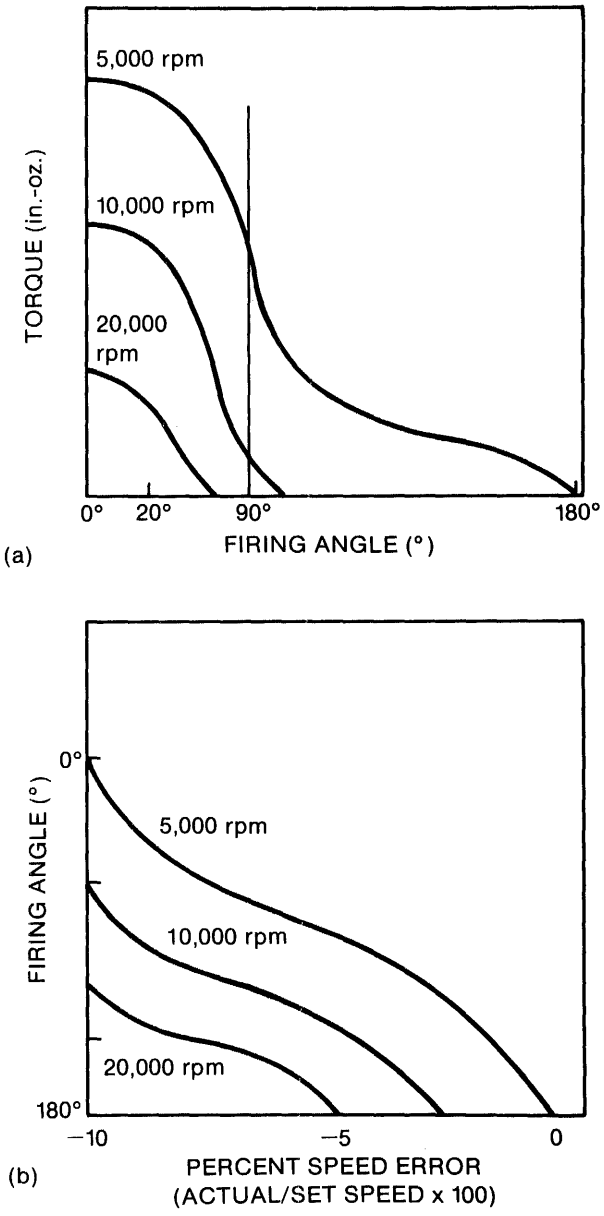
Done empirically, this mapping (Figure 43b) yields a curve of speed error versus torque that is almost linear. This curve's independence of a specific speed is assured by correlating speed error with firing angle for each of various speeds.

Fig. 42 CHANGE THE CURVE



NOTE: The speed-torque curve of a universal motor determines the motor's operating point for a constant voltage input and applied load. Only a closed-loop controller will allow the speed to be kept relatively constant in the face of a variable load.

Fig. 43 MAPPINGS



NOTE: Starting from no motor movement at all, the first load line of the motor—which corresponds to a small firing angle—is followed up to the first speed switch point, where the next firing angle takes over. This process continues until the motor runs out of torque.

Speed Measurement

The speed control algorithm built into the microcomputer uses the percentage error between the actual and set speed. For relatively small changes in speed, the percentage change in the period of revolution is approximately the same as the percentage change in speed.

If measurements for all possible set speeds in the same length of time are made with sufficient resolution, by picking up many pulses per motor revolution, the percentage error difference between the set period and actual period is approximately the negative percentage speed error.

This is easily shown mathematically. The fractional error in speed, E_S, is of course the difference between the set speed, S_S, and the actual speed, S_A, expressed as a fraction of S_S, or:

$$E_S = (S_S - S_A)/S_S \qquad (1)$$

The speed in revolutions per minute is 60 times the product of the reciprocals of N, the number of pulses per revolution, and P, the period in seconds of those pulses. So by substitution in Eq. 1:

$$\begin{aligned} E_S &= [(60/NP_S) - (60/NP_A)]/(60/NP_S) \\ &= (1/P_S - 1/P_A)/(1/P_S) \\ &= 1 - [P_S/(P_S - P_E)] \\ &= -P_E/(P_S - P_E) \end{aligned}$$

where P_A, P_S, and P_E are respectively the actual, set, and error periods in seconds. But if the error period is very much smaller than the set period (the usual case), $E_S = -P_E/P_S$, as was stated.

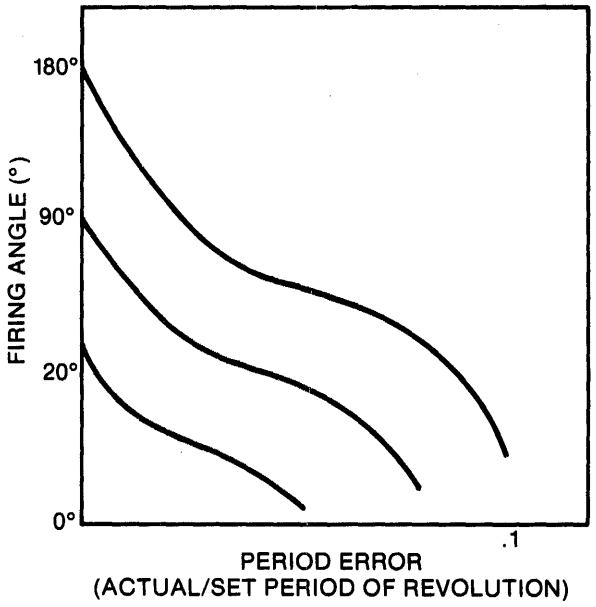
For these constant or near constant measurement period approximations, the error in period is proportional to the percentage speed error and can replace it in the firing angle mapping to achieve proper control (Figure 44). For fixed speeds, the values of N and P can be stored in a look-up table, and for variable speed control they can be calculated by means of a divide routine. Both of these are stored in the microcomputer.

Ripple Control

To refer back to Figure 43b, it is important to note the sharp change in torque for a given change in firing angle around 90°. The resolution of the firing angle at this point determines how much ripple there is in motor speed. At low speed inadequate resolution can cause sputtering where the torque change is such that it produces very noticeable jerks in speed.

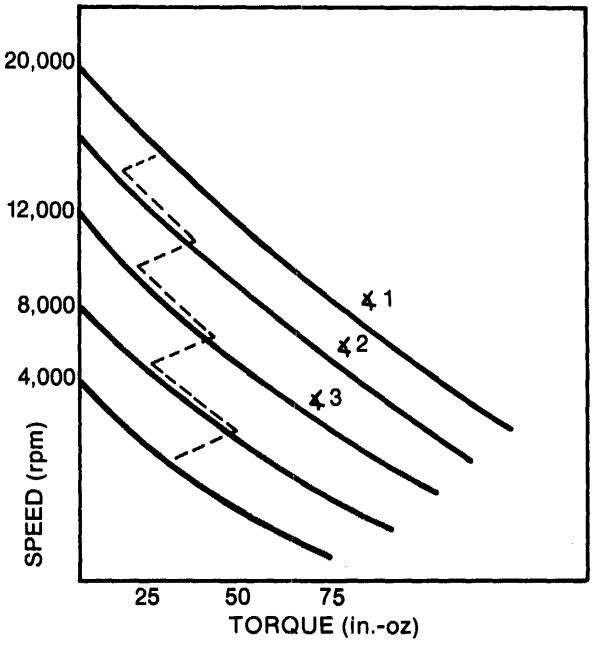
For instance, when the motor starts from zero speed, the first load line corresponding to a small firing angle (Figure 45) is followed up to the first speed point. There a second and larger firing angle is switched in. This discrete control is continued until the motor runs out of torque. From this diagram it is clear that any ripple will be determined by the step size in measurement made by the speed pickup and the resolution of the firing angle as set by the microcomputer.

Fig. 44 PERIOD



NOTE: For small changes in speed, the change in the period of motor revolution is the same as its change in speed. Consequently, the error in the motor period is proportional to its speed error and can therefore replace that variable in the firing-angle mapping.

Fig. 45 JUMPY



NOTE: Torque is a nonlinear function of both triac firing angle and motor speed (a). For linear motor speed regulation, the speed error must be mapped into firing angle (b). If done properly, a linear speed-error versus torque curve is achieved.

Microcomputer Requirements

A microcomputer used in universal motor speed control must have an 8-bit data word and an instruction execution rate of at least 250KHz to perform the functions discussed. And of course it should and does consume relatively little power.

The first two requirements are important because of the relatively complex calculations that must be performed quickly and the high resolution required for the triac firing angle at low motor speeds.

The General Instrument NMOS PIC1655A was specifically designed to meet these constraints. A one-chip microcomputer that uses only 35 milliamperes from a 4.5-to-7-volt supply, it has a pipelined architecture, 12-bit instructions, and an 8-bit data path.

Pipelining, or fetching the next instruction while executing the current one, shortens its instruction execution time to 4 microseconds. Also, the internal functions—the arithmetic and logic unit, memory, and input/output—need have data settling times of only 2 to 3μs to permit a conservative design and extended temperature ranges.

The 12-bit instruction word is long enough to eliminate the need for multiple fetches of instructions. The instruction set includes, in addition to common operations such as add, subtract, AND, OR, and exclusive-OR, other powerful bit operations like bit set, bit clear, and bit test. For example, the BSFSC 7, 2 instruction will skip the next instruction when bit 2 of I/O register 7 is low.

The 8-bit data path is adequate for most control applications. However, the PIC can handle the double precision necessary when 16-bit resolution is required. Its double-precision signed-integer math routines, including addition, subtraction, multiplication and division, are contained in 90 instructions.

Application Example A

What can a microcomputer do for a home vacuum cleaner? On the one hand, the vacuum motor can have a soft start. That is, current is limited during startup. With this feature, larger motors can be installed to allow higher vacuums and greater air flow without dimming the lights, blowing fuses, or exceeding Underwriters Laboratories specifications on turn-on.

In addition, the vacuum motor can be run at maximum efficiency. Depending on motor design, this might correspond to a constant speed of about 15,000 revolutions per minute for about 70% to 80% efficiency. Now the centrifugal blower can also be optimized for constant speed operation, further enhancing efficiency and lowering peak noise.

Note that the term "constant speed" means speed regulation within a certain limit, which will depend on the application. A speed decrease of about 10% from no load to full load is actually desirable since an increase of about 30% in vacuum pressure in fact accompanies decreased flow.

Application Example B

An alternative to constant pressure control is constant torque operation—allowing the speed to vary to maintain constant air flow. Furthermore, it permits the use of a motor designed for very high speeds, but one that normally draws too much current at lower speeds. Higher available vacuum pressure than would otherwise be possible is the result.

An improvement desirable in a vacuum cleaner is a reliable "bag full" indication. The indication of a full bag is low air flow over a period of time. Since the flow is most often proportional to torque in constant speed operation, the microcomputer can digitally filter the torque input signal and turn a lamp on. If the vacuum is run with constant torque, the bag will be full when the average speed goes over a certain limit. And finally, it is easy to hook up several push buttons to preset carpet beater speed and vacuum level.

7.7 Interfacing a PIC Microcomputer with the ER1400 EAROM

INTRODUCTION

Organized as 100 14-bit words, the ER1400 is an electrically erasable and reprogrammable non-volatile memory. Individual words may be erased and reprogrammed.

The ER1400 consists of a memory array, control circuitry, twenty bit serial to parallel shift register for addressing, and a 14-bit serial to parallel, parallel to serial shift register for data I/O. In the accept address mode, the address is shifted serially into the ER1400. The address consists of two consecutive one-of-ten codes controlling the "tens" digit and the "units" digit respectively. The Accept Address command may be followed by either Erase, Accept Data, Write (for reprogramming), or Read, and Shift Data Out (for reading).

With its serial address/data flow, the ER1400 only requires 5 I/O ports to interface with the microcomputer: one for clocking, three for control, and one for addressing and data flow. On the other hand, a 64 word x 8 bit EAROM such as the ER2055 requires 17 I/O ports: one for clocking, two for control, six for addressing, and eight for bidirectional data flow. However, the read cycle time for the ER2055 is much shorter than the ER1400.

Data is transferred to or from the ER1400 by first serially inputting two ten bit address words and then serially shifting in or out the 14-bit data word. Control of these operations is done by three chip control lines and 14KHz clock. It is essential that the clock is not interrupted between Accept Address and Shift Data Out and between Accept Address and Accept Data. Write and erase cycles require a 18 msec delay (with clocking) before changing modes to guarantee data retention.

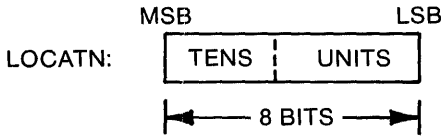
HARDWARE

A PIC with open drain outputs can directly drive the 10 volt I/O lines for the ER1400 as shown in Figure 46. The outputs of the PIC can be pulled more positive than the chip's power supply. High level outputs are pulled to the 10 volt supply by the 15K resistors, while low levels are pulled to ground by the output transistors on the PIC. In Figure 46, the point C2 is low for data or address transfers to the ER1400, and high for data transfers to the PIC. Thus the 100K resistor provides a pull-up for data write cycles and a 100K resistor is provided to ground when the ER1400 is outputting. Note that a logic "0" to the EAROM is a high voltage level, and a logic "1" is a low voltage level. According to Figure 46, a high voltage level is +10 volt and a low voltage level is 0 volt.

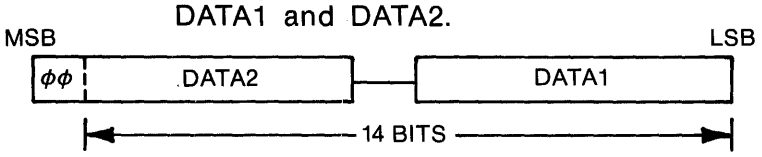
SOFTWARE

This software package consists of five subroutines as follow:

1. READ — Before calling READ, the read address should be stored in register LOCATN in BCD format as shown below.



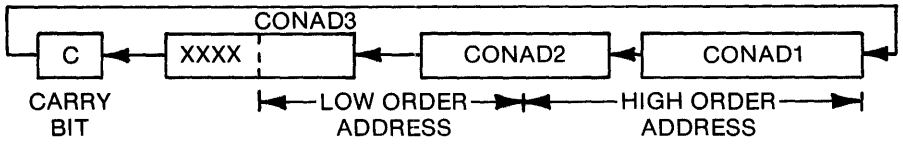
The subroutine ADEAR will be called to convert this BCD address into two 10-bit addresses in one-of-ten code as required by the ER1400 and transfer this address into the address register in the EAROM. After the content of this location has been read into the data register in the EAROM, this 14-bit data will be shifted out serially to two consecutive files in the PIC called DATA1 and DATA2.



When this is finished, the PIC will put the ER1400 into standby mode. A flowchart of the READ operation is shown on page 5.

2. WRITE — Before calling WRITE, the write address in BCD format should be stored in file LOCATN as described above. The 14-bit data waiting to be written into the EAROM should be stored in files DATA1 and DATA2. By calling ADEAR, the write address will be transferred into the EAROM. The content of this location is erased to logic '1' before data can be written in. After the content of DATA1 and DATA2 has been written into the EAROM, the PIC will put the EAROM into standby mode. A flowchart for the WRITE operation is shown on page 6.

3. ADEAR — According to the 2 digit BCD address in LOCATN, this subroutine will create a 20-bit address (2 consecutive one-of-ten codes) which is required by the EAROM. This 20-bit address is stored in three consecutive files called CONAD1, CONAD2 and CONAD3 in the following configuration:

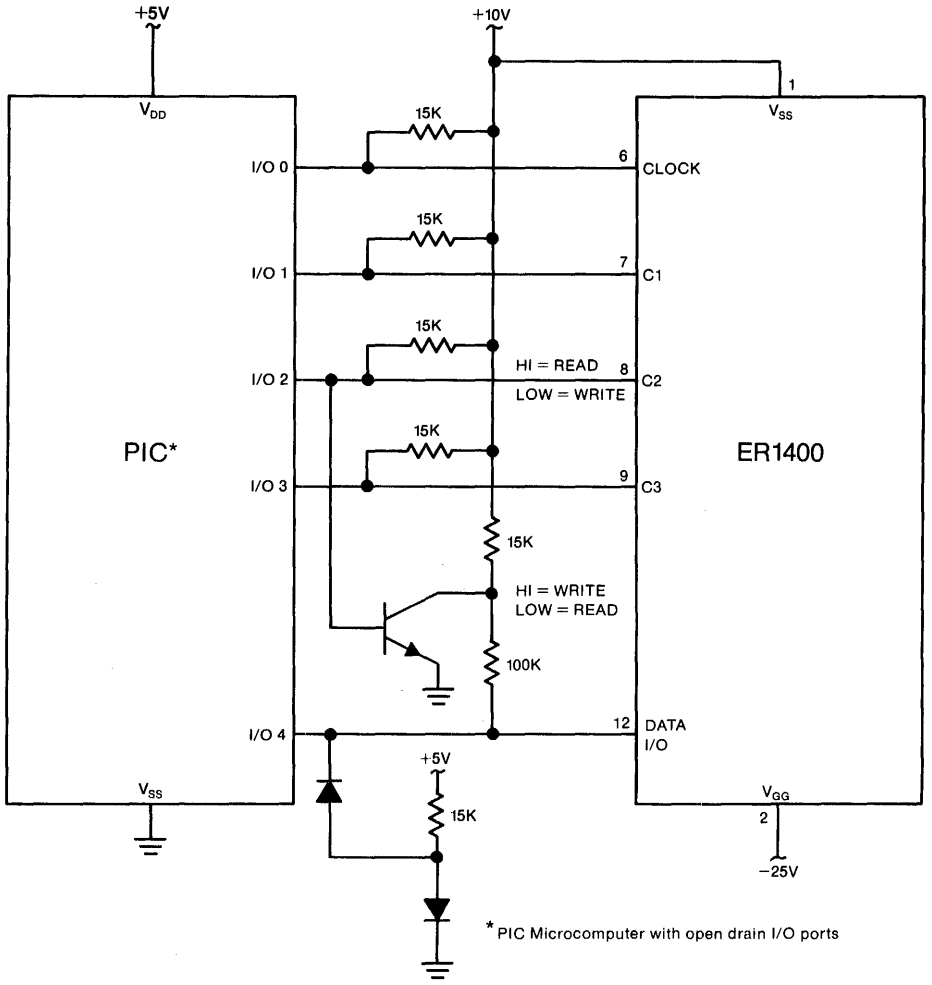


When this address is formed, this subroutine will automatically call ERTRAN which will send out the address to the EAROM.

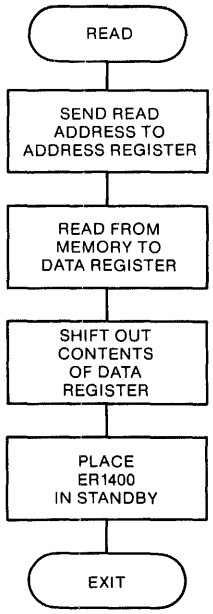
4. ERTRAN — This subroutine transfers the 20-bit address to the EAROM or the 14-bit data to/from the EAROM. On entry, the W register should contain the EAROM control code, file COUNT should contain the number of clock cycles for the EAROM, and the File Select Register (F4) should point to the start of the information file waiting to be transferred. This subroutine clocks the information to/from the EAROM at a rate of 13.8KHZ. The internal oscillator on the PIC runs at 1MHz providing an instruction cycle time of 4 microseconds. Thus a programming loop of 18 instruction cycle times can be used to generate the 14KHz clock for the ER1400. The complete software listing for the PIC-EAROM interface is given on pages 209-211.

5. WI8MS — This subroutine waits 18ms while the PIC is clocking the EAROM. This is required when an erase or write operation to the EAROM is called for.

Fig. 46 PIC MICROCOMPUTER TO ER1400 INTERFACE



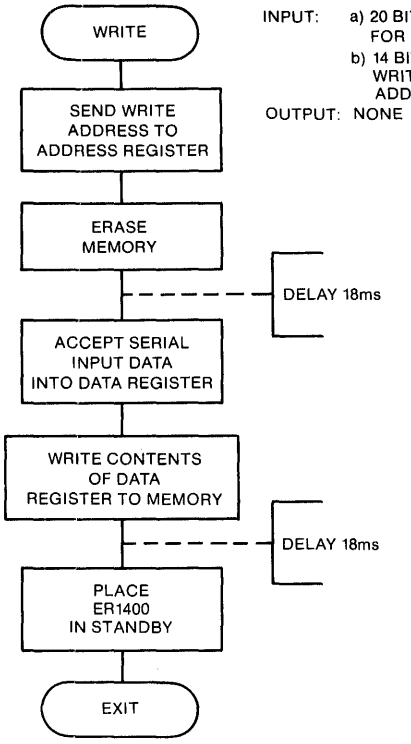
CLOCK CYCLES	ER1400 MODE CONTROL		
	C1	C2	C3
20	0	1	1
1	1	0	0
14	1	0	1
	0	0	0



INPUT: 20 BIT READ ADDRESS FOR ER1400

OUTPUT: 14 BIT DATA FROM READ ADDRESS

CLOCK CYCLES	ER1400 MODE CONTROL		
	C1	C2	C3
20	0	1	1
	0	1	0
14	1	1	1
	1	1	0
	0	0	0



INPUT: a) 20 BIT WRITE ADDRESS FOR ER1400
b) 14 BIT DATA TO BE WRITTEN INTO WRITE ADDRESS

OUTPUT: NONE

```
                                TITLE  '1650-ER1400'
                                LIST   E,X,P=1650
 000000
 000000
 000000
 000000
 000000
 000000
                        ;****************************************
                        ;* ************************************ *
                        ;* *                                  * *
                        ;* * PROJECT: PIC1650-ER1400 INTERFACE * *
                        ;* *                                  * *
                        ;* * ADDRESS:GENERAL INSTRUMENT CORP.  * *
                        ;* *         MICROELECTRONICS DIVISION * *
                        ;* *         600 WEST JOHN STREET     * *
                        ;* *         HICKSVILLE, NY 11802     * *
                        ;* * PHONE:(516) 733-3000             * *
                        ;* *                                  * *
                        ;* ************************************ *
                        ;****************************************
 000000
 000000
 000000
 000000
 000000
 000000
                        ;*****************************************************
                        ;* ************************************************* *
                        ;* *                                               * *
                        ;* * COPYRIGHT 1982 GENERAL INSTRUMENT CORPORATION * *
                        ;* * THIS PROGRAM IS PROTECTED AS AN UNPUBLISHED   * *
                        ;* * WORK UNDER THE COPYRIGHT ACT OF 1976 AND THE  * *
                        ;* * COMPUTER SOFTWARE ACT OF 1980.                * *
                        ;* *                                               * *
                        ;* ************************************************* *
                        ;*****************************************************
.INE    ADDR    B1      B2       1650-ER1400

 000000
 000000
                        ;************************
                        ;*                      *
                        ;* I/O FILE ASSIGNMENT  *
                        ;*                      *
                        ;************************
 000000
 000005                 IOREG   =       5               ;ADDRESS OF PORT A
 000000
 000000
 000000
 000000
                        ;****************************************
                        ;*                                      *
                        ;* I/O BITS ASSIGNMENT FOR PORT A (F5)  *
                        ;*                                      *
                        ;* A +5 VOLT ON THE CONTROL BIT MEANS   *
                        ;* LOGIC 0 FOR THE EAROM.   0 VOLT ON   *
                        ;* THE CONTROL BIT MEANS LOGIC 1 FOR    *
                        ;* THE ER1400 EAROM.                    *
                        ;*                                      *
                        ;****************************************
 000000
 000000                 ERCLK   =       0               ;14 KHZ CLOCK TO THE ER1400.
 000001                 C1      =       1               ;EAROM CONTROL BIT 1.
 000002                 C2      =       2               ;  "       "     "  2.
 000003                 C3      =       3               ;  "       "     "  3.
 000004                 ERDATA  =       4               ;SERIAL DATA TO OR FROM EAROM.
 000000
 000000
 000000
 000000
 000000
 000000
```

```
LINE    ADDR    B1      B2      1650-ER1400

  75  000000
  76                        ;*****************************************
  77                        ;*                                       *
  78                        ;* FILE REGISTER ASSIGNMENTS.            *
  79                        ;*                                       *
  80                        ;* THIS EAROM INTERFACE ROUTINE UTILIZES *
  81                        ;* F30 TO F37 IN THE PIC1650.  IT IS     *
  82                        ;* IMPORTANT THAT THESE EIGHT REGISTERS  *
  83                        ;* ARE DEDICATED TO THIS ROUTINE ONLY.   *
  84                        ;*                                       *
  85                        ;*****************************************
  86  000000
  87  000004                FSR     =       4               ;FILE SELECT REGISTER.
  88  000030                COUNT   =       30              ;EAROM ROUTINE INTERNAL COUNTER.
  89                                                        ;THIS COUNTER IS USED TO COUNT THE
  90                                                        ;NUMBER OF EAROM CLOCKS.
  91  000031                CONAD3  =       31              ;THE LSB OF THE 20-BIT EAROM
  92  000032                CONAD2  =       32              ;ADDRESS IN ONE-OUT-OF TEN
  93  000033                CONAD1  =       33              ;CODE FORMAT.
  94  000034                TEMP    =       34              ;TEMPORY REGISTER USED BY EAROM.
  95  000035                LOCATN  =       35              ;ON ENTRY, THIS REGISTER CONTAINS
  96                                                        ;THE BCD EAROM ADDRESS.  THIS
  97                                                        ;ROUTINE WILL CONVERT THIS BCD
  98                                                        ;INTO THE FINAL ONE OF TEN CODE
  99  000036                DATA1   =       36              ;THIS IS THE LSB OF THE 14 BITS
 100                                                        ;EAROM DATA.
 101  000037                DATA2   =       37              ;THIS IS THE MSB OF THE 14 BITS
 102                                                        ;EAROM DATA.
 103  000000
LINE    ADDR    B1      B2      1650-ER1400

 105                        ;*****************************************************
 106                        ;*                                                   *
 107                        ;* THIS IS THE READ EAROM ROUTINE.  THE FOLLOWING    *
 108                        ;* PARAMETER ARE NEEDED BEFORE CALLING THIS ROUTINE: *
 109                        ;*                                                   *
 110                        ;*     PARAMETER: LOCATN (F35)--- THE BCD ADDRESS OF *
 111                        ;*                      THE EAROM LOCATION THAT HAS TO *
 112                        ;*                      BE READ.                     *
 113                        ;*                                                   *
 114                        ;*     OUTPUT:    DATA1 (F36)--- THE LSB OF THE 14  *
 115                        ;*                                  BITS EAROM DATA. *
 116                        ;*                DATA2 (F37)--- THE MSB OF THE 14  *
 117                        ;*                                  BITS EAROM DATA. *
 118                        ;*                                                   *
 119                        ;*****************************************************
 120  000000
 121  000000                READ    RES     0               ;READ EAROM ROUTINE ENTRY POINT.
 122  000000 04446                  CALL    ADEAR           ;ADDRESS ER1400.  COUNT LEFT AT ZERO
 123  000001 02430                  BSF     COUNT,0         ;SET COUNTER TO ONE
 124  000002 06375                  MOVLW   B'11111101'     ;CONTROL CODE FOR READ
 125                                                        ;DATA AND CLOCK HIGH
 126  000003 04474                  CALL    ERTRAN          ;READ THE DATA REGISTER, COUNT LEFT AT ZE
 127  000004 02630                  BSF     COUNT,4         ;SHIFT OUT 16 BITS (14 PLUS 2 TO
 128                                                        ;NORMALIZE DATA TO LOWER
 129                                                        ;6 BITS OF DATA2 )
 130  000005 06036                  MOVLW   DATA1           ;
 131  000006 00044                  MOVWF   FSR             ;POINT TO DATA REGISTERS
 132  000007 06345                  MOVLW   B'11100101'     ;CON CODE FOR SHIFT DATA OUT
 133  000010 04474                  CALL    ERTRAN          ;SHIFT DATA OUT.  LEAVE 77 IN W
 134  000011 00577                  ANDWF   DATA2           ;ENSURE BITS 6-7 CLEAR
 135                                                        ;
 136  000012                EXEAR   RES     0               ;
 137  000012 06377                  MOVLW   B'11111111'     ;CONTROL CODE FOR STANDBY
 138                                                        ;WITH CLOCK BIT SET
 139  000013 00045                  MOVWF   IOREG           ;OUTPUT CONTROL CODE
 140  000014 04000                  RETLW   0
```

```
LINE    ADDR    B1      B2        1650-ER1400

 142  000015
 143                        ;*************************************************************
 144                        ;*                                                           *
 145                        ;* THIS IS THE EAROM WRITE ROUTINE.  THE FOLLOWING           *
 146                        ;* PARAMETERS MUST BE SET UP BEFORE THIS ROUTINE             *
 147                        ;* IS INVORKED.                                              *
 148                        ;*                                                           *
 149                        ;*   PARAMETERS: LOCATN (F55)--- THE BCD ADDRESS OF THE      *
 150                        ;*                       EAROM LOCATION THAT NEW             *
 151                        ;*                       DATA IS GOING TO BE STORED INTO     *
 152                        ;*               DATA1   (F56)--- THE LOWER 8 BITS OF        *
 153                        ;*                       NEW DATA.                           *
 154                        ;*               DATA2   (F57)--- THE UPPER 6 BITS OF THE    *
 155                        ;*                       NEW DATA PLUS TWO DON'T CARE BITS.  *
 156                        ;*                                                           *
 157                        ;*   OUTPUT: NONE                                            *
 158                        ;*                                                           *
 159                        ;*************************************************************
 160  000015
 161  000015                WRITE   RES     0               ;EAROM WRITE ENTRY POINT.
 162  000015 04446                  CALL    ADEAR           ;ADDRESS THE EAROM.
 163  000016 06373                  MOVLW   B'11111011'     ;CON CODE FOR EREASE
 164                                                        ;DATA & CLOCK HIGH
 165  000017 00045                  MOVWF   IOREG           ;
 166  000020 04436                  CALL    W18MS           ;DELAY 18MS.  ON RETURN,
 167                                                        ;14 IS STORED IN W.
 168  000021 00070                  MOVWF   COUNT           ;SEND OUT 14 CLOCK PULSES.
 169  000022 06036                  MOVLW   DATA1           ;STORE THE ADDRESS OF THE LOW
 170  000023 00044                  MOVWF   FSR             ;BYTE OF NEW DATA INTO 'FSR'.
 171                                                        ;
 172  000024 06361                  MOVLW   B'11110001'     ;CON CODE FOR ACCEPT DATA
 173                                                        ;DATA & CLOCK HIGH
 174  000025 04474                  CALL    ERTRAN          ;SHIFT THE DATA INTO THE EAROM.
 175                                                        ;
 176  000026 06371                  MOVLW   B'11111001'     ;CON CODE FOR WRITE
 177  000027 00045                  MOVWF   IOREG           ;DATA & CLOCK HIGH
 178  000030 04436                  CALL    W18MS           ;DELAY 18MS WITH CONTINOUS CLOCK.
 179  000031 05012                  GOTO    EXEAR           ;EXIT FROM THIS EAROM INTERFACE
 180                                                        ;ROUTINE AND RETURN TO MAIN PROGRAM.
 181                                                        ;THE ER1400 IS PUT INTO STANDBY MODE.
 182  000032
 183  000032
LINE    ADDR    B1      B2        1650-ER1400

 185  000032
 186                        ;*************************************************
 187                        ;*                                               *
 188                        ;* THIS IS AN 18MS DELAY ROUTINE REQUIRED WHEN   *
 189                        ;* WRITING DATA INTO THE ER1400 EAROM.  DURING   *
 190                        ;* THIS 18MS PERIOD, A 14 KHZ EAROM CLOCK MUST   *
 191                        ;* BE MAINTAINED.  ON RETURN, THIS ROUTINE PUT   *
 192                        ;* A DECIMAL NUMBER 14 INTO THE W REGISTER.      *
 193                        ;*                                               *
 194                        ;*************************************************
 195  000032
 196  000032                WMID    RES     0
 197  000032 00645                  XORWF   IOREG           ;TOGGLE THE EAROM CLOCK
 198  000033 03030                  BTFSC   COUNT,0
 199  000034 04016                  RETLW   .14             ;RETURN TO CALLING ROUTINE.
 200  000035 02430                  BSF     COUNT,0
 201  000036                W18MS   RES     0               ;ENTRY POINT FOR 18 MS DELAY.
 202  000036 00174                  CLRF    TEMP
 203  000037                W36US   RES     0
 204  000037 01374                  DECFSZ  TEMP
 205  000040 05042                  GOTO    WNZYET
 206  000041 05032                  GOTO    WMID
 207  000042
 208  000042                WNZYET  RES     0
 209  000042 06001                  MOVLW   1
 210  000043 00645                  XORWF   IOREG           ;TOGGLE THE EAROM CLOCK.
 211  000044 05045                  GOTO    WPAD
 212  000045 05037          WPAD    GOTO    W36US
 213  000046
 214  000046
 215  000046
 216  000046
```

```
LINE    ADDR    B1      B2        1650-ER1400

 218  000046
 219                        ;*****************************************************
 220                        ;*                                                   *
 221                        ;* THIS ROUTINE TRANSFORMS THE BCD EAROM ADDRESS     *
 222                        ;* STORED IN REGISTER 'LOCATN' INTO THE 20-BIT       *
 223                        ;* ONE-OUT-OF-TEN CODE REQUIRED BY THE ER1400 EAROM. *
 224                        ;* THIS ONE-OF-TEN CODE IS STORED IN 'CONAD1',       *
 225                        ;* 'CONAD2' AND 'CONAD3' WITH THE LSB IN 'CONAD3'.   *
 226                        ;*                                                   *
 227                        ;* WHEN THIS 20-BIT ADDRESS IS FORMED, IT IS AUTO-   *
 228                        ;* MATICALLY SENT TO THE EAROM BY EXECUTING THE      *
 229                        ;* 'ERTRAN' ROUTINE.                                 *
 230                        ;*                                                   *
 231                        ;*****************************************************
 232  000046
 233  000046                ADEAR   RES     0               ;ENTRY POINT FOR ADDRESS TRANSFORM.
 234  000046 01035                  MOVFW   LOCATN          ;PUT LOW NIBBLE OF ADDRESS
 235  000047 07017          LOADDC  ANDLW   17              ;IN LOW NIBBLE OF TEMP
 236  000050 00074                  MOVWF   TEMP
 237  000051 06012                  MOVLW   .10             ;NO OF LOOPS BEFORE
 238  000052 00070                  MOVWF   COUNT           ;THIS ADDRESS PART COMPLETE
 239  000053 06001                  MOVLW   1               ;DECREMENT FOR ADDRESS
 240  000054 00274          ROT3SR  SUBWF   TEMP            ;CLRS CARRY IF THIS PART OF ADDRESS
 241                                                        ;HAS NOW REACHED ZERO
 242  000055 01573                  RLF     CONAD1          ;SHIFT THE 'SHIFT REGISTER'
 243  000056 01572                  RLF     CONAD2
 244  000057 01571                  RLF     CONAD3
 245  000060 01370                  DECFSZ  COUNT           ;.10 SHIFTS DONE YET ?
 246  000061 05054                  GOTO    ROT3SR          ;NOT YET
 247  000062 03505                  BTFSS   IOREG,2         ;YES.  WAS THIS SECOUND ADDRESS ?
 248  000063 05067                  GOTO    OPADD           ;YES.  NOW OUTPUT CONVERTED ADDRESS
 249  000064 02105                  BCF     IOREG,2         ;NO.  NOW CONVERT HIGH ADDRESS
 250  000065 01635                  SWAPF   LOCATN,0        ;READY FOR HIGH NIBBLW OF ADDRESS
 251  000066 05047                  GOTO    LOADDC          ;GO DO HIGH ADDRESS
 252  000067 06033          OPADD   MOVLW   CONAD1          ;PT FSR TO START OF CONVERTED ADDRESS
 253  000070 00044                  MOVWF   FSR             ;3-REGISTER 'SHIFT REGISTER'
 254  000071 06024                  MOVLW   .20
 255  000072 00070                  MOVWF   COUNT           ;SET FOR 10 BIT TRANSFER TO ER1400
 256  000073 06363                  MOVLW   B'11110011'     ;ACCEPT ADDRESS CONTROL CODE
 257                                                        ;DATA HIGH, CLOCK HIGH
 258                                                        ;GO INTO I/O ROUTINE 'ERTRAN'
LINE    ADDR    B1      B2        1650-ER1400

 260                        ;*****************************************************
 261                        ;*                                                   *
 262                        ;* TRANSFER DATA OR ADDRESS TO OR FROM THE ER1400    *
 263                        ;*                                                   *
 264                        ;*                    ON ENTRY                       *
 265                        ;*                    --------                       *
 266                        ;* FSR (F4)    - POINTS TO START OF INFORMATION FILE *
 267                        ;*               (CONAD1 IF ADDRESS, DATA1 IF DATA)  *
 268                        ;*                                                   *
 269                        ;* FILE COUNT - NUMBER OF ER1400 CLOCK CYCLES OR BITS *
 270                        ;*                                                   *
 271                        ;* W          - ER1400 CONTROL CODE                  *
 272                        ;*                                                   *
 273                        ;*****************************************************
 274  000074
 275  000074                ERTRAN  RES     0
 276  000074 00045                  MOVWF   IOREG           ;OUTPUT CONTROL WORD
 277  000075 06010                  MOVLW   .8              ;OUTPUT 8 BITS BEFORE
 278  000076 00074                  MOVWF   TEMP            ;MOVING TO NEXT INFO FILE
 279  000077                STLOOP  RES     0
 280  000077 02405                  BSF     IOREG,ERCLK     ;SET THE EAROM CLOCK BIT
 281  000100 03105                  BTFSC   IOREG,C2        ;INPUTTING TO THE PIC?
 282  000101 05107                  GOTO    RECEIV          ;YES, INPUT TO PIC FROM ER1400.
 283  000102                GIVE    RES     0               ;ELSE, OUTPUT DATA FROM
 284  000102 02605                  BSF     IOREG,ERDATA    ;PIC TO EAROM
 285  000103 01440                  RRF     0               ;ROTATE INFO FILE INTO CARRY
 286  000104 03403                  SKPC                    ;IS THE INFO BIT A ZERO ?
 287  000105 02205                  BCF     IOREG,ERDATA    ;YES, SHIFT A ZERO TO EAROM.
 288  000106 05114                  GOTO    NEXTI           ;GET NEXT INFO BIT
 289  000107                RECEIV  RES     0               ;RECEIVE DATA FROM EAROM.
 290  000107 02605                  BSF     IOREG,ERDATA    ;ENSURE PIN NOT LATCHED AT ZERO
 291  000110 02003                  CLRC                    ;READ THE INPUT FROM EAROM
 292  000111 03205                  BTFSC   IOREG,ERDATA    ;IS IT A LOGIC '1' ?
 293  000112 02403                  SETC                    ;YES
 294  000113 01440                  RRF     0               ;STORE THE DATA INTO PIC.
 295  000114                NEXTI   RES     0
```

```
296  000114 02005                    BCF     IOREG,ERCLK     ;CLEAR THE EAROM CLOCK BIT
297  000115 01374                    DECFSZ  TEMP            ;DONE 8 BITS YET ?
298  000116 05125                    GOTO    STPAD           ;NO, MORE TO GO
299  000117 02574                    BSF     TEMP,3          ;ELSE, RESET COUNTER TO EIGTH
300  000120 01244                    INCF    FSR             ;INCREMENT FSR TO NEXT INFO FILE
301  000121                  FINL?   RES     0
302  000121 01370                    DECFSZ  COUNT           ;FINISH ALL INFO FILES ?
303  000122 05077                    GOTO    STLOOP          ;NO.
304  000123 02405                    BSF     IOREG,ERCLK     ;ELSE, SET EAROM CLOCK BIT HIGH
305  000124 04077                    RETLW   77              ;END OF EAROM I/O WITH 77 IN W.
306  000125                  STPAD   RES     0
307  000125 05121                    GOTO    FINL?           ;TIMING COMPENSATION.
308  000126
309  000126                          END

 ASSEMBLER ERRORS =    0
                        1650-ER1400


                        CROSS REFERENCE

LABEL     VALUE       REFERENCE

ADEAR     000046        122   162  -233
C1        000001        -64
C2        000002        -65   281
C3        000003        -66
CONAD1    000033        -93   242   252
CONAD2    000032        -92   243
CONAD3    000031        -91   244
COUNT     000030        -88   123   127   168   198   200   238   245
                        255   302
DATA1     000036        -99   130   169
DATA2     000037       -101   134
ERCLK     000000        -63   280   296   304
ERDATA    000004        -67   284   287   290   292
ERTRAN    000074        126   133   174  -275
EXEAR     000012       -136   179
FINL?     000121       -301   307
FSR       000004        -87   131   170   253   300
GIVE      000102       -283
IOREG     000005        -47   139   165   177   197   210   247   249
                        276   280   281   284   287   290   292   296
                        304
LOADDC    000047       -235   251
LOCATN    000035        -95   234   250
NEXTI     000114        288  -295
OPADD     000067        248  -252
READ      000000       -121
RECEIV    000107        282  -289
ROT3SR    000054       -240   246
STLOOP    000077       -279   303
STPAD     000125        298  -306
TEMP      000034        -94   202   204   236   240   278   297   299
W18MS     000036        166   178  -201
W36US     000037       -203   212
WMID      000032       -196   206
WNZYET    000042        205  -208
WPAD      000045        211  -212
WRITE     000015       -161

EOF:366
0:)
```

7.8 Interfacing the PIC1650 Microcomputer with the ER2055 EAROM

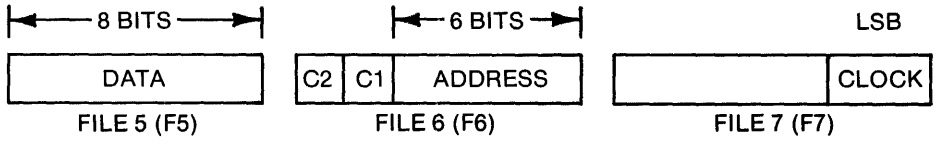
The ER2055 is a 64 x 8 EAROM with parallel address and I/O. Seventeen I/O pins are required in this routine to interface with the PIC1650. Figure 47 shows the configuration of these I/O ports.

The address of the EAROM is stored in the Tower 6 bits of F6. Bit 6 and 7 of the F6 are used to store the mode control inputs C1 and C2 respectively.

On entry to READ or WRITE, the address should be stored in the W register and the two most significant bits must be zero. Before calling WRITE, data waiting to be written into the EAROM must be in File 5. On return from READ, the data read from the EAROM is in File 5 and can be transferred to another register, if desired.

Figure 48 shows the hardware connections of the I/O ports. The ER2055 is fully TTL compatible and thus no external hardware is needed. The EAROM has two chip select lines which are hard-wired so that the EAROM is always selected. The controlling software will always set the EAROM in the read mode except when writing data to the EAROM. However, the 2-20μs clock pulse required to read the EAROM need be generated only when the READ subroutine is called. In order to give the correct clock pulse, the clock bit must be initialized to zero at the beginning of the program. Before writing data into the EAROM, that location has to be erased first. The erase and write cycle time is set to 22 msec by calling the DELAY subroutine. The EAROM will again set back to the read mode when the write cycle is finished. It takes 40 microseconds to read data from and 43.2 msec to write data to the EAROM.

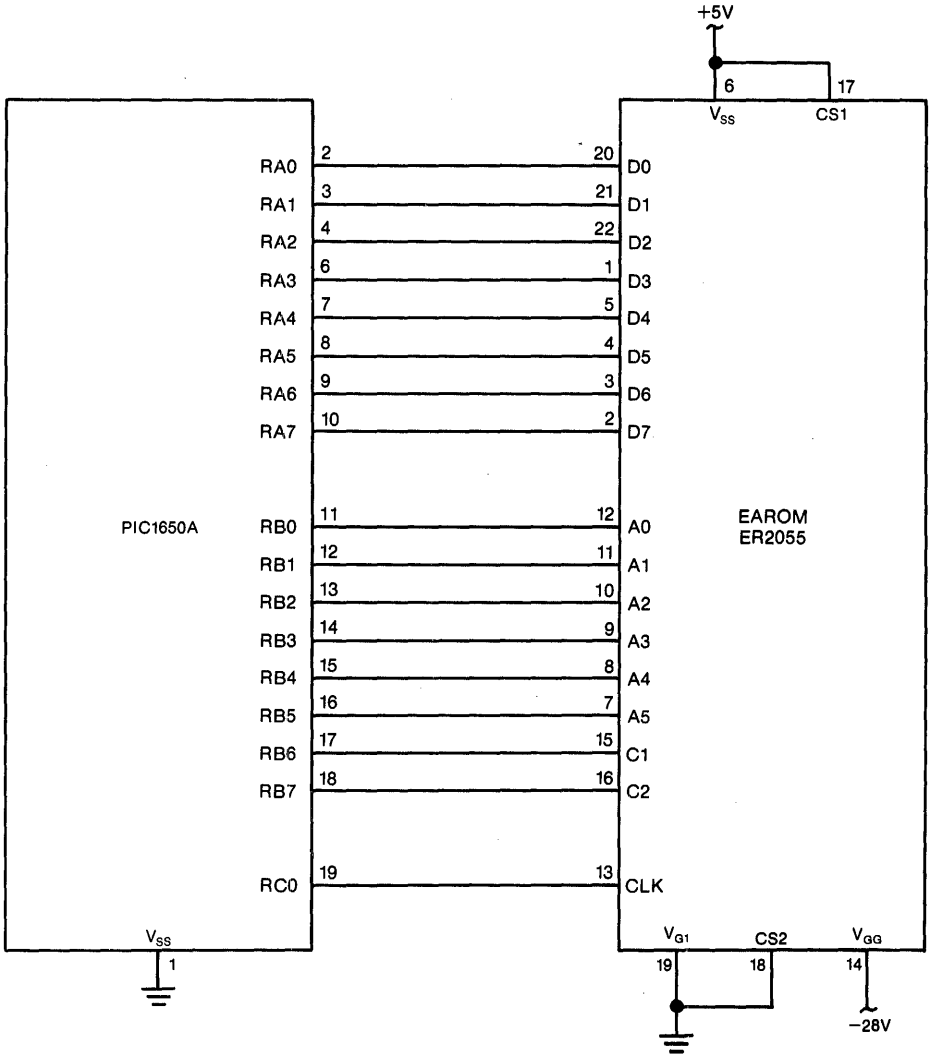
Fig. 47 I/O PORTS ARRANGEMENT



ER2055 VS ER1400

Since the ER2055 uses parallel addressing and I/O, seventeen I/O pins are required to interface with PIC. There are eight bidirectional data lines, six address lines, two mode control lines and one clock input. Since the eight data lines are only used during read/write operations, these data lines may also be used for some other purposes such as 7-segment display. On the other hand, it only needs six I/O lines to interface the ER1400 with the PIC since data and address are sent serially. However, the read cycle time for the ER1400 is much longer than the ER2055. To read a location, the ER1400 needs 3.4ms while the ER2055 only takes 40 microseconds.

Fig. 48 PIC1650 TO ER2055 INTERFACE



```
 1  000000
 2  000000
 3                                  TITLE  'PIC1650-ER2055 INTERFACE ROUTINE'
 4  000000
 5  000000
 6  000005                  DATA    EQU    5
 7  000006                  ADDR    EQU    6
 8  000007                  CTRL    EQU    7
 9  000000                  CLOCK   EQU    0
10  000006                  C1      EQU    6
11  000007                  C2      EQU    7
12  000020                  TEMP1   EQU    20
13  000021                  TEMP2   EQU    21
14  000000
15  000000
16                                                      ; ON ENTRY, ADDRESS SHOULD BE IN THE
17                                                      ; LOWER 6 BITS OF THE W REGISTER AND
18                                                      ; THE MOST SIGNIFICANT 2 BITS SHOULD
19                                                      ; BE ZERO.  ON EXIT, DATA READ IS IN
20                                                      ; THE W REGISTER
21  000000
22  000000 06500            READ    IORLW  100
23  000001 00046                    MOVWF  ADDR             ; SET IN READ MODE
24  000002 06377                    MOVLW  377              ; SET THE I/O PORT FOR INPUT
25  000003 00045                    MOVWF  DATA
26  000004 02407                    BSF    CTRL,CLOCK       ; CLOCK THE READ OPERATION
27  000005 02007                    BCF    CTRL,CLOCK
28  000006 04000                    RETLW  0
29  000007
30  000007
31                                                      ; ON ENTRY, ADDRESS SHOULD BE IN THE
32                                                      ; W REGISTER.  DATA IN THE DATA REGISTER.
33                                                      ;
34  000007 06600            WRITE   IORLW  200
35  000010 00046                    MOVWF  ADDR             ; SET IN ERASE MODE
36  000011 04416                    CALL   DELAY
37  000012 02346                    BCF    ADDR,C2          ; SET IN WRITE MODE
38  000013 04416                    CALL   DELAY
39  000014 02706                    BSF    ADDR,C1          ; SET IN READ MODE
40  000015 04000                    RETLW  0
41  000016
42                                                      ; THIS GIVE 22MS DELAY TIME
43                                                      ;
44  000016 06007            DELAY   MOVLW  7
45  000017 00060                    MOVWF  TEMP1
46  000020 00161                    CLRF   TEMP2
47  000021 01361            LOOP    DECFSZ TEMP2
48  000022 05021                    GOTO   LOOP
49  000023 01360                    DECFSZ TEMP1
50  000024 05021                    GOTO   LOOP
51  000025 04000                    RETLW  0
52  000026
53  000026
55                                  ORG    100
56  000100 06123            TESTWR  MOVLW  123              ;TEST WRITING ROUTINE
57  000101 00045                    MOVWF  DATA
58  000102 06005                    MOVLW  5                ; THIS IS THE ADDRESS OF THE EAROM
59  000103 04407                    CALL   WRITE
60  000104
61  000104
62  000104
63                                                      ;TEST THE READING ROUTINE
64  000104 06005            TESTRD  MOVLW  5                ; ADDRESS OF THE EAROM
65  000105 04400                    CALL   READ
66  000106 01005                    MOVF   DATA,W           ; STORE THE DATA INTO W REGISTER
67  000107
68  000107                          END

ASSEMBLER ERRORS =     0


                              SYMBOL TABLE

ADDR    000006    C1      000006    C2      000007    CLOCK   000000
CTRL    000007    DATA    000005    DELAY   000016    LOOP    000021
READ    000000    TEMP1   000020    TEMP2   000021    TESTRD  000104
TESTWR  000100    WRITE   000007

EOF:84
0:>
```

NOTES

NOTES

GENERAL INSTRUMENT

Microelectronics Division/General Instrument Corporation
WORLDWIDE SALES OFFICES

NORTH AMERICA

UNITED STATES:

MICROELECTRONICS DIVISION

NORTHEAST—600 West John Street
Hicksville, New York 11802
Tel: 516-733-3107, TWX: 510-221-1866

20th Century Plaza
Daniel Webster Highway
Merrimack, New Hampshire 03054
Tel: 603-424-3303, TWX: 710-366-0676

858 Welsh Road
Maple Glen, Pennsylvania 19002
Tel: 215-643-5326

SOUTHEAST—7901 4th Street. N., Suite 208
St. Petersburg, Florida 33702
Tel: 813-577-4024, TWX: 810-863-0398

1616 Forest Drive
Annapolis, Maryland 21403
Tel: 301-269-6250, TWX: 710-867-8566

4921C Professional Court
Raleigh, North Carolina 27609
Tel: 919-876-7380

408 North Cedar Bluff Road, Suite 390
Knoxville, Tennessee 37923
Tel: 615-690-2233

SOUTH CENTRAL—5520 LBJ Frwy., Suite 330
Dallas, Texas 75240
Tel: 214-934-1654, TWX: 910-860-9259

CENTRAL—4524 S. Michigan Street
South Bend, Indiana 46614
Tel: 219-291-0585, TWX: 810-299-2518

5820 West 85th Street, Suite 102
Indianapolis, Indiana 46278
Tel: 317-872-7740, TWX: 810-341-3145

2355 S. Arlington Hts. Road, Suite 408
Arlington Heights, Illinois 60005
Tel: 312-981-0040, TWX: 910-687-0254

32969 Hamilton Court, Suite 210
Farmington Hills, Michigan 48018
Tel: 313-553-4330, Telex: 231193

230 North River Ridge Circle, Suite 116
Burnsville, Minnesota 55337
Tel: 612-894-1840, TWX: 910-5760240

SOUTHWEST—201 Standard Street
El Segundo, California 90245
Tel: 213-322-7745, TWX: 910-348-6296

NORTHWEST—3080 Olcott Street, Suite 230C
Santa Clara, California 95051
Tel: 408-496-0844, TWX: 910-379-0010

EUROPE

NORTHERN EUROPE

Times House, Ruislip, Middlesex, HA4 8LE
Tel: (08956), 35700, Telex: 23272

Sandhamnsgatan 67
S-115 28, Stockholm
Tel: (08) 67 99 25, Telex: 17779

SOUTHERN EUROPE

5-7 Rue De L'Amiral Courbet
94160 Saint Mande, Paris
Tel: (1) 365 72 50, Telex: 213073

Via Quintiliano 27, 20138 Milano
Tel: (02) 5062648, Telex: 843-320348

CENTRAL EUROPE

GENERAL INSTRUMENT DEUTSCHLAND GmbH
Freischuetzstr. 96
Postfach 81 03 29
8000 Muenchen 81
Tel: (089) 956001, Telex: 528054

6070 Langen Bei Frankfurt A Main
Wilhelm-Leuschner Platz 8, Postf. 1167
Tel: (6103) 23 051, Telex: 415000

ASIA

HONG KONG:

GENERAL INSTRUMENT HONG KONG LTD.
139 Connaught Road Central, 3/F, San-Toi Building
Tel: (5) 434360, Telex: 84606

JAPAN:

GENERAL INSTRUMENT INTERNATIONAL CORP.
Fukide Bldg. 8th Floor, 1-13 Toranomon 4-Chome
Minato-ku, Tokyo 105
Tel: (03) 437-0281, Telex: 2423413

KOREA:

GENERAL INSTRUMENT MICROELECTRONICS
Dong Young Building, 903
82, 1-KA, Ulgiro, Chung Ku
Seoul, South Korea
Tel: (2) 777-3848, Telex: K 26880 DAEHO

SINGAPORE:

GENERAL INSTRUMENT HONG KONG LTD.
Suite 1714, Shaw Centre
1 Scotts Road, Singapore 0922
Tel: (65) 235-8030, Telex: GIS'PORE RS 24424

TAIWAN:

GENERAL INSTRUMENT MICROELECTRONICS TAIWAN
77 Pao Chiao Road, Hsin Tien
Taipei, Taiwan
Tel: (02) 914-6234, Telex: 785-3111

MANUFACTURING FACILITIES

U.S.A.—Hicksville, New York • Chandler, Arizona • EUROPE—Glenrothes, Scotland • ASIA—Kaohsiung, Taiwan

APPLICATIONS CENTERS

U.S.A.—Hicksville, New York • Chandler, Arizona • Los Angeles, California
EUROPE—Glenrothes, Scotland • London, England • Paris, France • Munich, Germany • Stockholm, Sweden
ASIA—Kaohsiung, Taiwan • Tokyo, Japan • Hong Kong