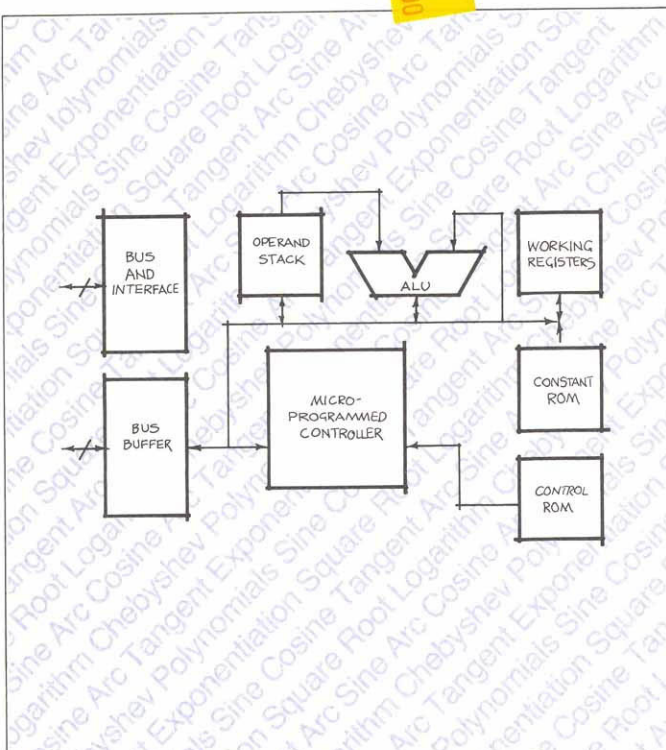
Floating Point Processor Manual Am9511A/Am9512

Technical Manual



ADVANCED MICRO DEVICES

Advanced Micro Devices

Am9511A/Am9512 Floating Point Processor Manual

901 Thompson Place, P.O. Box 3453, Sunnyvale, California 94088
(408) 732-2400 TWX: 910-339-9280 TELEX: 34-6306

Printed in U.S.A.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

ILLUSTRATIONS

TABLES

CHAPTER 1
AN INTRODUCTION TO FLOATING POINT

1.1 WHAT IS A FLOATING POINT NUMBER?

The numbers we encounter every day, such as 12, 34.56, 0.0789, etc., are known as fixed point numbers because the decimal point is in a fixed position. Such numbers are fairly closely matched in magnitude and within about ten orders of magnitude from unity. Examples of such numbers are found in bank accounts, unit prices of store items and paychecks.

In scientific applications, the numbers encountered can be very large. Avogadro's number expressed in fixed point notation is approximately 602,250,000,000,000,000,000,000. A scientist may also use Planck's constant which would be approximately 0.000000000000000000000000006626196 erg sec in fixed point notation. These examples demonstrate the undesirability of writing fixed point notation and why most scientists use the concise floating point notation to represent numbers such as Avogadro's number and Planck's constant.

When a scientist writes the value of Avogadro's number, he writes 6.0225×10^{23}. Similarly he would express Planck's constant as 6.626196×10^{-27} erg sec.

As we can observe, the number $+6.0225 \times 10^{23}$, consists of 4 parts:

Sign –
The sign of the number (+ or −). The plus sign is usually assumed when no sign is shown.

Mantissa –
Sometimes also known as the fraction. The mantissa describes the actual number. In the example, the mantissa is 6.0225.

Exponent –
Sometimes also known as the characteristic. The exponent describes the order of magnitude of the number. In the example, the exponent is 23.

Base –
Sometimes also known as the radix. The base is the number base in which the exponent is raised. In the example, the base is 10.

The parts of a floating point number can then be represented by the following equation:

$$F = (-1)^S \times M \times B^E$$

where

- F = floating point number
- S = sign of the floating point number, so that S = 0 if the number is positive and S = 1 if the number is negative
- M = mantissa of the floating point number
- B = base of the floating point number
- E = exponent of the floating point number

1.2 WHEN SHOULD FLOATING POINT BE USED?

Although floating point numbers are useful when numbers of very different magnitude are used, they should not be used indiscriminately. There is an inherent loss of accuracy and increased execution time for floating point computations on most computers. Floating point computation suffers the greatest loss of accuracy when two numbers of closely matched magnitude are subtracted from each other or two numbers of opposite sign but almost equal magnitude are added together. Therefore, the Associative Law in arithmetic

$$A + (B + C) = (A + B) + C$$

does not always hold true if B is of opposite sign to A and C and very similar in magnitude to either A or C.

In most computers, hardware floating point multiply and divide takes approximately the same amount of execution time as hardware fixed point multiply and divide, but hardware floating point add and subtract usually takes considerably more time then hardware fixed point add and subtract. If the computer lacks floating point hardware, all floating point computations will consume more CPU time than fixed point computations.

CHAPTER 2
FLOATING POINT FORMATS

2.1 COMMONLY USED FLOATING POINT BASES

The following three number bases are commonly used in floating point number systems:

1) Binary – The base is 2.
2) Binary Code Decimal (BCD) – The base is 10.
3) Hexadecimal – The base is 16.

2.2 COMPARISONS OF THE THREE COMMONLY USED BASES

Binary –
The main advantages of the binary floating point format are relative ease of hardware implementation and maximum accuracy for a given number of bits. On the negative side, the conversion of an ASCII (American Standard Code for Information Interchange) decimal string to and from a binary floating number is difficult and time consuming. In commercial applications where input and output are always decimal character strings, the binary floating point numbers will have an inherent rounding error because numbers such as 0.1_{10} cannot be represented exactly with a binary floating point number.

BCD –
The advantages and disadvantages of the BCD floating point numbers are just the opposite of the binary floating point numbers. BCD floating point is most commonly used in commercial applications where the computations involved are usually simple and input/output is always in the form of decimal ASCII strings.

Hexadecimal –
The hexadecimal floating point numbers have similar advantages and disadvantages as the binary floating point when compared with the BCD floating point format. When the same number of bits of exponent and mantissa are used, the hexadecimal floating point gives a considerably larger dynamic range than the binary floating point format. For example, for a 7-bit exponent, the largest positive number that can be represented in the hexadecimal floating point is approximately 16^{64} (approximately 1.16×10^{77}. The smallest non-zero positive number that can be represented is 16^{-64} (approximately 8.64×10^{-78}). By comparison, the largest and smallest positive numbers that can be represented in a 7-bit exponent binary system are approximately 1.84×10^{19} and 5.42×10^{-20} respectively.

An advantage of the hexadecimal floating point system over the binary point system is that during normalization and denormalization of the floating point numbers the hexadecimal system requires far fewer shifts compared with the binary system, because the hexadecimal system shifts four places at a time and most binary systems shift only one place at a time. For more sophisticated systems where normalization and denormalization can be done in one operation, this advantage does not exist. Most present-day systems do not fall in this category.

This disadvantage of the hexadecimal system is the loss of precision as compared with the binary system when the number of mantissa bits are the same. Since the three most significant bits could be zero when the first digit of the hexadecimal is a 1, this leads to a loss of 3 bits of accuracy in the worst case. However, assuming uniform distribution of numbers, the average loss of accuracy is only 11/15 bits. The above comparison assumes the binary system does not use an "implied 1" (Section 2.4). The loss of accuracy in a hexadecimal system compared with a binary system using an "implied 1" and same number of bits of mantissa is 4 bits in the worst case and 1 and 11/15 bits on the average.

2.3 DIFFERENT EXPONENT FORMATS

Two types of exponents used in floating point number systems are the biased exponent and the unbiased exponent. An unbiased exponent has a two's complement number. An exponent said to be biased by N (or excess N notation), means that the coded exponent is formed by adding N to the actual exponent in two's complement form. Any overflow generated from the addition is ignored. The result becomes an unsigned number. Most common floating point systems use a biased exponent. Biased exponents are used to simplify floating point hardware. During floating point computations, arithmetic operations such as add and subtract need to be performed on the exponents of the operands. If a biased exponent is used, the arithmetic logic unit (ALU) needs only to perform unsigned arithmetic. If an unbiased exponent is used, the ALU must perform two's complement arithmetic, and overflow conditions are more difficult to detect.

2.4 "IMPLIED 1"

Most floating point numbers must always be presented to the computer in "normalized" form (i.e., the most significant digit of the mantissa is always non-zero, except if the number is zero). For a binary floating point system, this would mean the leading binary bit of the mantissa is always 1 (except when the number is zero). In some floating point number systems, such as Am9512 format, this 1 bit is not represented on input or output to the floating point processor. The extra bit can be used for one more bit of precision or one more bit of exponent range.

CHAPTER 3
FLOATING POINT ARITHMETIC

3.1 INTRODUCTION

This chapter describes the basic principles of performing arithmetic with floating point numbers. First, the internal mechanism of floating point is analyzed. The following discussion uses the Am9512 single precision format although the discussion can apply to other formats with only minor modifications. The operands are assumed to be located in a stack. The first operand is called TOS (top of stack) and the second operand is called NOS (next on stack).

3.2 FLOATING POINT ADD AND SUBTRACT

Floating point add and subtract use essentially the same algorithm. The only difference is that floating point subtract changes the sign of the floating point number at top of stack and then performs the floating point add.

The following is a step-by-step description of a floating point add algorithm (Figure 3.1):

a. Unpack TOS and NOS.
b. The exponent of TOS is compared to the exponent of NOS.
c. If the exponents are equal, go to step f.
d. Right-shift the mantissa of the number with the smaller exponent.
e. Increment the smaller exponent and go to step b.
f. Set sign of result to sign of larger number.
g. Set exponent of result to exponent of larger number.
h. If sign of the two numbers are not equal, go to m.
i. Add mantissas.
j. Right-shift resultant mantissa by 1 and increment exponent of result by 1.

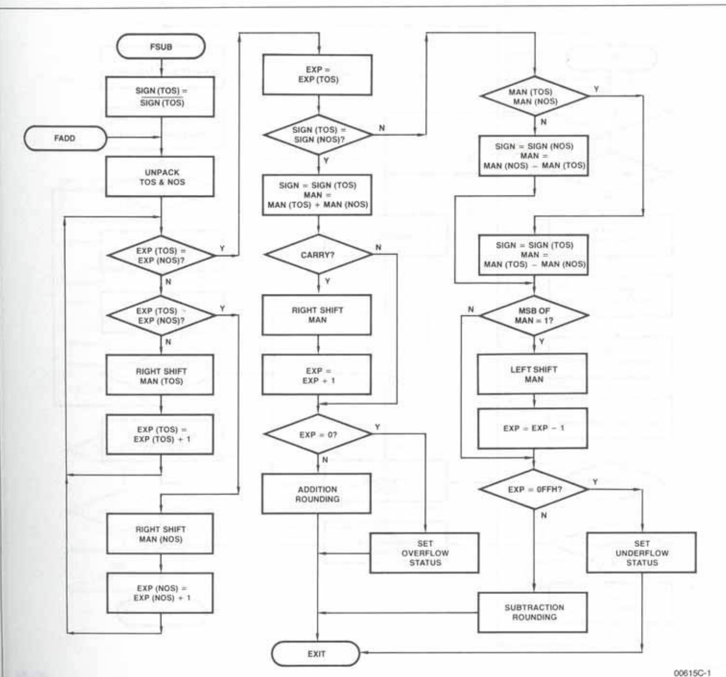


Figure 3.1. Floating Point Add/Subtract Flowchart

k. If the most significant bit (MSB) of exponent changes from 1 to 0 as a result of the increment, set overflow status.
l. Round if necessary and exit.
m. Subtract smaller mantissa from larger mantissa.
n. Left-shift mantissa and decrement exponent of result.
o. If MSB of exponent changes from 0 to 1 as a result of the decrement, set underflow status and exit.
p. If the MSB of the resultant mantissa = 0, go to n.
q. Round if necessary and exit.

3.3 FLOATING POINT MULTIPLY

Floating point multiply basically involves the addition of the exponents and multiplication of the mantissas. The following is a step-by-step description of a floating point multiplication algorithm (Figure 3.2):

a. Check if TOS or NOS = 0.
b. If either TOS or NOS = 0, Set result to 0 and exit.
c. Unpack TOS and NOS.
d. Convert EXP (TOS) and EXP (NOS) to unbiased form:
   EXP (TOS) = EXP (TOS) − 127_{10}
   EXP (NOS) = EXP (NOS) − 127_{10}
e. Add exponents:
   EXP = EXP (TOS) + EXP (NOS)
f. If MSB of EXP (TOS) = MSB of EXP (NOS) = 0 and MSB of EXP = 1, then set overflow status and exit.
g. If MSB of EXP (TOS) = MSB of EXP (NOS) = 1 and MSB of EXP = 0, then set underflow status and exit.
h. Convert exponent back to biased form:
   EXP = EXP + 127_{10}
i. If sign of TOS = sign of NOS, set sign of result to 0; otherwise, set sign of result to 1.
j. Multiply mantissas.
k. If MSB of resultant mantissa = 1, right-shift mantissa by 1 and increment exponent of resultant.
l. If MSB of exponent changes from 1 to 0 as a result of the increment, set overflow status.
m. Round if necessary and exit.

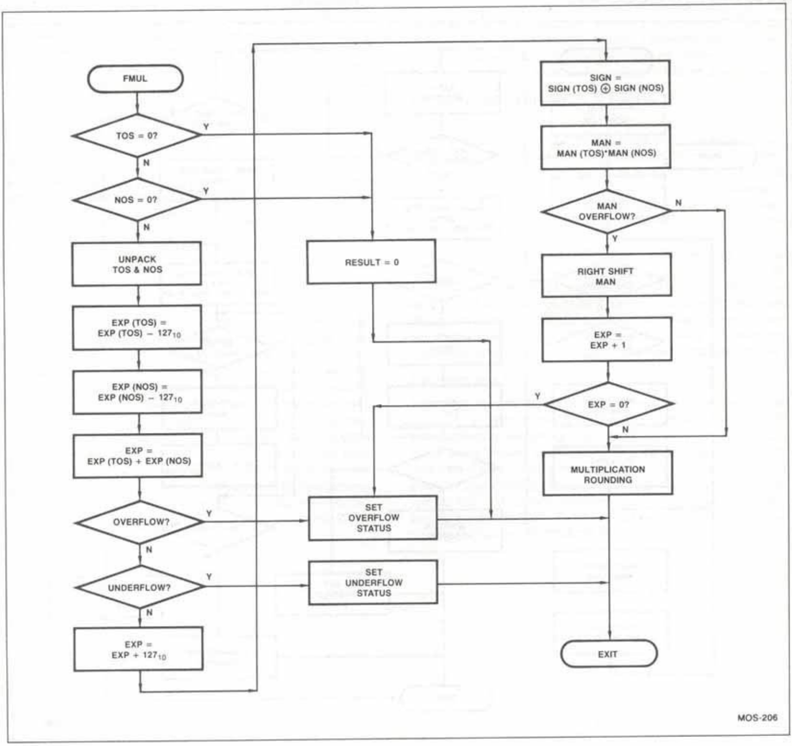


Figure 3.2. Floating Point Multiply Flowchart

3.4. FLOATING POINT DIVIDE

The floating point divide basically involves the subtraction of exponents and the division of mantissas. The following is a step-by-step description of a division algorithm (Figure 3.3):

a. If TOS = 0, set divide exception error and exit.
b. If NOS = 0, set result to 0 and exit.
c. Unpack TOS and NOS.
d. Convert EXP (TOS) and EXP (NOS) to unbiased form:
   EXP (TOS) = EXP (TOS) − 127_{10}
   EXP (NOS) = EXP (NOS) − 127_{10}
e. Subtract exponent of TOS from exponent of NOS:
   EXP = EXP (NOS) − EXP (TOS)
f. If MSB of EXP (NOS) = 0, MSB of EXP (TOS) = 1, and MSB of EXP = 1, then set overflow status and exit.
g. If MSB of EXP (NOS) = 1, MSB of EXP (TOS) = 0, and MSB of EXP = 0, then set underflow status and exit.
h. Add bias to exponent of result:
   EXP = EXP + 127_{10}
i. If sign of TOS = sign of NOS, set sign of result to 0, else set sign of result to 1.
j. Divide mantissa of NOS by mantissa of TOS
k. If MSB = 0, left-shift mantissa and decrement exponent of resultant, or else go to n.
l. If MSB of exponent changes from 0 to 1 as a result of the decrement, set underflow status.
m. Go to k.
n. Round if necessary and exit.

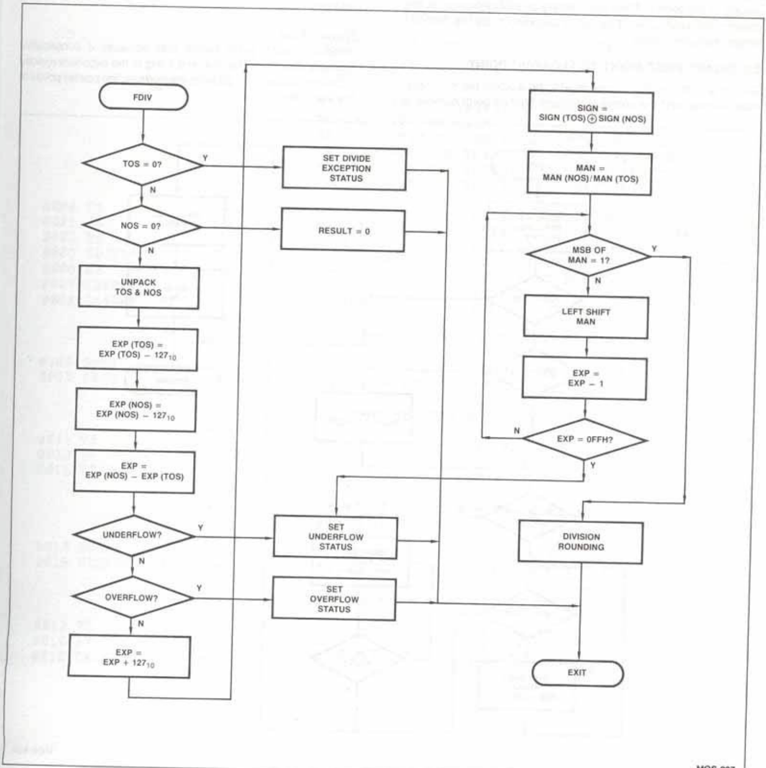


Figure 3.3. Floating Point Divide Flowchart

CHAPTER 4
DATA CONVERSION

4.1 INTRODUCTION

This chapter describes how to convert fixed point binary integer to floating point, floating point to fixed point binary integer, decimal ASCII (American Standard Code for Information Interchange) string to floating point and floating point to decimal ASCII string. These conversion methods are useful because few real-world inputs and outputs are in floating point format. When human interface is involved, the real-world interface is usually a decimal ASCII string. If the data are collected through some automatic means such as an A/D converter, counters, etc., the input is usually in the form of fixed point binary or BCD integers. In this chapter, the floating point format is assumed to be the Am9512 single precision format.

4.2 BINARY FIXED POINT TO FLOATING POINT

The input to this routine is assumed to be a 32-bit two's complement number and the output is a binary floating point number of Am9512 format. Figure 4.1 shows the flow chart of such a program and Figure 4.2 shows an Am9080A assembly language subroutine that accomplishes this task.

The data format used in the assembly language conversion is as follows:

Fixed Point –
Two's complement number that occupies 4 consecutive memory locations with the most significant byte residing in low memory. To address the number, the pointer points to the low address.

Floating Point –
Am9512 floating point format that occupies 4 consecutive memory locations. The sign and 7 bits of the exponent resides in the low address. To address the number, the pointer points to the low address.

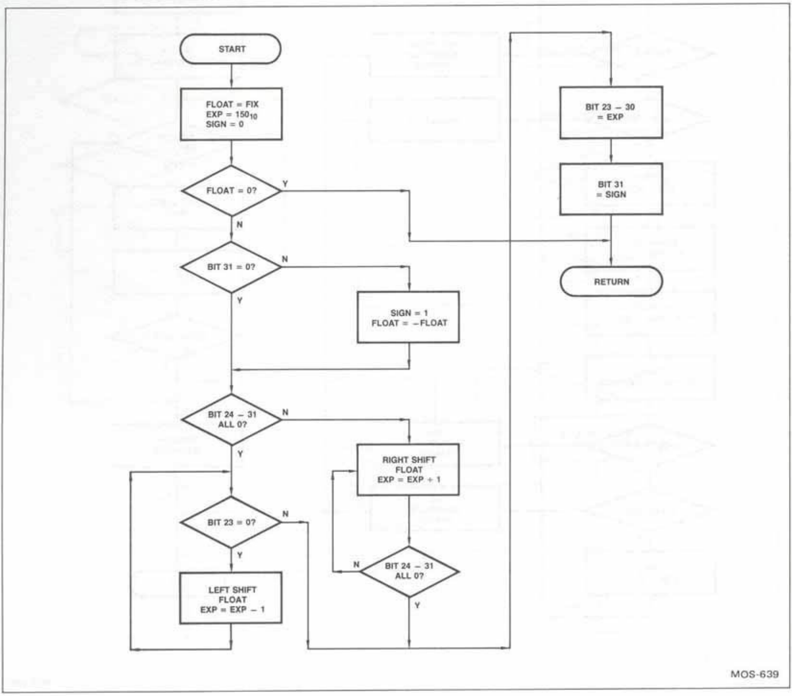


Figure 4.1. Fix to Float Conversion Flowchart

```
LOC  OBJ          LINE         SOURCE STATEMENT

                       $        PAGEWIDTH(80)   MACROFILE
                       ;
                       ;        ***********************************
                       ;
                       ;        SUBROUTINES TO CONVERT FIX TO FLOAT
                       ;          AND FLOAT TO FIX POINT FORMATS
                       ;
                       ;        ***********************************
                       ;
                                NAME CONVT
                       ;
                                PUBLIC FXTOFL,FLTOFX
                       ;
                                EXTRN QMOVE,QTEST,QNEG,QLSL,QLSR,QCLR
                       ;
                                CSEG PAGE
                       ;
                       ;        FIX TO FLOAT CONVERSION ROUTINE
                       ;        TO CALL THE PROGRAM,
                       ;        HL = POINTER TO THE FIXED POINT NUMBER
                       ;        DE = POINTER TO THE FLOATING POINT NUMBER
                       ;        ACC AND PSW ARE ALTERED BY THE SUBROUTINE
                       ;        ALL OTHER REGISTERS ARE NOT DISTURBED
                       ;
0000 C5                FXTOFL:  PUSH B          ;SAVE BC REGISTER PAIR
0001 D5                         PUSH D          ;SAVE DESTINATION POINTER
0002 E5                         PUSH H          ;SAVE SOURCE POINTER
0003 CD0000     E               CALL QMOVE      ;COPY FIXED PT NO. INTO FLOAT
0006 EB                         XCHG            ;PUT FLOAT POINTER IN HL
0007 CD0000     E               CALL QTEST      ;TEST IF NO. = 0?
000A CA4D00     C               JZ RETN         ;YES - JUMP
                       ;
                       ;        THE NUMBER IS NOT ZERO, INIT. SIGN AND EXP
                       ;
000D 0600                       MVI B,0         ;B REG = SIGN
000F 0E96                       MVI C,23+127    ;C REG = EXPONENT + BIAS
                       ;
                       ;        TEST IF THE NUMBER IS NEGATIVE
                       ;
0011 7E                         MOV A,M         ;GET MSB FROM FLOAT
0012 B7                         ORA A           ;SET FLAGS
0013 F21B00     C               JP FX10         ;JUMP IF NO. IS POSITIVE
                       ;
                       ;        THE FIXED POINT NUMBER IS NEGATIVE
                       ;        NEGATE NUMBER AND SET SIGN = 1
                       ;
0016 0680                       MVI B,80H       ;SET SIGN TO 80H
0018 CD0000     E               CALL QNEG       ;NEGATE NUMBER IN FLOAT
                       ;
                       ;        TEST IF MOST SIGNIFICANT BYTE OF FLOAT = 0
                       ;
001B 7E                FX10:    MOV A,M         ;GET MSB OF FLOAT
001C B7                         ORA A           ;SET FLAGS
001D CA2C00     C               JZ FX20         ;JUMP IF MSB = 0
```

Figure 4.2. Float to Fix Conversion Flowchart

```
LOC  OBJ                SOURCE STATEMENT

                   ;
                   ;       MSB NOT ZERO, RIGHT SHIFT REQUIRED
                   ;
0020 0C            FX15:   INR C             ;INC. EXP BY 1
0021 CD0000   E            CALL QLSR         ;LOGICAL SHIFT RIGHT OF FLOAT
0024 7E                    MOV A,M           ;TEST IF MSB = 0
0025 B7                    ORA A             ;SET FLAGS
0026 C22000   C            JNZ FX15          ;NOT ZERO, SHIFT SOME MORE
0029 C33B00   C            JMP FX30          ;ZERO, SHIFT COMPLETE
                   ;
                   ;       MSB = 0, TEST IF LEFT SHIFT REQUIRED
                   ;
002C 54            FX20:   MOV D,H
002D 5D                    MOV E,L           ;PUT FLOAT POINTER INTO DE
002E 13                    INX D             ;POINT TO NEXT MSB OF FLOAT
002F 1A            FX25:   LDAX D            ;GET NEXT MSB
0030 B7                    ORA A             ;SET FLAGS
0031 FA3B00   C            JM FX30           ;DONE IF BIT 23 = 1
0034 0D                    DCR C             ;DEC. EXP BY 1
0035 CD0000   E            CALL QLSL         ;LOGICAL LEFT SHIFT OF FLOAT
0038 C32F00   C            JMP FX25          ;TRY AGAIN
                   ;
                   ;       SHIFT COMPLETE, MANTISSA FORMED IN FLOAT
                   ;
003B 1A            FX30:   LDAX D            ;GET NEXT MSB OF FLOAT
003C E67F                  ANI 7FH           ;STRIP OFF HIDDEN "1"
003E 12                    STAX D            ;PUT IT BACK IN MEMORY
003F 79                    MOV A,C           ;GET EXPONENT
0040 0F                    RRC               ;ROTATE RIGHT
0041 4F                    MOV C,A           ;PUT ROTATED EXP. BACK IN C
0042 E680                  ANI 80H           ;EXTRACT LSB OF EXPONENT
0044 EB                    XCHG              ;PUT NEXT MSB POINTER IN HL
0045 B6                    ORA M             ;COMBINE MSB OF MANTISSA WITH EX
                   P
0046 77                    MOV M,A
0047 EB                    XCHG              ;RESTORE POINTERS
0048 79                    MOV A,C           ;GET ROTATED EXPONENT
0049 E67F                  ANI 7FH           ;STRIP OF LSB
004B B0                    ORA B             ;COMBINE EXP WITH SIGN
004C 77                    MOV M,A           ;SET MSB OF FLOAT
                   ;
                   ;       CONVERSION COMPLETE, RETURN TO CALLER
                   ;
004D E1            RETN:   POP H             ;RESTORE ALL REGISTERS
004E D1                    POP D
004F C1                    POP B
0050 C9                    RET               ;RETURN TO CALLER
                   ;
                   ;       FLOAT TO FIX CONVERSION ROUTINE
                   ;       TO CALL THE PROGRAM
                   ;       HL = POINTER TO THE FLOATING POINT NUMBER
                   ;       DE = POINTER TO THE FIXED POINT NUMBER
                   ;       ON RETURN
                   ;       A REG = 0 AND Z FLAG = 1 IF NO ERROR
                   ;       A = 1 AND Z FLAG = 0 IF OVERFLOW ERROR
```

Figure 4.2. Float to Fix Conversion Flowchart (Cont.)

```
LOC  OBJ            LINE          SOURCE STATEMENT

                        ;         OTHER REGISTERS ARE NOT DISTURBED
                        ;
0051 C5                 FLTOFX:   PUSH B         ;SAVE ALL REGISTERS
0052 D5                           PUSH D
0053 E5                           PUSH H
0054 CD0000     E                 CALL QMOVE     ;COPY FLOAT TO FIX
0057 CD0000     E                 CALL QTEST     ;TEST IF INPUT NO. = 0?
005A CAA200     C                 JZ FL40        ;RETURN IF INPUT IS ZERO
                        ;
                        ;         EXTRACT SIGN AND EXPONENT FROM FLOATING PT NO.
                        ;
005D EB                           XCHG           ;HL POINTS TO FIX
005E 7E                           MOV A,M        ;GET MSB
005F E680                         ANI 80H        ;EXTRACT SIGN BIT
0061 47                           MOV B,A        ;SAVE SIGN IN B
0062 7E                           MOV A,M        ;GET MSB AGAIN
0063 07                           RLC            ;MULTIPLY BY 2
0064 E6FE                         ANI 0FEH       ;STRIP OF LSB
0066 4F                           MOV C,A        ;SAVE IN C
0067 23                           INX H          ;POINT TO NEXT MSB
0068 7E                           MOV A,M        ;GET NEXT MSB
0069 07                           RLC            ;MOVE LSB OF EXP INTO CARRY
006A D26E00     C                 JNC $+4        ;SKIP IF NO CARRY
006D 0C                           INR C          ;PROPAGATE CARRY INTO EXP
006E 7E                           MOV A,M        ;GET NEXT MSB
006F F680                         ORI 80H        ;SET HIDDEN BIT
0071 77                           MOV M,A        ;RESTORE NEXT MSB
0072 2B                           DCX H          ;NOW HL POINTS TO MSB AGAIN
0073 3600                         MVI M,0        ;CLEAR MSB
0075 79                           MOV A,C        ;GET BIASED EXPONENT
0076 D67F                         SUI 127        ;STRIP OFF BIAS
0078 FAA700     C                 JM ZERO        ;EXP < 0, RETURN ZERO AS RESULT
007B FE1F                         CPI 31         ;CHECK IF EXP > 31
007D D2AD00     C                 JNC OVFL       ;JUMP IF NUMBER IS TOO LARGE
0080 D617                         SUI 23         ;SUBTRACT EXP BY 23
0082 CA9A00     C                 JZ FL30        ;NO SHIFT REQUIRED, CHECK SIGN
0085 4F                           MOV C,A        ;SAVE SHIFT COUNT
0086 DA9300     C                 JC FL20        ;COUNT < 0, RIGHT SHIFT
                        ;
                        ;         COUNT > 0, LEFT SHIFT REQUIRED
                        ;
0089 CD0000     E       FL10:     CALL QLSL      ;LOGICAL SHIFT LEFT
008C 0D                           DCR C
008D C28900     C                 JNZ FL10
0090 C39A00     C                 JMP FL30
                        ;
                        ;         COUNT < 0, RIGHT SHIFT REQUIRED
                        ;
0093 CD0000     E       FL20:     CALL QLSR      ;LOGICAL SHIFT RIGHT
0096 0C                           INR C
0097 C29300     C                 JNZ FL20
                        ;
                        ;         SHIFT COMPLETE, CHECK SIGN AND EXIT
                        ;
009A 78                 FL30:     MOV A,B        ;GET SIGN
```

Figure 4.2. Float to Fix Conversion Flowchart (Cont.)

```
LOC  OBJ          LINE      SOURCE STATEMENT

009B B7                          ORA A          ;SET FLAGS
009C F2A200   C                  JP FL40        ;PLUS SIGN, SKIP NEGATION
009F CD0000   E                  CALL QNEG      ;MINUS SIGN, NEGATE NUMBER
                     ;
                     ;           CLEAR ERROR FLAG AND RETURN
                     ;
00A2 AF              FL40:       XRA A
00A3 E1                          POP H          ;RESTORE ALL REGISTERS
00A4 D1                          POP D
00A5 C1                          POP B
00A6 C9                          RET
                     ;
                     ;           ZERO FIX POINT NUMBER AND RETURN
                     ;
00A7 CD0000   E      ZERO:       CALL QCLR      ;CLEAR FIX POINT NUMBER
00AA C3A200   C                  JMP FL40       ;RETURN
                     ;
                     ;           SET OVERFLOW FLAG AND RETURN
                     ;
00AD 3E01            OVFL:       MVI A,1        ;SET A REG
00AF B7                          ORA A          ;SET Z FLAG
00B0 C3A300   C                  JMP FL40+1     ;RESTORE REG. AND RETURN
                                 END

PUBLIC SYMBOLS
FLTOFX C 0051     FXTOFL C 0000

EXTERNAL SYMBOLS
QCLR   E 0000     QLSL   E 0000     QLSR   E 0000     QMOVE  E 0000
QNEG   E 0000     QTEST  E 0000

USER SYMBOLS
FL10   C 0089     FL20   C 0093     FL30   C 009A     FL40   C 00A2
FLTOFX C 0051     FX10   C 001B     FX15   C 0020     FX20   C 002C
FX25   C 002F     FX30   C 003B     FXTOFL C 0000     OVFL   C 00AD
QCLR   E 0000     QLSL   E 0000     QLSR   E 0000     QMOVE  E 0000
QNEG   E 0000     QTEST  E 0000     RETN   C 004D     ZERO   C 00A7

ASSEMBLY COMPLETE,    NO ERRORS
```

Figure 4.2. Float to Fix Conversion Flowchart (Cont.)

```
LOC  OBJ          LINE          SOURCE STATEMENT

                         $      PAGEWIDTH(80) MACROFILE
                         ;
                         ;      ********************************
                         ;
                         ;      QUADRUPLE PRECISION SUBROUTINES
                         ;
                         ;      ********************************
                         ;
                                PUBLIC QMOVE,QTEST,QNEG,QLSL,QLSR,QCLR
                         ;
                                CSEG
                         ;
                         ;      MOVE 4 BYTES POINTED TO BY HL
                         ;      TO 4 BYTES POINTED BY DE
                         ;      M(DE) = M(HL)
                         ;
0000 C5                  QMOVE: PUSH B              ;SAVE ALL REGISTERS
0001 D5                         PUSH D
0002 E5                         PUSH H
0003 0604                       MVI B,4
0005 7E                  QM10:  MOV A,M             ;GET BYTE FROM M(HL)
0006 12                         STAX D              ;STORE BYTE IN M(DE)
0007 23                         INX H               ;BUMP SOURCE POINTER
0008 13                         INX D               ;BUMP DESTINATION POINTER
0009 05                         DCR B
000A C20500    C                JNZ QM10            ;UNTIL 4 TIMES
000D E1                         POP H               ;RESTORE ALL REGISTERS
000E D1                         POP D
000F C1                         POP B
0010 C9                         RET
                         ;
                         ;      TEST 4 BYTES POINTED TO HL FOR 0
                         ;      M(HL) = 0?
                         ;
0011 E5                  QTEST: PUSH H              ;SAVE HL
0012 7E                         MOV A,M             ;GET FIRST BYTE
0013 23                         INX H
0014 B6                         ORA M               ;COMBINE WITH 2ND BYTE
0015 23                         INX H
0016 B6                         ORA M               ;COMBINE WITH 3RD BYTE
0017 23                         INX H
0018 B6                         ORA M               ;COMBINE WITH 4TH BYTE
0019 E1                         POP H               ;RESTORE HL
001A C9                         RET
                         ;
                         ;      NEGATE THE QUAD PRECISION NUMBER POINTED TO BY H
                         L
                         ;      M(HL) = - M(HL)
                         ;
001B C5                  QNEG:  PUSH B              ;SAVE BC
001C 23                         INX H               ;MOVE HL TO LSB
001D 23                         INX H
001E 23                         INX H
001F 0604                       MVI B,4
```

Figure 4.2. Float to Fix Conversion Flowchart (Cont.)

```
LOC  OBJ                   SOURCE STATEMENT

0021 B7                     ORA A              ;CLEAR CARRY
0022 3E00          QN10:    MVI A,0            ;CLEAR A WITHOUT AFFECTING CARRY
0024 9E                     SBB M
0025 77                     MOV M,A
0026 2B                     DCX H
0027 05                     DCR B
0028 C22200    C            JNZ QN10
002B 23                     INX H              ;RESTORE HL
002C C1                     POP B              ;RESTORE BC
002D C9                     RET
                   ;
                   ;        LOGICAL SHIFT LEFT 4 BYTES POINTED TO HL
                   ;        M(HL) = LSL(M(HL))
                   ;
002E C5            QLSL:    PUSH B             ;SAVE BC
002F 23                     INX H              ;MOVE POINTED TO LSB
0030 23                     INX H
0031 23                     INX H
0032 0604                   MVI B,4
0034 B7                     ORA A              ;CLEAR CARRY
0035 7E            QLSL10:  MOV A,M
0036 17                     RAL
0037 77                     MOV M,A
0038 2B                     DCX H
0039 05                     DCR B
003A C23500    C            JNZ QLSL10
003D 23                     INX H              ;RESTORE HL
003E C1                     POP B              ;RESTORE BC
003F C9                     RET
                   ;
                   ;        LOGICAL RIGHT SHIFT OF 4 BYTES POINTED TO BY HL
                   ;        M(HL) = LSR(M(HL))
                   ;
0040 C5            QLSR:    PUSH B             ;SAVE BC
0041 E5                     PUSH H             ;SAVE HL
0042 0604                   MVI B,4
0044 B7                     ORA A              ;CLEAR CARRY
0045 7E            QLSR10:  MOV A,M
0046 1F                     RAR
0047 77                     MOV M,A
0048 23                     INX H
0049 05                     DCR B
004A C24500    C            JNZ QLSR10
004D E1                     POP H              ;RESTORE HL
004E C1                     POP B              ;RESTORE BC
004F C9                     RET
                   ;
                   ;        CLEAR 4 BYTES POINTED TO BY HL
                   ;        M(HL) = 0
                   ;
0050 E5            QCLR:    PUSH H
0051 AF                     XRA A
0052 77                     MOV M,A
0053 23                     INX H
0054 77                     MOV M,A
```

Figure 4.2. Float to Fix Conversion Flowchart (Cont.)

```
LOC  OBJ          LINE        SCURCE STATEMENT

0055 23           109          INX H
0056 77           110          MOV M,A
0057 23           111          INX H
0058 77           112          MOV M,A
0059 E1           113          POP H
005A C9           114          RET
                  115 ;
                  116          END

PUBLIC SYMBOLS
QCLR   C 0050    QLSL   C 002E    QLSR   C 0040    QMOVE  C 0000
QNEG   C 001B    QTEST  C 0011

EXTERNAL SYMBOLS

USER SYMBOLS
QCLR   C 0050    QLSL   C 002E    QLSL10 C 0035    QLSR   C 0040
QLSR10 C 0045    QM10   C 0005    QMOVE  C 0000    QN10   C 0022
QNEG   C 001B    QTEST  C 0011

ASSEMBLY COMPLETE,   NO ERRORS
```

Figure 4.2. Float to Fix Conversion Flowchart (Cont.)

The following is a step-by-step description of the algorithm used in the conversion example:

a. Copy the fixed point number into the location of the floating point number.
b. Test the floating point number to see if it is zero.
c. Return to caller if the number is zero.
d. The sign is defaulted to 0 (plus).
e. Default the actual exponent to 23. This is the exponent that would be valid if no shift is required, i.e., the most significant 1 is in bit position 23. Since the Am9512 format has a bias of 127_{10} the bias is added to the default value to make the default exponent $23_{10} + 127_{10} = 150_{10}$.
f. If bit 31 in the floating point register = 1, then the input number is a negative number. The number in the floating point register is negated (two's complement negation) and the sign is set to 1.
g. If bits 24-31 of the floating point register are all zeroes, then the input number has an exponent less than or equal 23. The program transfers to step j for possible left shifts. Otherwise the program falls through to h.
h. Bits 24-31 are not all zeroes. This means the magnitude of the fixed point number is greater than 2^{23}. The floating point register is right-shifted one place and the exponent is incremented by 1.
i. Test bits 24-31 again for all zeroes. If they are not all zeroes, repeat step h. If bits 24-31 are all zeroes, shifting is complete and the program transfers to step l.
j. Bits 24-31 are all zero. If bits 23 = 1, no more shifting is required and the program transfers to step l.
k. Left-shift floating point register. Decrement exponent by 1 and repeat step j.
l. Shifting is complete. The exponent is stored into bits 23-30. (The original bit 23, the "hidden 1" is overwritten).
m. Store the sign into bit 31 of the floating point register.
n. Return to caller.

4.3 FLOATING POINT TO BINARY FIXED POINT

Figure 4.2 shows the flowchart of a floating point to fixed point conversion flowchart. An Am9080A assembly language subroutine that implements to flowchart is shown in Figure 4.3. The following is a step-by-step description of the algorithm:

a. Copy the floating point number into the fixed point register.
b. If the floating number is zero, return to caller.
c. Unpack the floating point number from the fixed point register by removing the exponent and sign. The exponent (in the unbiased form) and the sign are stored in CPU registers. The "Hidden 1" is restored in the fixed point register.
d. If exponent is less than 0, zero fixed point register and exit.
e. If exponent is larger than 31, set overflow flag and exit.
f. Subtract 23 from exponent to derive the shift count.
g. If the adjusted exponent is greater than zero, the original exponent is greater than 23, the program transfers to step j to left shift fixed point register, or else it falls through to step h.
h. If the exponent = 0, shift is complete and the program transfers to step l.
i. Right-shift the fixed point number one position and increment the exponent by 1. Repeat step h.
j. Left-shift the fixed point number by one position and decrement the exponent by 1.
k. If the exponent is not zero, repeat step j; or else, the program falls through to step l.
l. Test the original sign of the floating point number. If sign is positive skip step m.
m. If the sign is negative, negate the number in the fixed point register (two's complement).
n. Return to caller.

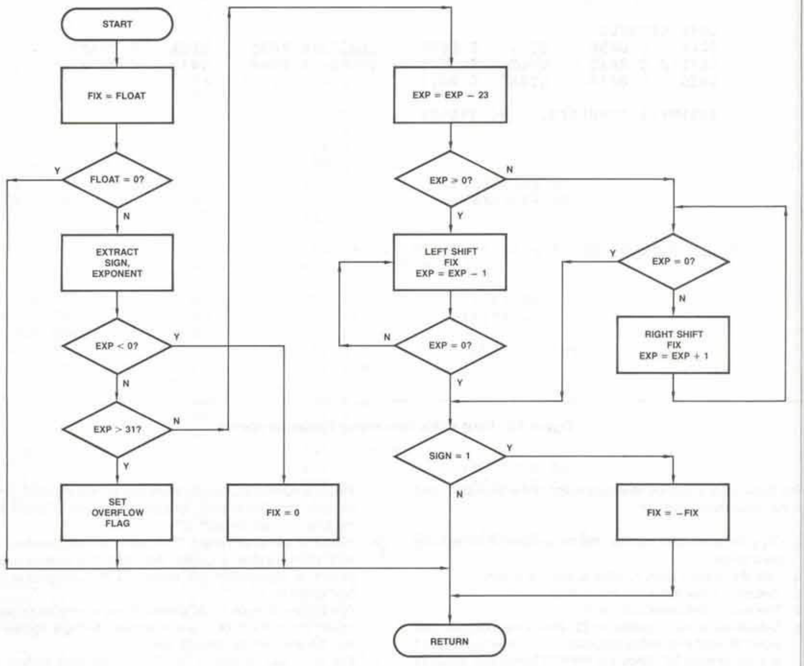


Figure 4.3. Fix to Float/Float to Fix Conversion Subroutines

4.4 DECIMAL TO BINARY FLOATING POINT CONVERSION

When a programmer works with binary floating point numbers, it is often necessary to convert decimal numbers into binary floating point notation to enter the desired numbers into the machine. Figure 4.4 shows the flowchart of such a conversion program and Figure 4.5 shows a BASIC program that does the conversion.

The program uses an array A of 32 elements. Each element of the array corresponds to one bit of the floating point number: A(31) is the sign bit, A(30) to A(23) represent the exponent and A(22) to A(0) represent the mantissa. Other variables used are as follows:

- D – The decimal number entered from console
- E – The exponent of the binary floating point number
- H – An index to the hexadecimal string with range 0-15
- H$ – An ASCII string of all hexadecimal characters used for hexadecimal output
- I – An integer used for loop index
- J – A number used for comparison when unpacking the exponent and the mantissa
- M – The mantissa of the binary floating point number

The following equation converts a floating point number from one base to another:

Let E_2 = Exponent of new number
M_2 = Mantissa of new number
B_2 = Base of new number
N_1 = Original number

Given N_1 and B_2, the equations used to solve E_2 and M_2 are:

$$E_2 = \mathrm{INT}\,(\mathrm{LOG}\,(N_1)/\mathrm{LOG}\,(B_2))$$
$$M_2 = N_1/(B_2 ** E_2)$$

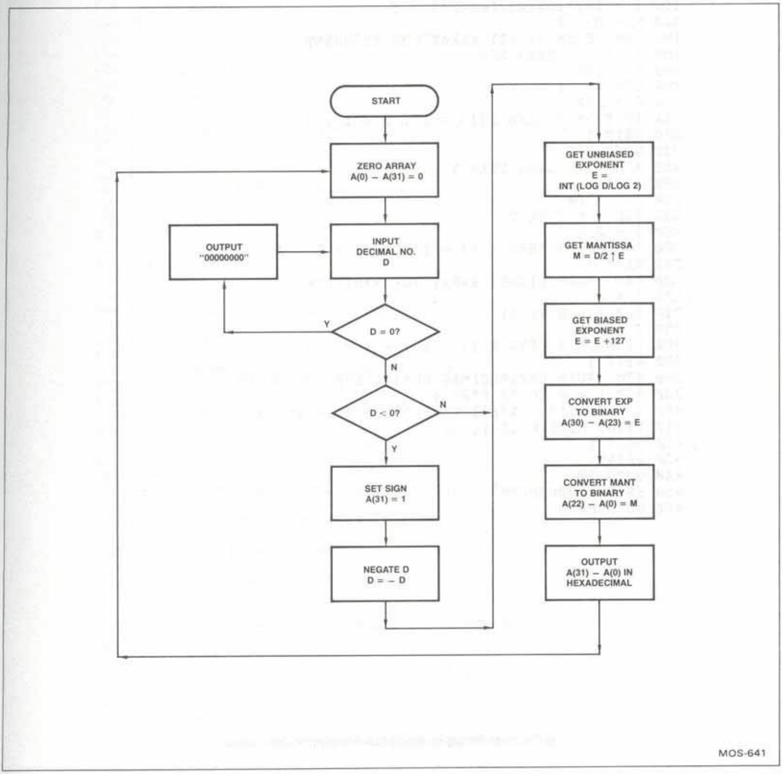


Figure 4.4. Decimal to Binary Floating Point Conversion Flowchart

```
10  REM
20  REM
30  DIM A(32)
40  H$ = "0123456789ABCDEF"
50  PRINT "INPUT DECIMAL NO. ";
60  INPUT D
70  REM    CLEAR BINARY ARRAY
80  FOR I = 0 TO 31
90  A(I) = 0
100 NEXT I
110 IF D = 0 THEN 450
120 IF D < 0 THEN A(0) = 1
130 D = ABS(D)
140 REM  FIND THE EXPONENT
150 E = INT(LOG(D)/LOG(2)) + 1
160 M = D/2^E
170 REM  FORM BINARY ARRAY FOR EXPONENT
180 IF E < 1 THEN 250
190 J = 128
200 FOR I = 1 TO 7
210 J = J/2
220 IF E >= J THEN A(I) = 1 : E = E - J
230 NEXT I
240 GOTO 320
250 REM E IS LESS THAN 1
260 A(1) = 1
270 J = - 64
280 FOR I = 2 TO 7
290 J = J/2
300 IF E >= J THEN A(I) = 1 ELSE E = E - J
310 NEXT I
320 REM  FORM BINARY ARRAY FOR MANTISSA
330 J = 1
340 FOR I = 8 TO 31
350 J = J/2
360 IF M >= J THEN A(I) = 1 : M = M - J
370 NEXT I
380 REM  FORM HEXADECIMAL NUMBER AND OUTPUT IT
390 FOR I = 0 TO 31 STEP 4
400 E = 8*A(I) + 4*A(I+1) + 2*A(I+2) + A(I+3)
410 PRINT MID$(H$,H+1,1);
420 NEXT I
430 PRINT
440 GOTO 50
450 PRINT "00000000"
460 GOTO 50
```

a) Decimal String to Am9511A Floating Point Format

Figure 4.5. Decimal to Binary Floating Point Conversion Programs

```
10  REM
20  REM
30  REM
40  REM
50  DEFINT A,I,H
60  DIM A(32)
70  H$ = "0123456789ABCDEF"
80  REM
90  REM     CLEAR BINARY ARRAY A(0) TO A(31)
100 REM
110 FOR I = 0 TO 31
120 A(I) = 0
130 NEXT I
140 REM
150 REM     INPUT A DECIMAL NUMBER FROM CONSOLE
160 REM
170 PRINT
180 INPUT "ENTER DECIMAL NUMBER";D
190 REM
200 REM     CHECK IF INPUT NUMBER IS ZERO
210 REM
220 IF D <> 0 THEN 280
230 PRINT "00000000"
240 GOTO 180
250 REM
260 REM     INPUT IS NOT ZERO, CHECK IF IT IS NEGATIVE
270 REM
280 IF D < 0 THEN A(31) = 1 : D = -D
290 REM
300 REM     FIND THE UNBIASED EXPONENT
310 REM
320 E = INT(LOG(D)/LOG(2))
330 REM
340 REM     FIND THE MANTISSA
350 REM
360 M = D/2^E
370 REM
380 REM     FIND THE BIASED EXPONENT
390 REM
400 E = E + 127
410 REM
420 REM     FORM BINARY ARRAY FOR EXPONENT
430 REM
440 J = 256
450 FOR I = 30 TO 23 STEP - 1
460 J = J/2
470 IF E >= J THEN A(I) = 1 : E = E - J
480 NEXT I
490 REM
500 REM     FORM BINARY ARRAY FOR MANTISSA
510 REM
520 M = M - 1 : REM  STRIP OFF "HIDDEN 1"
530 J = 1
540 FOR I = 22 TO 0 STEP -1
550 J = J/2
560 IF M >= J THEN A(I) = 1 : M = M - J
570 NEXT I
580 REM
590 REM     FORM HEXADECIMAL NUMBER AND OUPUT TO CONSOLE
600 REM
610 FOR I = 31 TO 0 STEP -4
620 H = 8*A(I) + 4*A(I-1) + 2*A(I-2) + A(I-3)
630 PRINT MID$(H$,H+1,1);
640 NEXT I
650 GOTO 110
```

b) Decimal String to Am9512 Floating Point Format

Figure 4.5. Decimal to Binary Floating Point Conversion Programs (Cont.)

```
10  REM
20  REM
30  REM
40  REM
50  DEFINT H,I,S : DIM H(8)
60  REM
70  REM    INPUT BINARY FLOATING POINT IN HEXADECIMAL
80  REM
90  INPUT "ENTER AN 8 DIGIT HEXADECIMAL NUMBER";H$
100 REM
110 REM    UNPACK HEXADECIMAL NUMBER INTO A BINARY ARRAY
120 REM
130 FOR I = 0 TO 7
140 C$ = MID$(H$,I+1,1)
150 H(I) = ASC(C$)
160 IF (H(I) < 48 OR H(I) > 70) THEN 530
170 IF (H(I) > 57 AND H(I) < 65) THEN 530
180 H(I) = H(I) - 48
190 IF H(I) > 9 THEN H(I) = H(I) - 7
200 NEXT I
210 REM
220 REM    FIND THE SIGN OF THE NUMBER
230 REM
240 S = 0
250 IF H(0) > 7 THEN S = 1
260 REM
270 REM    FIND THE EXPONENT OF THE NUMBER
280 REM
290 E = 32*(H(0) AND 7) + 2*H(1) + (H(2) AND 8)/8 - 127
300 REM
310 REM    FIND THE MANTISSA OF THE NUMBER
320 REM
330 H(2) = H(2) AND 7
340 M = 1
350 FOR I = 2 TO 7
360 M = M + H(I)/2^(3+4*(I-2))
370 NEXT I
380 REM
390 REM    FIND THE NUMBER BY COMBINING EXPONENT & MANTISSA
400 REM
410 N = (2^E) * M
420 REM
430 REM    CHECK SIGN TO SEE IF NEGATION REQUIRED
440 REM
450 IF S = 1 THEN N = -N
460 REM
470 REM    OUTPUT DECIMAL NUMBER
480 REM
490 PRINT N : GOTO 90
500 REM
510 REM    ILLEGAL INPUT DETECTED, ABORT
520 REM
530 PRINT "INPUT ERROR, UNKNOWN CHARACTER '";C$;"'" : GOTO 90
```

b) Hexadecimal Floating Point

Figure 4.5. Binary to Decimal Floating Point Conversion Program

```
10  REM
20  REM
30  REM
40  DEFINT A,I
50  DEFDBL B-H,J-Z
60  DIM A(64)
70  H$ = "0123456789ABCDEF"
80  INPUT "ENTER DECIMAL NUMBER";D
90  REM    CLEAR BINARY ARRAY
100 FOR I = 0 TO 63
110 A(I) = 0
120 NEXT I
130 IF D = 0 THEN 540
140 IF D < 0 THEN A(0) = 1
150 D = ABS(D)
160 REM  FIND THE UNBAISED EXPONENT
170 E = INT(LOG(D)/LOG(2))
180 REM USE ITERATIVE LOOP TO FIND 2^E BECAUSE
190 REM  EXPONENTIATION IS NOT EXACT T = 2^E
200 T = 1
210 IF E = 0 THEN 320
220 IF E > 0 THEN 280
230 REM  THE EXPONENT IS NEGATIVE
240 FOR I = -1 TO E STEP -1
250 T = T/2
260 NEXT I
270 GOTO 320
280 FOR I = 1 TO E
290 T = 2*T
300 NEXT I
310 REM  FIND THE MANTISSA AND BIASED EXPONENT
320 M = D/T
330 E = E + 1023
340 REM  FORM BINARY ARRAY FOR EXPONENT
350 J = 2048
360 FOR I = 1 TO 11
370 J = J/2
380 IF E >= J THEN A(I) = 1 : E = E - J
390 NEXT I
400 REM  FORM BINARY ARRAY FOR MANTISSA
410 M = M - 1#
420 J = 1
430 FOR I = 12 TO 63
440 J = J/2
450 IF M >= J THEN A(I) = 1 : M = M - J
460 NEXT I
470 REM  FORM HEXADECIMAL NUMBER AND OUTPUT IT
480 FOR I = 0 TO 63 STEP 4
490 H = 8*A(I) + 4*A(I+1) + 2*A(I+2) + A(I+3)
500 PRINT MID$(H$,H+1,1);
510 NEXT I
520 PRINT
530 GOTO 80
540 PRINT "0000000000000000"
550 GOTO 80
```

c) Decimal String to Am9512 Floating Point — Double Precision Format

Figure 4.5. Decimal to Binary Floating Point Conversion Programs (Cont.)

```
10  REM
20  REM
30  DEFDBL A-G,K-Z
35  DEFINT I,J
40  DIM C(16)
50  INPUT "INPUT 16 DIGIT HEXADECIMAL NUMBER ";H$
60  REM  UNPACK HEXADECIMAL NUMBER INT A BINARY ARRAY
70  FOR I = 0 TO 15
80  C$ = MID$(H$,I+1,1)
90  C(I) = ASC(C$) - 48
100 IF C(I) < 0 THEN 290
110 IF C(I) > 10 THEN C(I) = C(I) - 7
120 IF C(I) > 15 THEN 290
130 NEXT I
140 REM  FIND SIGN OF NUMBER
150 S = 0
160 IF C(0) > 7 THEN S = 1
170 REM FIND EXPONENT OF NUMBER
180 E = 256*(C(0) AND 7) + 16*C(1) + C(2) - 1023
190 REM  FIND MANTISSA OF NUMBER
200 C(2) = C(2) AND 7
210 M = 1
220 FOR I = 3 TO 15
230 M = M + C(I)/2^(4*(I-2))
240 NEXT I
250 N = (2^E) * M
260 IF S = 1 THEN N = -N
270 PRINT N
280 GOTO 50
290 PRINT "INPUT ERROR"
300 GOTO 50
```

c) Double Precision Decimal Number

Figure 4.5. Binary to Decimal Floating Point Conversion Program (Cont.)

4.5 BINARY TO DECIMAL FLOATING POINT CONVERSION

In order to read the value of a binary floating point number stored in a computer, it is often useful to convert it to a decimal number so a person can visualize the number. The conversion from binary to decimal is somewhat simpler than from decimal to binary. The following is an algorithm to convert a binary number into a decimal number:

a. Unpack the binary floating point number into sign (S), unbiased exponent (E) and mantissa (M).
b. Obtain the decimal value of the exponent using an integer binary to decimal conversion routine.
c. Obtain the decimal value of the mantissa using a fractional binary to decimal conversion routine.
d. Obtain the decimal value using

$(-1)^S \times 2^E \times M$

The flowchart in Fig. 4.6 and the basic program in Fig. 4.7 illustrate an example of such a conversion. The following is a description of the variables used in the basic program:

C$	– A single ASCII character used during unpacking of the input string.
E	– The exponent of the binary floating point number.
H(0)-H(7)	– Each element of the array represents the value of each hexadecimal ASCII character entered. That is, each element has the value 0 to 15.
H$	– The input string, which should be an 8-digit hexadecimal number. Characters entered after the eighth character are ignored.
I	– An integer used for loop index.
M	– The mantissa of the binary floating point number.
N	– The decimal floating point number.

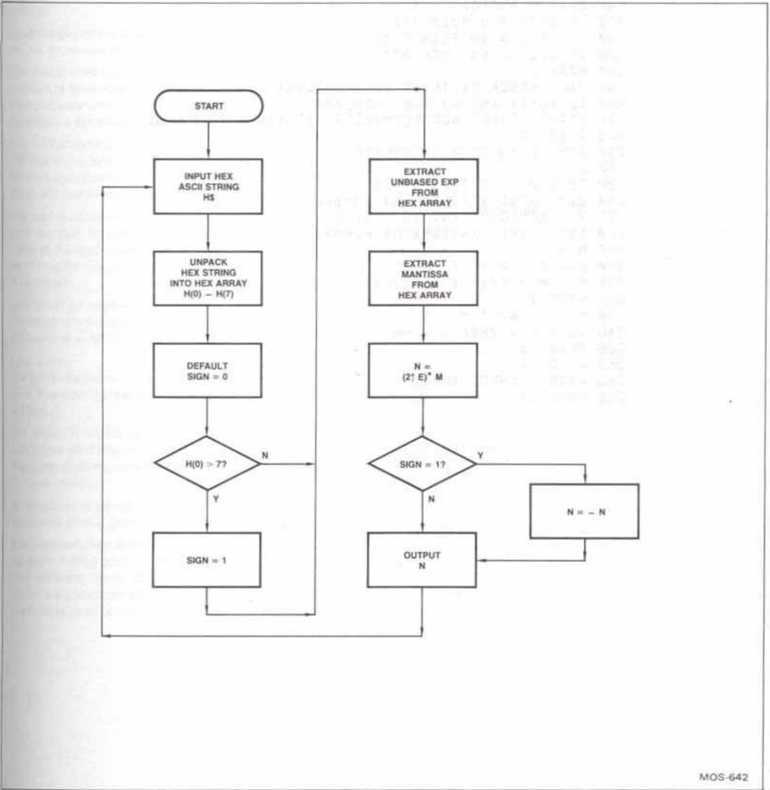


Figure 4.6. Binary to Decimal Floating Point Conversion Flowchart

```
10  REM
20  REM
30  REM
40  DIM C(8)
50  PRINT "INPUT 8 DIGIT HEXADECIMAL NUMBER: ";
60  INPUT H$
70  REM  UNPACK HEXADECIMAL NUMBER INTO BINARY ARRAY
80  FOR I = Ø TO 7
90  C$ = MID$(H$,I+1,1)
100 REM  CHECK IF INPUT IS ZERO
110 IF H$ <> "ØØØØØØØØ" THEN 14Ø
120 PRINT "Ø"
130 GOTO 5Ø
140 C(I) = ASC(C$) - 48
150 IF C(I) < Ø THEN 37Ø
160 IF C(I) > 1Ø THEN C(I) = C(I) - 7
170 IF C(I) > 15 THEN 37Ø
180 NEXT I
190 REM  CHECK IF INPUT IS NORMALIZED
200 IF (C(2) AND 8) > Ø THEN 23Ø
210 PRINT "INPUT NOT NORMALIZED FLOATING POINT NO."
220 GOTO 5Ø
230 REM  FIND SIGN OF NUMBER
240 S = Ø
250 IF C(Ø) > 7 THEN S = 1
260 REM  FIND EXPONENT OF NUMBER
270 E = 16*(C(Ø) AND 7) + C(1)
280 REM  FIND MANTISSA OF NUMBER
290 M = Ø
300 FOR I = 2 TO 7
310 M = M + C(I)/2^(4*(I-1))
320 NEXT I
330 N = (2^E) * M
340 IF S = 1 THEN N = -N
350 PRINT N
360 GOTO 5Ø
370 PRINT "INPUT ERROR"
380 GOTO 5Ø
```

Figure 4.7. Binary to Decimal Floating Point Conversion Programs

CHAPTER 5
SINGLE-CHIP FLOATING POINT PROCESSORS

5.1 INTRODUCTION

Until recently, floating point computation has been implemented either in software or in hardware with MSI/SSI (medium-scale integration/small-scale integration) devices. The former method involves considerable programming effort and the resulting product is usually very slow. It also consumes valuable main memory space for the floating point routines. The latter method involves using hundreds of ICs, which requires considerable development effort, and the resulting product is expensive to manufacture and requires considerable power and space. With the advent of LSI (large-scale integration) technology in recent years, it becomes possible to put a complete hardware floating point processor into a single IC.

The advantages of the single-chip LSI floating point processor compared to previous hardware implementation are as follows:

Low development cost –
The cost of developing an interface to a single-chip floating point processor should be less than 10 percent of the cost of developing a complete hardware floating point processor.

Low production cost –
The cost of producing and testing of hardware floating point boards is at least several hundred dollars whereas the cost of a single-chip processor is only a small fraction of that cost.

Improved reliability –
Most electronic failures occur at the interface level. By combining all the logic inside a single device, the number of connections in the system is drastically reduced. Hence reliability is increased.

Less power consumption –
The single-chip processor typically draws less than 5 percent of the power of an MSI/SSI implementation.

Less space –
The single-chip processor usually fits on the same board as the CPU, thus requiring one or two fewer boards than the MSI/SSI solution.

Get product to market sooner –
Due to less effort required both for development and production, using single-chip processors will shorten the design cycle of a new product.

The advantages of the single-chip LSI floating point processor over software floating point computation methods are:

Enhanced execution speed –
Hardware floating point processors typically execute floating point arithmetic five to 50 times faster than software. If the floating point processor allows concurrent CPU execution, the overall throughput is even further enhanced for applications where the CPU can do other meaningful tasks during a floating point computation.

Low development cost –
The cost of developing a comprehensive software floating point package often involves many manmonths of programming effort. With a hardware processor, programming is drastically reduced because the floating point computation algorithm is precoded inside the hardware processor.

Less main memory required –
Since the floating point processors contain the computation algorithm on chip (often in microcode), it could save a few thousand bytes of main memory. This should be important in applications where CPU has limited addressing space.

Improved portability –
With the advent of new microprocessors in rapid frequency, software often must be rewritten when upgrading from one CPU to another. When using the hardware processors, rewriting the floating point routines is eliminated.

The first LSI single-chip floating processors available commercially were introduced by Advanced Micro Devices. AMD introduced the Am9511 Arithmetic Processor unit in 1977 and the Am9512 Floating Point Processor unit in 1979.

5.2 Am9511A ARITHMETIC PROCESSOR

This pioneer single-chip arithmetic processor interfaces with most popular 8-bit microprocessors such as Am9080A, Am8085, MC6800 by Motorola and Z80 by Zilog. It can also be used for 16-bit microprocessors such as AmZ8000,* but its performance with such 16-bit microprocessors is somewhat hindered by its 8-bit external data bus.

Although the external interface is only 8 bits wide, the Am9511A internally is a 16-bit microprogrammed, stack-oriented floating point machine. It includes not only floating point operations but fixed point as well. In addition to the basic add, subtract, multiply and divide operations, transcendental derived functions are also included. A data sheet of Am9511A is included in Appendix A.

5.3 Am9512 FLOATING POINT PROCESSOR

The Am9512 is a follow-up to the Am9511A. Although the hardware interface between the two chips is similar, the data formats are different.

The Am9512 supports two data types: 32-bit binary floating point and 64-bit binary floating point. The formats adopted are compatible with one of the proposed IEEE formats. Unlike the Am9511A, the Am9512 does not have any of the derived transcendental functions. A description of the Am9512 is included in Appendix B.

*Z8000 is a trademark of Zilog, Inc.

CHAPTER 6
SOME INTERFACE EXAMPLES

6.1 INTRODUCTION

This chapter describes examples of interfacing some of the popular microprocessors to the Am9511A and Am9512 single-chip floating point processors. The examples given are for conceptual illustration only, minor timing details may need to be modified for systems running at nonstandard clock rates.

6.2 Am9080A TO Am9511A INTERFACE

Figure 6.1 illustrates a sample interface for an Am9080A 8-bit microprocessor to an Am9511A. The system controller that interfaces to the Am9511A is an Am8238 and not an Am8228 because the $\overline{\text{IOW}}$ (or $\overline{\text{MEMW}}$) from the Am8228 will appear too late to put the Am9080A into the WAIT state. This could cause possible overwriting of Am9511A internal registers.

In the example illustrated, the $\overline{\text{CS}}$ input comes from an address comparator Am25LS2521 (8-bit comparator). Note that the chip select decoder must not be strobed with $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$, because doing so will cause $\overline{\text{CS}}$ to go LOW after $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ went LOW. The Am9511A chip select to read or write time has a minimum setup time of 0. Strobing the chip select decoder will cause the setup time to be negative and cause the Am9511A to malfunction.

Note that the Am9511 $\overline{\text{CS}}$ (but not the Am9511A) requires a high-to-low transition for every read or write cycle. This means that the address decode should be as explicit as possible to guarantee a low-to-high transition on the address decode. In Fig. 6.1, only low-order address locations are used and an Am9080A program cannot form a read/write loop in 2 bytes; a transition on the address comparator is guaranteed. If using 4-bit comparator instead of 7-bit comparator, the program could form a read/write loop in 16 bytes. If the loop memory address always coincides with the Am9511 I/O address, there will not be a transition on the comparator output and the Am9511 will not function properly. Although the Am9080A duplicates the I/O address on A_8-A_{15}, these address lines should not be used for Am9511 address decode because if the program is executing in a region where the upper 7 bits of address match the Am9511 I/O port number, no chip-select transition may occur.

The example shows an interrupt driven interface. At the end of every Am9511A operation, the END signal goes LOW. This causes the Am9080A to go into an interrupt-acknowledge sequence. Since the $\overline{\text{INTA}}$ on the Am8238 is pulled to +12V through a 1K resistor, the data bus is pulled to all 1's during the interrupt-acknowledge cycle. This generates an RST 7 instruction to the

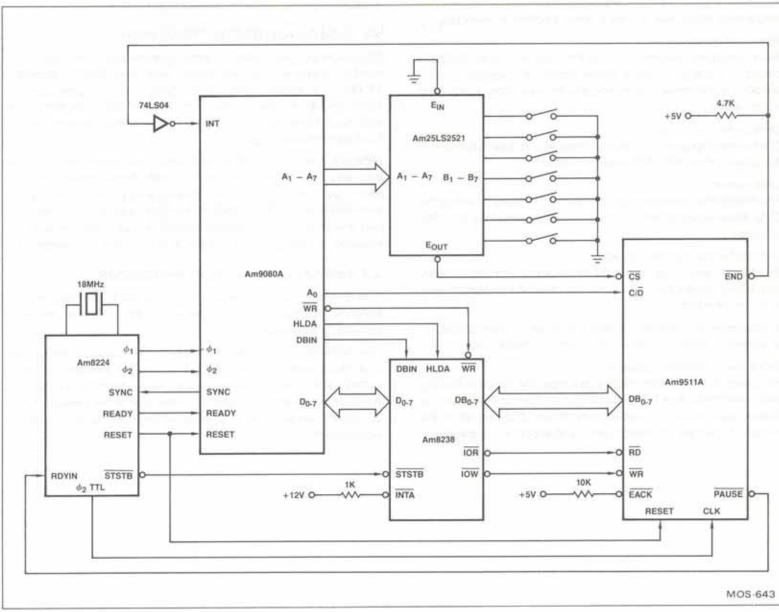


Figure 6.1. Am9080A to Am9511A Interface

Am9080A. The Am9080A stores the current program counter on the stack and jumps to location 38H to execute the interrupt handling routine. By pulling the $\overline{\text{EACK}}$ HIGH, the $\overline{\text{END}}$ output will stay LOW until the first read/write operation is performed on the Am9511A, thus clearing the interrupt request.

6.3 Am9080A TO Am9512 INTERFACE

Figure 6.2 illustrates an example of interfacing the Am9512 to the Am9080A. The principal timing difference between the Am9511A and the Am9512 is that the $\overline{\text{PAUSE}}$ follows $\overline{\text{RD}}$ or $\overline{\text{WR}}$ in the Am9511A whereas the $\overline{\text{PAUSE}}$ follows $\overline{\text{CS}}$ in the Am9512.

Two additional gates (74LS08 and 74LS32) are inserted in the $\overline{\text{PAUSE}}$ to RDYIN line. Otherwise, during a memory cycle in which the memory address bits 1 to 7 match the I/O address of the Am9512, the $\overline{\text{PAUSE}}$ will go LOW. Since there will be no $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ in that cycle to reset the $\overline{\text{PAUSE}}$, the system will be deadlocked. The additional gates allow the $\overline{\text{PAUSE}}$ to pass through only if the current cycle is an I/O cycle. Strobing the chip select decoder with $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ will not work because that will create a negative chip select to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ setup time, which is not permitted with the Am9512. Other considerations about the chip-select decoding are the same as discussed in Section 6.2.

The 74LS32 gate shown at the top of Figure 6.2 allows either END or ERR to interrupt to CPU. The CPU can read the status register of the Am9512 to determine the source of the interrupt.

6.4 Am8085A to Am9511-1 INTERFACE

In a typical Am8085A system, the system clock rate is 3MHz. The Am9511A-1 is selected because the Am9511A-1 has as a maximum clock rate of 3MHz. The Am8085A has an earlier ready setup window compared with the Am9080A. If the $\overline{\text{PAUSE}}$ signal is connected directly to the READY input to the Am8085A, the READY line will be pulled down too late for the Am8085A to go into the WAIT state. The 74LS74 is used for forcing one WAIT state when the Am9511-1 is accessed. After the first WAIT state, the 74LS74 Q output is reset to HIGH and the $\overline{\text{PAUSE}}$ of the Am9511-1 controls any additional WAIT states if necessary. The chip-state decoder is strobed with IO/$\overline{\text{M}}$ signal to prevent Am9511-1 responding to memory accesses when bits 9 to 15 of the memory address coincides with Am9511A-1 I/O address.

6.5 Am8085A TO Am9512-1 INTERFACE

The Am9512 is designed specifically to interface to Am8085A. The interface is straightforward and no additional logic is required. The Am9512-1 is used instead of Am9512 because the typical Am8085A system runs at 3 MHz.

The ERR output and END output are connected to separate interrupt inputs so that the CPU can identify the souce of interrupt without reading the status register of the Am9512-1.

Since the chip-select decoder is strobed with the IO/$\overline{\text{M}}$ signal, a transition is guaranteed with each I/O operation without the concern of insufficient address decode as in the Am9080A to Am9511A or Am9512 interfaces.

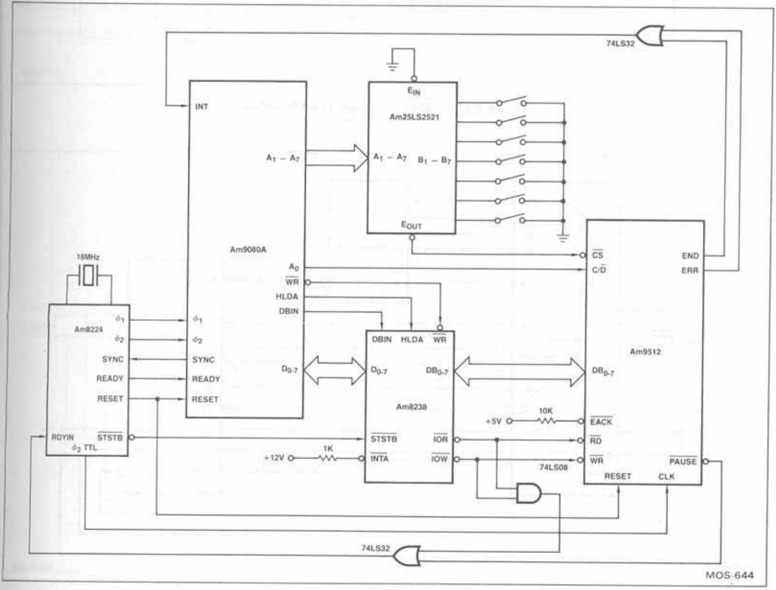


Figure 6.2. Am9080A to Am9512 Interface

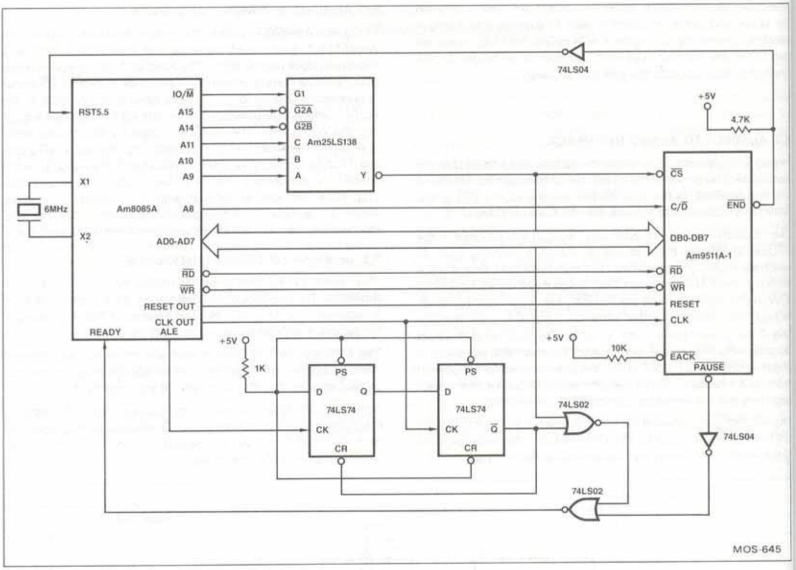


Figure 6.3. Am8085A to Am9511A-1 Interface

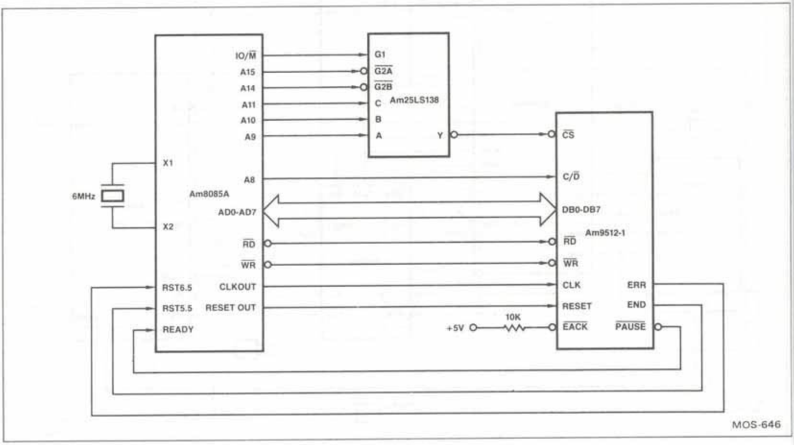


Figure 6.4. Am8085A to Am9512-1 Interface

6.6 Z80 TO Am9511A INTERFACE

Figure 6.5 illustrates a programmed I/O interface technique for Am9511A with a Z80 CPU.

The Chip Select ($\overline{CS}$) signal is a decode of Z80 address lines A1-A7. This assigns the Am9511A to two consecutive addresses, an even (Data) address, and the next higher odd (Command) address. Selection between the Data (even) and the Command/Status (odd) ports is by the least significant address bit A0.

The $\overline{IORQ}$ (Input/Output Request) from the Z80 is an enable input to the Am25LS139 decoder. The $\overline{WR}$ and $\overline{RD}$ from the Z80 are the two inputs to the decoder. The outputs Y1 and Y2 are tied to $\overline{WR}$ and $\overline{RD}$ of the Am9511A. The $\overline{PAUSE}$ output of the Am9511 is connected to $\overline{WAIT}$ line of Z80. The Am9511A outputs a LOW on $\overline{PAUSE}$ 150ns (max) after RD or WR has become active. The $\overline{PAUSE}$ remains LOW for 3.5 TCY + 50ns (min) for data read and is LOW for 1.5 TCY + 50ns (min) for status read from Am9511A where TCY is the clock period at which Am9511A is running. Therefore, Z80 will insert one to two extra WAIT states. The Am9511A $\overline{PAUSE}$ output responds to a data read, data write, or command write request received while the Am9511A is still occupied (executing a previous command) by pulling the $\overline{PAUSE}$ output LOW. Since $\overline{PAUSE}$ and $\overline{WAIT}$ are tied together, as soon as Z80 tries to interfere with APU execution, Z80 enters the WAIT state.

6.7 Z80 TO Am9512 INTERFACE

The Am9512 interface to Z80 (Fig. 6.6) requires two more gates than the Am9511A interface to Z80. An inverter is added to the interrupt request line because the sense of the END/ERR signals are different. The 74LS32 is added in the wait line because the Am9512 $\overline{PAUSE}$ will go LOW whenever chip select on the Am9512 goes LOW. In Fig. 6.6 the chip-select input can go LOW during second or third cycles of an instruction when the memory address matches the Am9512 I/O addressed. If the 74LS32 OR-gate is omitted, the WAIT input on the Z80 will go LOW and the system will be deadlocked. Strobing the chip-select decoder will not work because this would cause a negative chip select to $\overline{RD}$ or $\overline{WR}$ time on the Am9512.

The chip select decoder in this example is strobed with $\overline{M1}$. This accomplishes a dual purpose. It not only guarantees a chip select transition on every I/O cycle, it also prevents the chip select to go LOW during an interrupt acknowledge cycle. This is vital because $\overline{IORQ}$ is also LOW during that cycle. Without the $\overline{M1}$ strobe, $\overline{CS}$ might go LOW and cause $\overline{PAUSE}$ to go LOW which will again cause the system to deadlock.

6.8 MC6800 TO Am9511A INTERFACE

Figure 6.7 shows interface of a Motorola MC6800 microprocessor to an Am9511A. The MC6800 has no explicit I/O instructions. All I/O devices are treated as memory locations. Therefore the chip-select input of the Am9511A is derived from a decode of address lines A_1 to A_{15}. The decoder is strobed by VMA (Valid Memory Address) to produce a glitch-free output. The C/$\overline{D}$ input of the Am9511A is connected directly to the A_0 of the MC6800 so that the even address selects the data port and odd address selects the status or command port. The $\overline{RD}$ and $\overline{WR}$ inputs to the Am9511A is derived by demultiplexing the $\phi 2$ and VMA and the R/$\overline{W}$ signals.

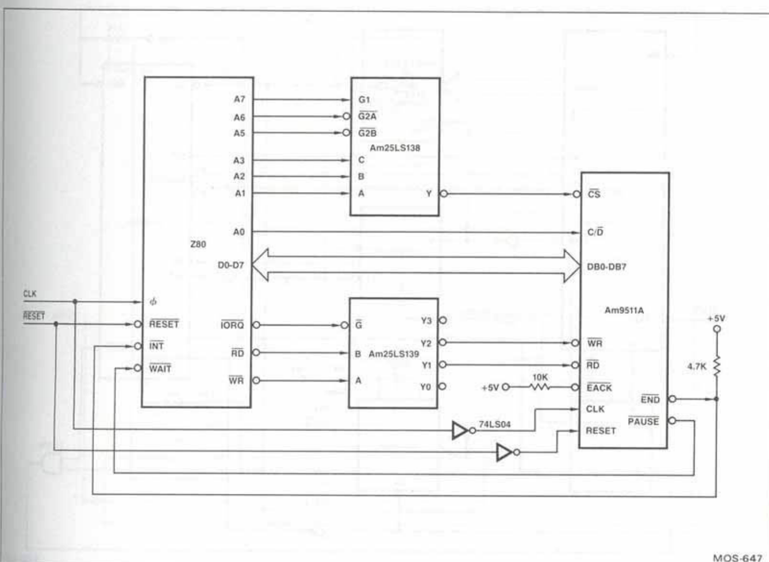


Figure 6.5. Z80 to Am9511A Interface

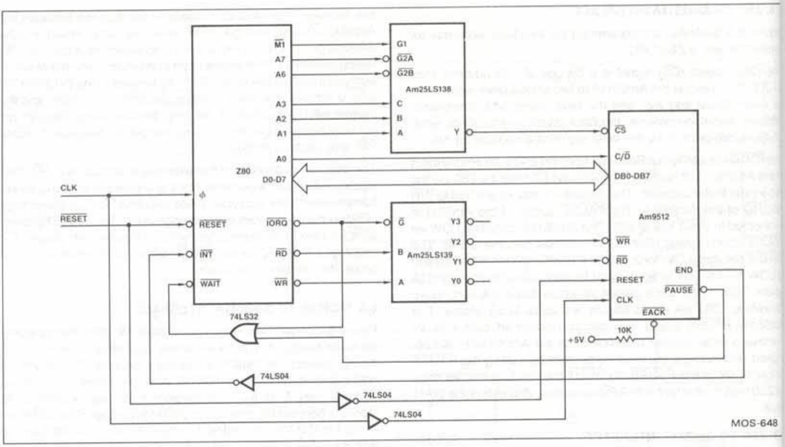


Figure 6.6. Z80 to Am9512 Interface

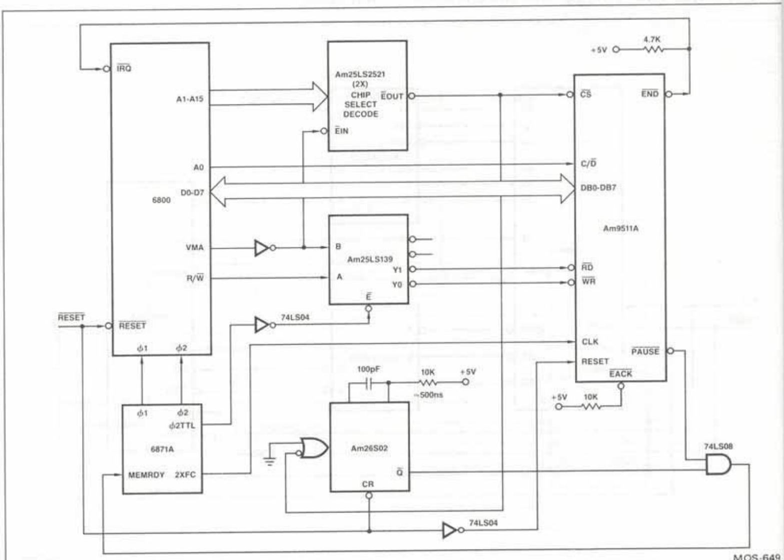


Figure 6.7. MC6800 to Am9511A Interface

The Am9511A has a relatively long read access time. To read the Am9511A data or status registers, the $\overline{RD}$ pulse to the Am9511A must be stretched and the clock to the Am9511A clock must keep running because the read access time is a function of the propagation delay and the number of clock cycles. The MC6871A clock driver chip provides a perfect solution to the problem. It has a memory ready input to stretch the $\phi 2$ HIGH time and a 2XFC free-running clock output that is not affected by memory ready input. The standard MC6800 uses a 1MHz clock so that 2XFC is at 2MHz, which is the ideal frequency for an Am9511A. When a $\overline{CS}$ to the Am9511A is decoded, the Am26S02 one-shot is triggered to pull the memory ready line LOW for approximately 500ns. This allows the $\overline{PAUSE}$ output to take control of the memory ready. The one-shot is necessary because $\overline{PAUSE}$ will not go LOW soon enough to stretch out $\phi 2$ in the current cycle.

Since the MC6800 is a dynamic device and the clock input must not be stopped for more than 5 microseconds, the programmer must not perform operations other than a status read while a current command is still in progress. This avoids producing a $\overline{PAUSE}$ output longer than 5 microseconds. The programmer should check the status register to verify that the Am9511A is not busy before performing any operation other than a status read.

6.9 MC6800 TO Am9512 INTERFACE

The MC6800 interface to Am9512 (Fig. 6.8) is somewhat simpler than the MC6800 to Am9511A interface. All the discussions in Section 6.8 also apply to this section except for the one-shot. Since the $\overline{PAUSE}$ output from the Am9512 follows the $\overline{CS}$ instead of $\overline{RD}$ or $\overline{WR}$, the memory ready signal can be directly driven by the $\overline{PAUSE}$ output. The only other addition is the inverter between the END output of the Am9512 to the $\overline{IRQ}$ input.

The software considerations concerning the possibility of excessive $\overline{PAUSE}$ time discussed in the previous section also apply to the Am9512 interface.

6.10 AmZ8002 TO Am9511A INTERFACE

The Am9511A can also be interfaced to a 16-bit microprocessor such as the AmZ8002. Since the data bus of the Am9511A is only 8 bits wide, the operations performed must be byte-oriented.

The $\overline{RD}$ and $\overline{WR}$ inputs to the Am9511A can be obtained by demultiplexing the data strobe ($\overline{DS}$) output of the AmZ8002. The data bus of the Am9511A can be connected to either the upper 8 bits or the lower 8 bits of the AmZ8002 data bus. If the Am9511A data bus is connected to the upper 8 bits (Fig. 6.9), the I/O address of the Am9511A is always even. If the Am9511A data bus is connected to the low 8 bits, the I/O address is always odd. The chip select is derived from a decode of A_2 to A_{15}. A_1 is used to select between data/status during READ and data/command during WRITE.

Due to the long READ access time of the Am9511A, the AmZ8002 must be put in a WAIT state for each READ access to the Am9511A. If the $\overline{PAUSE}$ output of the Am9511A is connected directly to the $\overline{WAIT}$ input of the AmZ8002, the $\overline{PAUSE}$ output will

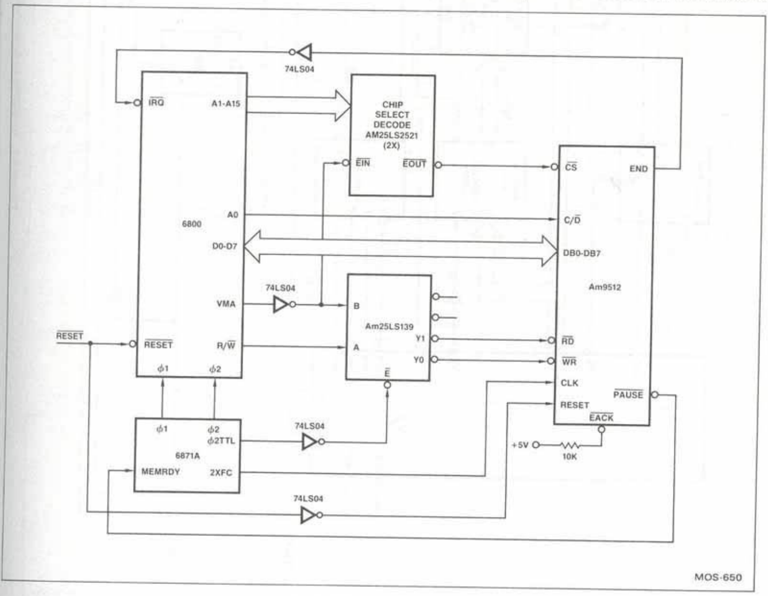


Figure 6.8. MC6800 to Am9512 Interface

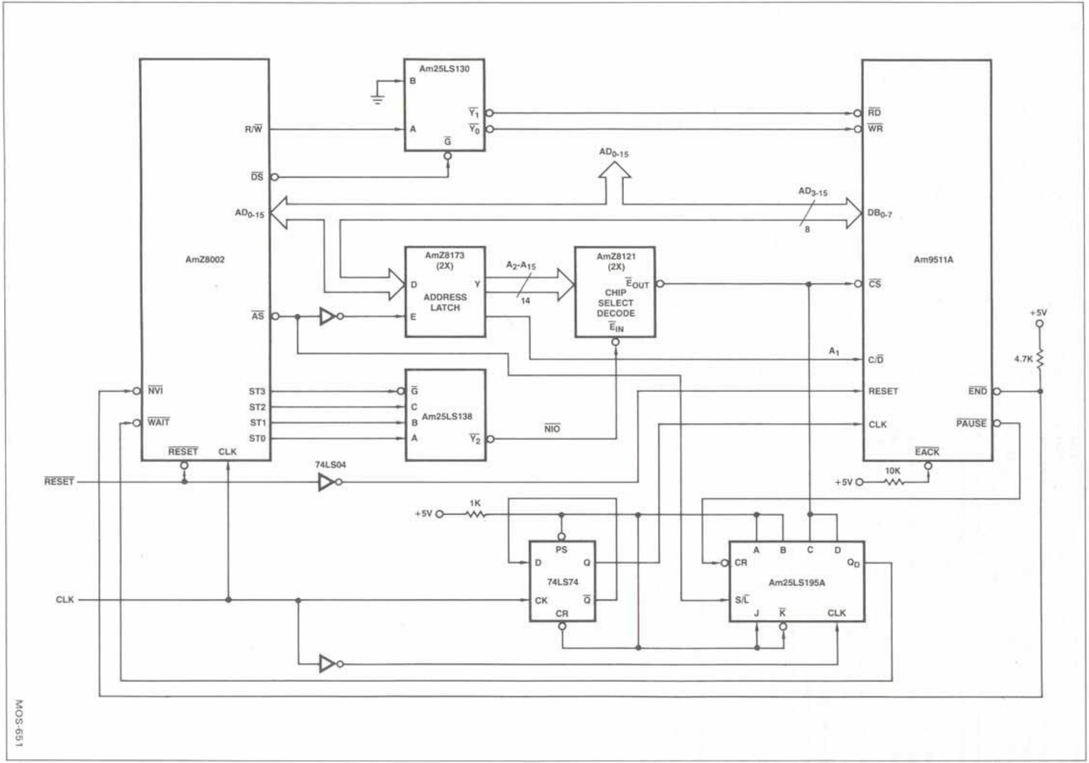


Figure 6.9. AmZ8002 to Am9511A Interface

arrive too late to put the AmZ8002 into the WAIT state. The Am25LS195A 4-bit shift register is used to solve this problem. During each address strobe, the Q_D output will be forced LOW if chip select to the Am9511A is present. The Q_D will remain LOW for two clock periods. If $\overline{PAUSE}$ is LOW during this period, the $\overline{WAIT}$ line will remain LOW because the Am25LS195A is held at the RESET state. After the $\overline{PAUSE}$ returns to high the Q_D output will go HIGH after two clocks and the AmZ8002 can proceed with the current operation. An alternative method of handling the $\overline{PAUSE}$ line is use a one shot as in Figure 6.7.

6.11 AmZ8002 TO Am9512 INTERFACE

The AmZ8002 to Am9512 interface is similar to the AmZ8002 to Am9511A interface, except the $\overline{PAUSE}$ output of the Am9512 can be connected directly to the $\overline{WAIT}$ input of the AmZ8002. This is because the $\overline{PAUSE}$ output of the Am9512 follows the chip select instead of $\overline{RD}$ or $\overline{WR}$ and the AmZ8002 has sufficient time to go into the WAIT state. Figure 6.10 illustrates interfacing the Am9512 with the AmZ8002.

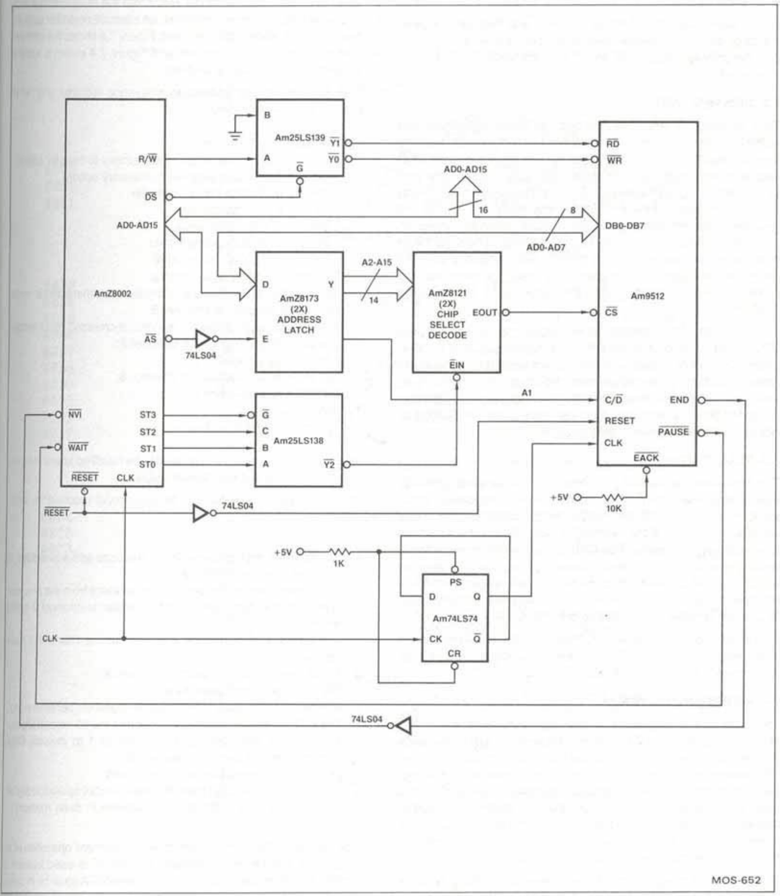


Figure 6.10. AmZ8002 to Am9512 Interface

CHAPTER 7
Am9511A INTERFACE METHODS

7.1 INTRODUCTION

Interfacing the Am9080A to the Am9511A can be accomplished in one of the following ways:

1. Demand/wait
2. Poll status
3. Interrupt driven
4. DMA transfer

The various tradeoffs of these methods are discussed below. Although only the Am9080A and Am9511A are used as an example, the principle applies to any of the processors discussed in Chapter 6.

7.2 DEMAND/WAIT

This interface is the simplest both in terms hardware and software. The connection is shown in Fig. 6.1, except that the interrupt input to the Am9080A need not be connected to the $\overline{END}$ output of the Am9511A. When this interface is used, the programmer can regard the Am9511A as always ready for READ and WRITE operations. If the Am9511A is not ready, the $\overline{PAUSE}$ will go LOW to put Am9080A in the WAIT state. When the Am9511A has completed the current operation, the $\overline{PAUSE}$ will go HIGH and the suspended READ and WRITE will proceed. Figure 7.1 shows an example of a program that loads the data into the Am9511A, executes a command and retrieves the data from the Am9511A.

The drawback of this method is that concurrent processing by the CPU is not allowed, and the CPU also cannot respond to other interrupts or DMA requests in the system while it is in the WAIT state. In systems where above considerations are not important, this would be the preferred method. This interface is not applicable to MC6800 systems because the clock of the MC6800 may not be stretched beyond 5 microseconds.

7.3 POLL STATUS

The hardware interface of this method is the same as demand/wait. The software (Fig. 7.2) is slightly more complicated. When the CPU wants to READ or WRITE to the Am9511A, the status register is first read. If the most significant bit is a 1, the Am9511A is executing a command. The CPU should refrain from performing any operation on the Am9511A except loop back for another status read. When the MSB of the status is a 0, the Am9511A has finished executing the command and the program can fall through to perform a READ or WRITE to the Am9511A.

This method does not allow the CPU to perform useful concurrent tasks, but it does allow the CPU to respond to interrupts and DMA requests when it is in the status poll loop.

7.4 INTERRUPT DRIVEN

The hardware configuration of the interrupt driven method is shown in Fig. 6.1. The CPU would first load the APU data stack and then issue a command. Dúring the command execution, the CPU would be able to perform other useful tasks in the system. When the Am9511A has finished the command, the $\overline{END}$ output goes LOW to issue an interrupt request. When the interrupt request is acknowledged by the CPU, the CPU executes a routine to fetch from the Am9511A data stack and, if necessary, load up the data stack and issue another command.

This method is most suitable for real-time multitasking systems because concurrent execution of the CPU and APU is allowed. Figure 7.3 shows an example interrupt handler for Am9511A.

7.5 DMA TRANSFER

If ultimate system performance is required, the Am9511A data stack can be loaded and unloaded by a DMA controller such as the Am9517A. To achieve maximum throughput, two channels of the Am9517A DMA controller are used in the configuration shown. Channel 2 is used to load the Am9511A and channel 3 is used to unload the Am9511A result into the main memory. For real-time interrupt driven systems, an interrupt controller such as the Am9519A should also be used. Figure 7.5 shows the connection diagram of such a system and Figure 7.4 shows a sample program to drive such a system.

The following is the initializing sequence required only after power up or system reset:

1. The Command Register
   - Bit 0 = Don't care (applies to memory to transfer option)
   - Bit 1 = Don't care (applies to memory option)
   - Bit 3 = 0, Enable DMA controller
   - Bit 4 = 0, Normal timing
   - Bit 5 = 1, Extended write
   - Bit 6 = 0, DREQ active HIGH
   - Bit 7 = 0, DACK active LOW
2. The mode register of channel 2:
   Read mode, auto initialize, address decrement, block mode
3. The mode register of channel 3:
   Write mode, auto initialize, address increment, block mode
4. The word count register of channel 2:
   Initialized to a count of 8
5. The word count register of channel 3:
   Initialized to a count of 4
6. Mask register:
   Channels 2 and 3 cleared

The word count registers may need to be modified later if the word count desired is not the default value.

The following is a sequence of operations required for each Am9511A operation:

1. The operand address is written to the base address register of channel 2 of the Am9517A.
2. If the word count of the operand is different from the previous operation, the new word count is written to channel 2 of the Am9517A.
3. The address of the result is written to the channel 3 base address register.
4. A software request is sent to channel 2.
5. The CPU performs other tasks.
6. An interrupt is received from channel 2 end of operation signal.
7. The CPU writes the command word into the command register with MSB of the command word set to 1 to indicate DMA service required at end of operation.
8. The CPU is free to perform other tasks.
9. An interrupt is received from channel 3 end of operation signal. The result is now is the desired location in main memory.

The above method offers maximum concurrent operation of an Am9080A and Am9511A system. If Am9512 is used instead of Am9511A, the mode of transfer of the Am9517A must be in single transfer mode to obtain a transition at the chip select input of the Am9512.

```
LOC  OBJ        LINE        SOURCE STATEMENT

                  $          PAGEWIDTH(80) MACROFILE NOOBJECT
                  ;
                  ;          ********************************
                  ;
                  ;              PROGRAMS FOR CHAPTER 7 OF
                  ;               FLOATING POINT TUTORIAL
                  ;
                  ;          ********************************
                  ;
                             NAME CHAP7
                  ;
                  ;          AM9511A ARITHMETIC PROCESSING UNIT
                  ;                 I/O  PORT ASSIGNMENT
                  ;
00C0              APUDR      EQU 0C0H          ;AM9511A DATA PORT
00C1              APU R      EQU APUDR+1       ;AM9511A STATUS PORT
00C1              APUCR      EQU APUSR         ;AM9511A COMMAND PORT
                  ;
                  ;          AM9517A MULTIMODE DMA CONTROLLER
                  ;                I/O  PORT ASSIGNMENT
                  ;
00B0              DMAC       EQU 0B0H          ;AM9517A BASE ADDRESS
00B4              CH2ADR     EQU DMAC+4        ;CHANNEL 2 ADDRESS
00B5              CH2CNT     EQU DMAC+5        ;CHANNEL 2 BYTE COUNT
00B6              CH3ADR     EQU DMAC+6        ;CHANNEL 3 ADDRESS
00B7              CH3CNT     EQU DMAC+7        ;CHANNEL 3 BYTE COUNT
00B8              CMD17      EQU DMAC+8        ;COMMAND REGISTER
00B9              REQ17      EQU DMAC+9        ;REQUEST REGISTER
00BB              MOD17      EQU DMAC+0BH      ;MODE REGISTER
00BD              CLR17      EQU DMAC+0DH      ;MASTER CLEAR
00BF              MSK17      EQU DMAC+0FH      ;MASK REGISTER
                  ;
                  ;          AM9519 UNIVERSAL INTERRUPT CONTROLLER
                  ;                  I/O PORT ASSIGNMENT
                  ;
00C2              UICDR      EQU 0C2H          ;AM9519 DATA PORT
00C3              UIC R      EQU UICDR+1       ;AM9519 STATUS PORT
00C3              UICCR      EQU UICSR         ;AM9519 COMMAND PORT
                  ;
                             CSEG
                  ;
                  ;          PROGRAM EXAMPLE FOR DEMAND WAIT INTERFACE
                  ;                 ***** FIGURE 7.1 *****
                  ;
                  ;          TO CALL THE FOLLOWING PROGRAM,
                  ;          ON ENTRY:
                  ;          HL = POINTER TO TPE FIRST OPERAND (NOS)
                  ;          DE = POINTER TO THE SECOND OPERAND (TOS)
                  ;          BC = POINTER TO THE RESULT
                  ;          A  = THE 2 OPERAND OPCODE
                  ;
                  ;          ON RETURN:
                  ;          A  = THE STATUS REGISTER OF AM9511A
                  ;          ALL POINTERS ARE DESTROYED
```

Figure 7.1. Demand/Wait Programming

```
LOC  OBJ                 SOURCE STATEMENT

                    ;
0000 C5             DEMAND: PUSH B              ;SAVE RESULT POINTER
0001 F5                     PUSH PSW            ;SAVE OPCODE
0002 010300                 LXI B,3
0005 09                     DAD B               ;MOVE SOURCE POINTER TO LSB
                    ;
                    ;       PUSH OPERAND #1 ONTO APU DATA STACK
                    ;
0006 0604                   MVI B,4             ;INIT LOOP1 COUNTER
0008 7E             DLOOP1: MOV A,M             ;FETCH A BYTE FROM OPER 1
0009 D3C0                   OUT APUDR           ;PUSH ONTO APU DATA STACK
000B 2B                     DCX H               ;DEC. BYTE POINTER
000C 05                     DCR B               ;DEC. LOOP COUNTER
000D C20800    C            JNZ DLOOP1
                    ;
0010 EB                     XCHG                ;PUT OPERAND 2 POINTER IN HL
0011 010300                 LXI B,3
0014 09                     DAD B               ;MOVE POINTER TO LSB
                    ;
                    ;       PUSH OPERAND #2 ONTO APU DATA STACK
                    ;
0015 0604                   MVI B,4
0017 7E             DLOOP2: MOV A,M             ;FETCH A BYTE FROM OPER 2
0018 D3C0                   OUT APUDR           ;PUSH ONTO APU DATA STACK
001A 2B                     DCX H               ;DEC. BYTE POINTER
001B 05                     DCR B               ;DEC. LOOP COUNTER
001C C21700    C            JNZ DLOOP2
                    ;
                    ;       OPERAND LOAD COMPLETE, WRITE COMMAND
                    ;
001F F1                     POP PSW             ;RETRIEVE COMMAND OPCODE
0020 D3C1                   CUT APUCR           ;WRITE TO APU COMMAND PORT
                    ;
                    ;       READ DATA FROM STACK
                    ;       IF THE APU IS NOT READY, THE PAUSE
                    ;       SIGNAL WILL PUT AM9080A INTO THE
                    ;       "WAIT" STATE UNTIL THE DATA IS READY
                    ;
0022 C1                     POP B               ;RETRIEVE RESUIT POINTER
0023 1E04                   MVI E,4             ;INIT LOOP3 COUNTER
0025 DBC0           DLOOP3: IN APUDR            ;READ APU STACK
0027 02                     STAX B              ;STORE RESULT IN MEMORY
0028 03                     INX B
0029 1D                     DCR E
002A C22500    C            JNZ DLOOP3
                    ;
                    ;       RETURN STATUS IN A
                    ;
002D DBC1                   IN APUSR
002F C9                     RET
                    $       EJECT
```

Figure 7.1. Demand/Wait Programming (Cont.)

```
LOC  OBJ          LINE      SOURCE STATEMENT

                       ;
                       ;          SUBROUTINE FOR POLL STATUS INTERFACE
                       ;                 **** FIGURE 7.2 *****
                       ;
0030 C5                POLL:      PUSH B              ;SAVE RESULT POINTER
0031 F5                           PUSH PSW            ;SAVE OPCODE
0032 010300                       LXI B,3
0035 09                           DAD B               ;MOVE POINTER TO LSB
                       ;
                       ;          CHECK IF AM9511A IS READY TO ACCEPT DATA
                       ;
0036 DBC1              CHK1:      IN APUSR            ;READ APU STATUS
0038 B7                           ORA A               ;SET CPU FLAGS
0039 FA3600     C                 JM CHK1             ;LOOP BACK IF NOT READY
                       ;
                       ;          THE AM9511A IS READ IF FALLEN THROUGH
                       ;
003C 0604                         MVI B,4             ;INIT LOOP1 COUNTER
003E 7E                PLOOP1:    MOV A,M             ;FETCH FROM OPERAND 1
003F D3C0                         OUT APUDR           ;PUSH ONTO APU DATA STACK
0041 2B                           DCX H               ;DEC. BYTE POINTER
0042 05                           DCR B               ;DEC. LOOP COUNTER
0043 C23E00     C                 JNZ PLOOP1
                       ;
0046 EB                           XCHG                ;PUT OPERAND 2 POINTER IN HL
0047 010300                       LXI B,3
004A 09                           DAD B               ;MOVE POINTER TO LSB
                       ;
                       ;          PUSH OPERAND #2 ONTO APU DATA STACK
                       ;
004B 0604                         MVI B,4             ;INIT LOOP2 COUNTER
004D 7E                PLOOP2:    MOV A,M             ;FETCH FROM OPERAND 2
004E D3C0                         OUT APUDR           ;PUSH ONTO APU DATA STACK
0050 2B                           DCX H               ;DEC. BYTE POINTER
0051 05                           DCR B               ;DEC. LOOP COUNTER
0052 C24D00     C                 JNZ PLOOP2
                       ;
                       ;          OPERANDS LOADED, WRITE COMMAND
                       ;
0055 F1                           POP PSW             ;RETRIEVE OPCODE
0056 D3C1                         OUT APUCR           ;WRITE COMMAND TO APU
                       ;
                       ;          SET UP RESULT POINTER AND LOOP3 COUNTER
                       ;
0058 C1                           POP B               ;RETRIEVE RESULT POINTER
0059 1E04                         MVI E,4             ;INIT LOOP3 COUNTER
                       ;
                       ;          WAIT UNTIL AM9511A FINISH EXECUTION
                       ;
005B DBC1              CHK2:      IN APUSR            ;READ APU STATUS PORT
005D B7                           ORA A               ;SET STATUS FLAGS
005E FA5B00     C                 JM CHK2             ;LOOP BACK IF NOT READY
0061 F5                           PUSH PSW            ;SAVE APU STATUS
                       ;
                       ;          THE AM9511A HAS FINISHED EXECUTION
```

Figure 7.2. Status Poll Programming Interface

```
LOC  OBJ                SOURCE STATEMENT

                   ;                    READ RESULT
                   ;
0062 DBC0          PLOOP3: IN APUDR          ;READ APU DATA STACK
0064 02                    STAX B            ;STORE RESULT IN MEMORY
0065 03                    INX B             ;INC. MEMORY POINTER
0066 1D                    DCR E             ;DEC. LOOP COUNTER
0067 C26200    C           JNZ PLOOP3
                   ;
                   ;       EXECUTION COMPLETE, RESTORE STATUS IN A
                   ;
006A F1                    POP PSW           ;RESTORE APU STATUS
006B C9                    RET
                   $       EJECT
```

Figure 7.2. Status Poll Programming Interface (Cont.)

```
LOC  OBJ                SOURCE STATEMENT

                        ;
                        ;       SUBROUTINES FOR INTERRUPT DRIVEN INTERFACE
                        ;                   ***** FIGURE 7.3 *****
                        ;
                        ;       LOCATE INTERRUPT HANDLER IN RST 7 LOCATION
                        ;
                                ASEG
0038                            ORG 38H
                        ;
0038 F5                 RST7:   PUSH PSW        ;SAVE ALL REGISTERS USED
0039 C5                         PUSH B
003A E5                         PUSH H
003B 0604                       MVI B,4         ;INIT LOOP COUNTER
003D 2A0000     D               LHID RSTPTR     ;FETCH RESULT POINTER
                        ;
0040 DBC0               ILOOP1: IN APUDR        ;READ RESULT FROM APU
0042 77                         MOV M,A         ;STORE IT IN MEMORY
0043 23                         INX H           ;BUMP MEMORY POINTER
0044 05                         DCR B           ;DEC. LOOP COUNTER
0045 C24000                     JNZ ILOOP1
                        ;
                        ;       DONE, SET DONE FLAG AND RESTORE REGISTERS
                        ;
0048 3E01                       MVI A,1
004A 320200     D               STA DONE
004D E1                         POP H
004E C1                         POP B
004F F1                         POP PSW
0050 C9                         RET
                        ;
                        ;       SUBROUTINE TO LOAD APU STACK AND SEND
                        ;                   COMMAND WORD
                        ;
                        ;       CALLING SQUENCE:
                        ;       ON ENTRY  HL = POINTER TO MSB OF 8 BYTES
                        ;                      OF OPERAND
                        ;                 DE = POINTER TO 4 BYTES OF RESULT
                        ;                  A = EXECUTION OPCODE
                        ;
                        ;       ON RETURN:  ALL REGISTER ARE NOT AFFECTED,
                        ;                   DONE FLAG CLEARED.
                        ;
                                CSEG
                        ;
006C E5                 LOAD:   PUSH H          ;SAVE OPERAND POINTER
006D D5                         PUSH D          ;SAVE RESULT POINTER
006E F5                         PUSH PSW        ;SAVE OPCODE
                        ;
006F 110800                     LXI D,8         ;OPER. OFFSET, E = LOOP2 CTR
0072 19                         DAD D           ;MOVE OPERAND POINTER TO LSB
                        ;
                        ;       CHECK AM9511A STATUS
                        ;
0073 DBC1               LLOOP1: IN APUSR        ;READ AM9511 STATUS REG.
0075 B7                         ORA A           ;TEST FOR BUSY
```

Figure 7.3. Interrupt Driven Programming

```
LOC  OBJ             LINE         SOURCE STATEMENT

0076 FA7300    C                  JM LLOOP1          ;WAIT UNTIL NOT BUSY
                             ;
                             ;    LOAD AM9511 STACK
                             ;
0079 2B                      LLOOP2: DCX H           ;DEC. OPERAND POINTER
007A 7E                           MOV A,M            ;FETCH 1 BYTE OF OPERAND
007B D3C0                         CUT APUDR          ;LOAD APU DATA STACK
007D 1D                           DCR E              ;DEC. LOOP COUNTER
007E C27900    C                  JNZ LLOOP2
                             ;
0081 F1                           POP PSW            ;GET OPCODE
0082 D3C1                         OUT APUCR          ;WRITE TO APU COMMAND REG.
0084 210200    D                  LXI H,DONE
0087 3600                         MVI M,0            ;CLEAR DONE FLAG
0089 E1                           POP H              ;GET RESULT POINTER
008A 220000    D                  SHLD RSTPTR        ;STORE IN RESULT POINTER
008D EB                           XCHG               ;RESTORE DE REG. PAIR
008E E1                           P P H              ;RESTORE HL
008F C9                           RET
                             ;
                             ;    RAM AREA
                             ;
                                  DSEG
                             ;
0000                         RSTPTR: DS 2            ;RESULT POINTER
0002                         DONE:   DS 1            ;DONE FLAG, 1 = DONE
                             $    EJECT
```

Figure 7.3. Interrupt Driven Programming (Cont.)

```
LOC  OBJ          LINE         SOURCE STATEMENT

                        ;
                        ;         HIGH PERFORMANCE INTERFACE WITH
                        ;              AM9517A AND AM9519
                        ;             **** FIGURE 7.4 ****
                        ;
                                  CSEG
                        ;
                        ;         AM9517A INITIALIZATION ROUTINE
                        ;         CALLING SEQUENCE:
                        ;         NO PARAMETERS REQUIRED ON ENTRY.
                        ;         SOURCE OPERANDS ASSUMED TO BE 8 BYTES AND
                        ;         RESLUT OPERAND ASSUMED TO BE 4 BYTES
                        ;
                        ;         ON RETURNED:  NO REGISTER AFFECTED
                        ;
0090 F5                 INIT17:   PUSH PSW         ;SAVE PSW
0091 D3BD                         OUT CLR17        ;MASTER CLEAR
0093 3E20                         MVI A,00100000B  ;LOAD COMMAND WORD
0095 D3B8                         OUT CMD17        ;WRITE TO COMMAND REG.
0097 3EBA                         MVI A,10111010B  ;LOAD CH 2 MODE WORD
0099 D3BB                         OUT MOD17        ;INIT CHANNEL 2 MODE
009B 3E97                         MVI A,10010111B  ;LOAD CH 3 MODE WORD
009D D3BB                         OUT MOD17        ;INIT CHANNEL 3 MODE
009F 3E08                         MVI A,8          ;LOAD CH 2 BYTE COUNT
00A1 D3B5                         OUT CH2CNT       ;INIT CH 2 LOW BYTE COUNT
00A3 AF                           XRA A
00A4 D3B5                         OUT CH2CNT       ;INIT CH 2 HIGH BYTE COUNT
00A6 3E04                         MVI A,4          ;LOAD CH 3 BYTE COUNT
00A8 D3B7                         OUT CH3CNT       ;INIT CH 3 LOW BYTE COUNT
00AA AF                           XRA A
00AB D3B7                         OUT CH3CNT       ;INIT CH 3 HIGH BYTE COUNT
00AD 3E03                         MVI A,00000011B  ;LOAD MASK REGISTER PATTERN
00AF D3BF                         OUT MSK17        ;INIT MASK REGISTFR
00B1 F1                           POP PSW          ;RESTORE PSW
00B2 C9                           RET
                        ;
                        ;         SUBROUTINE TO INITIALIZE AM9519
                        ;         CALLING SEQUENCE:
                        ;         ON ENTRY:  HL = STARTING ADDRESS OF WRITE
                        ;                         COMMAND SUBROUTINE
                        ;                    DE = STARTING ADDRESS OF SET
                        ;                         DONE FLAG SUBROUTINE
                        ;         ON RETURN: NO REGISTERS ARE AFFECTED
                        ;
00B3 F3                 INIT19:   DI               ;DISABLE ALL CPU INTERRUPTS
00B4 F5                           PUSH PSW         ;SAVE PSW
00B5 AF                           XRA A
00B6 D3C3                         OUT UICCR        ;SOFTWARE RESET AM9519
00B8 3E88                         MVI A,10001000B  ;MODE WORD FOR M0-M4
00BA D3C3                         OUT UICCR        ;SET M0-M4
00BC 3EC0                         MVI A,11000000B  ;SELECT AUTO CLEAR REG
00BE D3C3                         OUT UICCR
00C0 3E03                         MVI A,00000011B  ;SELECT CH 0 & 1 FOR AUTO CLR
00C2 D3C2                         OUT UICDR
00C4 3EB0                         MVI A,10110000B  ;SELECT MASK REGISTER
```

Figure 7.4. DMA Interface Programming

```
LOC  OBJ          LINE        SOURCE STATEMENT

ØØC6 D3C3          311         OUT UICCR
ØØC8 3EFC          312         MVI A,11111100B ;CLR CH Ø & 1 MASK REG.
ØØCA D3C2          313         OUT UICDR
ØØCC 3EFØ          314         MVI A,1111ØØØØB ;SEL CH Ø FOR 3 BYTES
ØØCE D3C3          315         OUT UICCR
ØØDØ 3ECD          316         MVI A,ØCDH      ;9Ø8ØA 'CALL' OPCODE
ØØD2 D3C2          317         CUT UICDR
ØØD4 7B            318         MOV A,E         ;GET CH Ø LOW ADDRESS
ØØD5 D3C2          319         OUT UICDR
ØØD7 7A            32Ø         MOV A,D         ;GET CH Ø HIGH ADDRESS
ØØD8 D3C2          321         OUT UICDR
ØØDA 3EF1          322         MVI A,11110ØØ1B ;SEL CH 1 FOR 3 BYTES
ØØDC D3C3          323         OUT UICCR
ØØDE 3ECD          324         MVI A,ØCDH      ;9Ø8ØA 'CALL' OPCODE
ØØEØ D3C2          325         OUT UICDR
ØØE2 7D            326         MOV A,L         ;GET CH 1 LOW ADDRESS
ØØE3 D3C2          327         OUT UICDR
ØØE5 7C            328         MOV A,H         ;GET CH 1 HIGH ADDRESS
ØØE6 D3C2          329         OUT UICDR
ØØE8 3EA1          33Ø         MVI A,1Ø1ØØØØ1B ;ARM AM9519
ØØEA D3C3          331         OUT UICCR
ØØEC F1            332         POP PSW         ;RESTORE PSW
ØØED FB            333         EI              ;ENABLE CPU INTERRUPTS
ØØEE C9            334         RET
                   335 ;
                   336 ;       SUBROUTINE TO PERFORM AN EXECUTION WITH
                   337 ;       8 BYTES OF OPERANDS AND 4 BYTES OF RESULT
                   338 ;       CALLING SEQUENCE:
                   339 ;       O  ENTRY:  HL = ADDRESS OF OPERANDS
                   34Ø ;                  DE = ADDRESS OF RESULT
                   341 ;                   A = OPCODE
                   342 ;       ON RETURN:  ALL REGISTERS ARE NOT AFFECTED
                   343 ;
ØØEF F5            344 EXEC:   PUSH PSW        ;SAVE OPCODE
ØØFØ 32Ø3ØØ     D  345         STA OPCODE      ;INIT OPCODE STORAGE
ØØF3 AF            346         XRA A
ØØF4 32Ø4ØØ     D  347         STA DONE2       ;CLEAR DONE FLAG
ØØF7 7D            348         MOV A,L
ØØF8 D3B4          349         OUT CH2ADR      ;INIT CH 2 LOW ADDR
ØØFA 7C            35Ø         MOV A,H
ØØFB D3B4          351         OUT CH2ADR      ;INIT CH 2 HIGH ADDR
ØØFD 7B            352         MOV A,E
ØØFE D3B6          353         CUT CH3ADR      ;INIT CH 3 LOW ADDR
Ø1ØØ 7A            354         MOV A,D
Ø1Ø1 D3B6          355         OUT CH3ADR      ;INIT CH 3 HIGH ADDR
Ø1Ø3 3EØ6          356         MVI A,ØØØØØ11ØB
Ø1Ø5 D3B9          357         OUT REQ17       ;SOFTWARE REQ TO CH 2
Ø1Ø7 F1            358         POP PSW         ;RESTORE PSW
Ø1Ø8 C9            359         RET
                   36Ø ;
                   361 ;       INTERRUPT HANDLER #1 TO WRITE COMMAND WORD
                   362 ;          TO AM9511A WHEN AM9517A HAS FINISHED
                   363 ;                 LOADING THE OPERANDS
                   364 ;
Ø1Ø9 F5            365 INTR1:  PUSH PSW        ;SAVE PSW
```

Figure 7.4. DMA Interface Programming (Cont.)

```
 LOC  OBJ          LINE        SOURCE STATEMENT

 010A 3A0300   D    366          LDA OPCODE         ;GET OPCODE
 010D D3C1          367          OUT APUCR          ;WRITE TO COMMAND REGISTER
 010F F1            368          POP PSW            ;RESTORE PSW
 0110 FB            369          EI                 ;RE-ENABLE CPU INTERRUPTS
 0111 C9            370          RET
                    371 ;
                    372 ;        INTERRUPT HANDLER #2 TO SET DONE FLAG
                    373 ;          TO INDICATE OPERATION IS COMPLETE
                    374 ;
 0112 F5            375 INTR2:   PUSH PSW           ;SAVE PSW
 0113 3E01          376          MVI A,1
 0115 320400   D    377          STA DONE2          ;SET DONE FLAG
 0118 F1            378          POP PSW            ;RESTORE PSW
 0119 FB            379          EI                 ;RE-ENABLE CPU INTERRUPTS
 011A C9            380          RET
                    381 ;
                    382 ;        RAM AREA
                    383 ;
                    384          DSEG
                    385 ;
 0003               386 OPCODE:  DS 1               ;APU OPCODE SAVE AREA
 0004               387 DONE2:   DS 1               ;DONE FLAG
                    388 ;
                    389          END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS
APUCR  A 00C1    APUDR  A 00C0    APUSR  A 00C1    CH2ADR A 00B4
CH2CNT A 00B5    CH3ADR A 00B6    CH3CNT A 00B7    CHK1   C 0036
CHK2   C 005B    CLR17  A 00BD    CMD17  A 00B8    DEMAND C 0000
DLOOP1 C 0008    DLOOP2 C 0017    DLOOP3 C 0025    DMAC   A 00B0
DONE   D 0002    DONE2  D 0004    EXEC   C 00EF    ILOOP1 A 0040
INIT17 C 0090    INIT19 C 00B3    INTR1  C 0109    INTR2  C 0112
LLOOP1 C 0073    LLOOP2 C 0079    LOAD   C 006C    MOD17  A 00BB
MSK17  A 00BF    OPCODE D 0003    PLOOP1 C 003E    PLOOP2 C 004D
PLOOP3 C 0062    POLL   C 0030    REQ17  A 00B9    RST7   A 0038
RSTPTR D 0000    UICCR  A 00C3    UICDR  A 00C2    UICSR  A 00C3

ASSEMBLY COMPLETE,    NO ERRORS
```

Figure 7.4. DMA Interface Programming (Cont.)

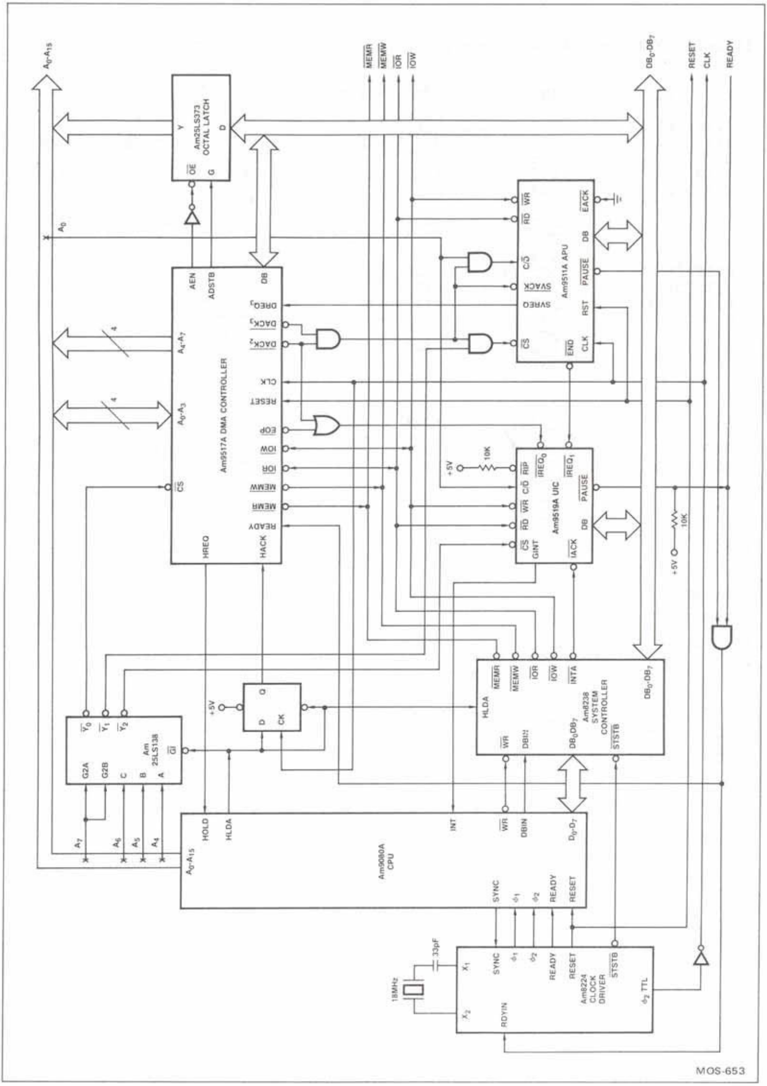


Figure 7.5. High-Performance Configuration

CHAPTER 8
FLOATING POINT EXECUTION TIMES

8.1 INTRODUCTION

This chapter offers some numerical values of comparing execution times between Am9511A, Am9512 and their software counterparts. The software packages selected are the Intel FPAL LIB(R) floating point library and the Lawrence Livermore Laboratory BASIC (LLL BASIC). These two software packages are selected because the Intel format is the same as the Am9512 single precision format and the LLL BASIC format is the same as the Am9511A floating point format. This should offer a reasonably comprehensive comparison.

In the execution-time cycles tables, the cycles given for the Am9511A and Am9512 are from the issue of the command to the completion of the command execution. The times for loading and unloading the operands are not included because these times depend on external hardware and also depend on whether the calculation is a chain calculation. Similarly, the software cycles are counted from the "Call" instruction to the "Ret" instruction of the floating point package. Operand setup time is also not counted.

The measurement is conducted on an Intel MDS 800(R) system with an Advanced Micro Computers 95/6011 APU board and 95/6012 FPU board. The host is a 2-MHz 8080A. The clock for the 95/6011 or 95/6012 board is derived from the 9.8304-MHz bus clock divided by five to achieve a frequency of 1.96608 MHz. Because the main memory of the MDS 800 is dynamic, there is approximately ±0.5% uncertainty of software timing measurements. Because the bus clock is asynchronous to the CPU clock and the internal clock of the Am9511A and Am9512 is a two-phase clock derived from the single phase bus clock, there is a ±2-clock uncertainty in the hardware measurements.

8.2 FLOATING POINT ADD/SUBTRACT EXECUTION TIMES

Floating point add and subtract usually share the same routine. Floating point subtract is merely a change of sign of the subtrahend and is performed as floating point add. For the sake of discussion in this chapter, we assume the two operands are of like signs. If the operands are different signs, the discussion about addition will apply to subtraction and vice versa.

The execution time of floating point addition is mostly dependent on exponent alignment time of the two operands, maximum of one shift would be required for post-normalization. If the addend and the augend have the same exponent, no exponent alignment time is required. If the magnitude of the addend and the augend are fairly close, only a few alignment shifts are required. If the addend and augend are very different, the number of required shifts is large, hence longer execution time.

The execution time of floating point subtraction not only has the same exponent alignment time as in the floating point addition, it also has a post-normalization time. Like floating point addition, the execution time lengthens as the magnitude of the minuend diverges from the magnitude of the subtrahend. Unlike the floating point add routine, the execution time also lengthens as the subtrahend approaches the value of the minuend. This is due to the number of left shifts required to produce a normalized result.

Table 8.1 shows the cycle times of Am9511A and LLL BASIC floating point add and subtract routines. Table 8.2 shows the cycle time of Am9512 and Intel floating point library execution times. The software execution times given have been normalized for a 2-MHz 8080A.

8.3 FLOATING POINT MULTIPLY/DIVIDE EXECUTION TIMES

Unlike floating point add or subtract, the execution times of floating point multiply or divide falls within a relatively narrow range and is not dependent on the relative magnitudes of the operands. Most multiplication algorithms use a shift and add method. For such algorithms, the execution time dependency is mainly on the number of 1's in the multiplier. The number of 1's in the multiplicand would not affect the execution time. The division execution time dependency is more complicated because of the number of division algorithms in use. In general, there is no simple way to predict the division execution time of a particular pair of operands (Tables 8.3 and 8.4).

8.4 DOUBLE-PRECISION FLOATING POINT EXECUTION TIMES

The Am9512 supports a double-precision (64-bit) floating point format. No known 64-bit floating point library routines are available at this time. Some sample execution times are given. The operands are selected over a representative range to give a comprehensive average (Tables 8.5 and 8.6).

TABLE 8.1. Am9511A vs LLL BASIC FLOATING POINT ADD/SUBTRACT EXECUTION TIME COMPARISON

OPERAND #1		OPERAND #2		AM9511		LLLBASIC	
DEC.	HEX.	DEC.	HEX.	FADD	FSUB	FADD	FSUB
5	03A00000	.0006	769D4951	214	228	3395	3884
5	03A00000	.006	79C49BA4	179	192	3000	3506
5	03A00000	.06	7CF5C28E	143	156	2608	3088
5	03A00000	.6	00999999	95	108	2100	2578
5	03A00000	6	03C00000	57	91	1826	2105
5	03A00000	60	06F00000	116	120	2362	2281
5	03A00000	600	0A960000	153	169	2540	2805
5	03A00000	6000	0DBB8000	189	204	2945	3186
123	07F60000	456	09F40000	103	108	2215	2137
.123	7DFBE76C	456	09E40000	213	227	3220	3467
123	07F60000	.456	7FE978D4	154	169	2748	3241
12345	0EC0E400	67890	11849900	106	131	2030	2460
1.3579	01ADCFAA	24680	0FC0D000	238	253	3469	3727
.000012	70C9539A	340000	13A60400	344	347	4783	5025
234	08EA0000	-678	8AA98000	118	96	2605	1920
-1.234	819DF3B6	12345	0EC0E400	238	229	3890	3367
			TOTAL	2660	2828	45736	48777
			AVERAGE	166.2	176.8	2858.5	3048.6

TABLE 8.2. Am9512 vs INTEL FPAL LIB FLOATING POINT ADD/SUBTRACT EXECUTION TIME COMPARISON

OPERAND #1		OPERAND #2		AM9512		FPAL.LIB	
DEC.	HEX.	DEC.	HEX.	SADD	SSUB	FADD	FSUB
5	40A00000	.0006	3A1D4952	254	275	2351	2568
5	40A00000	.006	3BC49BA6	229	217	1914	2152
5	40A00000	.06	3D75C28F	171	178	2506	2724
5	40A00000	.6	3F19999A	98	119	1954	2178
5	40A00000	6	40C00000	58	89	1430	1734
5	40A00000	60	42700000	128	123	2002	2165
5	40A00000	600	44160000	169	177	2455	2712
5	40A00000	6000	45BB8000	212	219	1866	2159
123	42F60000	456	43E40000	114	109	1844	2036
.123	3DFBE76D	456	43E40000	264	283	2145	2424
123	42F60000	.456	3EE978D4	192	183	1651	1878
12345	4640E400	67890	47849900	114	140	1889	2279
1.3579	3FADCFAB	24680	46C0D000	300	309	2435	2715
.000012	3749539B	340000	48A60400	475	477	1953	2231
234	436A0000	-678	C4298000	124	101	2155	1911
-1.234	BF9DF3B6	12345	4640E400	284	297	2564	2284
			TOTAL	3186	3296	33114	36150
			AVERAGE	199.1	206.0	2069.6	2259.4

TABLE 8.3. Am9511A vs LLL BASIC FLOATING POINT MULTIPLY/DIVIDE EXECUTION TIME COMPARISON

OPERAND #1		OPERAND #2		AM9511		LLLBASIC	
DEC.	HEX.	DEC.	HEX.	FMUL	FDIV	FMUL	FDIV
5	03A00000	.0006	769D4951	174	157	8451	13013
5	03A00000	.006	79C49BA4	174	178	8441	12856
5	03A00000	.06	7CF5C28E	149	177	8264	12867
5	03A00000	.6	00999999	174	157	8407	13302
5	03A00000	6	03C00000	173	178	8423	12835
5	03A00000	60	06F00000	148	179	8218	12892
5	03A00000	600	0A960000	173	155	8415	12214
5	03A00000	6000	0DBB8000	175	179	8437	13020
123	07F60000	456	09E40000	148	156	8939	12713
.123	7DFBE76C	456	09E40000	148	157	10948	13373
123	07F60000	.456	7FE978D4	149	155	8965	12878
12345	0EC0E400	67890	11849900	173	157	9163	14305
1.3579	01ADCFAA	24680	0FC0D000	147	179	10591	13149
.000012	70C9539A	340000	13A60400	149	157	10018	13395
234	08EA0000	-678	8AA98000	148	156	8781	13509
-1.234	819DF3B6	12345	0EC0E400	175	178	10971	12952
			TOTAL	2577	2655	145432	209273
			AVERAGE	161.1	165.9	9089.5	13079.6

TABLE 8.4. Am9512 vs INTEL FPAL LIB FLOATING POINT MULTIPLY/DIVIDE EXECUTION TIME COMPARISON

OPERAND #1		OPERAND #2		AM9512		FPAL.LIB	
DEC.	HEX.	DEC.	HEX.	SMUL	SDIV	FMUL	FDIV
5	40A00000	.0006	3A1D4952	234	250	3206	7757
5	40A00000	.006	3BC49BA6	256	235	3252	7905
5	40A00000	.06	3D75C28F	198	247	3088	7975
5	40A00000	.6	3F19999A	234	248	3245	7708
5	40A00000	6	40C00000	220	232	3052	7955
5	40A00000	60	42700000	200	246	2897	7999
5	40A00000	600	44160000	220	248	3072	7799
5	40A00000	6000	45BB8000	220	246	3137	7853
123	42F60000	456	43E40000	201	248	2903	7820
.123	3DFBE76D	456	43E40000	199	243	3087	7834
123	42F60000	.456	3EE978D4	219	236	3072	7822
12345	4640E400	67890	47849900	242	249	3124	7585
1.3579	3FADCFAB	24680	46C0D000	253	240	3139	7854
.000012	3749539B	340000	48A60400	219	228	3131	7776
234	436A0000	-678	C4298000	201	234	2925	7721
-1.234	BF9DF3B6	12345	4640E400	223	227	3314	7852
			TOTAL	3539	3857	49644	125215
			AVERAGE	221.2	241.1	3102.8	7825.9

TABLE 8.5. Am9512 DOUBLE PRECISION ADD/SUBTRACT EXECUTION TIMES

OPERAND #1		OPERAND #2		AM9512	
DEC.	HEX.	DEC.	HEX.	DADD	DSUB
5	4014000000000000	.0006	3F43A92A30553261	1273	1310
5	4014000000000000	.006	3F789374BC6A7EF9	1174	1211
5	4014000000000000	.06	3FAEB851EB851EB8	1038	1105
5	4014000000000000	.6	3FE3333333333333	868	891
5	4014000000000000	6	4018000000000000	720	773
5	4014000000000000	60	404E000000000000	951	922
5	4014000000000000	600	4082C00000000000	1091	1107
5	4014000000000000	6000	40B7700000000000	1229	1244
123	405EC00000000000	456	407C800000000000	906	877
.123	3FBF7CED916872B0	456	407C800000000000	1233	1280
123	405EC00000000000	.456	3FDD2F1A9FBE76C8	1072	1103
12345	40C81C8000000000	67890	40F0932000000000	907	960
1.3579	3FF5B9F559B3D07C	24680	40D81A0000000000	1322	1352
.000012	3EE92A737110E453	340000	4114C08000000000	2158	2232
234	406D400000000000	-678	C085300000000000	914	861
-1.234	BFF3BE76C8B43958	12345	40C81C8000000000	1309	1290
			TOTAL	18165	18518
			AVERAGE	1135.3	1157.4

TABLE 8.6. Am9512 DOUBLE PRECISION MULTIPLY/DIVIDE EXECUTION TIMES

OPERAND #1		OPERAND #2		AM9512	
DEC.	HEX.	DEC.	HEX.	DMUL	DDIV
5	4014000000000000	.0006	3F43A92A30553261	1810	4857
5	4014000000000000	.006	3F789374BC6A7EF9	1814	4983
5	4014000000000000	.06	3FAEB851EB851EB8	1779	5048
5	4014000000000000	.6	3FE3333333333333	1841	5007
5	4014000000000000	6	4018000000000000	1785	4700
5	4014000000000000	60	404E000000000000	1751	4699
5	4014000000000000	600	4082C00000000000	1787	4618
5	4014000000000000	6000	40B7700000000000	1786	4702
123	405EC00000000000	456	407C800000000000	1750	4671
.123	3FBF7CED916872B0	456	407C800000000000	1756	4748
123	405EC00000000000	.456	3FDD2F1A9FBE76C8	1744	4936
12345	40C81C8000000000	67890	40F0932000000000	1807	4696
1.3579	3FF5B9F559B3D07C	24680	40D81A0000000000	1762	4788
.000012	3EE92A737110E453	340000	4114C08000000000	1755	4764
234	406D400000000000	-678	C085300000000000	1750	4670
-1.234	BFF3BE76C8B43958	12345	40C81C8000000000	1802	4768
			TOTAL	28479	76655
			AVERAGE	1779.9	4790.9

CHAPTER 9
TRANSCENDENTAL FUNCTIONS OF Am9511A

9.1 INTRODUCTION

The word "'transcendental" is defined as "a function that cannot be expressed by a finite number of algebraic operations." Three examples of such functions are sine, logarithmic and exponentiation. The Am9511A performs a number of such functions, and this chapter describes the algorithms adopted by the device.

9.2 CHEBYSHEV POLYNOMIALS

Computer approximations of transcendental functions are often based on some form of polynomial equations, such as

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + \ldots$$

The most well-known polynomial for evaluating transcendental functions is the Taylor series

$$f(x) = f(a) + \frac{f^k(a)\,(X - a)^k}{k!}$$

Where $f^k(a)$ is the k^{th} derivative of the function f. Taylor series usually works well when $(x - a)$ is a small number. When the value of $(x - a)$ is large, the number of Taylor series terms required to evaluate to a given accuracy becomes large. The primary shortcoming of an approximation in this form is that it typically exhibits very large errors when the magnitude of $|X|$ is large, although the errors are small when $|X|$ is small. With polynomials in this form, the error distribution is markedly uneven over any arbitrary interval. To avoid this shortcoming, there is a set of approximating functions that not only minimizes the maximum error but also provides an even distribution of errors within the selected data representation interval. These are known as Chebysheve polynomial functions and are based upon the cosine functions. The Chebyshev polynomials T(x) are defined as follows

$$T_n(x) = \cos(n\cos^{-1}x)$$

The various terms of the Chebyshev series can be computed as

$$T_0(x) = \cos(0) = 1$$
$$T_1(x) = \cos(\cos^{-1}x) = x$$
$$T_2(x) = \cos(2\cos^{-1}x) = 2\cos^2(\cos^{-1}x) - 1 = 2x^2 - 1$$

in general, the next term in the 'C' series can be recursively derived from the previous term as the following: –

$$T_n(x) = 2x(T_{n-1}(x)) - T_{n-2}(x) \text{ for } n \geq 2$$

the terms $T_3(x)$, $T_4(x)$, $T_5(x)$ and $T_6(x)$ are given below for reference

$$T_3(x) = 4x^3 - 3x$$
$$T_4(x) = 8x^4 - 8x^2 + 1$$
$$T_5(x) = 16x^5 - 20x^3 + 5x$$
$$T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1$$

It is not the intent of this book to go into the detailed derivation of the Chebyshev series. For readers interested in the formal derivation, references 1 and 3 are recommended. The Chebyshev series is given as follows:

$$f(x) = \frac{1}{2}C_0 + \sum_{n=1}^{\infty} C_nT_n(x)$$

where

$$C_n = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)\;T_n(x)}{\sqrt{1 - x^2}}\,dx$$

For a given accuracy, only a finite number of terms is required. The Am9511A selects the number of terms required by different functions to provide a mean relative error of about one part in 10^7. The coefficients C_n are all precalculated and stored in the constant ROM.

Each of the transcendental functions in the Am9511A uses the Chebyshev polynomial series except the square root function. Each function is a three-step process as follows:

Range Reduction –
The input argument of the function is transformed to fall within a range of values for which the function can be computed to a valid result. For example, since functions like sine and cosine are periodic for multiples of radians, input arguments for these functions are converted to lie within a range of

$$0 \text{ to } \pi \text{ or } -\frac{\pi}{2} \text{ to } +\frac{\pi}{2}$$

Chebyshev polynomial evaluation –
This step is the same for all functions. The algebraic sum of the appropriate number of terms of the Chebyshev series is computed.

Postprocessing –
Some functions, such as sine and cosine, need postprocessing of the result such as sign correction.

The following sections give a detailed function-by-function description of each transcendental function in the Am9511A.

9.3 THE FUNCTIONS CHEBY AND ENTIER

Two functions are used in the following sections. The first one is CHEBY. This function evaluates the Chebyshev polynomial series

$$f(x) = 1/2C_0 + \sum_{k=1}^{n-1} C_kT_k(x)$$

The function is called by CHEBY (x, c, n) where x is the input argument after any necessary preprocessing; c is the coefficient list for the given function; and n is the number of Chebyshev polynomial terms used.

The FORTRAN program to implement the cheby function is as follows:

```
     FUNCTION CHEBY (X, C, N)
     Dimension C(12), T(12)
     T(1) = 1
     T(2) = X
     CHEBY = 0.5 * X(1) + C(2) * T(2)
     DO 100 I = 3, N
     T(I) = 2 * X * T(I - 1) - T(I - 2)
100 CHEBY = CHEBY + C(I) * T(I)
```

This program is not written to minimize execution time or code space but for its clarity. A program that improves execution speed but is somewhat more obscure is as follows:

```
     FUNCTION CHEBY (X, C, N)
     DIMENSION C(12), T(12)
     B = 0
     D = C(N)
     X2 = 2 * X
     DO 100 I = N, 2, -1
     A = B
     B = D
100 D = X2 * B - A + C(I - 1)
     CHEBY = (D - A)/2
     END
```

The second function is called ENTIER. Entier is the French word for integer. The entier function is similar to the FORTRAN integer function, except the integer function rounds down to the nearest integer closer to zero whereas the entier function rounds down to the nearest integer of a lower value. In other words, if the number is greater than or equal to zero, both functions are identical. If the number is negative, such as −2.5, INT (−2.5) = −2, ENTIER (−2.5) = −3.

A FORTRAN program to implement the entier function is as follows:

```
FUNCTION ENTIER (X)
IF (X.LT.0) X = X − 1
ENTIER = INT (X)
END
```

9.4 SINE

Any argument of the sine function can be reduced to a value from $-\pi/2$ to $+\pi/2$. Hence the range reduction is

$X = X * 2/\pi$
$X = X - 4 * \text{Entier}((X + 1)/4)$
If (X.GT.1) $X = 2 - X$

This reduces the input argument to a range from −1 to +1. The Chebyshev polynomial evaluation is

$\sin(X) = X * \text{CHEBY}(2X^2 - 1, \text{Csin}, \text{Nsin})$

where Csin is an array of precalculated Chebyshev coefficients for sine, and Nsin is the number of Chebyshev polynomial series used. In the case of Am9511A

$\text{Nsin} = 6$
$\text{Csin}_0 = 2.5525579$
$\text{Csin}_1 = -0.2852616$
$\text{Csin}_2 = 9.118016 \times 10^{-3}$
$\text{Csin}_3 = -1.365875 \times 10^{-4}$
$\text{Csin}_4 = 1.184962 \times 10^{-6}$
$\text{Csin}_5 = -6.702792 \times 10^{-9}$

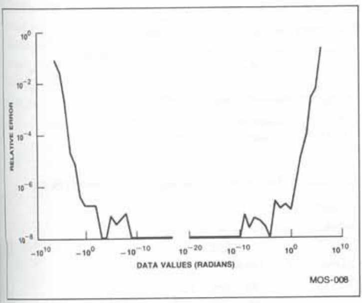


Figure 9.1. Sine

9.5 COSINE

Any argument of cosine function can be reduced to a range from 0 to π. Hence, the formulas for cosine range reduction are

$X = X * 2/\pi$
$X = 4 * \text{Entier}((X + 2)/4) - X + 1$
If (X.GT.1) $X = 2 - X$

The cosine function is now evaluated the same way as the sine function

$\cos(x) = X * \text{CHEBY}(2x^2 - 1, \text{Csin}, \text{Nsin})$

where Csin and Nsin are the same as the sine function

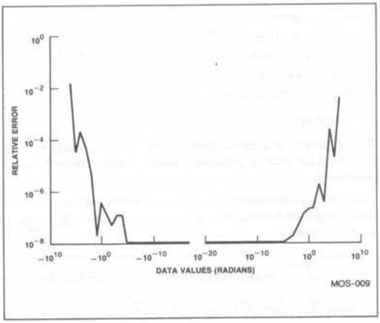


Figure 9.2. Cosine

9.6 TANGENT

Any argument for tangent can be reduced to a value from $-\pi/2$ to $+\pi/2$. This is the same range reduction algorithm as the sine function (Figure 9.4).

$X = X * 2/\pi$
$X = X - 4 * \text{Entier}((X + 1)/4)$
$Y = X$
If (Y.GT.1) $X = 2 - X$

The Chebyshev polynomial evaluation is

$\text{Tan}(X) = X * \text{CHEBY}(2X^2 - 1, \text{Ctan}, \text{Ntan})$

A postprocessing step is also required

If (Y.GT.1) $\text{Tan}(X) = 1/\text{Tan}(X)$

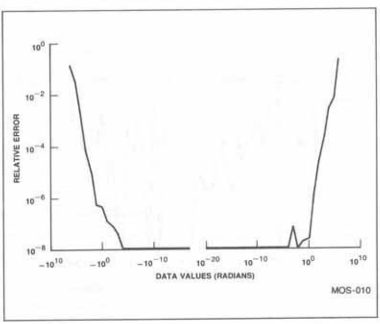


Figure 9.4. Tangent

The constants used in the Am9511A are as follows:

Ntan = 9
$Ctan_0 = 1.7701474$
$Ctan_1 = 1.0675393 \times 10^{-1}$
$Ctan_2 = 7.5861016 \times 10^{-3}$
$Ctan_3 = 5.4417038 \times 10^{-4}$
$Ctan_4 = 3.9066370 \times 10^{-5}$
$Ctan_5 = 2.8048161 \times 10^{-6}$
$Ctan_6 = 2.0137658 \times 10^{-7}$
$Ctan_7 = 1.4458187 \times 10^{-8}$
$Ctan_8 = 1.0380510 \times 10^{-9}$

9.7 ARCSINE

The argument of arcsine must be less than or equal to 1, or else an input error is detected. Hence, range reduction is not necessary.

There are two different Chebyshev polynominal expansion used depending on the initial value of X. If $X^2 \leq 1/2$ then the following formula is used

$$Asin(X) = x * 2 * CHEBY(4x^2 - 1, Casin, Nasin)$$

If $1/2 < x^2 \leq 1$ then

$$Asin(X) = sign(X) * \frac{\pi}{2} * \sqrt{2 - 2x^2} * CHEBY(3 - 4x^2, Casin, Nasin)$$

Where sign (X) is the sign of X. The values of Casin and Nasin used in the Am9511A are as follows:

Nasin = 10
$Casin_0 = 1.4866665$
$Casin_1 = 3.8853034 \times 10^{-2}$
$Casin_2 = 2.8854414 \times 10^{-3}$
$Casin_3 = 2.8842183 \times 10^{-4}$
$Casin_4 = 3.3223672 \times 10^{-5}$
$Casin_5 = 4.1584779 \times 10^{-6}$
$Casin_6 = 5.4965045 \times 10^{-7}$
$Casin_7 = 7.5500784 \times 10^{-8}$
$Casin_8 = 1.0671938 \times 10^{-8}$
$Casin_9 = 1.5421800 \times 10^{-9}$

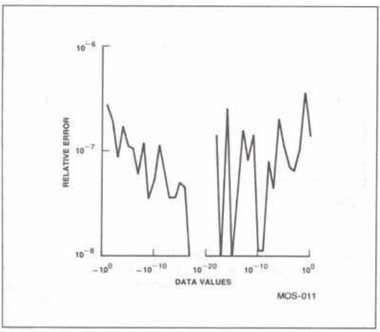


Figure 9.3. Inverse Sine

9.8 ARCCOSINE

The arccosine is obtained from arcsine by using the trigonometric identity.

$$Arccosine(x) = \frac{\pi}{2} - arcsine(x)$$

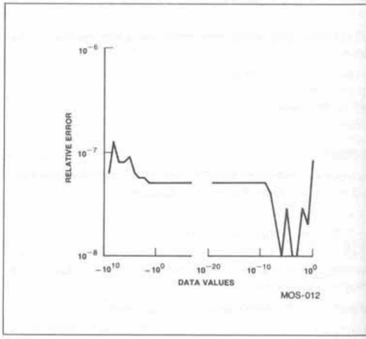


Figure 9.5. Inverse Cosine

9.9 ARCTANGENT

The range reduction of the arctangent function involves taking the reciprocal of the input argument if the absolute value of the input argument is greater than 1.

U = X
If (ABS (U).GT.1)X = 1/X

The Chebyshev polynomial evaluation is

$$Atan(X) = X * Cheby(2X^2 - 1, Catan, Natan)$$

The postprocessing requirement is

If (U.GT.1) Atan (X) = $\pi/2$ − Atan (X)
If (U.LT.−1) Atan (X) = $-\pi/2$ − Atan (X)

The value of Natan and Catan used in the Am9511A are:

Natan = 11
$Catan_0 = 1.7627472$
$Catan_1 = -1.0589292 \times 10^{-1}$
$Catan_2 = 1.1135842 \times 10^{-2}$
$Catan_3 = -1.3811950 \times 10^{-3}$
$Catan_4 = 1.8574297 \times 10^{-4}$
$Catan_5 = -2.6215196 \times 10^{-5}$
$Catan_6 = 3.8210366 \times 10^{-6}$
$Catan_7 = -5.6991862 \times 10^{-7}$
$Catan_8 = 8.6488779 \times 10^{-8}$
$Catan_9 = -1.3303384 \times 10^{-8}$
$Catan_{10} = 2.0685060 \times 10^{-9}$
$Catan_{11} = -3.2448600 \times 10^{-10}$

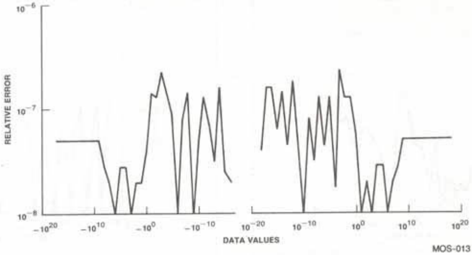


Figure 9.6. Inverse Tangent

9.10. EXPONENTIATION (Figure 9.7)

The range reduction for the exponentiation function is performed by the following formulas

$X = X * \text{Log}_2 e$

$N = 1 + \text{Entier}(X)$

The Chebyshev polynomial evaluation is

$\text{Exp}(X) = 2^N * \text{Cheby}(2*(N - X) - 1, \text{Cexp}, \text{Nexp})$

No postprocessing is required for the exponentiation function. The values of Nexp and Cexp used by Am9511A are:

Nexp = 8
Cexp_0 = 1.4569999
Cexp_1 = $-2.4876243 \times 10^{-1}$
Cexp_2 = 2.1446556×10^{-2}
Cexp_3 = $-1.2357141 \times 10^{-3}$
Cexp_4 = 5.3453058×10^{-5}
Cexp_5 = $-1.8506907 \times 10^{-6}$
Cexp_6 = 5.3411877×10^{-8}
Cexp_7 = $-1.3215160 \times 10^{-9}$

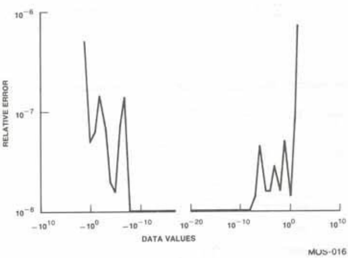


Figure 9.7. e^x

9.11. NATURAL LOGARITHM (Figure 9.8)

Any input argument to a logarithm function that is less than or equal to zero will be returned as an error input. No preprocessing or postprocessing is necessary for all positive input X.

$\text{LN}(X) = \text{CHEBY}(4*\text{Mant}(X) - 3, \text{CLN}, \text{NLN}) + (\text{Expo}(X) - 1) *\text{LN2}$

Where Mant(X) is the mantissa value of X and expo (X) is the exponent value of X.

The value of NLN and CLN used in the Am9511A are:

NLN = 11
CLN_0 = 7.5290563×10^{-1}
CLN_1 = 3.4314575×10^{-1}
CLN_2 = $-2.9437253 \times 10^{-2}$
CLN_3 = 3.3670893×10^{-3}
CLN_4 = $-4.3327589 \times 10^{-4}$
CLN_5 = 5.9470712×10^{-5}
CLN_6 = $-8.5029675 \times 10^{-6}$
CLN_7 = 1.2504674×10^{-6}
CLN_8 = $-1.8772800 \times 10^{-7}$
CLN_9 = 2.8630251×10^{-8}
CLN_{10} = $-4.4209570 \times 10^{-9}$

9.12 LOGARITHM TO BASE 10 (COMMON LOGARITHM)

The common logarithm is derived from the natural logarithm by the equation

$\text{LOG}(X) = \text{LN}(X) * \text{LOG}_{10}e$

where

$\text{LOG}_{10}e = 0.4342945$

9.13 X TO THE POWER OF Y

The function X to the power of Y is derived from the following equation

$X^Y = e^{(Y*\text{LN}(X))}$

9.14 SQUARE ROOT

The square root function (Figure 9.9) in the Am9511A is the only derived function that does not use the Chebyshev polynomials. It

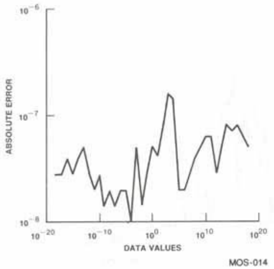


Figure 9.8. Natural Logarithm

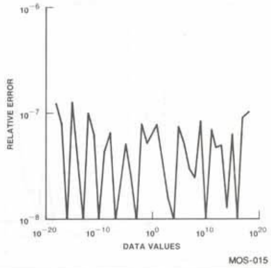


Figure 9.9. Square Root

uses a combination of linear approximation and the Newton-Ralfson successive approximation methods. The square root algorithm adopted is divided into three parts:

(a) Range reduction –
The input argument is divided into the exponent and the mantissa. If the exponent is odd, the exponent is incremented by 1 and the mantissa is divided by 2. If the input exponent is even, the above step is skipped.

(b) Linear Approximation –
The mantissa is now a number greater than or equal to 1/4 and less than 1. The curve line in Figure 9.10 represents the square root of all numbers between 1/4 and 1. The straight line represents the first-order approximation for the square root of the number. To select the best straight line, we must minimize the maximum relative error between the straight line and the curve line. This would reduce the worst case error to a minimum. This line is known as the minimax line.

The method used to compute the best linear approximation line is as follows:

Let m = Slope of the minimax line
Let b = Y intercept of the minimax line
Let Y = The function of the minimax line

such that

$$Y = mx + b$$

The relative error between the actual square root value and the first-order approximation is

$$E(x) = \frac{mx + b - \sqrt{x}}{\sqrt{x}}$$

Figure 9.10 shows that the absolute value of E(x) is a maximum at the two extremities (x = 1/4 and x = 1) and at a point where the slope of the curve E(x) = 0, or dE/dx = 0.

$$\frac{dE}{Dx} = \frac{d}{dx}\frac{(mx + b - \sqrt{x})}{\sqrt{x}} \qquad (9.1)$$

$$= \frac{d}{dx} mx^{1/2} + \frac{d}{dx} bx^{-1/2} - \frac{d}{dx}(1)$$

$$= m\frac{d}{dx}x^{1/2} + b\frac{d}{dx}x^{-1/2} - 0$$

$$= \frac{1}{2}mx^{-1/2} - \frac{1}{2}bx^{-3/2} = 0$$

therefore

$$mx^{1/2} = bx^{-3/2}$$

$$x = \frac{b}{m}$$

The relative errors at the extremities are given by

$$E\left(\frac{1}{2}\right) = \frac{\frac{m}{4} + b - \sqrt{\frac{1}{4}}}{\sqrt{\frac{1}{4}}}$$

$$= \frac{\frac{m}{4} + b - \frac{1}{2}}{\frac{1}{2}}$$

$$= \frac{m}{2} + 2b - 1 \qquad (9.2)$$

$$\Sigma(1) = \frac{m + b - \sqrt{1}}{\sqrt{1}} = m + b - 1 \qquad (9.3)$$

The minimax line requires these maximum errors to be equal

$$\frac{m}{2} + 2b - 1 = m + b - 1$$

$$b - \frac{m}{2} = 0$$

$$\frac{b}{m} = \frac{1}{2} \qquad (9.4)$$

$$m = 2b \qquad (9.5)$$

from equations 9.1 and 9.4

$$x = \frac{b}{m} = \frac{1}{2}$$

Therefore, the maximum error in the middle occurs when X = 1/2. The minimax line requires these errors to be equal in magnitude. Thus

$$E\left(\frac{1}{4}\right) = E(1) = -E\left(\frac{1}{2}\right)$$

$$E\left(\frac{1}{2}\right) = \frac{\frac{m}{2} + b - \sqrt{\frac{1}{2}}}{\sqrt{\frac{1}{2}}} \qquad (9.6)$$

Since m = 2b from equation 9.5

$$E\left(\frac{1}{2}\right) = \frac{2b - \sqrt{\frac{1}{2}}}{\sqrt{\frac{1}{2}}} \qquad (9.7)$$

From equations 9.3 and 9.5

$$E(1) = 3b - 1 \qquad (9.8)$$

From equations 9.6, 9.7 and 9.8

$$\frac{2b - \sqrt{\frac{1}{2}}}{\sqrt{\frac{1}{2}}} = -(3b - 1) = 1 - 3b$$

$$2\sqrt{2}\,b - 1 = 1 - 3b$$

$$b = \frac{2}{2\sqrt{2} + 3} = 0.34314575$$

From 9.5

$$m = 2b = 0.6829150$$

Therefore, the minimax line is given by

$$Y = 68629150x + 0.34314575$$

This is the equation used in Am9511A for the first-order linear approximation. Therefore

$$X_0 = 0.68629150x + 0.34314575$$

(c) Newton-Ralfson successive approximation –
After the first-order approximation (X_0) is obtained, the Am9511A executes two iterations of the Newton-Ralfson approximation

$$X_1 = (X/X_0 + X_0)/2$$
$$X_2 = (X/X_1 + X_1)/2$$

And the result is given by

$$SQRT(X) = x_2 \cdot 2^{E/2}$$

A FORTRAN function to illustrate the above algorithm is given below:

```
      FUNCTION ROOT (X)
      INTEGER EXPO, LSB
      REAL MANT, X0, X1, X2
      EXPO = INT (LOG(X)/LOG(2)) + 1
      MANT = X/2**EXP
      LSB = MOD(EXPO, 2)
      IF (LSB.EQ.0) GOTO 100
C     EXPONENT IS ODD
      EXPO = EXPO + 1
      MANT = MANT/2.0
100   X0 = 0.68629150* Mant + 0.34314575
      X1 = (X/X0 + X0)/2.0
      X2 = (X/X1 + X1)/2.0
      Root = (2**(EXPO/2))*X2
      End
```

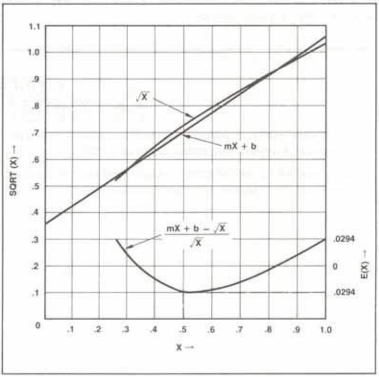


Figure 9.10. Square Root Computation

9.15 DERIVED FUNCTION ERROR PERFORMANCE

Since each of the derived functions is an approximation of the true function, results computed by the Am9511A are not always exact. In order to quantify the error performance of the component more comprehensively, the following graphs have been prepared. Each function has been executed with a statistically significant number of diverse data values, spanning the allowable input data range, and resulting errors have been tabulated. Absolute errors (that is, the number of bits in error) have been converted to relative errors according to the following equation:

$$\text{Relative Error} = \frac{\text{Absolute Error}}{\text{True Result}}$$

This conversion permits the error to be viewed with respect to the magnitude of the true result. This provides a more objective measurement of error performance since it directly translates to a measure of significant digits of algorithm accuracy.

For example, if a given absolute error is 0.0001 and the true result is also 0.0001, it is clear that the relative error is equal to 1.0 (which implies that even the first significant digit of the result is wrong. However, if the same absolute error is computed for a true result of 10000.0, then the first six significant digits of the result are correct (0.001/10000 = 0.0000001).

Each of the following graphs was prepared to illustrate relative algorithm error as a function of input data range. Natural logarithm is the only exception; since logarithms are typically additive, absolute error is plotted for this function.

Two graphs have not been included in the following figures: common logarithms and the power function (X^Y). Common logarithms are computed by multiplication of the natural logarithms by the conversion factor 0.43429448 and the error function is therefore the same as that for natural logarithm. The power function is realized by combination of natural log and exponential functions according to the equation

$$X^Y = e^{Y\ln(X)}$$

The error for the power function is a combination of that for the logarithm and exponential functions. Specifically, the relative error for PWR is expressed as

$$RE_{PWR} = RE_{EXP} + X(AE_{ln})$$

where

RE_{PWR} = relative error for power function
RE_{EXP} = relative error for exponential function
AE_{ln} = absolute error for natural logarithm
X = value of independent variable in X^Y

REFERENCES

1. Penningtor Ralph H. *Introduction to Computer Methods and Numerical Analysis*. Macmillan Company, 1970.
2. Clenshaw, Miller and Woodger. "Algorithms for Special Functions (I and II)," *Numerische Mathematic*, 1963.
3. Parker, Richard O. and Joseph H. Kroeger. *Algorithm Details for the Am9511A Arithmetic Processing Unit*. Advanced Micro Devices, 1978.

Appendix A

Am9511A

Arithmetic Processor

DISTINCTIVE CHARACTERISTICS

- 2, 3 and 4MHz operation
- Fixed point 16 and 32 bit operations
- Floating point 32 bit operations
- Binary data formats
- Add, Subtract, Multiply and Divide
- Trigonometric and inverse trigonometric functions
- Square roots, logarithms, exponentiation
- Float to fixed and fixed to float conversions
- Stack-oriented operand storage
- DMA or programmed I/O data transfers
- End signal simplifies concurrent processing
- Synchronous/Asynchronous operations
- General purpose 8-bit data bus interface
- Standard 24 pin package
- +12 volt and +5 volt power supplies
- Advanced N-channel silicon gate MOS technology

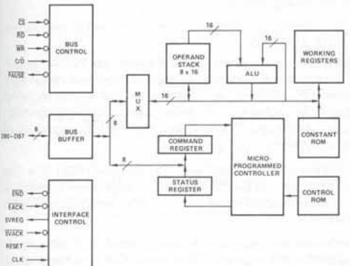
GENERAL DESCRIPTION

The Am9511A Arithmetic Processing Unit (APU) is a monolithic MOS/LSI device that provides high performance fixed and floating point arithmetic and a variety of floating point trigonometric and mathematical operations. It may be used to enhance the computational capability of a wide variety of processor-oriented systems.

All transfers, including operand, result, status and command information, take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack and a command is issued to perform operations on the data in the stack. Results are then available to be retrieved from the stack, or additional commands may be entered.

Transfers to and from the APU may be handled by the associated processor using conventional programmed I/O, or may be handled by a direct memory access controller for improved performance. Upon completion of each command, the APU issues an end of execution signal that may be used as an interrupt by the CPU to help coordinate program execution.

BLOCK DIAGRAM

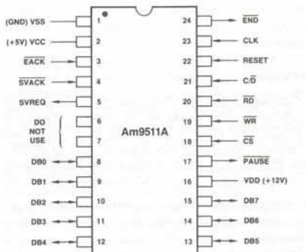


01892B-1

CONNECTION DIAGRAM

Top View

D-24-2



01892B-2

Note: Pin 1 is marked for orientation.

ORDERING INFORMATION

Package Type	Ambient Temperature	Maximum Clock Frequency		
		2MHz	3MHz	4MHz
Hermetic DIP	0°C ≤ T_A ≤ +70°C	Am9511ADC	Am9511A-1DC	Am9511A-4DC
	−40°C ≤ T_A ≤ +85°C	Am9511ADI	Am9511A-1DI	
	−55°C ≤ T_A ≤ +125°C	Am9511ADMB	Am9511A-1DMB	

01892B-MMP

INTERFACE SIGNAL DESCRIPTION

VCC: +5V Power Supply
VDD: +12V Power Supply
VSS: Ground

CLK (Clock, Input)

An external timing source connected to the CLK input provides the necessary clocking. The CLK input can be asynchronous to the $\overline{RD}$ and $\overline{WR}$ control signals.

RESET (Reset, Input)

A HIGH on this input causes initialization. Reset terminates any operation in progress, and clears the status register to zero. The internal stack pointer is initialized and the contents of the stack may be affected but the command register is not affected by the reset operation. After a reset the $\overline{END}$ output will be HIGH, and the SVREQ output will be LOW. For proper initialization, the RESET input must be HIGH for at least five CLK periods following stable power supply voltages and stable clock.

C/$\overline{D}$ (Command/Data Select, Input)

The C/$\overline{D}$ input together with the $\overline{RD}$ and $\overline{WR}$ inputs determines the type of transfer to be performed on the data bus as follows:

C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	Function
L	H	L	Push data byte into the stack
L	L	H	Pop data byte from the stack
H	H	L	Enter command byte from the data bus
H	L	H	Read Status
X	L	L	Undefined

L = LOW
H = HIGH
X = DON'T CARE

$\overline{END}$ (End of Execution, Output)

A LOW on this output indicates that execution of the current command is complete. This output will be cleared HIGH by activating the $\overline{EACK}$ input LOW or performing any read or write operation or device initialization using the RESET. If $\overline{EACK}$ is tied LOW, the $\overline{END}$ output will be a pulse (see $\overline{EACK}$ description). This is an open drain output and requires a pull up to +5V.

Reading the status register while a command execution is in progress is allowed. However any read or write operation clears the flip-flop that generates the $\overline{END}$ output. Thus such continuous reading could conflict with internal logic setting the $\overline{END}$ flip-flop at the completion of command execution.

$\overline{EACK}$ (End Acknowledge, Input)

This input when LOW makes the $\overline{END}$ output go HIGH. As mentioned earlier LOW on the $\overline{END}$ output signals completion of a command execution. The $\overline{END}$ output signal is derived from an internal flip-flop which is clocked at the completion of a command. This flip-flop is clocked to the reset state when $\overline{EACK}$ is LOW. Consequently, if the $\overline{EACK}$ is tied LOW, the $\overline{END}$ output will be a pulse that is approximately one CLK period wide.

SVREQ (Service Request, Output)

A HIGH on this output indicates completion of a command. In this sense this output is same as the $\overline{END}$ output. However, whether the SVREQ output will go HIGH at the completion of a command or not is determined by a service request bit in the command register. This bit must be 1 for SVREQ to go HIGH. The SVREQ can be cleared (i.e., go LOW) by activating the $\overline{SVACK}$ input LOW or initializing the device using the RESET. Also, the SVREQ will be automatically cleared after completion of any command that has the service request bit as 0.

$\overline{SVACK}$ (Service Acknowledge, Input)

A LOW on this input activates the reset input of the flip-flop generating the SVREQ output. If the $\overline{SVACK}$ input is permanently tied LOW, it will conflict with the internal setting of the flip-flop to generate the SVREQ output. Thus the SVREQ indication cannot be relied upon if the $\overline{SVACK}$ is tied LOW.

DB0-DB7 (Bidirectional Data Bus, Input/Output)

These eight bidirectional lines are used to transfer command, status and operand information between the device and the host processor. DB0 is the least significant and DB7 is the most significant bit position. HIGH on the data bus line corresponds to 1 and LOW corresponds to 0.

When pushing operands on the stack using the data bus, the least significant byte must be pushed first and most significant byte last. When popping the stack to read the result of an operation, the most significant byte will be available on the data bus first and the least significant byte will be the last. Moreover, for pushing operands and popping results, the number of transactions must be equal to the proper number of bytes appropriate for the chosen format. Otherwise, the internal byte pointer will not be aligned properly. The Am9511A single precision format requires 2 bytes, double precision and floating-point formats require 4 bytes.

$\overline{CS}$ (Chip Select, Input)

This input must be LOW to accomplish any read or write operation to the Am9511A.

To perform a write operation data is presented on DB0 through DB7 lines, C/$\overline{D}$ is driven to an appropriate level and the $\overline{CS}$ input is made LOW. However, actual writing into the Am9511A cannot start until $\overline{WR}$ is made LOW. After initiating the write operation by a $\overline{WR}$ HIGH to LOW transition, the $\overline{PAUSE}$ output will go LOW momentarily (TPPWW).

The $\overline{WR}$ input can go HIGH after $\overline{PAUSE}$ goes HIGH. The data lines, C/$\overline{D}$ input and the $\overline{CS}$ input can change when appropriate hold time requirements are satisfied. See write timing diagram for details.

To perform a read operation an appropriate logic level is established on the C/$\overline{D}$ input and $\overline{CS}$ is made LOW. The Read operation does not start until the $\overline{RD}$ input goes LOW. $\overline{PAUSE}$ will go LOW for a period of TPPWR. When $\overline{PAUSE}$ goes back HIGH again, it indicates that read operation is complete and the required information is available on the DB0 through DB7 lines. This information will remain on the data lines as long as $\overline{RD}$ input is LOW. The $\overline{RD}$ input can return HIGH anytime after $\overline{PAUSE}$ goes HIGH. The $\overline{CS}$ input and C/$\overline{D}$ inputs can change anytime after $\overline{RD}$ returns HIGH. See read timing diagram for details.

$\overline{RD}$ (Read, Input)

A LOW on this input is used to read information from an internal location and gate that information on to the data bus. The $\overline{CS}$ input must be LOW to accomplish the read operation. The C/$\overline{D}$ input determines what internal location is of interest. See C/$\overline{D}$, $\overline{CS}$ input descriptions and read timing diagram for details. If the END output was LOW, performing any read operation will make the $\overline{END}$ output go HIGH after the HIGH to LOW transition of the $\overline{RD}$ input (assuming $\overline{CS}$ is LOW).

$\overline{\text{WR}}$ (Write, Input)

A LOW on this input is used to transfer information from the data bus into an internal location. The $\overline{\text{CS}}$ must be LOW to accomplish the write operation. The C/$\overline{\text{D}}$ determines which internal location is to be written. See C/$\overline{\text{D}}$, $\overline{\text{CS}}$ input descriptions and write timing diagram for details.

If the $\overline{\text{END}}$ output was LOW, performing any write operation will make the $\overline{\text{END}}$ output go HIGH after the LOW to HIGH transition of the $\overline{\text{WR}}$ input (assuming $\overline{\text{CS}}$ is LOW).

$\overline{\text{PAUSE}}$ (Pause, Output)

This output is a handshake signal used while performing read or write transactions with the Am9511A. A LOW at this output indicates that the Am9511A has not yet completed its information transfer with the host over the data bus. During a read operation, after $\overline{\text{CS}}$ went LOW, the $\overline{\text{PAUSE}}$ will become LOW shortly (TRP) after $\overline{\text{RD}}$ goes LOW. $\overline{\text{PAUSE}}$ will return high only after the data bus contains valid output data. The $\overline{\text{CS}}$ and $\overline{\text{RD}}$ should remain LOW when $\overline{\text{PAUSE}}$ is LOW. The RD may go high anytime after $\overline{\text{PAUSE}}$ goes HIGH. During a write operation, after $\overline{\text{CS}}$ went LOW, the $\overline{\text{PAUSE}}$ will be LOW for a very short duration (TPPWW) after $\overline{\text{WR}}$ goes LOW. Since the minimum of TPPWW is 0, the $\overline{\text{PAUSE}}$ may not go LOW at all for fast devices. $\overline{\text{WR}}$ may go HIGH anytime after $\overline{\text{PAUSE}}$ goes HIGH.

FUNCTIONAL DESCRIPTION

Major functional units of the Am9511A are shown in the block diagram. The Am9511A employs a microprogram controlled stack oriented architecture with 16-bit wide data paths.

The Arithmetic Logic Unit (ALU) receives one of its operands from the Operand Stack. This stack is an 8-word by 16-bit 2-port memory with last in-first out (LIFO) attributes. The second operand to the ALU is supplied by the internal 16-bit bus. In addition to supplying the second operand, this bidirectional bus also carries the results from the output of the ALU when required. Writing into the Operand Stack takes place from this internal 16-bit bus when required. Also connected to this bus are the Constant ROM and Working Registers. The ROM provides the required constants to perform the mathematical operations (Chebyshev Algorithms) while the Working Registers provide storage for the intermediate values during command execution.

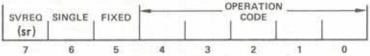
Communication between the external world and the Am9511A takes place on eight bidirectional input/output lines DB0 through DB7 (Data Bus). These signals are gated to the internal eight-bit bus through appropriate interface and buffer circuitry. Multiplexing facilities exist for bidirectional communication between the internal eight and sixteen-bit buses. The Status Register and Command Register are also accessible via the eight-bit bus.

The Am9511A operations are controlled by the microprogram contained in the Control ROM. The Program Counter supplies the microprogram addresses and can be partially loaded from the Command Register. Associated with the Program Counter is the Subroutine Stack where return addresses are held during subroutine calls in the microprogram. The Microinstruction Register holds the current microinstruction being executed. This register facilitates pipelined microprogram execution. The Instruction Decode logic generates various internal control signals needed for the Am9511A operation.

The Interface Control logic receives several external inputs and provides handshake related outputs to facilitate interfacing the Am9511A to microprocessors.

COMMAND FORMAT

Each command entered into the Am9511A consists of a single 8-bit byte having the format illustrated below:



01892B-3

Bits 0-4 select the operation to be performed as shown in the table. Bits 5-6 select the data format for the operation. If bit 5 is a 1, a fixed point data format is specified. If bit 5 is a 0, floating point format is specified. Bit 6 selects the precision of the data to be operated on by fixed point commands (if bit 5 = 0, bit 6 must be 0). If bit 6 is a 1, single-precision (16-bit) operands are indicated; if bit 6 is a 0, double-precision (32-bit) operands are indicated. Results are undefined for all illegal combinations of bits in the command byte. Bit 7 indicates whether a service request is to be issued after the command is executed. If bit 7 is a 1, the service request output (SVREQ) will go high at the conclusion of the command and will remain high until reset by a low level on the service acknowledge pin (SVACK) or until completion of execution of a succeeding command where bit 7 is 0. Each command issued to the Am9511A requests post execution service based upon the state of bit 7 in the command byte. When bit 7 is a 0, SVREQ remains low.

COMMAND SUMMARY

Command Code								Command Mnemonic	Command Description
7	6	5	4	3	2	1	0		
FIXED-POINT 16-BIT									
sr	1	1	0	1	1	0	0	SADD	Add TOS to NOS. Result to NOS. Pop Stack.
sr	1	1	0	1	1	0	1	SSUB	Subtract TOS from NOS. Result to NOS. Pop Stack.
sr	1	1	0	1	1	1	0	SMUL	Multiply NOS by TOS. Lower half of result to NOS. Pop Stack.
sr	1	1	1	0	1	1	0	SMUU	Multiply NOS by TOS. Upper half of result to NOS. Pop Stack.
sr	1	1	0	1	1	1	1	SDIV	Divide NOS by TOS. Result to NOS. Pop Stack.
FIXED-POINT 32-BIT									
sr	0	1	0	1	1	0	0	DADD	Add TOS to NOS. Result to NOS. Pop Stack.
sr	0	1	0	1	1	0	1	DSUB	Subtract TOS from NOS. Result to NOS. Pop Stack.
sr	0	1	0	1	1	1	0	DMUL	Multiply NOS by TOS. Lower half of result to NOS. Pop Stack.
sr	0	1	1	0	1	1	0	DMUU	Multiply NOS by TOS. Upper half of result to NOS. Pop Stack.
sr	0	1	0	1	1	1	1	DDIV	Divide NOS by TOS. Result to NOS. Pop Stack.
FLOATING-POINT 32-BIT									
sr	0	0	1	0	0	0	0	FADD	Add TOS to NOS. Result to NOS. Pop Stack.
sr	0	0	1	0	0	0	1	FSUB	Subtract TOS from NOS. Result to NOS. Pop Stack.
sr	0	0	1	0	0	1	0	FMUL	Multiply NOS by TOS. Result to NOS. Pop Stack.
sr	0	0	1	0	0	1	1	FDIV	Divide NOS by TOS. Result to NOS. Pop Stack.
DERIVED FLOATING-POINT FUNCTIONS									
sr	0	0	0	0	0	0	1	SQRT	Square Root of TOS. Result in TOS.
sr	0	0	0	0	0	1	0	SIN	Sine of TOS. Result in TOS.
sr	0	0	0	0	0	1	1	COS	Cosine of TOS. Result in TOS.
sr	0	0	0	0	1	0	0	TAN	Tangent of TOS. Result in TOS.
sr	0	0	0	0	1	0	1	ASIN	Inverse Sine of TOS. Result in TOS.
sr	0	0	0	0	1	1	0	ACOS	Inverse Cosine of TOS. Result in TOS.
sr	0	0	0	0	1	1	1	ATAN	Inverse Tangent of TOS. Result in TOS.
sr	0	0	0	1	0	0	0	LOG	Common Logarithm (base 10) of TOS. Result in TOS.
sr	0	0	0	1	0	0	1	LN	Natural Logarithm (base e) of TOS. Result in TOS.
sr	0	0	0	1	0	1	0	EXP	Exponential (e^x) of TOS. Result in TOS.
sr	0	0	0	1	0	1	1	PWR	NOS raised to the power in TOS. Result in NOS. Pop Stack.
DATA MANIPULATION COMMANDS									
sr	0	0	0	0	0	0	0	NOP	No Operation
sr	0	0	1	1	1	1	1	FIXS	Convert TOS from floating point to 16-bit fixed point format.
sr	0	0	1	1	1	1	0	FIXD	Convert TOS from floating point to 32-bit fixed point format.
sr	0	0	1	1	1	0	1	FLTS	Convert TOS from 16-bit fixed point to floating point format.
sr	0	0	1	1	1	0	0	FLTD	Convert TOS from 32-bit fixed point to floating point format.
sr	1	1	1	0	1	0	0	CHSS	Change sign of 16-bit fixed point operand on TOS.
sr	0	1	1	0	1	0	0	CHSD	Change sign of 32-bit fixed point operand on TOS.
sr	0	0	1	0	1	0	1	CHSF	Change sign of floating point operand on TOS.
sr	1	1	1	0	1	1	1	PTOS	Push 16-bit fixed point operand on TOS to NOS (Copy)
sr	0	1	1	0	1	1	1	PTOD	Push 32-bit fixed point operand on TOS to NOS. (Copy)
sr	0	0	1	0	1	1	1	PTOF	Push floating point operand on TOS to NOS. (Copy)
sr	1	1	1	1	0	0	0	POPS	Pop 16-bit fixed point operand from TOS. NOS becomes TOS.
sr	0	1	1	1	0	0	0	POPD	Pop 32-bit fixed point operand from TOS. NOS becomes TOS.
sr	0	0	1	1	0	0	0	POPF	Pop floating point operand from TOS. NOS becomes TOS.
sr	1	1	1	1	0	0	1	XCHS	Exchange 16-bit fixed point operands TOS and NOS.
sr	0	1	1	1	0	0	1	XCHD	Exchange 32-bit fixed point operands TOS and NOS.
sr	0	0	1	1	0	0	1	XCHF	Exchange floating point operands TOS and NOS.
sr	0	0	1	1	0	1	0	PUPI	Push floating point constant "π" onto TOS. Previous TOS becomes NOS.

NOTES:

1. TOS means Top of Stack. NOS means Next on Stack.
2. AMD Application Brief "Algorithm Details for the Am9511A APU" provides detailed descriptions of each command function, including data ranges, accuracies, stack configurations, etc.
3. Many commands destroy one stack location (bottom of stack) during development of the result. The derived functions may destroy several stack locations. See Application Brief for details.
4. The trigonometric functions handle angles in radians, not degrees.
5. No remainder is available for the fixed-point divide functions.
6. Results will be undefined for any combination of command coding bits not specified in this table.

COMMAND INITIATION

After properly positioning the required operands on the stack, a command may be issued. The procedure for initiating a command execution is as follows:

1. Enter the appropriate command on the DB0-DB7 lines.
2. Establish HIGH on the C/$\overline{\text{D}}$ input.
3. Establish LOW on the $\overline{\text{CS}}$ input.
4. Establish LOW on the $\overline{\text{WR}}$ input after an appropriate set up time (see timing diagrams).
5. Sometime after the HIGH to LOW level transition of $\overline{\text{WR}}$ input, the $\overline{\text{PAUSE}}$ output will become LOW. After a delay of TPPWW, it will go HIGH to acknowledge the write operation. The WR input can return to HIGH anytime after $\overline{\text{PAUSE}}$ going HIGH. The DB0-DB7, C/$\overline{\text{D}}$ and $\overline{\text{CS}}$ inputs are allowed to change after the hold time requirements are satisfied (see timing diagram).

An attempt to issue a new command while the current command execution is in progress is allowed. Under these circumstances, the $\overline{\text{PAUSE}}$ output will not go HIGH until the current command execution is completed.

OPERAND ENTRY

The Am9511A commands operate on the operands located at the TOS and NOS and results are returned to the stack at NOS and then popped to TOS. The operands required for the Am9511A are one of three formats – single precision fixed-point (2 bytes), double precision fixed-point (4 bytes) or floating-point (4 bytes). The result of an operation has the same format as the operands except for float to fix or fix to float commands.

Operands are always entered into the stack least significant byte first and most significant byte last. The following procedure must be followed to enter operands onto the stack:

1. The lower significant operand byte is established on the DB0-DB7 lines.
2. A LOW is established on the C/$\overline{\text{D}}$ input to specify that data is to be entered into the stack.
3. The $\overline{\text{CS}}$ input is made LOW.
4. After appropriate set up time (see timing diagrams), the $\overline{\text{WR}}$ input is made LOW. The $\overline{\text{PAUSE}}$ output will become LOW.
5. Sometime after this event, the $\overline{\text{PAUSE}}$ will return HIGH to indicate that the write operation has been acknowledged.
6. Anytime after the $\overline{\text{PAUSE}}$ output goes HIGH the $\overline{\text{WR}}$ input can be made HIGH. The DB0-DB7, C/$\overline{\text{D}}$ and $\overline{\text{CS}}$ inputs can change after appropriate hold time requirements are satisfied (see timing diagrams).

The above procedure must be repeated until all bytes of the operand are pushed into the stack. It should be noted that for single precision fixed-point operands 2 bytes should be pushed and 4 bytes must be pushed for double precision fixed-point or floating-point. Not pushing all the bytes of a quantity will result in byte pointer misalignment.

The Am9511A stack can accommodate 8 single precision fixed-point quantities or 4 double precision fixed-point or floating-point quantities. Pushing more quantities than the capacity of the stack will result in loss of data which is usual with any LIFO stack.

DATA REMOVAL

Result from an operation will be available at the TOS. Results can be transferred from the stack to the data bus by reading the stack. When the stack is popped for results, the most significant byte is available first and the least significant byte last. A result is always of the same precision as the operands that produced it except for format conversion commands. Thus when the result is taken from the stack, the total number of bytes popped out should be appropriate with the precision – single precision results are 2 bytes and double precision and floating-point results are 4 bytes. The following procedure must be used for reading the result from the stack:

1. A LOW is established on the C/$\overline{\text{D}}$ input.
2. The $\overline{\text{CS}}$ input is made LOW.
3. After appropriate set up time (see timing diagrams), the $\overline{\text{RD}}$ input is made LOW. The $\overline{\text{PAUSE}}$ will become LOW.
4. Sometime after this, $\overline{\text{PAUSE}}$ will return HIGH indicating that the data is available on the DB0-DB7 lines. This data will remain on the DB0-DB7 lines as long as the $\overline{\text{RD}}$ input remains LOW.
5. Anytime after $\overline{\text{PAUSE}}$ goes HIGH, the $\overline{\text{RD}}$ input can return HIGH to complete transaction.
6. The $\overline{\text{CS}}$ and C/$\overline{\text{D}}$ inputs can change after appropriate hold time requirements are satisfied (see timing diagram).
7. Repeat this procedure until all bytes appropriate for the precision of the result are popped out.

Reading of the stack does not alter its data; it only adjusts the byte pointer. If more data is popped than the capacity of the stack, the internal byte pointer will wrap around and older data will be read again, consistent with the LIFO stack.

STATUS READ

The Am9511A status register can be read without any regard to whether a command is in progress or not. The only implication that has to be considered is the effect this might have on the END output discussed in the signal descriptions.

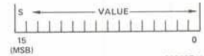
The following procedure must be followed to accomplish status register reading.

1. Establish HIGH on the C/$\overline{\text{D}}$ input.
2. Establish LOW on the $\overline{\text{CS}}$ input.
3. After appropriate set up time (see timing diagram) $\overline{\text{RD}}$ input is made LOW. The $\overline{\text{PAUSE}}$ will become LOW.
4. Sometime after the HIGH to LOW transition of $\overline{\text{RD}}$ input, the $\overline{\text{PAUSE}}$ will become HIGH indicating that status register contents are available on the DB0-DB7 lines. The status data will remain on DB0-DB7 as long as $\overline{\text{RD}}$ input is LOW.
5. The $\overline{\text{RD}}$ input can be returned HIGH anytime after $\overline{\text{PAUSE}}$ goes HIGH.
6. The C/$\overline{\text{D}}$ input and $\overline{\text{CS}}$ input can change after satisfying appropriate hold time requirements (see timing diagram).

DATA FORMATS

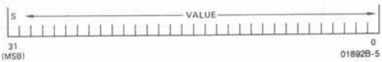
The Am9511A Arithmetic Processing Unit handles operands in both fixed-point and floating-point formats. Fixed-point operands may be represented in either single (16-bit operands) or double precision (32-bit operands), and are always represented as binary, two's complement values.

16-BIT FIXED-POINT FORMAT



01892B-4

32-BIT FIXED-POINT FORMAT



01892B-5

The sign (positive or negative) of the operand is located in the most significant bit (MSB). Positive values are represented by a sign bit of zero (S = 0). Negative values are represented by the two's complement of the corresponding positive value with a sign bit equal to 1 (S = 1). The range of values that may be accomodated by each of these formats is −32,767 to +32,767 for single precision and −2,147,483,647 to +2,147,483,647 for double precision.

Floating point binary values are represented in a format that permits arithmetic to be performed in a fashion analogous to operations with decimal values expressed in scientific notation.

$$(5.83 \times 10^2)(8.16 \times 10^1) = (4.75728 \times 10^4)$$

In the decimal system, data may be expressed as values between 0 and 10 times 10 raised to a power that effectively shifts the implied decimal point right or left the number of places necessary to express the result in conventional form (e.g., 47,572.8). The value-portion of the data is called the mantissa. The exponent may be either negative or positive.

The concept of floating point notation has both a gain and a loss associated with it. The gain is the ability to represent the significant digits of data with values spanning a large dynamic range limited only by the capacity of the exponent field. For example, in decimal notation if the exponent field is two digits wide, and the mantissa is five digits, a range of values (positive or negative) from 1.0000×10^{-99} to $9.9999 \times 10^{+99}$ can be accommodated. The loss is that only the significant digits of the value can be represented. Thus there is no distinction in this representation between the values 123451 and 123452, for example, since each would be expressed as: 1.2345×10^5. The sixth digit has been discarded. In most applications where the dynamic range of values to be represented is large, the loss of significance, and hence accuracy of results, is a minor consideration. For greater precision a fixed point format could be chosen, although with a loss of potential dynamic range.

The Am9511 is a binary arithmetic processor and requires that floating point data be represented by a fractional mantissa value between .5 and 1 multiplied by 2 raised to an appropriate power. This is expressed as follows:

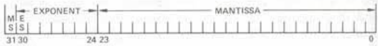
$$\text{value} = \text{mantissa} \times 2^{\text{exponent}}$$

For example, the value 100.5 expressed in this form is 0.11001001×2^7. The decimal equivalent of this value may be computed by summing the components (powers of two) of the mantissa and then multiplying by the exponent as shown below:

$$\begin{aligned} \text{value} &= (2^{-1} + 2^{-2} + 2^{-5} + 2^{-8}) \times 2^7 \\ &= (0.5 + 0.25 + 0.03125 + 0.00290625) \times 128 \\ &= 0.78515625 \times 128 \\ &= 100.5 \end{aligned}$$

FLOATING POINT FORMAT

The format for floating-point values in the Am9511A is given below. The mantissa is expressed as a 24-bit (fractional) value; the exponent is expressed as an unbiased two's complement 7-bit value having a range of −64 to +63. The most significant bit is the sign of the mantissa (0 = positive, 1 = negative), for a total of 32 bits. The binary point is assumed to be to the left of the most significant mantissa bit (bit 23). All floating-point data values must be normalized. Bit 23 must be equal to 1, except for the value zero, which is represented by all zeros.



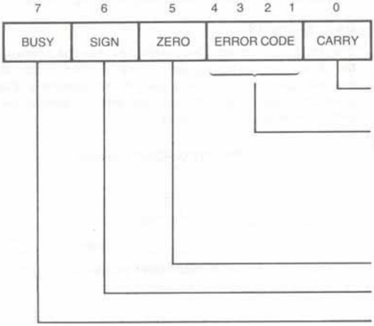
01892B-6

The range of values that can be represented in this format is $\pm(2.7 \times 10^{-20}$ to $9.2 \times 10^{18})$ and zero.

STATUS REGISTER

The Am9511A contains an eight bit status register with the following bit assignments.

If the BUSY bit in the status register is a one, the other status bits are not defined; if zero, indicating not busy, the operation is complete and the other status bits are defined as given below.



CARRY: Previous operation resulted in carry or borrow from most significant bit. (1 = Carry/Borrow, 0 = No Carry/No Borrow)

ERROR CODE: This field contains an indication of the validity of the result of the last operation. The error codes are:
- 0000 – No error
- 1000 – Divide by zero
- 0100 – Square root or log of negative number
- 1100 – Argument of inverse sine, cosine, or e^x too large
- XX10 – Underflow
- XX01 – Overflow

ZERO: Indicates that the value on the top of stack is zero (1 = Value is zero).

SIGN: Indicates that the value on the top of stack is negative (1 = Negative).

BUSY: Indicates that Am9511A is currently executing a command (1 = Busy).

Table 1.

Command Mnemonic	Hex Code (sr = 1)	Hex Code (sr = 0)	Execution Cycles	Summary Description
16-BIT FIXED-POINT OPERATIONS				
SADD	EC	6C	16-18	Add TOS to NOS. Result to NOS. Pop Stack.
SSUB	ED	6D	30-32	Subtract TOS from NOS. Result to NOS. Pop Stack.
SMUL	EE	6E	84-94	Multiply NOS by TOS. Lower result to NOS. Pop Stack.
SMUU	F6	76	80-98	Multiply NOS by TOS. Upper result to NOS. Pop Stack.
SDIV	EF	6F	84-94	Divide NOS by TOS. Result to NOS. Pop Stack.
32-BIT FIXED-POINT OPERATIONS				
DADD	AC	2C	20-22	Add TOS to NOS. Result to NOS. Pop Stack.
DSUB	AD	2D	38-40	Subtract TOS from NOS. Result to NOS. Pop Stack.
DMUL	AE	2E	194-210	Multiply NOS by TOS. Lower result to NOS. Pop Stack.
DMUU	B6	36	182-218	Multiply NOS by TOS. Upper result to NOS. Pop Stack.
DDIV	AF	2F	196-210	Divide NOS by TOS. Result to NOS. Pop Stack.
32-BIT FLOATING-POINT PRIMARY OPERATIONS				
FADD	90	10	54-368	Add TOS to NOS. Result to NOS. Pop Stack.
FSUB	91	11	70-370	Subtract TOS from NOS. Result to NOS. Pop Stack.
FMUL	92	12	146-168	Multiply NOS by TOS. Result to NOS. Pop Stack.
FDIV	93	13	154-184	Divide NOS by TOS. Result to NOS. Pop Stack.
32-BIT FLOATING-POINT DERIVED OPERATIONS				
SQRT	81	01	782-870	Square Root of TOS. Result to TOS.
SIN	82	02	3796-4808	Sine of TOS. Result to TOS.
COS	83	03	3840-4878	Cosine of TOS. Result to TOS.
TAN	84	04	4894-5886	Tangent of TOS. Result to TOS.
ASIN	85	05	6230-7938	Inverse Sine of TOS. Result to TOS.
ACOS	86	06	6304-8284	Inverse Cosine of TOS. Result to TOS.
ATAN	87	07	4992-6536	Inverse Tangent of TOS. Result to TOS.
LOG	88	08	4474-7132	Common Logarithm of TOS. Result to TOS.
LN	89	09	4298-6956	Natural Logarithm of TOS. Result to TOS.
EXP	8A	0A	3794-4878	e raised to power in TOS. Result to TOS.
PWR	8B	0B	8290-12032	NOS raised to power in TOS. Result to NOS. Pop Stack.
DATA AND STACK MANIPULATION OPERATIONS				
NOP	80	00	4	No Operation. Clear or set SVREQ.
FIXS	9F	1F	90-214	Convert TOS from floating point format to fixed point format.
FIXD	9E	1E	90-336	
FLTS	9D	1D	62-156	Convert TOS from fixed point format to floating point format.
FLTD	9C	1C	56-342	
CHSS	F4	74	22-24	Change sign of fixed point operand on TOS.
CHSD	B4	34	26-28	
CHSF	95	15	16-20	Change sign of floating point operand on TOS.
PTOS	F7	77	16	Push stack. Duplicate NOS in TOS.
PTOD	B7	37	20	
PTOF	97	17	20	
POPS	F8	78	10	Pop stack. Old NOS becomes new TOS. Old TOS rotates to bottom.
POPD	B8	38	12	
POPF	98	18	12	
XCHS	F9	79	18	Exchange TOS and NOS.
XCHD	B9	39	26	
XCHF	99	19	26	
PUPI	9A	1A	16	Push floating point constant π onto TOS. Previous TOS becomes NOS.

COMMAND DESCRIPTIONS

This section contains detailed descriptions of the APU commands. They are arranged in alphabetical order by command mnemonic. In the descriptions, TOS means Top Of Stack and NOS means Next On Stack.

All derived functions except Square Root use Chebyshev polynomial approximating algorithms. This approach is used to help minimize the internal microprogram, to minimize the maximum error values and to provide a relatively even distribution of errors over the data range. The basic arithmetic operations are used by the derived functions to compute the various Chebyshev terms. The basic operations may produce error codes in the status register as a result.

Execution times are listed in terms of clock cycles and may be converted into time values by multiplying by the clock period used. For example, an execution time of 44 clock cycles when running at a 3MHz rate translates to 14 microseconds (44 x 32μs = 14μs). Variations in execution cycles reflect the data dependency of the algorithms.

In some operations exponent overflow or underflow may be possible. When this occurs, the exponent returned in the result will be 128 greater or smaller than its true value.

Many of the functions use portions of the data stack as scratch storage during development of the results. Thus previous values in those stack locations will be lost. Scratch locations destroyed are listed in the command descriptions and shown with the crossed-out locations in the Stack Contents After diagram.

Table 1 is a summary of all the Am9511A commands. It shows the hex codes for each command, the mnemonic abbreviation, a brief description and the execution time in clock cycles. The commands are grouped by functional classes.

The command mnemonics in alphabetical order are shown below in Table 2.

Table 2.

Command Mnemonics in Alphabetical Order.

ACOS	ARCCOSINE	LOG	COMMON LOGARITHM
ASIN	ARCSINE	LN	NATURAL LOGARITHM
ATAN	ARCTANGENT	NOP	NO OPERATION
CHSD	CHANGE SIGN DOUBLE	POPD	POP STACK DOUBLE
CHSF	CHANGE SIGN FLOATING	POPF	POP STACK FLOATING
CHSS	CHANGE SIGN SINGLE	POPS	POP STACK SINGLE
COS	COSINE	PTOD	PUSH STACK DOUBLE
DADD	DOUBLE ADD	PTOF	PUSH STACK FLOATING
DDIV	DOUBLE DIVIDE	PTOS	PUSH STACK SINGLE
DMUL	DOUBLE MULTIPLY LOWER	PUPI	PUSH π
DMUU	DOUBLE MULTIPLY UPPER	PWR	POWER (X^Y)
DSUB	DOUBLE SUBTRACT	SADD	SINGLE ADD
EXP	EXPONENTIATION (e^x)	SDIV	SINGLE DIVIDE
FADD	FLOATING ADD	SIN	SINE
FDIV	FLOATING DIVIDE	SMUL	SINGLE MULTIPLY LOWER
FIXD	FIX DOUBLE	SMUU	SINGLE MULTIPLY UPPER
FIXS	FIX SINGLE	SQRT	SQUARE ROOT
FLTD	FLOAT DOUBLE	SSUB	SINGLE SUBTRACT
FLTS	FLOAT SINGLE	TAN	TANGENT
FMUL	FLOATING MULTIPLY	XCHD	EXCHANGE OPERANDS DOUBLE
FSUB	FLOATING SUBTRACT	XCHF	EXCHANGE OPERANDS FLOATING
		XCHS	EXCHANGE OPERANDS SINGLE

ACOS

32-BIT FLOATING-POINT INVERSE COSINE

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	0	1	1	0

Hex Coding: 86 with sr = 1
06 with sr = 0

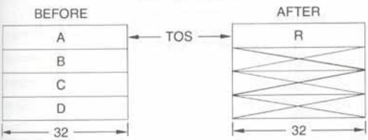
Execution Time: 6304 to 8284 clock cycles

Description:
The 32-bit floating-point operand A at the TOS is replaced by the 32-bit floating-point inverse cosine of A. The result R is a value in radians between 0 and π. Initial operands A, B, C and D are lost. ACOS will accept all input data values within the range of -1.0 to $+1.0$. Values outside this range will return an error code of 1100 in the status register.

Accuracy: ACOS exhibits a maximum relative error of 2.0 x 10^{-7} over the valid input data range.

Status Affected: Sign, Zero, Error Field

STACK CONTENTS



ASIN

32-BIT FLOATING-POINT INVERSE SINE

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	0	1	0	1

Hex Coding: 85 with sr = 1
05 with sr = 0

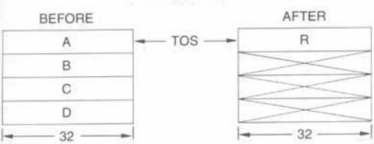
Execution Time: 6230 to 7938 clock cycles

Description:
The 32-bit floating-point operand A at the TOS is replaced by the 32-bit floating-point inverse sine of A. The result R is a value in radians between $-\pi/2$ and $+\pi/2$. Initial operands A, B, C and D are lost.
ASIN will accept all input data values within the range of -1.0 to $+1.0$. Values outside this range will return an error code of 1100 in the status register.

Accuracy: ASIN exhibits a maximum relative error of 4.0 x 10^{-7} over the valid input data range.

Status Affected: Sign, Zero, Error Field

STACK CONTENTS



ATAN

32-BIT FLOATING-POINT INVERSE TANGENT

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	0	1	1	1

Hex Coding: 87 with sr = 1
07 with sr = 0

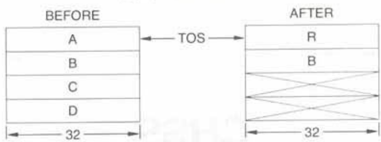
Execution Time: 4992 to 6536 clock cycles

Description:
The 32-bit floating-point operand A at the TOS is replaced by the 32-bit floating-point inverse tangent of A. The result R is a value in radians between $-\pi/2$ and $+\pi/2$. Initial operands A, C and D are lost. Operand B is unchanged.
ATAN will accept all input data values that can be represented in the floating point format.

Accuracy: ATAN exhibits a maximum relative error of 3.0 x 10^{-7} over the input data range.

Status Affected: Sign, Zero

STACK CONTENTS



CHSD

32-BIT FIXED-POINT SIGN CHANGE

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	1	1	0	1	0	0

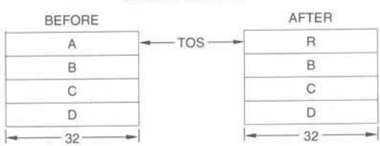
Hex Coding: B4 with sr = 1
34 with sr = 0

Execution Time: 26 to 28 clock cycles

Description:
The 32-bit fixed-point two's complement integer operand A at the TOS is subtracted from zero. The result R replaces A at the TOS. Other entries in the stack are not disturbed.
Overflow status will be set and the TOS will be returned unchanged when A is input as the most negative value possible in the format since no positive equivalent exists.

Status Affected: Sign, Zero, Error Field (overflow)

STACK CONTENTS



CHSF

32-BIT FLOATING-POINT SIGN CHANGE

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	0	1	0	1

Hex Coding: 95 with sr = 1
15 with sr = 0

Execution Time: 16 to 20 clock cycles

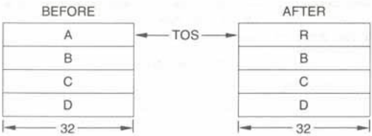
Description:

The sign of the mantissa of the 32-bit floating-point operand A at the TOS is inverted. The result R replaces A at the TOS. Other stack entries are unchanged.

If A is input as zero (mantissa MSB = 0), no change is made.

Status Affected: Sign, Zero

STACK CONTENTS



CHSS

16-BIT FIXED-POINT SIGN CHANGE

	7	6	5	4	3	2	1	0
Binary Coding:	sr	1	1	1	0	1	0	0

Hex Coding: F4 with sr = 1
74 with sr = 0

Execution Time: 22 to 24 clock cycles

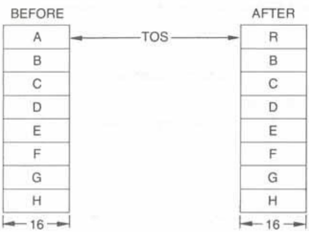
Description:

16-bit fixed-point two's complement integer operand A at the TOS is subtracted from zero. The result R replaces A at the TOS. All other operands are unchanged.

Overflow status will be set and the TOS will be returned unchanged when A is input as the most negative value possible in the format since no positive equivalent exists.

Status Affected: Sign, Zero, Overflow

STACK CONTENTS



COS

32-BIT FLOATING-POINT COSINE

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	0	0	1	1

Hex Coding: 83 with sr = 1
03 with sr = 0

Execution Time: 3840 to 4878 clock cycles

Description:

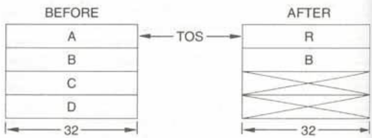
The 32-bit floating-point operand A at the TOS is replaced by R, the 32-bit floating-point cosine of A. A is assumed to be in radians. Operands A, C and D are lost. B is unchanged.

The COS function can accept any input data value that can be represented in the data format. All input values are range reduced to fall within an interval of $-\pi/2$ to $+\pi/2$ radians.

Accuracy: COS exhibits a maximum relative error of 5.0×10^{-7} for all input data values in the range of -2π to $+2\pi$ radians.

Status Affected: Sign, Zero

STACK CONTENTS



DADD

32-BIT FIXED-POINT ADD

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	1	0	1	1	0	0

Hex Coding: AC with sr = 1
2C with sr = 0

Execution Time: 20 to 22 clock cycles

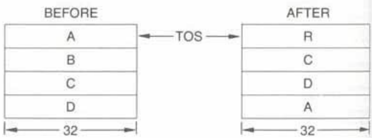
Description:

The 32-bit fixed-point two's complement integer operand A at the TOS is added to the 32-bit fixed-point two's complement integer operand B at the NOS. The result R replaces operand B and the Stack is moved up so that R occupies the TOS. Operand B is lost. Operands A, C and D are unchanged. If the addition generates a carry it is reported in the status register.

If the result is too large to be represented by the data format, the least significant 32 bits of the result are returned and overflow status is reported.

Status Affected: Sign, Zero, Carry, Error Field

STACK CONTENTS



DDIV

32-BIT FIXED-POINT DIVIDE

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	1	0	1	1	1	1

Hex Coding: AF with sr = 1
2F with sr = 0

Execution Time: 196 to 210 clock cycles when $A \neq 0$
18 clock cycles when A = 0.

Description:
The 32-bit fixed-point two's complement integer operand B at NOS is divided by the 32-bit fixed-point two's complement integer operand A at the TOS. The 32-bit integer quotient R replaces B and the stack is moved up so that R occupies the TOS. No remainder is generated. Operands A and B are lost. Operands C and D are unchanged.
If A is zero, R is set equal to B and the divide-by-zero error status will be reported. If either A or B is the most negative value possible in the format, R will be meaningless and the overflow error status will be reported.

Status Affected: Sign, Zero, Error Field

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B		C
C		D
D		
← 32 →		← 32 →

DMUL

32-BIT FIXED-POINT MULTIPLY, LOWER

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	1	0	1	1	1	0

Hex Coding: AE with sr = 1
2E with sr = 0

Execution Time: 194 to 210 clock cycles

Description:
The 32-bit fixed-point two's complement integer operand A at the TOS is multiplied by the 32-bit fixed-point two's complement integer operand B at the NOS. The 32-bit least significant half of the product R replaces B and the stack is moved up so that R occupies the TOS. The most significant half of the product is lost. Operands A and B are lost. Operands C and D are unchanged.
The overflow status bit is set if the discarded upper half was non-zero. If either A or B is the most negative value that can be represented in the format, that value is returned as R and the overflow status is set.

Status Affected: Sign, Zero, Overflow

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B		C
C		D
D		
← 32 →		← 32 →

DMUU

32-BIT FIXED-POINT MULTIPLY, UPPER

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	1	1	0	1	1	0

Hex Coding: B6 with sr = 1
36 with sr = 0

Execution Time: 182 to 218 clock cycles

Description:
The 32-bit fixed-point two's complement integer operand A at the TOS is multiplied by the 32-bit fixed-point two's complement integer operand B at the NOS. The 32-bit most significant half of the product R replaces B and the stack is moved up so that R occupies the TOS. The least significant half of the product is lost. Operands A and B are lost. Operands C and D are unchanged.
If A or B was the most negative value possible in the format, overflow status is set and R is meaningless.

Status Affected: Sign, Zero, Overflow

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B		C
C		D
D		
← 32 →		← 32 →

DSUB

32-BIT FIXED-POINT SUBTRACT

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	1	0	1	1	0	1

Hex Coding: AD with sr = 1
2D with sr = 0

Execution Time: 38 to 40 clock cycles

Description:
The 32-bit fixed-point two's complement operand A at the TOS is subtracted from the 32-bit fixed-point two's complement operand B at the NOS. The difference R replaces operand B and the stack is moved up so that R occupies the TOS. Operand B is lost. Operands A, C and D are unchanged.
If the subtraction generates a borrow it is reported in the carry status bit. If A is the most negative value that can be represented in the format the overflow status is set. If the result cannot be represented in the data format range, the overflow bit is set and the 32 least significant bits of the result are returned as R.

Status Affected: Sign, Zero, Carry, Overflow

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B		C
C		D
D		A
← 32 →		← 32 →

EXP

32-BIT FLOATING-POINT e^X

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	1	0	1	0

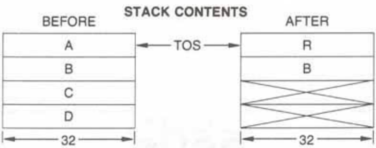
Hex Coding: 8A with sr = 1
0A with sr = 0

Execution Time: 3794 to 4878 clock cycles for $|A| \leq 1.0 \times 2^5$
34 clock cycles for $|A| > 1.0 \times 2^5$

Description:
The base of natural logarithms, e, is raised to an exponent value specified by the 32-bit floating-point operand A at the TOS. The result R of e^A replaces A. Operands A, C and D are lost. Operand B is unchanged.
EXP accepts all input data values within the range of $-1.0 \times 2^{+5}$ to $+1.0 \times 2^{+5}$. Input values outside this range will return a code of 1100 in the error field of the status register.

Accuracy: EXP exhibits a maximum relative error of 5.0×10^{-7} over the valid input data range.

Status Affected: Sign, Zero, Error Field



FADD

32-BIT FLOATING-POINT ADD

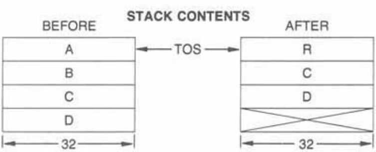
	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	0	0	0	0

Hex Coding: 90 with sr = 1
10 with sr = 0

Execution Time: 54 to 368 clock cycles for $A \neq 0$
24 clock cycles for $A = 0$

Description:
32-bit floating-point operand A at the TOS is added to 32-bit floating-point operand B at the NOS. The result R replaces B and the stack is moved up so that R occupies the TOS. Operands A and B are lost. Operands C and D are unchanged.
Exponent alignment before the addition and normalization of the result accounts for the variation in execution time. Exponent overflow and underflow are reported in the status register, in which case the mantissa is correct and the exponent is offset by 128.

Status Affected: Sign, Zero, Error Field



FDIV

32-BIT FLOATING-POINT DIVIDE

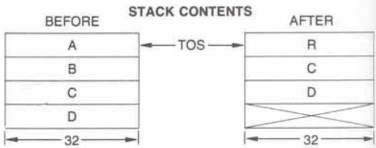
	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	0	0	1	1

Hex Coding: 93 with sr = 1
13 with sr = 0

Execution Time: 154 to 184 clock cycles for $A \neq 0$
22 clock cycles for $A = 0$

Description:
32-bit floating-point operand B at NOS is divided by 32-bit floating-point operand A at the TOS. The result R replaces B and the stack is moved up so that R occupies the TOS. Operands A and B are lost. Operands C and D are unchanged.
If operand A is zero, R is set equal to B and the divide-by-zero error is reported in the status register. Exponent overflow or underflow is reported in the status register, in which case the mantissa portion of the result is correct and the exponent portion is offset by 128.

Status Affected: Sign, Zero, Error Field



FIXD

32-BIT FLOATING-POINT TO 32-BIT FIXED-POINT CONVERSION

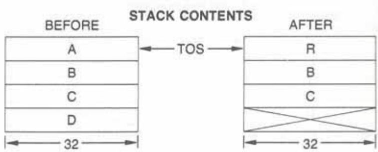
	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	1	1	1	0

Hex Coding: 9E with sr = 1
1E with sr = 0

Execution Time: 90 to 336 clock cycles

Description:
32-bit floating-point operand A at the TOS is converted to a 32-bit fixed-point two's complement integer. The result R replaces A. Operands A and D are lost. Operands B and C are unchanged.
If the integer portion of A is larger than 31 bits when converted, the overflow status will be set and A will not be changed. Operand D, however, will still be lost.

Status Affected: Sign, Zero Overflow



FIXS

32-BIT FLOATING-POINT TO 16-BIT FIXED-POINT CONVERSION

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	1	1	1	1

Hex Coding: 9F with sr = 1
1F with sr = 0

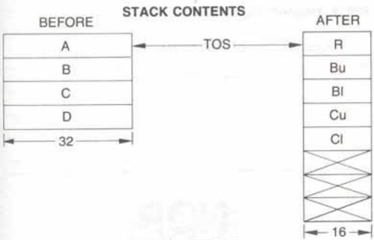
Execution Time: 90 to 214 clock cycles

Description:

32-bit floating-point operand A at the TOS is converted to a 16-bit fixed-point two's complement integer. The result R replaces the lower half of A and the stack is moved up by two bytes so that R occupies the TOS. Operands A and D are lost. Operands B and C are unchanged, but appear as upper (u) and lower (l) halves on the 16-bit wide stack if they are 32-bit operands.

If the integer portion of A is larger than 15 bits when converted, the overflow status will be set and A will not be changed. Operand D, however, will still be lost.

Status Affected: Sign, Zero, Overflow



FLTD

32-BIT FIXED-POINT TO 32-BIT FLOATING-POINT CONVERSION

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	1	1	0	0

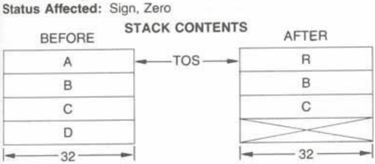
Hex Coding: 9C with sr = 1
1C with sr = 0

Execution Time: 56 to 342 clock cycles

Description:

32-bit fixed-point two's complement integer operand A at the TOS is converted to a 32-bit floating-point number. The result R replaces A at the TOS. Operands A and D are lost. Operands B and C are unchanged.

Status Affected: Sign, Zero



FLTS

16-BIT FIXED-POINT TO 32-BIT FLOATING-POINT CONVERSION

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	1	1	0	1

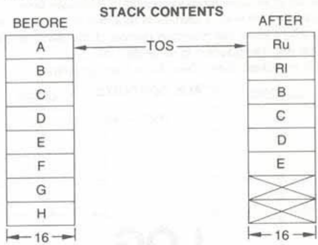
Hex Coding: 9D with sr = 1
1D with sr = 0

Execution Time: 62 to 156 clock cycles

Description:

16-bit fixed-point two's complement integer A at the TOS is converted to a 32-bit floating-point number. The lower half of the result R (Rl) replaces A, the upper half (Ru) replaces H and the stack is moved down so that Ru occupies the TOS. Operands A, F, G and H are lost. Operands B, C, D and E are unchanged.

Status Affected: Sign, Zero



FMUL

32-BIT FLOATING-POINT MULTIPLY

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	0	0	1	0

Hex Coding: 92 with sr = 1
12 with sr = 0

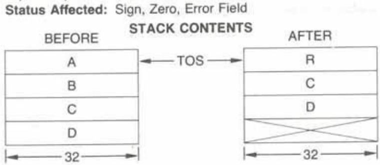
Execution Time: 146 to 168 clock cycles

Description:

32-bit floating-point operand A at the TOS is multiplied by the 32-bit floating-point operand B at the NOS. The normalized result R replaces B and the stack is moved up so that R occupies the TOS. Operands A and B are lost. Operands C and D are unchanged.

Exponent overflow or underflow is reported in the status register, in which case the mantissa portion of the result is correct and the exponent portion is offset by 128.

Status Affected: Sign, Zero, Error Field



FSUB

32-BIT FLOATING-POINT SUBTRACTION

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	0	0	0	1

Hex Coding: 91 with sr = 1
11 with sr = 0

Execution Time: 70 to 370 clock cycles for $A \neq 0$
26 clock cycles for $A = 0$

Description:
32-bit floating-point operand A at the TOS is subtracted from 32-bit floating-point operand B at the NOS. The normalized difference R replaces B and the stack is moved up so that R occupies the TOS. Operands A and B are lost. Operands C and D are unchanged.

Exponent alignment before the subtraction and normalization of the result account for the variation in execution time.

Exponent overflow or underflow is reported in the status register in which case the mantissa portion of the result is correct and the exponent portion is offset by 128.

Status Affected: Sign, Zero, Error Field (overflow)

STACK CONTENTS

BEFORE		AFTER
A	←TOS→	R
B		C
C		D
D		☒
←32→		←32→

LOG

32-BIT FLOATING-POINT COMMON LOGARITHM

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	1	0	0	0

Hex Coding: 88 with sr = 1
08 with sr = 0

Execution Time: 4474 to 7132 clock cycles for $A > 0$
20 clock cycles for $A \leq 0$

Description:
The 32-bit floating-point operand A at the TOS is replaced by R, the 32-bit floating-point common logarithm (base 10) of A. Operands A, C and D are lost. Operand B is unchanged.

The LOG function accepts any positive input data value that can be represented by the data format. If LOG of a non-positive value is attempted an error status of 0100 is returned.

Accuracy: LOG exhibits a maximum absolute error of 2.0×10^{-7} for the input range from 0.1 to 10, and a maximum relative error of 2.0×10^{-7} for positive values less than 0.1 or greater than 10.

Status Affected: Sign, Zero, Error Field

STACK CONTENTS

BEFORE		AFTER
A	←TOS→	R
B		B
C		☒
D		☒
←32→		←32→

LN

32-BIT FLOATING-POINT NATURAL LOGARITHM

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	1	0	0	1

Hex Coding: 89 with sr = 1
09 with sr = 0

Execution Time: 4298 to 6956 clock cycles for $A > 0$
20 clock cycles for $A \leq 0$

Description:
The 32-bit floating-point operand A at the TOS is replaced by R, the 32-bit floating-point natural logarithm (base e) of A. Operands A, C and D are lost. Operand B is unchanged.

The LN function accepts all positive input data values that can be represented by the data format. If LN of a non-positive number is attempted an error status of 0100 is returned.

Accuracy: LN exhibits a maximum absolute error of 2×10^{-7} for the input range from e^{-1} to e, and a maximum relative error of 2.0×10^{-7} for positive values less than e^{-1} or greater than e.

Status Affected: Sign, Zero, Error Field

STACK CONTENTS

BEFORE		AFTER
A	←TOS→	R
B		B
C		☒
D		☒
←32→		←32→

NOP

NO OPERATION

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	0	0	0	0

Hex Coding: 80 with sr = 1
00 with sr = 0

Execution Time: 4 clock cycles

Description:
The NOP command performs no internal data manipulations. It may be used to set or clear the service request interface line without changing the contents of the stack.

Status Affected: The status byte is cleared to all zeroes.

POPD

32-BIT STACK POP

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	1	1	1	0	0	0

Hex Coding: B8 with sr = 1
38 with sr = 0

Execution Time: 12 clock cycles

Description:
The 32-bit stack is moved up so that the old NOS becomes the new TOS. The previous TOS rotates to the bottom of the stack. All operand values are unchanged. POPD and POPF execute the same operation.

Status Affected: Sign, Zero

STACK CONTENTS

BEFORE		AFTER
A	←TOS→	B
B		C
C		D
D		A
←32→		←32→

POPF

32-BIT STACK POP

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	1	0	0	0

Hex Coding: 98 with sr = 1
18 with sr = 0

Execution Time: 12 clock cycles

Description:
The 32-bit stack is moved up so that the old NOS becomes the new TOS. The old TOS rotates to the bottom of the stack. All operand values are unchanged. POPF and POPD execute the same operation.

Status Affected: Sign, Zero

STACK CONTENTS

BEFORE		AFTER
A	←TOS→	B
B		C
C		D
D		A
←32→		←32→

POPS

16-BIT STACK POP

	7	6	5	4	3	2	1	0
Binary Coding:	sr	1	1	1	1	0	0	0

Hex Coding: F8 with sr = 1
78 with sr = 0

Execution Time: 10 clock cycles

Description:
The 16-bit stack is moved up so that the old NOS becomes the new TOS. The previous TOS rotates to the bottom of the stack. All operand values are unchanged.

Status Affected: Sign, Zero

STACK CONTENTS

BEFORE		AFTER
A	←TOS→	B
B		C
C		D
D		E
E		F
F		G
G		H
H		A
←16→		←16→

PTOD

PUSH 32-BIT TOS ONTO STACK

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	1	1	0	1	1	1

Hex Coding: B7 with sr = 1
37 with sr = 0

Execution Time: 20 clock cycles

Description:
The 32-bit stack is moved down and the previous TOS is copied into the new TOS location. Operand D is lost. All other operand values are unchanged. PTOD and PTOF execute the same operation.

Status Affected: Sign, Zero

STACK CONTENTS

BEFORE		AFTER
A	←TOS→	A
B		A
C		B
D		C
←32→		←32→

PTOF

PUSH 32-BIT TOS ONTO STACK

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	0	1	1	1

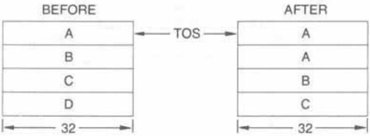
Hex Coding: 97 with sr = 1
17 with sr = 0

Execution Time: 20 clock cycles

Description:
The 32-bit stack is moved down and the previous TOS is copied into the new TOS location. Operand D is lost. All other operand values are unchanged. PTOF and PTOD execute the same operation.

Status Affected: Sign, Zero

STACK CONTENTS



PTOS

PUSH 16-BIT TOS ONTO STACK

	7	6	5	4	3	2	1	0
Binary Coding:	sr	1	1	1	0	1	1	1

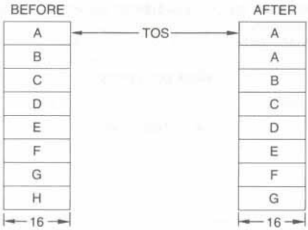
Hex Coding: F7 with sr = 1
77 with sr = 0

Execution Time: 16 clock cycles

Description:
The 16-bit stack is moved down and the previous TOS is copied into the new TOS location. Operand H is lost and all other operand values are unchanged.

Status Affected: Sign, Zero

STACK CONTENTS



PUPI

PUSH 32-BIT FLOATING-POINT π

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	1	1	0	1	0

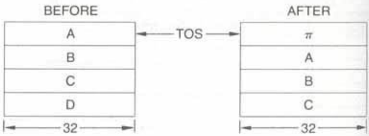
Hex Coding: 9A with sr = 1
1A with sr = 0

Execution Time: 16 clock cycles

Description:
The 32-bit stack is moved down so that the previous TOS occupies the new NOS location. 32-bit floating-point constant π is entered into the new TOS location. Operand D is lost. Operands A, B and C are unchanged.

Status Affected: Sign, Zero

STACK CONTENTS



PWR

32-BIT FLOATING-POINT X^Y

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	1	0	1	1

Hex Coding: 8B with sr = 1
0B with sr = 0

Execution Time: 8290 to 12032 clock cycles

Description:

32-bit floating-point operand B at the NOS is raised to the power specified by the 32-bit floating-point operand A at the TOS. The result R of B^A replaces B and the stack is moved up so that R occupies the TOS. Operands A, B, and D are lost. Operand C is unchanged.

The PWR function accepts all input data values that can be represented in the data format for operand A and all positive values for operand B. If operand B is non-positive an error status of 0100 will be returned. The EXP and LN functions are used to implement PWR using the relationship $B^A = EXP\,[A(LN\,B)]$. Thus if the term $[A(LN\,B)]$ is outside the range of $-1.0 \times 2^{+5}$ to $+1.0 \times 2^{+5}$ an error status of 1100 will be returned. Underflow and overflow conditions can occur.

Accuracy: The error performance for PWR is a function of the LN and EXP performance as expressed by:

$|(\text{Relative Error})_{PWR}| = |(\text{Relative Error})_{EXP} + |A(\text{Absolute Error})_{LN}|$

The maximum relative error for PWR occurs when A is at its maximum value while $[A(LN\,B)]$ is near 1.0×2^5 and the EXP error is also at its maximum. For most practical applications the relative error for PWR will be less than 7.0×10^{-7}

Status Affected: Sign, Zero, Error Field

STACK CONTENTS

BEFORE		AFTER
A	←TOS→	R
B		C
C		╳
D		╳
←32→		←32→

SADD

16-BIT FIXED-POINT ADD

	7	6	5	4	3	2	1	0
Binary Coding:	sr	1	1	0	1	1	0	0

Hex Coding: EC with sr = 1
6C with sr = 0

Execution Time: 16 to 18 clock cycles

Description:

16-bit fixed-point two's complement integer operand A at the TOS is added to 16-bit fixed-point two's complement integer operand B at the NOS. The result R replaces B and the stack is moved up so that R occupies the TOS. Operand B is lost. All other operands are unchanged.

If the addition generates a carry bit it is reported in the status register. If an overflow occurs it is reported in the status register and the 16 least significant bits of the result are returned.

Status Affected: Sign, Zero, Carry, Error Field

STACK CONTENTS

BEFORE		AFTER
A	←TOS→	R
B		C
C		D
D		E
E		F
F		G
G		H
H		A
←16→		←16→

SDIV

16-BIT FIXED-POINT DIVIDE

	7	6	5	4	3	2	1	0
Binary Coding:	sr	1	1	0	1	1	1	1

Hex Coding: EF with sr = 1
6F with sr = 0

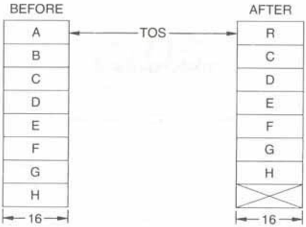
Execution Time: 84 to 94 clock cycles for $A \neq 0$
14 clock cycles for $A = 0$

Description:
16-bit fixed-point two's complement integer operand B at the NOS is divided by 16-bit fixed-point two's complement integer operand A at the TOS. The 16-bit integer quotient R replaces B and the stack is moved up so that R occupies the TOS. No remainder is generated. Operands A and B are lost. All other operands are unchanged.

If A is zero, R will be set equal to B and the divide-by-zero error status will be reported.

Status Affected: Sign, Zero, Error Field

STACK CONTENTS



SIN

32-BIT FLOATING-POINT SINE

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	0	0	1	0

Hex Coding: 82 with sr = 1
02 with sr = 0

Execution Time: 3796 to 4808 clock cycles for $|A| > 2^{-12}$ radians
30 clock cycles for $|A| \leq 2^{-12}$ radians

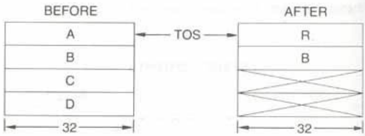
Description:
The 32-bit floating-point operand A at the TOS is replaced by R, the 32-bit floating-point sine of A. A is assumed to be in radians. Operands A, C and D are lost. Operand B is unchanged.

The SIN function will accept any input data value that can be represented by the data format. All input values are range reduced to fall within the interval $-\pi/2$ to $+\pi/2$ radians.

Accuracy: SIN exhibits a maximum relative error of 5.0×10^{-7} for input values in the range of -2π to $+2\pi$ radians.

Status Affected: Sign, Zero

STACK CONTENTS



SMUL

16-BIT FIXED-POINT MULTIPLY, LOWER

Binary Coding:

7	6	5	4	3	2	1	0
sr	1	1	0	1	1	1	0

Hex Coding: EE with sr = 1
6E with sr = 0

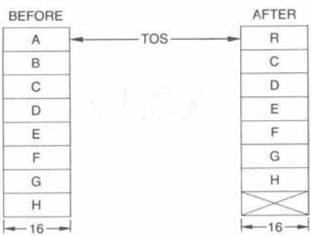
Execution Time: 84 to 94 clock cycles

Description:

16-bit fixed-point two's complement integer operand A at the TOS is multiplied by the 16-bit fixed-point two's complement integer operand B at the NOS. The 16-bit least significant half of the product R replaces B and the stack is moved up so that R occupies the TOS. The most significant half of the product is lost. Operands A and B are lost. All other operands are unchanged. The overflow status bit is set if the discarded upper half was non-zero. If either A or B is the most negative value that can be represented in the format, that value is returned as R and the overflow status is set.

Status Affected: Sign, Zero, Error Field

STACK CONTENTS



SMUU

16-BIT FIXED-POINT MULTIPLY, UPPER

Binary Coding:

7	6	5	4	3	2	1	0
sr	1	1	1	0	1	1	0

Hex Coding: F6 with sr = 1
76 with sr = 0

Execution Time: 80 to 98 clock cycles

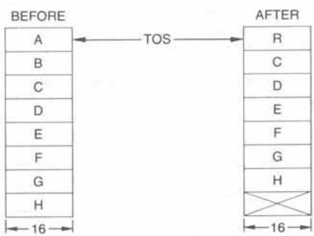
Description:

16-bit fixed-point two's complement integer operand A at the TOS is multiplied by the 16-bit fixed-point two's complement integer operand B at the NOS. The 16-bit most significant half of the product R replaces B and the stack is moved up so that R occupies the TOS. The least significant half of the product is lost. Operands A and B are lost. All other operands are unchanged.

If either A or B is the most negative value that can be represented in the format, that value is returned as R and the overflow status is set.

Status Affected: Sign, Zero, Error Field

STACK CONTENTS



SQRT

32-BIT FLOATING-POINT SQUARE ROOT

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	0	0	0	1

Hex Coding: 81 with sr = 1
01 with sr = 0

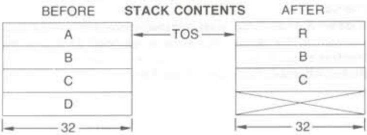
Execution Time: 782 to 870 clock cycles

Description:

32-bit floating-point operand A at the TOS is replaced by R, the 32-bit floating-point square root of A. Operands A and D are lost. Operands B and C are not changed.

SQRT will accept any non-negative input data value that can be represented by the data format. If A is negative an error code of 0100 will be returned in the status register.

Status Affected: Sign, Zero, Error Field



SSUB

16-BIT FIXED-POINT SUBTRACT

	7	6	5	4	3	2	1	0
Binary Coding:	sr	1	1	0	1	1	0	1

Hex Coding: ED with sr = 1
6D with sr = 0

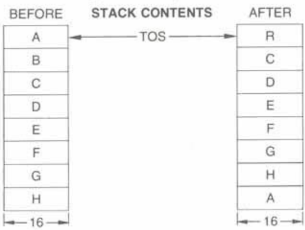
Execution Time: 30 to 32 clock cycles

Description:

16-bit fixed-point two's complement integer operand A at the TOS is subtracted from 16-bit fixed-point two's complement integer operand B at the NOS. The result R replaces B and the stack is moved up so that R occupies the TOS. Operand B is lost. All other operands are unchanged.

If the subtraction generates a borrow it is reported in the carry status bit. If A is the most negative value that can be represented in the format the overflow status is set. If the result cannot be represented in the format range, the overflow status is set and the 16 least significant bits of the result are returned as R.

Status Affected: Sign, Zero, Carry, Error Field



TAN

32-BIT FLOATING-POINT TANGENT

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	0	0	0	1	0	0

Hex Coding: 84 with sr = 1
04 with sr = 0

Execution Time: 4894 to 5886 clock cycles for $|A| > 2^{-12}$ radians
30 clock cycles for $|A| \le 2^{-12}$ radians

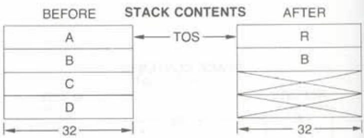
Description:

The 32-bit floating-point operand A at the TOS is replaced by the 32-bit floating-point tangent of A. Operand A is assumed to be in radians. A, C and D are lost. B is unchanged.

The TAN function will accept any input data value that can be represented in the data format. All input data values are range-reduced to fall within $-\pi/4$ to $+\pi/4$ radians. TAN is unbounded for input values near odd multiples of $\pi/2$ and in such cases the overflow bit is set in the status register. For angles smaller than 2^{-12} radians, TAN returns A as the tangent of A.

Accuracy: TAN exhibits a maximum relative error of 5.0×10^{-7} for input data values in the range of -2π to $+2\pi$ radians except for data values near odd multiples of $\pi/2$.

Status Affected: Sign, Zero, Error Field (overflow)



XCHD

EXCHANGE 32-BIT STACK OPERANDS

	7	6	5	4	3	2	1	0
Binary Coding:	sr	0	1	1	1	0	0	1

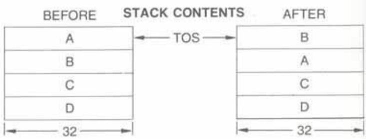
Hex Coding: B9 with sr = 1
39 with sr = 0

Execution Time: 26 clock cycles

Description:

32-bit operand A at the TOS and 32-bit operand B at the NOS are exchanged. After execution, B is at the TOS and A is at the NOS. All operands are unchanged. XCHD and XCHF execute the same operation.

Status Affected: Sign, Zero



XCHF

EXCHANGE 32-BIT STACK OPERANDS

Binary Coding:

7	6	5	4	3	2	1	0
sr	0	0	1	1	0	0	1

Hex Coding: 99 with sr = 1
19 with sr = 0

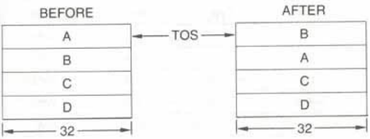
Execution Time: 26 clock cycles

Description:

32-bit operand A at the TOS and 32-bit operand B at the NOS are exchanged. After execution, B is at the TOS and A is at the NOS. All operands are unchanged. XCHD and XCHF execute the same operation.

Status Affected: Sign, Zero

STACK CONTENTS



XCHS

EXCHANGE 16-BIT STACK OPERANDS

Binary Coding:

7	6	5	4	3	2	1	0
sr	1	1	1	1	0	0	1

Hex Coding: F9 with sr = 1
79 with sr = 0

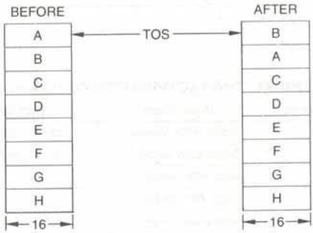
Execution Time: 18 clock cycles

Description:

16-bit operand A at the TOS and 16-bit operand B at the NOS are exchanged. After execution, B is at the TOS and A is at the NOS. All operand values are unchanged.

Status Affected: Sign, Zero

STACK CONTENTS



MAXIMUM RATINGS beyond which useful life may be impaired

Storage Temperature	−65 to +150°C
VDD with Respect to VSS	−0.5V to +15.0V
VCC with Respect to VSS	−0.5V to +7.0V
All Signal Voltages with Respect to VSS	−0.5V to +7.0V
Power Dissipation (Package Limitation)	2.0W

The products described by this specification include internal circuitry designed to protect input devices from damaging accumulations of static charge. It is suggested, nevertheless, that conventional precautions be observed during storage, handling and use in order to avoid exposure to excessive voltages.

OPERATING RANGE

Part Number	Ambient Temperature	VSS	VCC	VDD
Am9511ADC	$0°C \leq T_A \leq 70°C$	0V	+5.0V ±5%	+12V ±5%
Am9511A-1DC	$0°C \leq T_A \leq 70°C$	0V	+5.0V ±5%	+12V ±5%
Am9511A-4DC	$0°C \leq T_A \leq 70°C$	0V	+5.0V ±5%	+12V ±5%
Am9511ADI	$-40°C \leq T_A \leq 85°C$	0V	+5.0V ±10%	+12V ±10%
Am9511A-1DI	$-40°C \leq T_A \leq 85°C$	0V	+5.0V ±10%	+12V ±10%
Am9511ADM	$-55°C \leq T_A \leq 125°C$	0V	+5.0V ±10%	+12V ±10%
Am9511A-1DM	$-55°C \leq T_A \leq 125°C$	0V	+5.0V ±10%	+12V ±10%

ELECTRICAL CHARACTERISTICS Over Operating Range (Note 1)

Parameters	Description	Test Conditions	Min.	Typ.	Max.	Units
VOH	Output HIGH Voltage	IOH = −200μA	3.7			Volts
VOL	Output LOW Voltage	IOL = 3.2mA			0.4	Volts
VIH	Input HIGH Voltage		2.0		VCC	Volts
VIL	Input LOW Voltage		−0.5		0.8	Volts
IIX	Input Load Current	VSS ≤ VI ≤ VCC			±10	μA
IOZ	Data Bus Leakage	VO = 0.4V			10	μA
		VO = VCC			10	
ICC	VCC Supply Current	T_A = +25°C		50	90	mA
		T_A = 0°C			95	
		T_A = −55°C			100	
IDD	VDD Supply Current	T_A = +25°C		50	90	mA
		T_A = 0°C			95	
		T_A = −55°C			100	
CO	Output Capacitance	fc = 1.0MHz, Inputs = 0V		8	10	pF
CI	Input Capacitance			5	8	pF
CIO	I/O Capacitance			10	12	pF

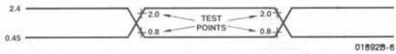
SWITCHING CHARACTERISTICS

Parameters	Description		Am9511A Min	Am9511A Max	Am9511A-1 Min	Am9511A-1 Max	Am9511A-4 Min	Am9511A-4 Max	Units
TAPW	EACK LOW Pulse Width		100		75		50		ns
TCDR	C/D to RD LOW Set-up Time		0		0		0		ns
TCDW	C/D to WR LOW Set-up Time		0		0		0		ns
TCPH	Clock Pulse HIGH Width		200		140		100		ns
TCPL	Clock Pulse LOW Width		240		160		120		ns
TCSR	CS LOW to RD LOW Set-up Time		0		0		0		ns
TCSW	CS LOW to WR LOW Set-up Time		0		0		0		ns
TCY	Clock Period		480	5000	320	3300	250	2500	ns
TDW	Data Bus Stable to WR HIGH Set-up Time		150		100 (Note 9)		100		ns
TEAE	EACK LOW to END HIGH Delay			200		175		150	ns
TEPW	END LOW Pulse Width (Note 4)		400		300		200		ns
TOP	Data Bus Output Valid to PAUSE HIGH Delay		0		0		0		ns
TPPWR	PAUSE LOW Pulse Width Read (Note 5)	Data	3.5TCY+50	5.5TCY+300	3.5TCY+50	5.5TCY+200	3.5TCY+50	5.5TCY+200	ns
		Status	1.5TCY+50	3.5TCY+300	1.5TCY+50	3.5TCY+200	1.5TCY+50	3.5TCY+200	
TPPWW	PAUSE LOW Pulse Width Write (Note 8)			50		50		50	ns
TPR	PAUSE HIGH to RD HIGH Hold Time		0		0		0		ns
TPW	PAUSE HIGH to WR HIGH Hold Time		0		0		0		ns
TRCD	RD HIGH to C/D Hold Time		0		0		0		ns
TRCS	RD HIGH to CS HIGH Hold Time		0		0		0		ns
TRO	RD LOW to Data Bus ON Delay		50		50		25		ns
TRP	RD LOW to PAUSE LOW Delay (Note 6)			150		100 (Note 9)		100	ns
TRZ	RD HIGH to Data Bus OFF Delay		50	200	50	150	25	100	ns
TSAPW	SVACK LOW Pulse Width		100		75		50		ns
TSAR	SVACK LOW to SVREQ LOW Delay			300		200		150	ns
TWCD	WR HIGH to C/D Hold Time		60		30		30		ns
TWCS	WR HIGH to CS HIGH Hold Time		60		30		30		ns
TWD	WR HIGH to Data Bus Hold Time		20		20		20		ns
TWI	Write Inactive Time	Command	3TCY		3TCY		3TCY		ns
		Data	4TCY		4TCY		4TCY		
TWP	WR LOW to PAUSE LOW Delay (Note 6)			150		100 (Note 9)		100	ns

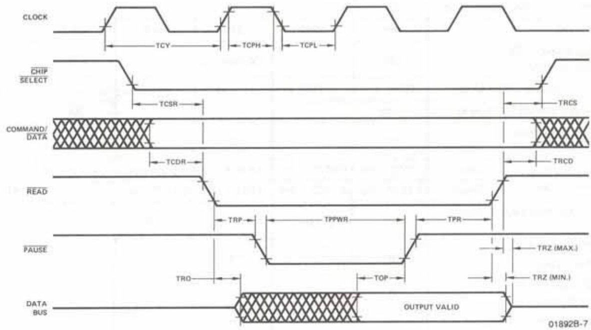
Notes:
1. Typical values are for $T_A = 25°C$, nominal supply voltages and nominal processing parameters.
2. Switching parameters are listed in alphabetical order.
3. Test conditions assume transition times of 20ns or less, output loading of one TTL gate plus 100pF and timing reference levels of 0.8V and 2.0V.
4. END low pulse width is specified for EACK tied to VSS. Otherwise TEAE applies.
5. Minimum values shown assume no previously entered command is being executed for the data access. If a previously entered command is being executed, PAUSE LOW Pulse Width is the time to complete execution plus the time shown. Status may be read at any time without exceeding the time shown.
6. PAUSE is pulled low for both command and data operations.
7. TEX is the execution time of the current command (see the Command Execution Times table).
8. PAUSE low pulse width is less than 50ns when writing into the data port or the control port as long as the duty requirement (TWI) is observed and no previous command is being executed. TWI may be safely violated up to 500ns as long as the extended TPPWW that results is observed. If a previously entered command is being executed, PAUSE LOW Pulse Width is the time to complete execution plus the time shown.
9. 150ns for the Am9511A-1DM.

SWITCHING WAVEFORMS

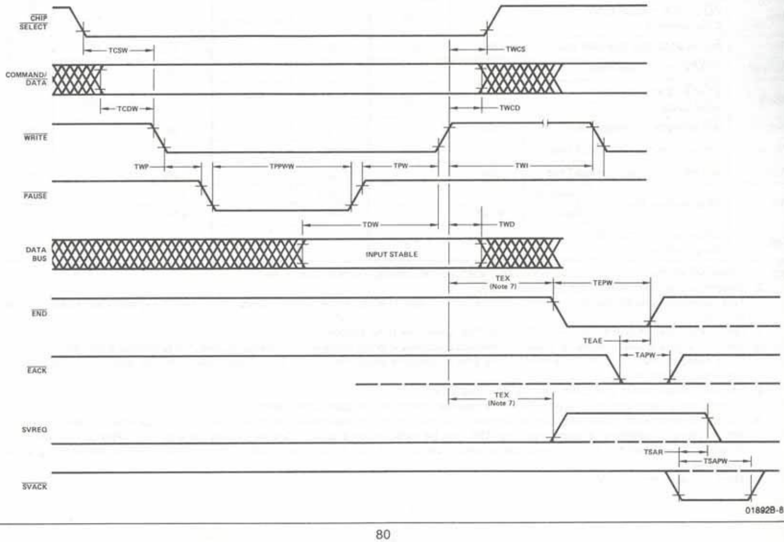
INPUT WAVEFORMS FOR AC TESTS



READ OPERATIONS



WRITE OPERATIONS



APPLICATION INFORMATION

The diagram in Figure 2 shows the interface connections for the Am9511A APU with operand transfers handled by an Am9517A DMA controller, and CPU coordination handled by an Am9519A Interrupt Controller. The APU interrupts the CPU to indicate that a command has been completed. When the performance enhancements provided by the DMA and Interrupt operations are not required, the APU interface can be simplified as shown in Figure 1. The Am9511A APU is designed with a general purpose 8-bit data bus and interface control so that it can be conveniently used with any general 8-bit processor.

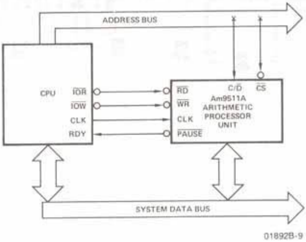


Figure 1. Am9511A Minimum Configuration Example.

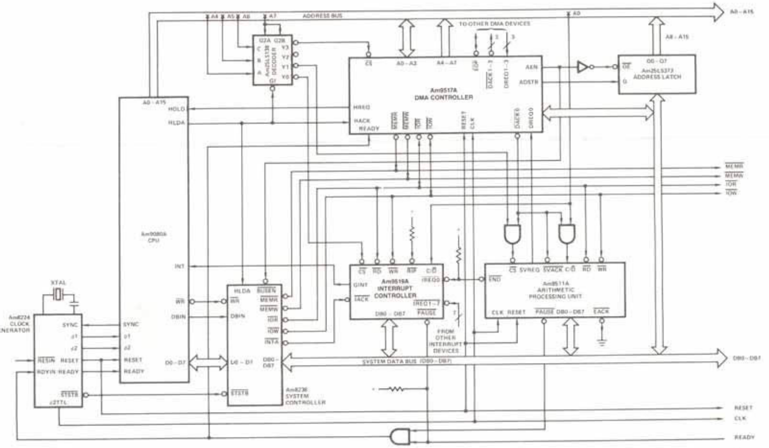
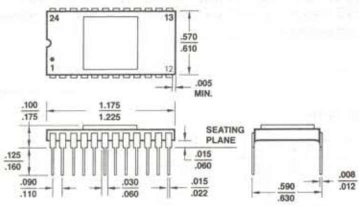


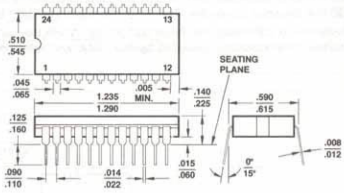
Figure 2. Am9511A High Performance Configuration Example.

PHYSICAL DIMENSIONS
Dual In-Line

SD-024



CD-024



Appendix B

Am9512

Floating Point Processor

DISTINCTIVE CHARACTERISTICS

- Single (32-bit) and double (64-bit) precision capability
- Add, subtract, multiply and divide functions
- Compatible with proposed IEEE format
- Easy interfacing to microprocessors
- 8-bit data bus
- Standard 24-pin package
- 12V and 5V power supplies
- Stack oriented operand storage
- Direct memory access or programmed I/O Data Transfers
- End of execution signal
- Error interrupt
- All inputs and outputs TTL level compatible
- Advanced N-channel silicon gate MOS technology

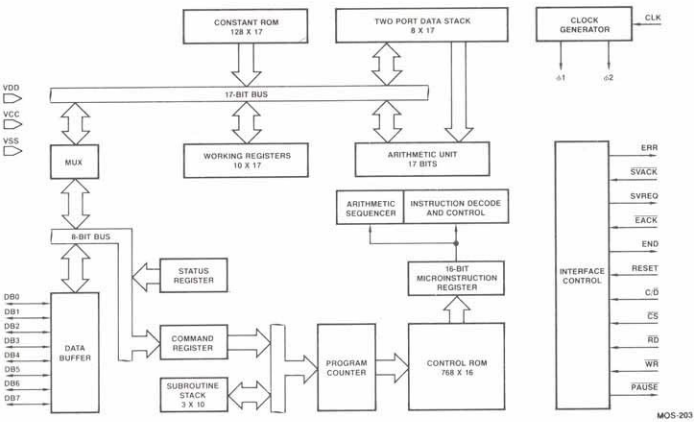
GENERAL DESCRIPTION

The Am9512 is a high performance floating-point processor unit (FPU). It provides single precision (32-bit) and double precision (64-bit) add, subtract, multiply and divide operations. It can be easily interfaced to enhance the computational capabilities of the host microprocessor.

The operand, result, status and command information transfers take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack by the host processor and a command is issued to perform an operation on the data stack. The results of this operation are available to the host processor by popping the stack.

Information transfers between the Am9512 and the host processor can be handled by using programmed I/O or direct memory access techniques. After completing an operation, the Am9512 activates an "end of execution" signal that can be used to interrupt the host processor.

BLOCK DIAGRAM



MOS-203

ORDERING INFORMATION

Package	Ambient Temperature	V_{SS}	V_{CC}	V_{DD}	Maximum Clock Frequency	
					2MHz	3MHz
Hermetic DIP	$0°C \leq T_A \leq 70°C$	0V	+5.0V ±5%	+12V ±5%	Am9512DC	Am9512-1DC
	$-40°C \leq T_A \leq +85°C$	0V	+5.0V ±10%	+12V ±10%	Am9512DI	Am9512-1DI
	$-55°C \leq T_A \leq +125°C$	0V	+5.0V ±10%	+12V ±10%	Am9512DMB	Am9512-1DMB

02047B-MMP

CONNECTION DIAGRAM
Top View

Note: Pin 1 is marked for orientation.

MOS-204

INTERFACE SIGNAL DESCRIPTION

VCC: +5V Power Supply

VDD: +12V Power Supply

VSS: Ground

CLK (Clock, Input)

An external timing source connected to the CLK input provides the necessary clocking.

RESET (Reset, Input)

A HIGH on this input causes initialization. Reset terminates any operation in progress, and clears the status register to zero. The internal stack pointer is initialized and the contents of the stack may be affected. After a reset the END output, the ERR output and the SVREQ output will be LOW. For proper initialization, RESET must be HIGH for at least five CLK periods following stable power supply voltages and stable clock.

C/$\overline{\text{D}}$ (Command/Data Select, Input)

The C/$\overline{\text{D}}$ input together with the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ inputs determines the type of transfer to be performed on the data bus as follows:

C/$\overline{\text{D}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	Function
L	H	L	Push data byte into the stack
L	L	H	Pop data byte from the stack
H	H	L	Enter command
H	L	H	Read Status
X	L	L	Undefined

L = LOW
H = HIGH
X = DON'T CARE

END (End of Execution, Output)

A HIGH on this output indicates that execution of the current command is complete. This output will be cleared LOW by activating the $\overline{\text{EACK}}$ input LOW or performing any read or write operation or device initialization using the RESET. If $\overline{\text{EACK}}$ is tied LOW, the END output will be a pulse (see $\overline{\text{EACK}}$ description).

Reading the status register while a command execution is in progress is allowed. However any read or write operation clears the flip-flop that generates the END output. Thus such continuous reading could conflict with internal logic setting of the END flip-flop at the end of command execution.

$\overline{\text{EACK}}$ (End Acknowledge, Input)

This input when LOW makes the END output go LOW. As mentioned earlier HIGH on the END output signals completion of a command execution. The END signal is derived from an internal flip-flop which is clocked at the completion of a command. This flip-flop is clocked to the reset state when $\overline{\text{EACK}}$ is LOW. Consequently, if $\overline{\text{EACK}}$ is tied LOW, the END output will be a pulse that is approximately one CLK period wide.

SVREQ (Service Request, Output)

A HIGH on this output indicates completion of a command. In this sense this output is the same as the END output. However, the Service Bit in the Command Register determines whether the SVREQ output will go HIGH at the completion of a command. This bit must be 1 for SVREQ to go HIGH. The SVREQ can be cleared (i.e., go LOW) by activating the $\overline{\text{SVACK}}$ input LOW or initializing the device using the RESET. Also, the SVREQ will be automatically cleared after completion of any command that has the service request bit as 0.

$\overline{\text{SVACK}}$ (Service Acknowledge, Input)

A LOW on this input clears SVREQ. If the $\overline{\text{SVACK}}$ input is permanently tied LOW, it will conflict with the internal setting of the SVREQ output. Thus the SVREQ indication cannot be relied upon if the $\overline{\text{SVACK}}$ is tied LOW.

DB0-DB7 (Data Bus, Input/Output)

These eight bidirectional lines are used to transfer command, status and operand information between the device and the host processor. DB0 is the least significant and DB7 is the most significant bit position. HIGH on a data bus line corresponds to 1 and LOW corresponds to 0.

When pushing operands on the stack using the data bus, the least significant byte must be pushed first and most significant byte last. When popping the stack to read the result of an operation, the most significant byte will be available on the data bus first and the least significant byte will be the last. Moreover, for pushing operands and popping results, the number of transactions must be equal to the proper number of bytes appropriate for the chosen format. Otherwise, the internal byte pointer will not be aligned properly. The Am9512 single precision format requires 4 bytes and double precision format requires 8 bytes.

ERR (Error, Output)

This output goes HIGH to indicate that the current command execution resulted in an error condition. The error conditions are: attempt to divide by zero, exponent overflow and exponent underflow. The ERR output is cleared LOW on read status register operation or upon RESET.

The ERR output is derived from the error bits in the status register. These error bits will be updated internally at an appropriate time during a command execution. Thus ERR output going HIGH may not correspond with the completion of a command. Reading of the status register can be performed while a command execution is in progress. However it should be noted that reading the status register clears the ERR output. Thus reading the status register while a command execution in progress may result in an internal conflict with the ERR output.

$\overline{CS}$ (Chip Select, Input)

This input must be LOW to accomplish any read or write operation to the Am9512.

To perform a write operation, appropriate data is presented on DB0 through DB7 lines, appropriate logic level on the C/$\overline{D}$ input and the $\overline{CS}$ input is made LOW. Whenever $\overline{WR}$ and $\overline{RD}$ inputs are both HIGH and $\overline{CS}$ is LOW, $\overline{PAUSE}$ goes LOW. However actual writing into the Am9512 cannot start until $\overline{WR}$ is made LOW. After initiating the write operation by the HIGH to LOW transition on the $\overline{WR}$ input, the $\overline{PAUSE}$ output will go HIGH indicating the write operation has been acknowledged. The $\overline{WR}$ input can go HIGH after $\overline{PAUSE}$ goes HIGH. The data lines, C/$\overline{D}$ input and the $\overline{CS}$ input can change when appropriate hold time requirements are satisfied. See write timing diagram for details.

To perform a read operation an appropriate logic level is established on the C/$\overline{D}$ input and $\overline{CS}$ is made LOW. The $\overline{PAUSE}$ output goes LOW because $\overline{WR}$ and $\overline{RD}$ inputs are HIGH. The read operation does not start until the $\overline{RD}$ input goes LOW. $\overline{PAUSE}$ will go HIGH indicating that read operation is complete and the required information is available on the DB0 through DB7 lines. This information will remain on the data lines as long as $\overline{RD}$ is LOW. The $\overline{RD}$ input can return HIGH anytime after $\overline{PAUSE}$ goes HIGH. The $\overline{CS}$ input and C/$\overline{D}$ input can change anytime after $\overline{RD}$ returns HIGH. See read timing diagram for details. If the $\overline{CS}$ is tied LOW permanently, $\overline{PAUSE}$ will remain LOW until the next Am9512 read or write access.

$\overline{RD}$ (Read, Input)

A LOW on this input is used to read information from an internal location and gate that information onto the data bus. The $\overline{CS}$ input must be LOW to accomplish the read operation. The C/$\overline{D}$ input determines what internal location is of interest. See C/$\overline{D}$, $\overline{CS}$ input descriptions and read timing diagram for details. If the END output was HIGH, performing any read operation will make the END output go LOW after the HIGH to LOW transition of the $\overline{RD}$ input (assuming $\overline{CS}$ is LOW). If the ERR output was HIGH performing a status register read operation will make the ERR output LOW. This will happen after the HIGH to LOW transition of the RD input (assuming $\overline{CS}$ is LOW).

$\overline{WR}$ (Write, Input)

A LOW on this input is used to transfer information from the data bus into an internal location. The $\overline{CS}$ must be LOW to accomplish the write operation. The C/$\overline{D}$ determines which internal location is to be written. See C/$\overline{D}$, $\overline{CS}$ input descriptions and write timing diagram for details.

If the END output was HIGH, performing any write operation will make the END output go LOW after the LOW to HIGH transition of the $\overline{WR}$ input (assuming $\overline{CS}$ is LOW).

$\overline{PAUSE}$ (Pause, Output)

This output is a handshake signal used while performing read or write transactions with the Am9512. If the $\overline{WR}$ and $\overline{RD}$ inputs are both HIGH, the $\overline{PAUSE}$ output goes LOW with the $\overline{CS}$ input in anticipation of a transaction. If $\overline{WR}$ goes LOW to initiate a write transaction with proper signals established on the DB0-DB7, C/$\overline{D}$ inputs, the $\overline{PAUSE}$ will return HIGH indicating that the write operation has been accomplished. The $\overline{WR}$ can be made HIGH after this event. On the other hand, if a read operation is desired, the $\overline{RD}$ input is made LOW after activating $\overline{CS}$ LOW and establishing proper C/$\overline{D}$ input. (The $\overline{PAUSE}$ will go LOW in response to $\overline{CS}$ going LOW.) The $\overline{PAUSE}$ will return HIGH indicating completion of read. The $\overline{RD}$ can return HIGH after this event. It should be noted that a read or write operation can be initiated without any regard to whether a command execution is in progress or not. Proper device operation is assured by obeying the $\overline{PAUSE}$ output indication as described.

FUNCTIONAL DESCRIPTION

Major functional units of the Am9512 are shown in the block diagram. The Am9512 employs a microprogram controlled stack oriented architecture with 17-bit wide data paths.

The Arithmetic Unit receives one of its operands from the Operand Stack. This stack is an eight word by 17-bit two port memory with last in – first out (LIFO) attributes. The second operand to the Arithmetic Unit is supplied by the internal 17-bit bus. In addition to supplying the second operand, this bidirectional bus also carries the results from the output of the Arithmetic Unit when required. Writing into the Operand Stack takes place from this internal 17-bit bus when required. Also connected to this bus are the Constant ROM and Working Registers. The ROM provides the required constants to perform the mathematical operations while the Working Registers provide storage for the intermediate values during command execution.

Communication between the external world and the Am9512 takes place on eight bidirectional input/output lines, DB0 through DB7 (Data Bus). These signals are gated to the internal 8-bit bus through appropriate interface and buffer circuitry. Multiplexing facilities exist for bidirectional communication between the internal eight and 17-bit buses. The Status Register and Command Register are also located on the 8-bit bus.

The Am9512 operations are controlled by the microprogram contained in the Control ROM. The Program Counter supplies the microprogram addresses and can be partially loaded from the Command Register. Associated with the Program Counter is the Subroutine Stack where return addresses are held during subroutine calls in the microprogram. The Microinstruction Register holds the current microinstruction being executed. The register facilitates pipelined microprogram execution. The Instruction Decode logic generates various internal control signals needed for the Am9512 operation.

The Interface Control logic receives several external inputs and provides handshake related outputs to facilitate interfacing the Am9512 to microprocessors.

COMMAND FORMAT

The Operation of the Am9512 is controlled from the host processor by issuing instructions called commands. The command format is shown below:

SVREQ ENB	OP CODE						
7	6	5	4	3	2	1	0

The command consists of 8 bits; the least significant 7 bits specify the operation to be performed as detailed in the accompanying table. The most significant bit is the Service Request Enable bit. This bit must be a 1 if SVREQ is to go high at end of executing a command.

The Am9512 commands fall into three categories: Single precision arithmetic, double precision arithmetic and data manipulation. There are four arithmetic operations that can be performed with single precision (32-bit), or double precision (64-bit) floating-point numbers: add, subtract, multiply and divide. These operations require two operands. The Am9512 assumes that these operands are located in the internal stack as Top of Stack

(TOS) and Next on Stack (NOS). The result will always be returned to the previous NOS which becomes the new TOS. Results from an operation are of the same precision and format as the operands. The results will be rounded to preserve the accuracy. The actual data formats and rounding procedures are described in a later section. In addition to the arithmetic operations, the Am9512 implements eight data manipulating operations. These include changing the sign of a double or single precision operand located in TOS, exchanging single precision operands located at TOS and NOS, as well as copying and popping single or double precision operands. See also the sections on status register and operand formats.

The Execution times of the Am9512 commands are all data dependent. Table 2 shows one example of each command execution time:

Table 1. Command Decoding Table.

Command Bits								Mnemonic	Description
7	6	5	4	3	2	1	0		
X	0	0	0	0	0	0	1	SADD	Add TOS to NOS Single Precision and result to NOS. Pop stack.
X	0	0	0	0	0	1	0	SSUB	Subtract TOS from NOS Single Precision and result to NOS. Pop stack.
X	0	0	0	0	0	1	1	SMUL	Multiply NOS by TOS Single Precision and result to NOS. Pop stack.
X	0	0	0	0	1	0	0	SDIV	Divide NOS by TOS Single Precision and result to NOS. Pop stack.
X	0	0	0	0	1	0	1	CHSS	Change sign of TOS Single Precision operand.
X	0	0	0	0	1	1	0	PTOS	Push Single Precision operand on TOS to NOS.
X	0	0	0	0	1	1	1	POPS	Pop Single Precision operand from TOS. NOS becomes TOS.
X	0	0	0	1	0	0	0	XCHS	Exchange TOS with NOS Single Precision.
X	0	1	0	1	1	0	1	CHSD	Change sign of TOS Double Precision operand.
X	0	1	0	1	1	1	0	PTOD	Push Double Precision operand on TOS to NOS.
X	0	1	0	1	1	1	1	POPD	Pop Double Precision operand from TOS. NOS becomes TOS.
X	0	0	0	0	0	0	0	CLR	CLR status.
X	0	1	0	1	0	0	1	DADD	Add TOS to NOS Double Precision and result to NOS. Pop stack.
X	0	1	0	1	0	1	0	DSUB	Subtract TOS from NOS Double Precision and result to NOS. Pop stack.
X	0	1	0	1	0	1	1	DMUL	Multiply NOS by TOS Double Precision and result to NOS. Pop stack.
X	0	1	0	1	1	0	0	DDIV	Divide NOS by TOS Double Precision and result to NOS. Pop Stack.

Notes: X = Don't Care   Operation for bit combinations not listed above is undefined.

Table 2. Am9512 Execution Time in Cycles.

Single Precision

	Min	Typ	Max
Add	58	220	512
Subtract	56	220	512
Multiply	192	220	254
Divide	228	240	264

Double Precision

	Min	Typ	Max
Add	578	1200	3100
Subtract	578	1200	3100
Multiply	1720	1770	1860
Divide	4560	4920	5120

Note: Typical for add and subtract, assumes the operands are within six decimal orders of magnitude. Max is derived from the maximum execution time of 1000 executions with random 32-bit or 64-bit patterns.

Table 3. Some Execution Examples.

Command	TOS	NOS	Result	Clock periods
SADD	3F800000	3F800000	40000000	58
SSUB	3F800000	3F800000	00000000	56
SMUL	40400000	3FC00000	40900000	198
SDIV	40000000	3F800000	3F000000	228
CHSS	3F800000	–	BF800000	10
PTOS	3F800000	–	–	16
POPS	3F800000	–	–	14
XCHS	3F800000	4000000	–	26
CHSD	3FF0000000000000	–	BFF0000000000000	24
PTOD	3FF0000000000000	–	–	40
POPD	3FF0000000000000	–	–	26
CLR	3FF0000000000000	–	–	4
DADD	3FF00000A0000000	8000000000000000	3FF00000A0000000	578
DSUB	3FF00000A0000000	8000000000000000	3FF00000A0000000	578
DMUL	BFF8000000000000	3FF8000000000000	C002000000000000	1748
DDIV	BFF8000000000000	3FF8000000000000	BFF0000000000000	4560

Note: TOS, NOS and Result are in hexadecimal; Clock period is in decimal.

COMMAND INITIATION

After properly positioning the required operands in the stack, a command may be issued. The procedure for initiating a command execution is as follows:

1. Establish appropriate command on the DB0-DB7 lines.
2. Establish HIGH on the C/$\overline{D}$ input.
3. Establish LOW on the $\overline{CS}$ input. Whenever $\overline{WR}$ and $\overline{RD}$ inputs are HIGH the $\overline{PAUSE}$ output follows the $\overline{CS}$ input. Hence $\overline{PAUSE}$ will become LOW.
4. Establish LOW on the $\overline{WR}$ input after an appropriate set up time (see timing diagrams).
5. Sometime after the HIGH to LOW level transition of $\overline{WR}$ input, the $\overline{PAUSE}$ output will become HIGH to acknowledge the write operation. The $\overline{WR}$ input can return to HIGH anytime after $\overline{PAUSE}$ goes HIGH. The DB0-DB7, C/$\overline{D}$ and $\overline{CS}$ inputs are allowed to change after the hold time requirements are satisfied (see timing diagram).

An attempt to issue a new command while the current command execution is in progress is allowed. Under these circumstances, the $\overline{PAUSE}$ output will not go HIGH until the current command execution is completed.

OPERAND ENTRY

The Am9512 commands operate on the operands located at the TOS and NOS and results are returned to the stack at NOS and then popped to TOS. The operands required for the Am9512 are one of two formats – single precision floating-point (4 bytes) or double precision floating-point (8 bytes). The result of an operation has the same format as the operands. In other words, operations using single precision quantities always result in a single precision result while operations involving double precision quantities will result in double precision result.

Operands are always entered into the stack least significant byte first and most significant byte last. The following procedure must be followed to enter operands into the stack:

1. The lower significant operand byte is established on the DB0-DB7 lines.
2. A LOW is established on the C/$\overline{D}$ input to specify that data is to be entered into the stack.
3. The $\overline{CS}$ input is made LOW. Whenever the $\overline{WR}$ and $\overline{RD}$ inputs are HIGH, the $\overline{PAUSE}$ output will follow the $\overline{CS}$ input. Thus $\overline{PAUSE}$ output will become LOW.
4. After appropriate set up time (see timing diagrams), the $\overline{WR}$ input is made LOW.
5. Sometime after this event, $\overline{PAUSE}$ will return HIGH to indicate that the write operation has been acknowledged.
6. Anytime after the $\overline{PAUSE}$ output goes HIGH the $\overline{WR}$ input can be made HIGH. The DB0-DB7, C/$\overline{D}$ and $\overline{CS}$ inputs can change after appropriate hold time requirements are satisfied (see timing diagrams).

The above procedure must be repeated until all bytes of the operand are pushed into the stack. It should be noted that for single precision operands 4 bytes should be pushed and 8 bytes must be pushed for double precision. Not pushing all the bytes of a quantity will result in byte pointer misalignment.

The Am9512 stack can accommodate 4 single precision quantities or 2 double precision quantities. Pushing more quantities than the capacity of the stack will result in loss of data which is usual with any LIFO stack.

REMOVING THE RESULTS

Result from an operation will be available at the TOS. Results can be transferred from the stack to the data bus by reading the stack. When the stack is popped for results, the most significant byte is available first and the least significant byte last. A result is always of the same precision as the operands that produced it. Thus when the result is taken from the stack, the total number of bytes popped out should be appropriate with the precision – single precision results are 4 bytes and double precision results are 8 bytes. The following prodedure must be used for reading the result from the stack:

1. A LOW is established on the C/$\overline{D}$ input.
2. The $\overline{CS}$ input is made LOW. When $\overline{WR}$ and $\overline{RD}$ inputs are both HIGH, the $\overline{PAUSE}$ output follows the $\overline{CS}$ input, thus $\overline{PAUSE}$ will be LOW.
3. After appropriate set up time (see timing diagrams), the $\overline{RD}$ input is made LOW.
4. Sometime after this, $\overline{PAUSE}$ will return HIGH indicating that the data is available on the DB0-DB7 lines. This data will remain on the DB0-DB7 lines as long as the $\overline{RD}$ input remains LOW.
5. Anytime after $\overline{PAUSE}$ goes HIGH, the $\overline{RD}$ input can return HIGH to complete transaction.
6. The $\overline{CS}$ and C/$\overline{D}$ inputs can change after appropriate hold time requirements are satisfied (see timing diagram).
7. Repeat this procedure until all bytes appropriate for the precision of the result are popped out.

Reading of the stack does not alter its data; it only adjusts the byte pointer. If more data is popped than the capacity of the stack, the internal byte pointer will wrap around and older data will be read again, consistent with the LIFO stack.

READING STATUS REGISTER

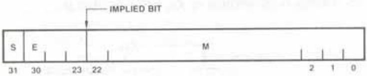
The Am9512 status register can be read without any regard to whether a command is in progress or not. The only implication that has to be considered is the effect this might have on the END and ERR outputs discussed in the signal descriptions.

The following procedure must be followed to accomplish status register reading.

1. Establish HIGH on the C/$\overline{D}$ input.
2. Establish LOW on the $\overline{CS}$ input. Whenever $\overline{WR}$ and $\overline{RD}$ inputs are HIGH, $\overline{PAUSE}$ will follow the $\overline{CS}$ input. Thus, $\overline{PAUSE}$ will go LOW.
3. After appropriate set up time (see timing diagram) $\overline{RD}$ is made LOW.
4. Sometime after the HIGH to LOW transition of $\overline{RD}$, $\overline{PAUSE}$ will become HIGH indicating that status register contents are available on the DB0-DB7 lines. These lines will contain this information as long as $\overline{RD}$ is LOW.
5. The $\overline{RD}$ input can be returned HIGH anytime after $\overline{PAUSE}$ goes HIGH.
6. The C/$\overline{D}$ input and $\overline{CS}$ input can change after satisfying appropriate hold time requirements (see timing diagram).

DATA FORMATS

The Am9512 handles floating-point quantities in two different formats – single precision and double precision. The single precision quantities are 32-bits long as shown below.



Bit 31:

S = Sign of the mantissa. 1 represents negative and 0 represents positive.

Bits 23-30

E = These 8-bits represent a biased exponent. The bias is $2^7 - 1 = 127$

Bits 0-22

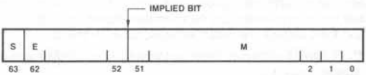
M = 23-bit mantissa. Together with the sign bit, the mantissa represents a signed fraction in sign-magitude notation. There is an implied 1 beyond the most significant bit (bit 22) of the mantissa. In other words, the mantissa is assumed to be a 24-bit normalized quantity and the most significant bit which will always be 1 due to normalization is implied. The Am9512 restores this implied bit internally before performing arithmetic; normalizes the result and strips the implied bit before returning the results to the external data bus. The binary point is between the implied bit and bit 22 of the mantissa.

The quantity N represented by the above notation is

$$N = (-1)^S \; 2^{E-\underbrace{(2^7-1)}_{\text{Bias}}} \; (1\underset{\text{Binary Point}}{.}M)$$

Provided E ≠ 0 or all 1's.

A double precision quantity consists of the mantissa sign bit(s), an 11 bit biased exponent (E), and a 52-bit mantissa (M). The bias for double precision quantities is $2^{10} - 1$. The double precision format is illustrated below.



Bit 63:

S = Sign of the mantissa. 1 represents negative and 0 represents positive.

Bits 52-62

E = These 11 bits represent a biased exponent. The bias is $2^{10} - 1 = 1023$.

Bit 0-51

M = 52-bit mantissa. Together with the sign bit, the mantissa represents a signed fraction in sign-magnitude notation. There is an implied 1 beyond the most significant bit (bit 51) of the mantissa. In other words, the mantissa is assumed to a 53-bit normalized quantity and the most significant bit, which will always be a 1 due to normalization, is implied. The Am9512 restores this implied bit internally before performing arithmetic; normalizes the result and strips the implied bit before returning the result to the external data bus. The binary point is between the implied bit and bit 51 of the mantissa.

The quantity N represented by the above notation is

$$N = (-1)^S \; 2^{E-\underbrace{(2^{10}-1)}_{\text{Bias}}} \; (1\underset{\text{Binary point}}{.}M)$$

Provided E ≠ 0 or all 1's.

STATUS REGISTER

The Am9512 contains an 8-bit status register with the following format.

BUSY	SIGN S	ZERO Z	RESERVED	DIVIDE EXCEPTION D	EXPONENT UNDERFLOW U	EXPONENT OVERFLOW V	RESERVED
7	6	5	4	3	2	1	0

Bit 0 and bit 4 are reserved. Occurrence of exponent oerflow (V), exponent underflow (U) and divide exception (D) are indicated by bits 1, 2 and 3 respectively. An attempt to divide by zero is the only divide exception. Bits 5 and 6 represent a zero result and the sign of a result respectively. Bit 7 (Busy) of the status register indicates if the Am9512 is currently busy executing a command. All the bits are initialized to zero upon reset. Also, executing a CLR (Clear Status) command will result in all zero status register bits. A zero in Bit 7 indicates that the Am9512 is not busy and a new command may be initiated. As soon as a new command is issued, Bit 7 becomes 1 to indicate the device is busy and remains 1 until the command execution is complete, at which time it will become 0. As soon as a new command is issued, status register bits 0, 1, 2, 3, 4, 5 and 6 are cleared to zero. The status bits will be set as required during the command execution. Hence, as long as bit 7 is 1, the remainder of the status register bit indications should not be relied upon unless the ERR occurs. The following is a detailed status bit description.

Bit 0 Reserved

Bit 1 Exponent overflow (V): When 1, this bit indicates that exponent overflow has occurred. Cleared to zero otherwise.

Bit 2 Exponent Underflow (U): When 1, this bit indicates that exponent underflow has occurred. Cleared to zero otherwise.

Bit 3 Divide Exception (D): When 1, this bit indicates that an attempt to divide by zero is made. Cleared to zero otherwise.

Bit 4 Reserved

Bit 5 Zero (Z): When 1, this bit indicates that the result returned to TOS after a command is all zeros. Cleared to zero otherwise.

Bit 6 Sign (S): When 1, this bit indicates that the result returned to TOS is negative. Cleared to zero otherwise.

Bit 7 Busy: When 1, this bit indicates the Am9512 is in the process of executing a command. It will become zero after the command execution is complete.

All other status register bits are valid when the Busy bit is zero.

ALGORITHMS OF FLOATING-POINT ARITHMETIC

1. Floating Point to Decimal Conversion

   As an introduction to floating-point arithmetic, a brief description of the Decimal equivalent of the Am9512 floating-point format should help the reader to understand and verify the validity of the arithmetic operations. The Am9512 single precision format is used for the following discussions. With a minor modification of the field lengths, the discussion would also apply to the double precision format.

   There are three parts in a floating point number:

   a. The sign – the sign applies to the sign of the number. Zero means the number is positive or zero. One means the number is negative.

b. The exponent – the exponent represents the magnitude of the number. The Am9512 single precision format has an excess 127_{10} notation which means the code representation is 127_{10} higher than the actual value. The following are a few examples of actual versus coded exponent.

Actual	Coded
$+127_{10}$	$+254_{10}$
0	127_{10}
-126_{10}	$+1_{10}$

c. The mantissa – the mantissa is a 23-bit value with the binary point to the left of the most significant bit. There is a hidden 1 to the left of the binary point so the mantissa is always less than 2 and greater than or equal to 1.

To find the Decimal equivalent of the floating point number, the mantissa is multiplied by 2 to the power of the actual exponent. The number is negated if the sign bit = 1. The following are two examples of conversion:

Example 1

Floating Point No. = 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 B

(Sign = 0; Exponent = 1 0 0 0 0 0 1 1; Mantissa = 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)

Coded Exponent = 1 0 0 0 0 0 1 1 B
Actual Exponent = 1 0 0 0 0 0 1 1 B – 0 1 1 1 1 1 1 1 B = 0 0 0 0 0 1 0 0 B = 4_{10}
Mantissa = 1.1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 B
= 1 + 1/2 + 1/4 = 1.75_{10}
Decimal No. = 2^4 x 1.75 = 16 x 1.75 = 28_{10}

Example 2

Floating Point No. = 1 0 1 1 1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 B

(Sign = 1; Exponent = 0 1 1 1 1 0 1 0; Mantissa = 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)

Code Exponent = 0 1 1 1 1 0 1 0 B
Actual Exponent = 0 1 1 1 1 0 1 0 B – 0 1 1 1 1 1 1 1 B = 1 1 1 1 1 0 1 1 B = -5_{10}
Mantissa = 1.0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 B
= 1 + 1/4 + 1/8 = 1.375_{10}
Decimal No. = -2^{-5} x 1.375 = $-.04296875_{10}$

2. Unpacking of the Floating-Point Numbers
The Am9512 unpacks the floating point number into three parts before any of the arithmetic operation. The number is divided into three parts as described in Section 1. The sign and exponent are copied from the original number as 1 and 8-bit numbers respectively. The mantissa is stored as a 24-bit number. The least significant 23 bits are copied from the original number and the MSB is set to 1. The binary point is assumed to the right of the MSB.

The abbreviations listed below are used in the following sections of algorithm description:

SIGN – Sign of Result
EXP – Exponent of Result
MAN – Mantissa of Result
SIGN (TOS) – Sign of Top of Stack
EXP (TOS) – Exponent of Top of Stack
MAN (TOS) – Mantissa of Top of Stack
SIGN (NOS) – Sign of Next on Stack
EXP (NOS) – Exponent of Next on Stack
MAN (NOS) – Mantissa of Next on Stack

3. Floating-Point Add/Subtract
The floating-point add and subtract essentially use the same algorithm. The only difference is that floating-point subtract changes the sign of the floating-point number at top of stack and then performs the floating-point add.

The following is a step by step description of a floating-point add algorithm (Figure 1):

a. Unpack TOS and NOS.
b. The exponent of TOS is compared to the exponent of NOS.
c. If the exponents are equal, go to step f.
d. Right shift the mantissa of the number with the smaller exponent.
e. Increment the smaller exponent and go to step b.
f. Set sign of result to sign of larger number.
g. Set exponent of result to exponent of larger number.
h. If sign of the two numbers are not equal, go to m.
i. Add Mantissas.
j. Right shift resultant mantissa by 1 and increment exponent of result by 1.
k. If MSB of exponent changes from 1 to 0 as a result of the increment, set overflow status.
l. Round if necessary and exit.
m. Subtract smaller mantissa from larger mantissa.
n. Left shift mantissa and decrement exponent of result.
o. If MSB of exponent changes from 0 to 1 as a result of the decrement, set underflow status and exit.
p. If the MSB of the resultant mantissa = 0, go to n.
q. Round if necessary and exit.

4. Floating-Point Multiply
Floating-point multiply basically involves the addition of the exponents and multiplication of the mantissas. The following is a step by step description of a floating multiplication algorithm (Figure 2):

a. Check if TOS or NOS = 0.
b. If either TOS or NOS = 0, Set result to 0 and exit.
c. Unpack TOS and NOS.
d. Convert EXP (TOS) and EXP (NOS) to unbiased form.
EXP (TOS) = EXP (TOS) -127_{10}
EXP (NOS) = EXP (NOS) -127_{10}
e. Add exponents.
EXP = EXP (TOS) + EXP (NOS)
f. If MSB of EXP (TOS) = MSB of EXP (NOS) = 0 and MSB of EXP = 1, then set overflow status and exit.
g. If MSB of EXP (TOS) = MSB of EXP (NOS) = 1 and MSB of EXP = 0, then set underflow status and exit.
h. Convert Exponent back to biased form.
EXP = EXP + 127_{10}
i. If sign of TOS = sign of NOS, set sign of result to 0, else set sign of result to 1.
j. Multiply mantissa.
k. If MSB of resultant = 1, right shift mantissa by 1 and increment exponent of resultant.
l. If MSB of exponent changes from 1 to 0 as a result of the increment, set overflow status.
m. Round if necessary and exit.

5. Floating-Point Divide
The floating-point divide basically involves the subtraction of exponents and the division of mantissas. The following is a step by step description of a division algorithm (Figure 3).

a. If TOS = 0, set divide exception error and exit.
b. If NOS = 0, set result to 0 and exit.
c. Unpack TOS and NOS.
d. Convert EXP (TOS) and EXP (NOS) to unbiased form.
EXP (TOS) = EXP (TOS) – 127_{10}
EXP (NOS) = EXP (NOS) – 127_{10}
e. Subtract exponent of TOS from exponent of NOS.
EXP = EXP (NOS) – EXP (TOS)
f. If MSB of EXP (NOS) = 0, MSB of EXP (TOS) = 1 and MSB of EXP = 1, then set overflow status and exit.
g. If MSB of EXP (NOS) = 1, MSB of EXP (TOS) = 0, and MSB of EXP = 0, then set underflow status and exit.

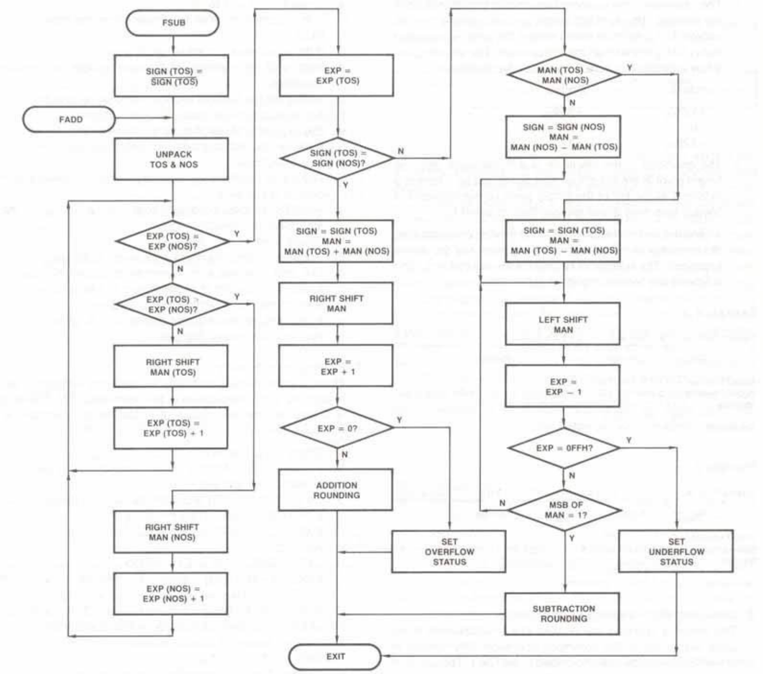


Figure 1. Conceptual Floating-Point Addition/Subtraction.

MOS-205

h. Add bias to exponent of result.
   $EXP = EXP + 127_{10}$

i. If sign of TOS = sign of NOS, set sign of result to 0, else set sign of result to 1.

j. Divide mantissa of NOS by mantissa of TOS.

k. If MSB = 0, left shift mantissa and decrement exponent of resultant, else go to n.

l. If MSB of exponent changes from 0 to 1 as a result of the decrement, set underflow status.

m. Go to k.

n. Round if necessary and exit.

The algorithms described above provide the user a means of verifying the validity of the result. They do not necessarily reflect the exact internal sequence of the Am9512.

6. Rounding

The Am9512 adopts a rounding algorithm that is consistent with the Intel® standard for floating-point arithmetic. The following description is an excerpt from the paper published in proceedings of Compsac 77, November 1977, pp. 107-112 by Dr. John F. Palmer of Intel Corporation.

The method used for doing the rounding during floating-point arithmetic is known as "Round to Even", i.e., if the resultant number is exactly halfway between two floating point numbers, the number is rounded to the nearest floating-point number whose LSB of the mantissa is 0. In order to simplify the explanation, the algorithms will be illustrated with 4-bit arithmetic. The existence of an accumulator will be assumed as shown:

OF	B1	B2	B3	B4	G	R	ST

The bit labels denote:

OF – The overflow bit
B1-B4 – The 4 mantissa bits
G – The Guard bit
R – The Rounding bit
ST – The "Sticky" bit

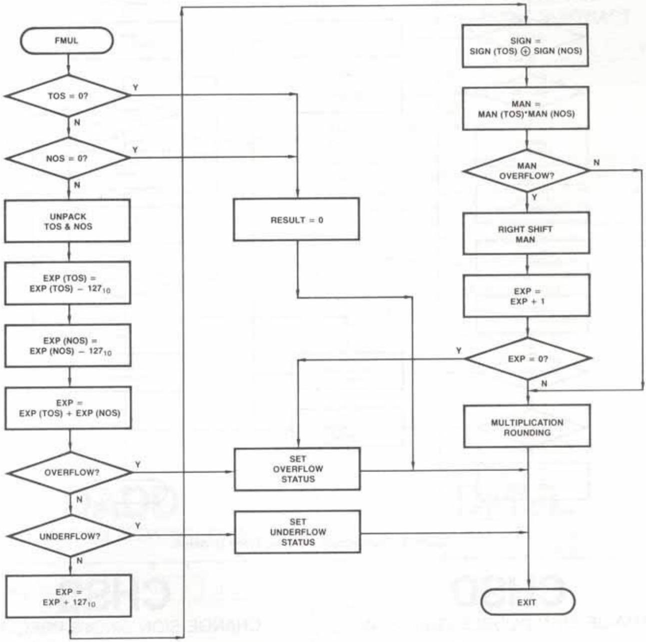


Figure 2. Conceptual Floating-Point Multiplication.

MOS-206

The Sticky bit is set to one if any ones are shifted right of the rounding bit in the process of denormalization. If the Sticky bit becomes set, it remains set throughout the operation. All shifting in the Accumulator involves the OF, G, R and ST bits. The ST bit is not affected by left shifts but, zeros are introduced into OF by right shifts.

Rounding during addition of magnitudes – add 1 to the G position, then if G=R=ST=0, set B4 to 0 ("Rounding to Even").

Rounding during subtraction of magnitudes – if more than one left shift was performed, no rounding is needed, otherwise round the same way as addition of magnitudes.

Rounding during multiplication – let the normalized double length product be:

B1	B2	B3	B4	B5	B6	B7	B8

Then G=B5, R=B6, ST=B7 V B8. The rounding is then performed as in addition of magnitudes.

Rounding during division – let the first six bits of the normalized quotient be

B1	B2	B3	B4	B5	B6

Then G=B5, R=B6, ST=0 if and only if remainder = 0. The rounding is then performed as in addition of magnitudes.

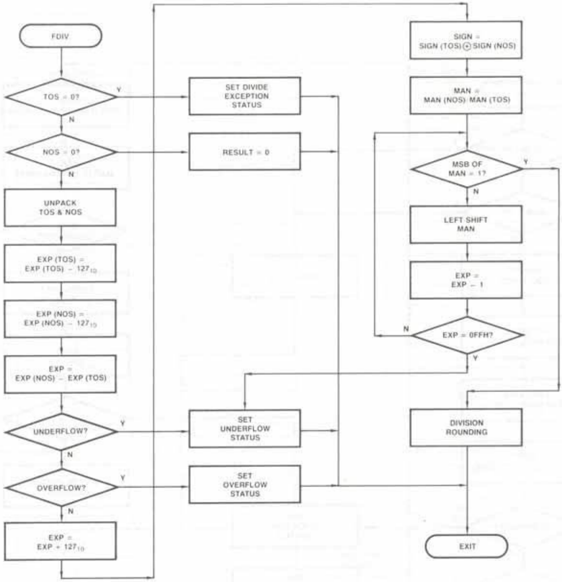


Figure 3. Conceptual Floating-Point Division.

MOS-207

CHSD

CHANGE SIGN DOUBLE PRECISION

	7	6	5	4	3	2	1	0
Binary Coding:	SRE	0	1	0	1	1	0	1

Hex Coding: AD IF SRE = 1
2D IF SRE = 0

Execution Time: See Table 2

Description:

The sign of the double precision TOS operand A is complemented. The double precision result R is returned to TOS. If the double precision operand A is zero, then the sign is not affected. The status bit S and Z indicate the sign of the result and if the result is zero. The status bits U, V and D are always cleared to zero.

Status Affected: S, Z. (U, V, D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	TOS	R
B	NOS	B

CHSS

CHANGE SIGN SINGLE PRECISION

	7	6	5	4	3	2	1	0
Binary Coding:	SRE	0	0	0	0	1	0	1

Hex Coding: 85 IF SRE = 1
05 IF SRE = 0

Execution Time: See Table 2

Description:

The sign of the single precision operand A at TOS is complemented. The single precision result R is returned to TOS. If the exponent field of A is zero, all bits of R will be zeros. The status bits S and Z indicate the sign of the result and if the result is zero. The status bits U, V and D are cleared to zero.

Status Affected: S, Z. (U, V, D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B	← NOS →	B
C		C
D		D

CLR

CLEAR STATUS

Binary Coding:

7	6	5	4	3	2	1	0
SRE	0	0	0	0	0	0	0

Hex Coding: 80 IF SRE = 1
00 IF SRE = 0

Execution Time: 4 clock cycles

Description:

The status bits S, Z, D, U, V are cleared to zero. The stack is not affected. This essentially is a no operation command as far as operands are concerned.

Status Affected: S, Z, D, U, V always zero.

DADD

DOUBLE PRECISION FLOATING-POINT ADD

Binary Coding:

7	6	5	4	3	2	1	0
SRE	0	1	0	1	0	0	1

Hex Coding: A9 IF SRE = 1
29 IF SRE = 0

Execution Time: See Table 2

Description:

The double precision operand A from TOS is added to the double precision operand B from NOS. The result is rounded to obtain the final double precision result R which is returned to TOS. The status bits S, Z, U and V are affected to report sign of the result, if the result is zero, exponent underflow and exponent overflow respectively. The status bit D will be cleared to zero.

Status Affected: S, Z, U, V. (D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B	← NOS →	Undefined

DSUB

DOUBLE PRECISION FLOATING-POINT SUBTRACT

Binary Coding:

7	6	5	4	3	2	1	0
SRE	0	1	0	1	0	1	0

Hex Coding: AA IF SRE = 1
2A IF SRE = 0

Execution Time: See Table 2

Description:

The double precision operand A at TOS is subtracted from the double precision operand B at NOS. The result is rounded to obtain the final double precision result R which is returned to TOS. The status bits S, Z, U and V are affected to report sign of the result, if the result is zero, exponent underflow and exponent overflow respectively. The status bit D will be cleared to zero.

Status Affected: S, Z, U, V. (D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B	← NOS →	Undefined

DMUL

DOUBLE PRECISION FLOATING-POINT MULTIPLY

Binary Coding:

7	6	5	4	3	2	1	0
SRE	0	1	0	1	0	1	1

Hex Coding: AB IF SRE = 1
2B IF SRE = 0

Execution Time: See Table 2

Description:

The double precision operand A from TOS is multiplied by the double precision operand B from NOS. The result is rounded to obtain the final double precision result R which is returned to TOS. The status bits S, Z, U and V are affected to report sign of the result, if the result is zero, exponent underflow and exponent overflow respectively. The status bit D will be cleared to zero.

Status Affected: S, Z, U, V. (D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B	← NOS →	Undefined

DDIV

DOUBLE PRECISION FLOATING-POINT DIVIDE

	7	6	5	4	3	2	1	0
Binary Code:	SRE	0	1	0	1	1	0	0

Hex Coding: AC IF SRE = 1
2C IF SRE = 0

Execution Time: See Table 2

Description:
The double precision operand B from NOS is divided by the double precision operand A from TOS. The result (quotient) is rounded to obtain the final double precision result R which is returned to TOS. The status bits, S, Z, D, U and V are affected to report sign of the result, if the result is zero, attempt to divide by zero, exponent underflow and exponent overflow respectively.

Status Affected: S, Z, D, U, V

STACK CONTENT

BEFORE		AFTER
A	← TOS →	R (see note)
B	← NOS →	Undefined

Note: If A is zero, then R = B (Divide exception).

SADD

SINGLE PRECISION FLOATING-POINT ADD

	7	6	5	4	3	2	1	0
Binary Coding:	SRE	0	0	0	0	0	0	1

Hex Coding: 81 IF SRE = 1
01 IF SRE = 0

Execution Time: See Table 2

Description:
The single precision operand A from TOS is added to the single precision operand B from NOS. The result is rounded to obtain the final single precision result R which is returned to TOS. The status bits S, Z, U and V are affected to report the sign of the result, if the result is zero, exponent underflow and exponent overflow respectively. The status bit D will be cleared to zero.

Status Affected: S, Z, U, V. (D always zero.)

STACK CONTENT

BEFORE		AFTER
A	← TOS →	R
B	← NOS →	C
C		D
D		Undefined

SSUB

SINGLE PRECISION FLOATING-POINT SUBTRACT

	7	6	5	4	3	2	1	0
Binary Coding:	SRE	0	0	0	0	0	1	0

Hex Coding: 82 IF SRE = 1
02 IF SRE = 0

Execution Time: See Table 2

Description:
The single precision operand A at TOS is subtracted from the single precision operand B at NOS. The result is rounded to obtain the final single precision result R which is returned to TOS. The status bits S, Z, U and V are affected to report the sign of the result, if the result is zero, exponent underflow and exponent overflow respectively. The status bit D will be cleared to zero.

Status Affected: S, Z, U, V. (D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B	← NOS →	C
C		D
D		Undefined

SMUL

SINGLE PRECISION FLOATING-POINT MULTIPLY

	7	6	5	4	3	2	1	0
Binary Coding:	SRE	0	0	0	0	0	1	1

Hex Coding: 83 IF SRE = 1
03 IF SRE = 0

Execution Time: See Table 2

Description:
The single precision operand A from TOS is multiplied by the single precision operand B from NOS. The result is rounded to obtain the final single precision result R which is returned to TOS. The status bits S, Z, U and V are affected to report the sign of the result, if the result is zero, exponent underflow and exponent overflow respectively. The status bit D will be cleared to zero.

Status Affected: S, Z, U, V. (D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R
B	← NOS →	C
C		D
D		Undefined

SDIV

SINGLE PRECISION FLOATING-POINT DIVIDE

Binary Coding:

7	6	5	4	3	2	1	0
SRE	0	0	0	0	1	0	0

Hex Coding: 84 IF SRE = 1
04 IF SRE = 0

Execution Time: See Table 2

Description:
The single precision operand B from NOS is divided by the single precision operand A from TOS. The result (quotient) is rounded to obtain the final result R which is returned to TOS. The status bits S, Z, D, U and V are affected to report the sign of the result, if the result is zero, attempt to divide by zero, exponent underflow and exponent overflow respectively.

Status Affected: S, Z, D, U, V

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	R (see note)
B	← NOS →	C
C		D
D		Undefined

Note: If exponent field of A is zero then R = B (Divide exception).

POPS

POP STACK SINGLE PRECISION

Binary Coding:

7	6	5	4	3	2	1	0
SRE	0	0	0	0	1	1	1

Hex Coding: 87 IF SRE = 1
07 IF SRE = 0

Execution Time: See Table 2

Description:
The single precision operand A is popped from the stack. The internal stack control mechanism is such that A will be written at the bottom of the stack. The status bits S and Z are affected to report the sign of the new operand at TOS and if it is zero, respectively. The status bits U, V and D will be cleared to zero. Note that only the exponent field of the new TOS is checked for zero, if it is zero status bit Z will set to 1.

Status Affected: S, Z. (U, V, D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	B
B	← NOS →	C
C		D
D		A

PTOD

PUSH STACK DOUBLE PRECISION

Binary Coding:

7	6	5	4	3	2	1	0
SRE	0	1	0	1	1	1	0

Hex Coding: AE IF SRE = 1
2E IF SRE = 0

Execution Time: See Table 2

Description:
The double precision operand A from the TOS is pushed back on to the stack. This is effectively a duplication of A into two consecutive stack locations. The status S and Z are affected to report sign of the new TOS and if the new TOS is zero respectively. The status bits U, V and D will be cleared to zero.

Status Affected: S, Z. (U, V, D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	A
B	← NOS →	A

PTOS

PUSH STACK SINGLE PRECISION

Binary Coding:

7	6	5	4	3	2	1	0
SRE	0	0	0	0	1	1	0

Hex Coding: 86 IF SRE = 1
06 IF SRE = 0

Execution Time: See Table 2

Description:
This instruction effectively pushes the single precision operand from TOS on to the stack. This amounts to duplicating the operand at two locations in the stack. However, if the operand at TOS prior to the PTOS command has only its exponent field as zero, the new content of the TOS will all be zeroes. The contents of NOS will be an exact copy of the old TOS. The status bits S and Z are affected to report the sign of the new TOS and if the content of TOS is zero, respectively. The status bits U, V and D will be cleared to zero.

Status Affected: S, Z. (U, V, D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	← TOS →	A* See note
B	← NOS →	A
C		B
D		C

Note: A* = A if Exponent field of A is not zero.
A* = 0 if Exponent field of A is zero.

POPD

POP STACK DOUBLE PRECISION

	7	6	5	4	3	2	1	0
Binary Coding:	SRE	0	1	0	1	1	1	1

Hex Coding: AF IF SRE = 1
2F IF SRE = 0

Execution Time: See Table 2

Description:

The double precision operand A is popped from the stack. The internal stack control mechanism is such that A will be written at the bottom of the stack. This operation has the same effect as exchanging TOS and NOS. The status bits S and Z are affected to report the sign of the new operand at TOS and if it is zero, respectively. The status bits U, V and D will be cleared to zero.

Status Affected: S, Z (U, V and D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	←— TOS —→	B
B	←— NOS —→	A

XCHS

EXCHANGE TOS AND NOS SINGLE-PRECISION

	7	6	5	4	3	2	1	0
Binary Coding:	SRE	0	0	0	1	0	0	0

Hex Coding: 88 IF SRE = 1
08 IF SRE = 0

Execution Time: See Table 2

Description:

The single precision operand A at the TOS and the single precision operand B at the NOS are exchanged. After execution, B is at the TOS and A is at the NOS. All other operands are unchanged.

Status Affected: S, Z (U, V and D always zero.)

STACK CONTENTS

BEFORE		AFTER
A	←— TOS —→	B
B	←— NOS —→	A
C		C
D		D

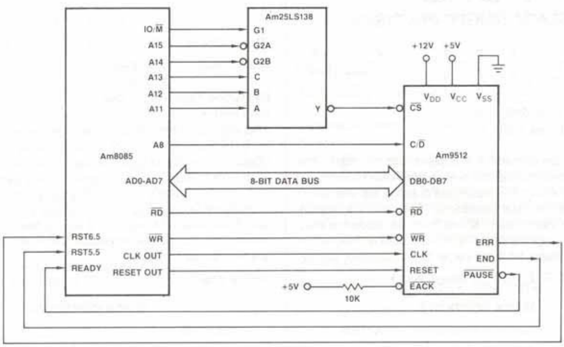


Figure 4. Am9512 to Am8085 Interface.

MOS-213

MAXIMUM RATINGS beyond which useful life may be impaired

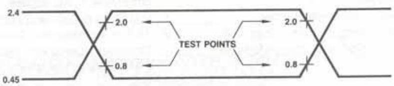
Storage Temperature	−65 to +150°C
V_{DD} with Respect to V_{SS}	−0.5 to +15.0V
V_{CC} with Respect to V_{SS}	−0.5 to +7.0V
All Signal Voltages with Respect to V_{SS}	−0.5 to +7.0V
Power Dissipation (Package Limitation)	2.0W

The products described by this specification include internal circuitry designed to protect input devices from damaging accumulations of static charge. It is suggested, nevertheless, that conventional precautions be observed during storage, handling and use in order to avoid exposure to excessive voltages.

ELECTRICAL CHARACTERISTICS Over Operating Range (Note 1)

Parameters	Description	Test Conditions	Min.	Typ.	Max.	Units
VOH	Output HIGH Voltage	IOH = −200μA	3.7			Volts
VOL	Output LOW Voltage	IOL = 3.2mA			0.4	Volts
VIH	Input HIGH Voltage		2.0		VCC	Volts
VIL	Input LOW Voltage		−0.5		0.8	Volts
IIX	Input Load Current	VSS ≤ VI ≤ VCC			±10	μA
IOZ	Data Bus Leakage	VO = 0.4V			10	μA
		VO = VCC			10	
ICC	VCC Supply Current	T_A = +25°C		50	90	mA
		T_A = 0°C			95	
		T_A = −55°C			100	
IDD	VDD Supply Current	T_A = +25°C		50	90	mA
		T_A = 0°C			95	
		T_A = −55°C			100	
CO	Output Capacitance	fc = 1.0MHz, Inputs = 0V		8	10	pF
CI	Input Capacitance			5	8	pF
CIO	I/O Capacitance			10	12	pF

INPUT AND OUTPUT WAVEFORMS FOR AC TESTS



SWITCHING CHARACTERISTICS

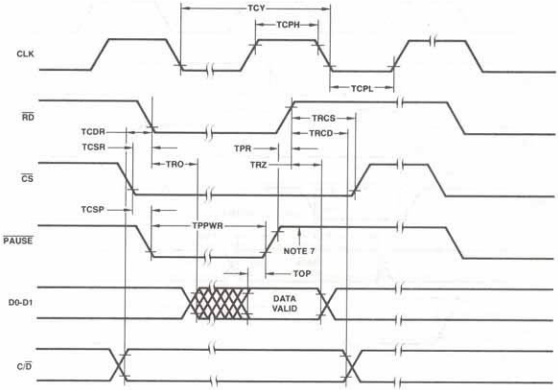
Parameters	Description		Am9512DC Min	Am9512DC Max	Am9512-1DC Min	Am9512-1DC Max	Units
TAPW	$\overline{EACK}$ LOW Pulse Width		100		75		ns
TCDR	C/$\overline{D}$ to $\overline{RD}$ LOW Set-up Time		0		0		ns
TCDW	C/$\overline{D}$ to $\overline{WR}$ LOW Set-up Time		0		0		ns
TCPH	Clock Pulse HIGH Width		200	500	140	500	ns
TCPL	Clock Pulse LOW Width		240		160		ns
TCSP	$\overline{CS}$ LOW to $\overline{PAUSE}$ LOW Delay (Note 5)			150		100	ns
TCSR	$\overline{CS}$ to $\overline{RD}$ LOW Set-up Time		0		0		ns
TCSW	$\overline{CS}$ LOW to $\overline{WR}$ LOW Set-up Time		0		0		ns
TCY	Clock Period		480	5000	320	2000	ns
TDW	Data Valid to $\overline{WR}$ HIGH Delay		150		100		ns
TEAE	$\overline{EACK}$ LOW to END LOW Delay			200		175	ns
TEHPHR	END HIGH to $\overline{PAUSE}$ HIGH Data Read when Busy			5.5TCY+300		5.5TCY+200	ns
TEHPHW	END HIGH to $\overline{PAUSE}$ HIGH Write when Busy			200		175	ns
TEPW	END HIGH Pulse Width		400		300		ns
TEX	Execution Time		See Table 2				ns
TOP	Data Bus Output Valid to $\overline{PAUSE}$ HIGH Delay		0		0		ns
TPPWR	$\overline{PAUSE}$ LOW Pulse Width Read	Data	3.5TCY+50	5.5TCY+300	3.5TCY+50	5.5TCY+200	ns
		Status	1.5TCY+50	3.5TCY+300	1.5TCY+50	3.5TCY+200	
TPPWRB	END HIGH to $\overline{PAUSE}$ HIGH Read when Busy	Data	See Table 2				ns
		Status	1.5TCY+50	3.5TCY+300	1.5TCY+50	3.5TCY+200	
TPPWW	$\overline{PAUSE}$ LOW Pulse Width Write when Not Busy			TCSW+50		TCSW+50	ns
TPPWWB	$\overline{PAUSE}$ LOW Pulse Width Write when Busy		See Table 2				ns
TPR	$\overline{PAUSE}$ HIGH to Read HIGH Hold Time		0		0		ns
TPW	$\overline{PAUSE}$ HIGH to Write HIGH Hold Time		0		0		ns
TRCD	$\overline{RD}$ HIGH to C/$\overline{D}$ Hold Time		0		0		ns
TRCS	$\overline{RD}$ HIGH to $\overline{CS}$ HIGH Hold Time		0		0		ns
TRO	$\overline{RD}$ LOW to Data Bus On Delay		50		50		ns
TRZ	$\overline{RD}$ HIGH to Data Bus Off Delay		50	200	50	150	ns
TSAPW	SVACK LOW Pulse Width		100		75		ns
TSAR	SVACK LOW to SVREQ LOW Delay			300		200	ns
TWCD	$\overline{WR}$ HIGH to C/D Hold Time		60		30		ns
TWCS	$\overline{WR}$ HIGH to CS HIGH Hold Time		60		30		ns
TWD	$\overline{WR}$ HIGH to Data Bus Hold Time		20		20		ns

NOTES:

1. Typical values are for T_A = 25°C, nominal supply voltages and nominal processing parameters.
2. Switching parameters are listed in alphabetical order.
3. Test conditions assume transition times of 20ns or less, output loading of one TTL gate plus 100pF and timing reference levels of 0.8V and 2.0V.
4. END HIGH pulse width is specified for $\overline{EACK}$ tied to VSS. Otherwise TEAE applies.
5. $\overline{PAUSE}$ is pulled low for both command and data operations.
6. TEX is the execution time of the current command (see the Command Execution Times table).
7. $\overline{PAUSE}$ will go low at this point if $\overline{CS}$ is low and $\overline{RD}$ and $\overline{WR}$ are high.

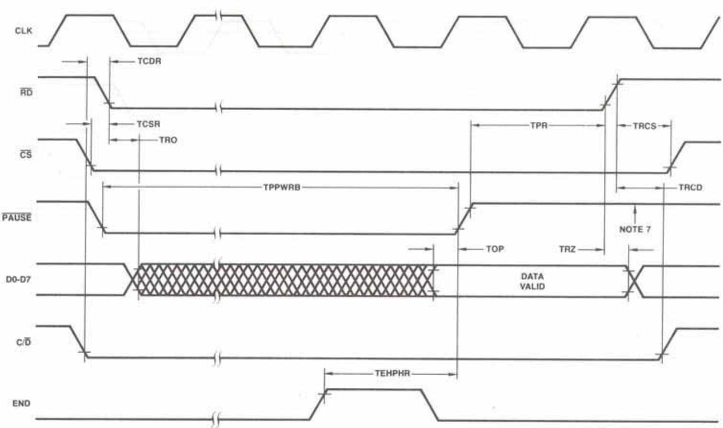
TIMING DIAGRAMS

READ OPERATION



MOS-208

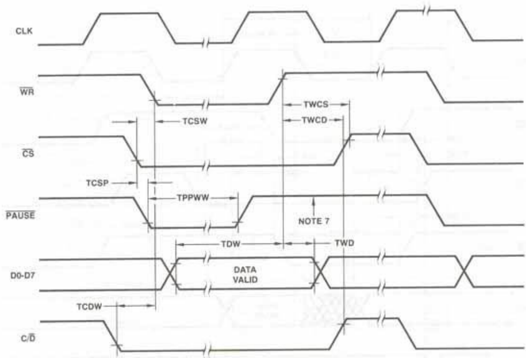
OPERAND READ WHEN Am9512 IS BUSY



MOS-209

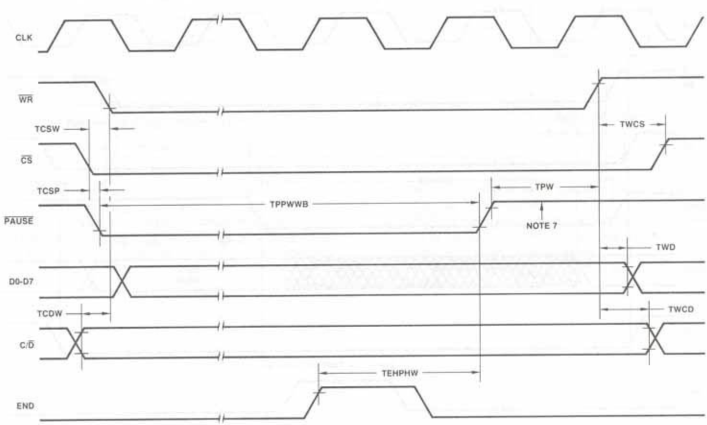
TIMING DIAGRAMS (Cont.)

OPERAND ENTRY



MOS-210

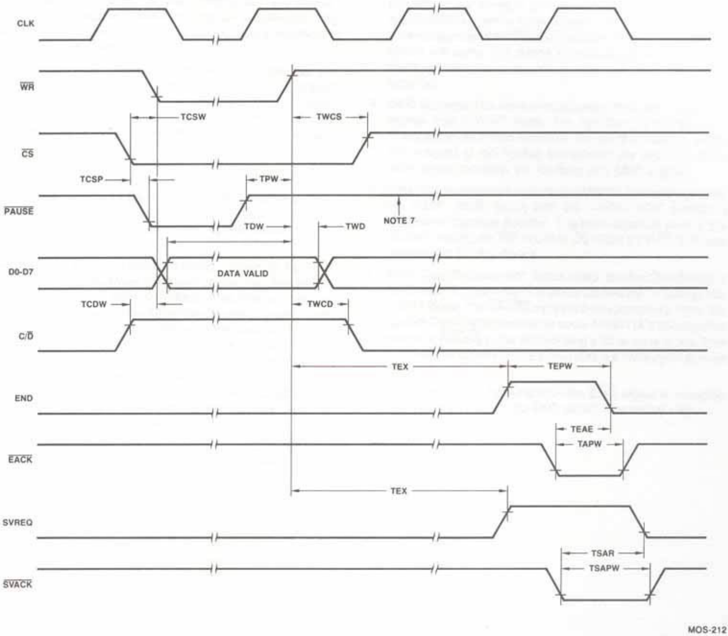
COMMAND OR DATA WRITE WHEN Am9512 IS BUSY



MOS-211

TIMING DIAGRAMS (Cont.)

COMMAND INITIATION



APPENDIX C DESIGNER'S CHECKLIST

The interfaces to the Am9511A and Am9512 are very similar. Most the common problems are shared by the two devices. The following problems usually occur due to the user's insufficient understanding of the data sheet:

1. 8085A Systems. The most common error is that the user ties the $\overline{\text{PAUSE}}$ output directly to the READY input. The 8085A READY sampling time is such that when the $\overline{\text{PAUSE}}$ comes out of the Am9511A, the 8085A has already passed the decision point for going into the WAIT State. The observed symptom is that the user will read the contents of the Command Register after a command has been issued. The solution is to use a flip-flop to advance the effective $\overline{\text{PAUSE}}$ by one clock period whenever there is an Am9511A access. This problem does not apply to the Am9512 since the Am9512 was designed for the 8085A. (Am9511A was designed for the 8080A).
2. Z80 Systems. The most common timing violation is the Chip Select to Read time. Many Z80 users derive the Chip Select for the Am9511A/12 from an address decoder strobed by IORQ. The Read input of the Am9511A/12 comes directly from RD. Since $\overline{\text{IORQ}}$ and $\overline{\text{RD}}$ are initiated simultaneously from the Z80, the Chip Select will reach the Am9511A/12 later than the $\overline{\text{RD}}$ signal by the propagation delay time of the address decoder. This violates the Chip Select to Read set-up time of 0ns. The solution to this problem is that $\overline{\text{IORQ}}$ should strobe the $\overline{\text{RD}}$ signal instead of the chip select decoder. For the Am9512, the $\overline{\text{PAUSE}}$ should also be gated with $\overline{\text{IORQ}}$.
3. 8080A Systems. The 8080A interface to the Am9512 requires the $\overline{\text{PAUSE}}$ output be gated with $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ because the $\overline{\text{PAUSE}}$ on the Am9512 follows the Chip Select inputs. In a typical 8080A system, Chip Select is derived from a straight address decode. If the $\overline{\text{PAUSE}}$ output is not gated, the $\overline{\text{PAUSE}}$ output will come out during a memory access whose address corresponds to the I/O address and the system will be hung up.
4. 6800 Systems. The most common error is the failure to put the system into a WAIT State. The symptom is the same as described in the 8085A systems. The solution requires a one-shot instead of the flip-flop because of the lack of suitable clock edges to trigger the flip-flops in a 6800 system.
5. PIAs. Some users try to interface a M6800-type microprocessor (6502, 6800, 6809) with the Am9511A/12 through a Peripheral Interface Adapter. The most common error is the failure to satisfy the $\overline{\text{WR}}$ HIGH to $\overline{\text{CS}}$ HIGH (or C/$\overline{\text{D}}$) hold time requirement for the device.
6. WAIT State Requirement. Some users take the Chip Select of the Am9511A to set a flip-flop which causes the CPU to go into a WAIT State. The $\overline{\text{PAUSE}}$ signal is then supposed to reset the flip-flop. This method will fail for some Am9511A's because the minimum $\overline{\text{PAUSE}}$ pulse width during a Write cycle is 0ns. That is, there may be no $\overline{\text{PAUSE}}$ signal during a Write cycle to reset the flip-flop.
7. END Signal. For the Am9511A the $\overline{\text{END}}$ signal is asserted Low. For the Am9512 the END signal is asserted High.

ADVANCED MICRO DEVICES, INC.
901 Thompson Place
P.O. Box 3453
Sunnyvale, California 94088
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL FREE
(800) 538-8450

Order # 00615C

CD-MU-5M-8-86-0