pdp11

KE11-E and KE11-F instruction set options manual

digital

EK-KE11E-TM-002

KE11-E and KE11-F instruction set options manual

digital equipment corporation · maynard, massachusetts

First Edition, January 1973
2nd Printing (Rev), June 1973
3rd Printing, December 1974
4th Printing, January 1975

Copyright © 1973, 1974, 1975 by Digital Equipment Corporation

The material in this manual is for informational purposes and is subject to change without notice.

Printed in U.S.A.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DEC	PDP
FLIP CHIP	FOCAL
DIGITAL	COMPUTER LAB
UNIBUS	

INTRODUCTION

This manual describes the KE11-E Extended Instruction Set (EIS) and KE11-F Floating Instruction Set (FIS) Options to the KD11-A Programmed Data Processor for the PDP-11/40 System. These two options are described in one manual because of their interdependency, in that KE11-F cannot be installed without the KE11-E being first installed. The purpose of this manual is to:

1. Provide an overall understanding of the functions of these options in a PDP-11/40 System.

2. Explain how the KE11-E and KE11-F can be used in software operating systems.

3. Describe the options in sufficient detail to enable maintenance personnel to perform on-site troubleshooting and repair.

In this manual each chapter is split in two with the first half of the chapter presenting information concerning the KE11-E Option and the second half being devoted to comparable information for the KE11-F Option. This organization is intended to facilitate greater ease in use by those customers who utilize only the EIS hardware. Note that due to the dependency of FIS hardware on the inclusion of EIS hardware, this split is not used in Chapter 4.

Chapter 1 provides an introduction to the options and lists brief specifications. Chapter 2 contains programming information, listing instructions and illustrating their formats. Chapter 3 gives a discussion of the theoretical principles implemented by these options. Chapter 4 comprises a block diagram discussion, a flow diagram discussion, and detailed descriptions of the logic functions. Content and organization of this chapter are based on the block schematics contained in a separate Engineering Drawings volume. Chapter 5 references the installation and maintenance procedures provided in the *PDP-11/40 System Maintenance Manual.* Specific procedures are given for modifications necessary to the processor, and for use of the Maintenance Module Overlay for these options.

Detailed descriptions of processor, console, Unibus, and memory logic that interface with these options are provided in the following related documents:

PDP-11/40 System Maintenance Manual	DEC-11-H40SA-A-D
KD11-A Central Processor Unit Maintenance Manual	EK-KD11A-MM-001

CONTENTS

CONTENTS (Cont)

ILLUSTRATIONS

TABLES

TABLES (Cont)

CHAPTER 1
GENERAL DESCRIPTION

This chapter contains a general description of both the KE11-E and KE11-F Options. Mechanical descriptions are given together with engineering specifications for each option. The chapter is divided in half with the EIS information presented first, followed by comparable information for the FIS hardware.

1.1 KE11-E EXTENDED INSTRUCTION SET

The KE11-E Extended Instruction Set is a hardware option to the basic PDP-11/40 Computer System. It is supplied as a pluggable option to the KD11-A Central Processor.

1.1.1 Purpose

The KE11-E Option expands the instruction set of the KD11-A Central Processor to provide extended manipulation of fixed-point numbers. When installed, it adds the capability of Arithmetic Shift, Arithmetic Shift Combined, Multiply, and Divide. With these additional instructions, the system can multiply and divide signed 16-bit numbers, and can shift signed 16-bit or 32-bit numbers. Condition codes are set in the processor on the result of each instruction.

1.1.2 Configuration

The KE11-E Option consists of one module. The single-hex X 8-1/2 in. M7238 module plugs directly into slot 2 (A–F) of the processor system unit. This is a dedicated prewired slot such that no other modules need be moved to accommodate its installation. When installed, the module functions as an extension of the basic KD11-A data paths, branch control, and control ROM. Basic timing of the processor is not degraded by use of this module, nor is the NPR latency affected when its instructions are being executed. Interrupts are serviced at the end of each instruction in the standard manner.

1.1.3 Specifications

Specifications for the KE11-E Option are given in Table 1-1.

Table 1-1
KE11-E (EIS) Specifications

Instructions	Arithmetic Shift (ASH) Arithmetic Shift Combined (ASHC) Multiply (MUL) Divide (DIV)
Operations	Multiplication and division of signed 16-bit numbers Arithmetic shifting of signed 16-bit or 32-bit numbers

Table 1-1 (Cont)
KE11-E (EIS) Specifications

Addressable Registers	None in option. Operands fetched from core or processor general registers.
Timing	Time = SRC Time + EF Time

SRC Mode	SRC Time
0	0.28 μs
1	0.78 μs
2	0.98 μs
3	1.74 μs
4	0.98 μs
5	1.74 μs
6	1.74 μs
7	2.64 μs

Instr	EF Time	Notes
MUL	8.88 μs	
DIV	11.30 μs	
ASH (right)	2.58 μs	+0.30 μs/shift
ASH (left)	2.78 μs	+0.30 μs/shift
ASHC (no shift)	2.78 μs	
ASHC (shift)	3.26 μs	+0.30 μs/shift

Size	Single Hex module (M7238)
Power Required	+5V, 2.3A

1.2 KE11-F FLOATING INSTRUCTION SET

The KE11-F Floating Instruction Set is a hardware option to the basic PDP-11/40 Computer System. It is supplied as a pluggable option to the KD11-A Central Processor and requires that the KE11-E described above be installed as a prerequisite.

1.2.1 Purpose

The KE11-F Floating Instruction Set (FIS) enables direct operations on single-precision 32-bit words in floating-point arithmetic. Since the KE11-E is a prerequisite to the KE11-F, extended manipulation of fixed-point numbers is available as well. The KE11-F Option further extends the PDP-11/40 instruction set to include Floating Add, Floating Subtract, Floating Multiply, and Floating Divide. As with the KE11-E, condition codes in the Processor Status Register are set on the result of each instruction. The prime advantage of this option is increased speed without the necessity of writing complex floating-point software routines.

1.2.2 Configuration

The KE11-F Option consists of one single-quad X 8-1/2 in. M7239 module with the KE11-E Option described above being a prerequisite. This FIS module plugs directly into slot 1 (A–D) also a dedicated prewired slot in the basic KD11-A. No degradation of processor timing or NPR latency is effected by the use of this option. Floating instructions are aborted if a BR request is issued before the instruction is within approximately 8 μs of completion, at which time the Program Counter (PC) is adjusted to point to the aborted floating instruction so that the instruction will be restarted upon return from the interrupt.

1.2.3 Specifications

Specifications for the KE11-F Option are given in Table 1-2.

Table 1-2
KE11-F (FIS) Specifications

Prerequisite	KE11-E Extended Instruction Set Option
Instructions	Floating-point Addition (FADD) Floating-point Subtraction (FSUB) Floating-point Multiply (FMUL) Floating-point Divide (FDIV)
Operations	Single-precision floating-point addition, subtraction, multiplication, and division of 24-bit numbers
Addressable Registers	None in option. Operands fetched from core.
Size	Single-quad module (M7239)
Power Required	+5V, 1.1A (typical)
Timing	Time = Basic Time + Binary Point Alignment Time + Normalization Time

Instr	Basic Time* μs	Binary Point Alignment Time Per Shift μs	Normalization Time Per Shift μs
FADD	18.78	0.30	0.34
FSUB	19.08	0.30	0.34
FMUL	29.00	—	0.34
FDIV	46.27	—	0.34

*Basic instruction times for FADD and FSUB assume exponents are equal or differ by one.

CHAPTER 2
PROGRAMMING

This chapter is devoted to general programming information for the KE11-E and KE11-F Options. It provides general descriptions of their operation, the formats and instructions for each. In addition, programming examples are supplied for each option. This chapter is intended merely as an introduction to the programming of this hardware. For more detailed information refer to the pertinent software documentation generated for these options. As with Chapter 1, information has been separated for each option.

2.1 KE11-E EXTENDED INSTRUCTION SET

There are no addressable registers in the KE11-E Option. EIS operands are fetched from either core memory or from the general processor registers. The result of each operation is stored in the general registers.

2.1.1 Operation

When the Arithmetic Shift (ASH) instruction is used, the contents of the selected register is shifted right or left the number of places specified by a count. This shift count is a 6-bit, 2's complement number which is the least significant 6 bits of the source operand. If the count is positive, the number is shifted left; if it is negative, the number is shifted right. This allows for shifts from 31 positions left to 32 positions right (+31 to -32) although a shift of greater than 16 places loses all significance. A count of 0 causes no change in the number.

When the Arithmetic Shift Combined (ASHC) instruction is used, the contents of the register (R) and the register ORed with 1 (RV1) are treated as a single 32-bit word. Register RV1 represents bits (15:00), register R represents bits (31:16). This 32-bit word is shifted right or left the number of places specified by a count. This shift count is the same as that described for the ASH instruction and permits shifts from +31 to -32. If the selected register (R) is an odd number, then R and RV1 are the same. In this case, the right shift becomes a rotate and the 16-bit word is rotated right the number of bits specified by the count for up to 16 shifts.

When the MULtiply (MUL) instruction is used, the contents of the Destination Register and the source are multiplied as 2's complement integers. The result is stored in the Destination Register R and the register ORed with 1 (RV1). If the register is odd, only the low-order product is stored. This instruction multiplies full 16-bit numbers.

When the DIVide (DIV) instruction is used, a 32-bit dividend in R and RV1 is divided by a 16-bit divisor to provide a 16-bit quotient and a 16-bit remainder. The sign of the remainder is always the same as the sign of the dividend unless the remainder is 0. Overflow is indicated if more than 16 bits are required to express the quotient. In this case, the instruction is aborted. If the content of the Source Register is 0, indicating divide by 0, an overflow is indicated.

2.1.2 Formats

The number formats for the KE11-E Option are shown in Figure 2-1. A single word is 16-bits long and a double word is 32-bits long. In the single word, bit 15 is the sign of the number; and in the double word, the sign bit is bit 15 of the high number part. The S bit is 0 for positive quantities and is 1 for negative quantities.

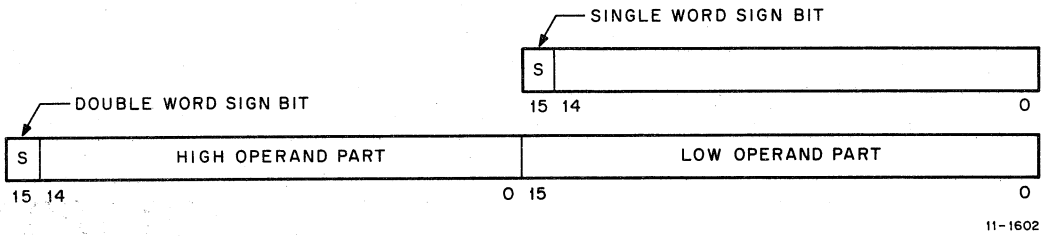


Figure 2-1 EIS Number Formats

2.1.3 Instructions

The EIS instruction format is shown in Figure 2-2. It is a double operand instruction in which bits (15:09) comprise the Op code, bits (08:06) designate the Destination Register field (RRR), bits (05:03) indicate the Source Address Mode (SSS), and bits (02:00) specify the Source Address Register (SSS). The octal coding is in the form 07XRSS. There are four EIS instructions, as follows:

MUL 070RSS

MULtiply

Operation: R, RV1 ← R X(SRC)

Condition Codes:

N: set if product is < 0; cleared otherwise.
Z: set if product is = 0; cleared otherwise.
V: cleared
C: set if the result is less than -2^{15} or is greater than or equal to $2^{15}-1$; cleared otherwise.

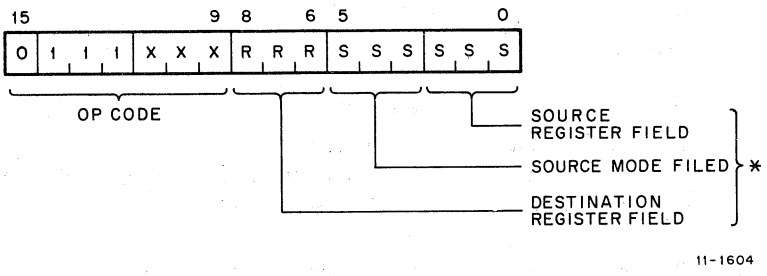
Description: The contents of the Destination Register R and source taken as 2's complement integers are multiplied and stored in the Destination Register R and the succeeding register RV1 (if R is even). If R is odd, only the low-order product is stored. Assembler syntax is:

MUL S, R. (Note that the actual destination is R, RV1 which reduces to just R when R is odd.)

Example: 16-bit product (R is odd)

```
000241          ,   CLC             ;Clear carry condition code
012701, 400     ,   MOV #400, R1
070127, 10      ,   MUL #10, R1
1034xx          ,   BCS ERROR       ;Carry will be set if
                                    ;product is less than
                                    ;-2^15 or greater than or
                                    ;equal to 2^15
                                    ;no significance lost
```

Before	After
(R1)=000400	(R1)=004000



***Note that for the EIS instructions the Source Register is considered the Destination since the answer is stored in that register. The Destination Mode and Register Field are considered to be the source. This is not consistent with other PDP-11 family instruction formats but is used throughout the discussions of the EIS instructions in this manual.**

Figure 2-2 EIS Instruction Format

DIV 071RSS

DIVide

Operation: R ← R, RV1 ÷ (SRC) RV1 ← Remainder

Condition Codes:

N: set if quotient < 0; cleared otherwise.
Z: set if quotient = 0; cleared otherwise.
V: set if source = 0 or if the absolute value of the register is larger than the absolute value of the source. (In this case, the instruction is aborted because the quotient would exceed 16 bits.)
C: set if divide by 0 attempted; cleared otherwise.

Description: The 32-bit 2's complement integer in R and RV1 is divided by the source operand (SSS). The quotient is placed in R; the remainder is placed in RV1 with the same sign of the dividend. R must be even.

Example:

```
005000          ,   CLR R0
012701,20001    ,   MOV #20001,R1
071027,2        ,   DIV #2, R0
```

Before	After	
(R0)=000000	(R0)=010000	Quotient
(R1)=020001	(R1)=000001	Remainder

ASH 072RSS

Arithmetic SHift

Operation: R ← R shifted arithmetically NN places to right or left, where NN = low-order 6 bits of source.

Condition Codes:

N: set if result < 0; cleared otherwise.
Z: set if result = 0; cleared otherwise.
V: set if sign of register changed during left shift; cleared otherwise.
C: loaded from last bit shifted out of register.

Description: The contents of the register are shifted right or left the number of times specified by the shift count. The shift count is taken as the low-order 6 bits of the source operand (SSS). This number ranges from -32 to +31. Negative is a right shift and positive is a left shift (Figure 2-3).

Example: ASH R0, R3

Before	After
(R3)=000003	(R3)=000003
(R0)=001234	(R0)=012340

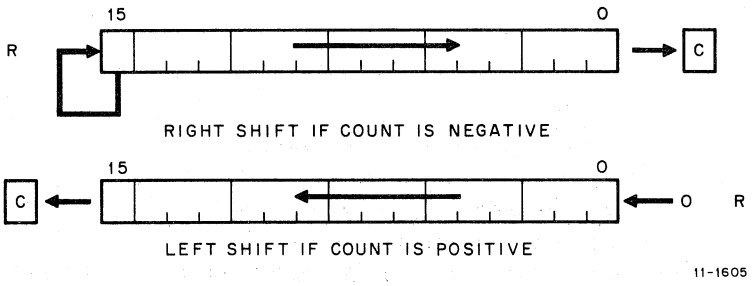


Figure 2-3 ASH Operation

ASHC 073RSS

Arithmetic SHift Combined

Operation: R, RV1 ← R, RV1. The double word is shifted NN places to the right or left, where NN = low-order six bits of source.

Condition Codes:

N: set if result < 0; cleared otherwise.
Z: set if result = 0; cleared otherwise.
V: set if sign bit changes during the left shift; cleared otherwise.
C: loaded with the last bit shifted out of the register.

Description: The contents of the register and the register ORed with 1 are treated as one 32-bit word. RV1 (bits 15:00) and R (bits 31:16) are shifted right or left the number of times specified by the shift count. The shift count is taken as the low-order 6 bits of the source operand. This number ranges from -32 to +31. Negative is a right shift and positive is a left shift (Figure 2-4). When the register chosen is an odd number, the register and the register ORed with 1 are the same. In this case, the right shift becomes a rotate. The 16-bit word is rotated right the number of bits specified by the shift count for up to 16 shifts.

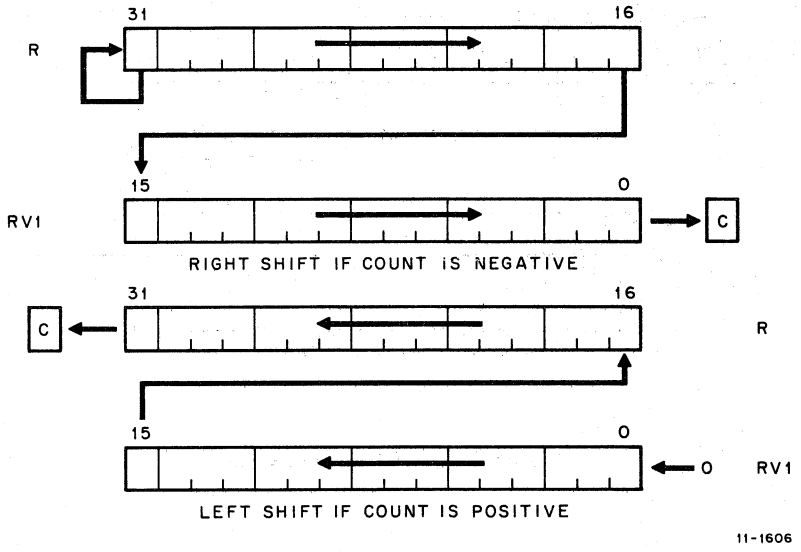


Figure 2-4 ASHC Operation

2.2 KE11-F FLOATING INSTRUCTION SET

There are no addressable registers in the KE11-F Option. FIS operands are fetched from core memory and the result of each operation is stored in core memory. Operands are ordered on the stack in Polish Notation (Paragraph 3.2), thereby reducing the number of operations necessary to achieve a result.

2.2.1 Operation

For Floating ADD, the A argument from the stack is added to the B argument from the stack with the result stored in the A argument position on the stack.

For Floating SUBtract, the B argument from the stack is subtracted from the A argument on the stack with the result stored in the A argument position on the stack.

The Floating MULtiply instruction multiplies the A argument on the stack by the B argument on the stack and stores the result in the A argument position on the stack.

The Floating DIVide instruction divides the A argument on the stack by the B argument on the stack and stores the result in the A argument position on the stack.

2.2.2 Formats

The number format for the KE11-F Option is shown in Figure 2-5. The KE11-F word is 32 bits long with bit 15 of the high argument designating the sign of the fraction. Note that the 8-bit exponent separates the fraction from its associated sign. In floating point, representation of binary numbers is in three parts: a sign bit, an exponent, and a mantissa. The mantissa is a fraction expressed in sign and magnitude format with the binary point positioned to the left of the most significant bit of the mantissa. The mantissa is assumed to be normalized. The MSB of the mantissa is not stored in core because it is redundant. Leading 0s are removed by shifting the mantissa left; however, each left shift of the mantissa must be followed by a decrement of the exponent value to maintain the true value of the number. The exponent value represents the power of 2 by which the mantissa is multiplied to obtain the value to be used.

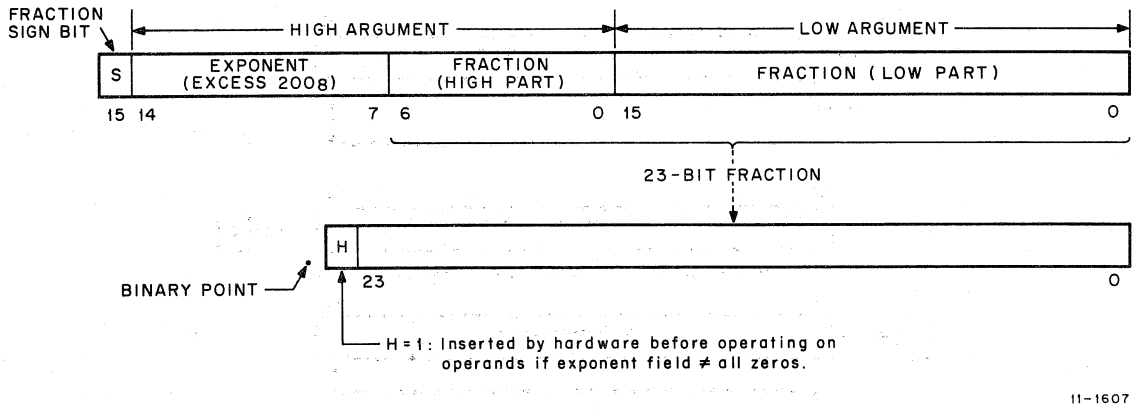


Figure 2-5 FIS Number Format

The KE11-F Option stores the exponent in excess 200_8 (128_{10}) notation. As a result, exponent values from -128 to +127 are represented by the binary equivalent of 0 to 255 (octal 0–377). Mantissas are represented in sign magnitude form.

The binary radix point is to the left. The results of the floating-point operations are always rounded away from 0, increasing the absolute value of the number.

If the exponent is equal to 0, the number is assumed to be 0 regardless of the sign bit or fraction value. The hardware generates a clean 0 (32-bit word of all 0s) in this case.

2.2.3 Instructions

The FIS instruction format is shown in Figure 2-6. It is a double operand instruction in which the low three bits (R,R,R) specify a register that is utilized as a stack pointer for the floating-point operands. The register may be any one of the eight general registers, but some caution must be used if using the PC (R7). It is unlikely that the PC would be desirable as a pointer.

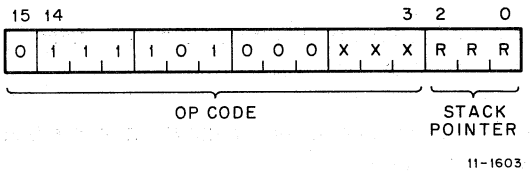
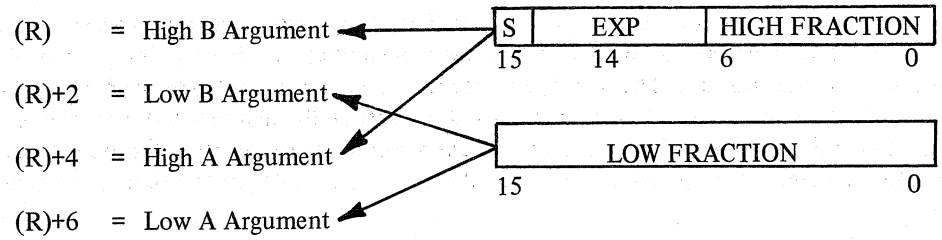


Figure 2-6 FIS Instruction Format

The operands are located on the stack as follows:



The floating-point answers are stored as follows:

(R)+4 = High Answer
(R)+6 = Low Answer

The floating-point stack pointer is repositioned to point to (R)+4 (High Answer).

The floating-point octal coding is in the form 0750XR. There are four FIS instructions, as follows:

FADD 07500R

Floating-ADD

Operation: [(R) +4 □ (R) +6] ← [(R) +4 □ (R) +6] + [(R) □ (R) +2], if result $\geq 2^{-128}$; else [(R) +4 □ (R) +6] ← 0

Condition Codes: (See Note Below)
N: set if result < 0; cleared otherwise.
Z: set if result = 0; cleared otherwise.
V: cleared
C: cleared

Description: Adds the B argument to the A argument and stores the result in the A argument position on the stack. A ← A+B

FSUB 07501R

Floating-SUBtract

Operation: [(R) +4 □ (R) +6] ← [(R) +4 □ (R) +6] - [(R) □ (R) +2], if result $\geq 2^{-128}$; else [(R) +4 □ (R) +6] ←0

Condition Codes: (See Note Below)
N: set if result < 0; cleared otherwise.
Z: set if result = 0; cleared otherwise.
V: cleared
C: cleared

Description: Subtracts the B argument from the A argument and stores the result in the A argument position on the stack. A ← A-B

FMUL 07502R

Floating-MULtiply

Operation: [(R) +4 □ (R) +6] ← [(R) +4 □ (R) +6] * [(R) □ (R) +2], if result $\geq 2^{-128}$; else [(R) +4, (R) +6]

Condition Codes: (See Note Below)
N: set if result < 0; cleared otherwise.
Z: set if result = 0; cleared otherwise.
V: cleared
C: cleared

Description: Multiplies the B argument by the A argument and stores the result in the A argument position on the stack. A ← A*B. If the result is $< 2^{-128}$, then underflow occurs and the destination address will contain the A argument.

FDIV 07503R

Floating-DIVide

Operation: [(R) +4 □ (R) +6] ← [(R) +4 □ (R) +6] / [(R) □ (R) +2], if result $\geq 2^{-128}$; else [(R) +4 □ (R) +6]

Condition Codes: (See Note Below)

N: set if result <0; cleared otherwise.
Z: set if result = 0; cleared otherwise.
V: cleared
C: cleared

Description: Divides the A argument by the B argument and stores the result in the A argument position on the stack. If the B argument (divisor) is equal to 0, the stack is left untouched. A ← A/B. If the result is $< 2^{-128}$, then the destination address will contain the A argument.

NOTE

If a trap occurs as a function of a floating instruction, the condition codes are reinterpreted as follows:

N: set if underflow, cleared if overflow.
Z: cleared
V: set if underflow, overflow, divide by 0 (error conditions).
C: set if divide by 0, otherwise cleared.

Traps occur through the vector 244. (R) is reset to point to high B argument on the stack. The arguments are left untouched.

2.2.4 Programming Example

A sample floating-point program is given below.

```
1
2          000000'          .CSECT
3                           .TITLE FISEXM
4                  ;
5                  ;        COPYRIGHT 1972 BY DIGITAL EQUIPMENT CORPORATION,
6                  ;                MAYNARD, MASSACHUSETTS,
7                  ;
8                  ;
9                  ;    EXAMPLE OF PDP-11/40 FLOATING INSTRUCTION SET USAGE (FIS)
10                 ;
11                 ;    COMPUTE LARGER ROOT OF QUADRACTIC EQUATION:
12                 ;
13                 ;        A*X*X + B*X + C = 0
14                 ;
15                 ;    ALGORITHM IS:
16                 ;
17                 ;        ROOT1 = (-B + SQRT(B*B - 4*A*C))/(2*A)
18                 ;
```

```
                        ;   INITIAL VALUES OF A, B, AND C ARE
                        ;       PLACED IN MEMORY LOCATIONS A, B, AND C,
                        ;       RESULT IS COMPUTED AND STORED AT ROOT1,
                        ;
                        ;       NORMAL TERMINATION IS A HALT AT LOCATION DONE,
                        ;       IF DISCRIMINANT IS NEGATIVE THEN HALT AT LOCATION
                        ;       IMAG. HALT AT AZERO IF A = 0,
                        ;
                        ;   NORMAL REGISTER DECLARATIONS:
        000000 R0       =%0
        000001 R1       =%1
        000002 R2       =%2
        000003 R3       =%3
        000004 R4       =%4
        000005 R5       =%5
        000006 SP       =%6
        000007 PC       =%7
                        ;
                        ;   PROGRAM STARTS HERE
                        ;
00000 012706 START:     MOV     #STACK,SP       ;INITIALIZE PROCESSOR STACK
      000442'
00004 016746            MOV     B+2,-(SP)       ;B TO STACK
      000204
00010 016746            MOV     B,-(SP)
      000176
00014 016746            MOV     B+2,-(SP)       ;AGAIN
      000174
00020 016746            MOV     B,-(SP)
      000166
00024 075026            FMUL    SP              ;FORM B*B
00026 005046            CLR     -(SP)           ;4.0 TO STACK
00030 012746            MOV     #^F4.0,-(SP)
      040600
00034 016746            MOV     A+2,-(SP)       ;A TO STACK
      000150
00040 016746            MOV     A,-(SP)
      000142
00044 001457            BEQ     AZERO           ;HALT IF A = 0,
00046 016746            MOV     C+2,-(SP)       ;C TO STACK
      000146
00052 016746            MOV     C,-(SP)
      000140
00056 075026            FMUL    SP              ;FORM A*C
00060 075026            FMUL    SP              ;FORM 4.*A*C
00062 075016            FSUB    SP              ;FORM B*B-4.*A*C (DISCRIMINANT)
00064 100446            BMI     IMAG            ;BRANCH IF NEGATIVE
00066 012667            MOV     (SP)+,TEMP1     ;STORE DISCRIMINANT
      000130
00072 012667            MOV     (SP)+,TEMP1+2
      000126
00076 004567            JSR     R5,SQRT         ;CALL FORTRAN SQUARE ROOT ROUTINE
      000000G
00102 000401            BR      .+4
00104 000222            .WORD   TEMP1
00106 010067            MOV     R0,TEMP2        ;STORE RESULT
      000114
00112 010167            MOV     R1,TEMP2+2
      000112
                        ;COMPUTE ROOT1
00116 016746            MOV     B+2,-(SP)       ;B TO STACK
      000072
00122 016746            MOV     B,-(SP)
      000064
00126 062716            ADD     #100000,@SP     ;NEGATE B ON STACK
      100000
```

```
67 00132 016746          MOV     TEMP2+2,-(SP)   ;SQUARE ROOT TO STACK
         000072
68 00136 016746          MOV     TEMP2,-(SP)
         000064
69 00142 075006          FADD    SP              ;FORM -B+SQRT
70 00144 016746          MOV     CONST+2,-(SP)   ;2,0 TO STACK
         000064
71 00150 016746          MOV     CONST,-(SP)
         000056
72 00154 016746          MOV     A+2,-(SP)       ;A TO STACK
         000030
73 00160 016746          MOV     A,-(SP)
         000022
74 00164 075026          FMUL    SP              ;FORM 2,*A
75 00166 075036          FDIV    SP              ;FORM (-B+SQRT)/(2,*A)
76 00170 012667          MOV     (SP)+,ROOT1     ;SAVE RESULT
         000042
77 00174 012667          MOV     (SP)+,ROOT1+2
         000040
78 00200 000000 DONE:    HALT
79 00202 000000 IMAG:    HALT
80 00204 000000 AZERO:   HALT
81               ;
82               ;
83 00206        A:       .BLKW   2
84 00212        B:       .BLKW   2
85 00216        C:       .BLKW   2
86 00222        TEMP1:   .BLKW   2
87 00226        TEMP2:   .BLKW   2
88 00232 040400 CONST:   .FLT2   2.0
   00234 000000
89 00236        ROOT1:   .BLKW   2
90                       .GLOBL  SQRT            ;EXTERNAL SUBROUTINE
91                       .BLKW   100             ;ROOM FOR STACK
92 00442        STACK:   .BLKW   1               ;START OF STACK IS TOP OF AREA
93       000001'         .END
```

CHAPTER 3
THEORY OF OPERATION

This chapter describes KE11-E and KE11-F theory of operation with the EIS principles described first, followed by those principles applying to the FIS hardware. A review of the basic requirements for the operations is presented as well as the algorithms for those operations. Each algorithmic description is first given in basic terms, followed by a more specific treatment of the operation involved.

3.1 KE11-E EXTENDED INSTRUCTION SET

The KE11-E Option is used for fixed-point operations in the KD11-A Central Processor. The principles involved in these instructions are given in the following paragraphs.

3.1.1 Binary 2's Complement Notation

The KE11-E Option requires a numerical notation that expresses both the sign and the magnitude of each number in binary digits. The simplest class of notation that meets this requirement is based on the following property: a number added to its own negative equals 0. Thus, adding the negative of a number to another number is the same as subtracting the number. The 2's complement of a number is created by complementing and incrementing the number, i.e., replacing each 0 bit with a 1 and each 1 bit with a 0, and then adding a 1 to the resultant number in the least significant position. Adding a number and its negative in 2's complement notation always produces all 0s (the only representation of the quantity 0 in 2's complement).

It is important to remember that the *representation* of a number differs greatly from *quantity* represented. For example: the *quantity* -1 is represented in 2's complement notation by 11 111 111 (in eight bits). The *quantity* +1 has a 2's complement *representation* of 00 000 001.

Example 1

Adding +1 to -1 yields the following:

```
  00 000 001 = +1
+11 111 111 = -1
---------------
100 000 000 = 0      (The left-most (carry) bit is not a significant bit and is ignored.)
```

Example 2

Adding +5 and -3 yields the following:

```
  00 000 101 = +5
+11 111 101 = -3
---------------
100 000 010 = +2     (Carry bit not significant.)
```

A disadvantage of 2's complement notation is that the representation of numbers is not symmetrical. That is, one more negative number than positive number can be expressed. In n bits, the maximum positive number that can be expressed is 2^{n-1}, but the maximum negative number is -2^{n-1} (because there is no negative 0).

3.1.2 Multiplication

Multiplication is repeated addition. Multiplying 3 times 7 is simply adding 7 three times. However, 2^{15} times a number requires 2^{15} additions (the KE11-E uses a short-cut method that requires only 16 operations).

In practice, the KE11-E adds multiples of the multiplicand. The multiples are formed by shifting. Each time a binary number is shifted one bit to the left, it is multiplied by two; thus, if it is shifted 5 places to the left, it is multiplied by 2^5 (32). The multiplier is broken down into individual bits that determine which multiples of the multiplicand are added to form the product.

The multiplication process is complicated by the representation of negative numbers used in the PDP-11 Systems. Negative 2's complement numbers cannot be multiplied by the addition of multiples unless a correction step is added at the end. To avoid this step, the KE11-E uses a method that provides for negative numbers and produces the same results as the addition-of-parts method for positive numbers. This method is based on a different breakdown of a binary number into positive and negative parts.

In binary numbers, 10−1=1. Representing each 1 bit of a binary number as the difference between that bit and the next most significant bit produces a string of alternating positive and negative powers of 2.

For example:

```
11010111=100000000-10000000+10000000-1000000+100000-10000
        +1000-100+100-10+10-1
        =100000000-1000000+100000-10000+1000-1
```

Multiplying a multiplicand by each of the numbers in the last string (preserving the signs) and then adding the products of the multiplications is equivalent to multiplying the chosen multiplicand by the original number (11010111). This can be done by shifting the multiplicand left and adding or subtracting at each position that corresponds to one of the numbers in the series.

The series of alternating positive and negative powers of 2 is easily generated because:

a. Each pair of powers of 2, one positive and a smaller negative, represents a string of 1s. The positive number is one digit higher than the most significant 1 in the string, and the negative number is in the same position as the least significant 1 in the string.

   For example, in the number 11010111:

```
100000000-1000000=11000000
  100000-  10000=00010000
   10000-      1=00000111
                 11010111
```

b. Strings of 1s are separated by strings of 0s. Each string is one or more digits long.

   For example, in the number 11010111:

```
-1000000+100000=0X2^5
  -10000+  1000=0X2^3
                11010111
```

c. Thus, each string of 1s can be replaced by a string of 0s with a -1 in the least significant place, and each string of 0s can be replaced by 0s with a +1 in the least significant place.

For example, the digits in the number 11010111 can be replaced as follows:

Original digit:	1	1	0	1	0	1	1	1
Replacement:	0	-1	+1	-1	+1	0	0	-1

d. Therefore, if for any bit of the multiplier the previous (less significant) bit is the same, the multiplicand is not added to the partial product (or it is multiplied by 0 and 0 is added to the product). If the previous bit is a 1 and the current bit is a 0, the multiplicand is added; if the previous bit is 0 and the current bit is 1, the multiplicand is subtracted (negated and added). In each case the multiplicand is shifted (with respect to the product) before the addition, because the number added to the product is actually the multiplicand multiplied by some power of 2.

For example, to multiply N by 11010111, the sum of the products of N times each replacement digit, times the appropriate power of 2, is the product as follows:

$$N\times11010111=N\times(0\times2^7-1\times2^6+1\times2^5-1\times2^4+1\times2^3+0\times2^2+0\times2^1-1\times2^0)$$

3.1.3 Division

Division is repeated subtraction. Division is more complicated than multiplication for two reasons:

a. The product of two integers is always an integer; the quotient of two integers is rarely an integer. Division produces two results, a quotient and a remainder, that interact. The correct quotient is dependent on a correct remainder.

b. The maximum value that results from the multiplication of two numbers can be no larger than the source of the maximum number. However, the maximum value that can result from a division is infinite, because the divisor can be much smaller than the dividend. Some quotients cannot be expressed in the number of bits available in the physical representation and are considered to have overflowed.

The quotient in a division is the number of times that the divisor can be subtracted from the dividend without going beyond 0 (changing sign). The result can be determined by counting subtractions until the remainder *does* go beyond 0 (which produces a condition called underflow), then reducing the count by one. The remainder must also be corrected by restoring the value of one subtraction.

Rather than do as many as 2^{15} subtractions, the KE11-E uses a short-cut method similar to that used in multiplication. The results of dividing by multiples of the divisor, where each multiple is a power of 2 times the divisor, can be combined to form the quotient.

This division procedure operates by subtracting a large multiple (2^{n-1}) of the divisor from the dividend. If the remainder does not go beyond 0 (there is no underflow), the next smaller power-of-2 multiple of the divisor is subtracted. For each successful subtraction, the quotient is increased by the same multiple (the same power of 2) as the multiple of the divisor used in the subtraction.

If a subtraction causes underflow, however, the corresponding quotient bit is cleared (the corresponding power of 2 is not added to the quotient). However, rather than restoring the previous value of the dividend, the KE11-E now approaches the correct remainder from the opposite direction. Successively smaller multiples of the divisor are added to the remainder (instead of subtracting) until the remainder again underflows, thus restoring the original sign. When the KE11-E is adding, instead of subtracting, the corresponding quotient bits are set only if the sign of the remainder returns to its original value; if the remainder does not change sign, the quotient bit is set to 0.

For example, dividing 17 (21_8) by 5 yields a quotient of 3 and a remainder of 2 as follows:

1. Subtract divisor X 2^3 from the dividend.

   00 010 001 = 21_8
   11 011 000 = $-5 \times 2^3 = -50_8$
   11 101 001 (The partial remainder has the wrong sign, underflow occurred.)

2. Add 0×2^3 to the quotient = 0.

3. Add divisor X 2^2 to the partial remainder.

   11 101 001
   00 010 100 = $5 \times 2^2 = 24_8$
   11 111 101 (The partial remainder has the wrong sign, no underflow.)

4. Add 0×2^2 to the quotient = 0.

5. Add divisor X 2^1 to the partial remainder.

   11 111 101
   00 001 010 = $5 \times 2^1 = 12_8$
   00 000 111 (The partial remainder has the right sign, underflow occurred.)

6. Add 1×2^1 to the quotient = 2.

7. Subtract divisor X 2^0 from the partial remainder.

   00 000 111
   11 111 011 = $-5 \times 2^0 = -5$
   00 000 010 (The remainder has the right sign and is 2, no underflow.)

8. Add 1×2^0 to the quotient = 3.

This procedure is valid for positive or negative numbers, provided that the dividend and divisor have the same sign. However, if the signs are originally different, subtracting multiples of the divisor drives the remainder away from 0. Therefore, the KE11-E adds multiples of the divisor until the remainder underflows (at which point, a quotient bit is set) and then subtracts until the remainder regains its original sign.

NOTE

In the KE11-E, if the dividend is negative, its 2's complement is taken before dividing. Adding the 2's complement is the same as subtracting.

This procedure can handle any combination of binary numbers, regardless of sign. Implementation of the procedure is simplified by the following considerations:

a. If the signs of the divisor and the dividend are originally the same, the KE11-E subtracts until they differ (because the sign of the remainder changes), then adds until they are the same. If the signs are originally different, the KE11-E adds until they are the same, then subtracts until they differ.

b. Therefore, for each operation, the KE11-E compares the signs of the remainder and divisor. If they differ, the KE11-E adds the divisor, shifted to form the proper multiple (power of 2 times the divisor); if the signs are the same, the KE11-E subtracts.

c. A quotient bit is set if the sign of the remainder remains the same after a subtraction or changes after an addition; the quotient bit is cleared if the sign changes after a subtraction or remains the same after an addition.

d. A subtraction is done if the signs of the remainder and divisor are the same, and an addition is done if the signs are different. A changed sign after an addition means that the signs are now the same, while no change after a subtraction also means the signs are the same.

e. Therefore, after each operation, the corresponding bit of the quotient is set if the signs are the same or cleared if the signs differ.

3.1.4 Basic Shift Operation

In the KE11-E, a basic shift operation is used as a primary operation in the sequences for all other operations. The register that is being shifted is treated as a sequence of bits, each shifted separately. The following descriptions illustrate the features of a basic shift:

a. In general, the bit at a particular location is replaced by another bit that is shifted into that location. No bit of information is moved more than one location.

b. One bit is shifted out of the register and is lost.

c. One bit is vacated. The original contents of that bit are shifted to the next bit, and a 0 replaces the previous bit if shifting left.

d. The bit lost is at the end toward which the bits are shifted, and the bit vacated is at the end away from which the bits are shifted (bit positions are numbered in ascending order from right to left).

3.1.5 Algorithms For KE11-E Operations

Figures 3-1 through 3-4 illustrate the sequence of operations for multiplication, division, and shifting. These flow charts emphasize the conceptual organization of the device that does each calculation. Chapter 4 relates the KE11-E logic to these algorithms and explains how the logic structure reduces the hardware and timing requirements. All KE11-E arithmetic operations are performed in the 16-bit ALU in the PDP-11/40 Central Processor KD11-A. This ALU has two 16-bit inputs. The B input is supplied (for the sake of the following discussions) by the B Register. The A input is supplied by all of the following sources:

a. The CPU general registers ($17_8 - 00_8$).

b. The KE11-E registers BR, DR, and (BR (14:00), DR15).

The KE11-E also supplies a carry-in signal to bit 0 of the CPU ALU. The BR is simply a holding register whereas the DR is a left/right shift register.

The ASH right operation is implemented by the right data port of the CPU DMUX. Similarly for ASHC the high half of the operand is shifted by the DMUX while the DR shifts the low half of the operand.

The high half of the operand for ASHC left and the entire operand for ASH left are shifted by the ALU function A plus B, while the low half of the ASHC operand is shifted by the DR Register.

In MULtiplication, the DR holds the multiplicand and, therefore, controls the summation of partial products; as the multiplicand is shifted out, the low order word of the product is shifted in. In DIVision, the DR holds the low order dividend, and is used to assemble the quotient; as the dividend is being shifted out, the quotient is being shifted in.

3.1.5.1 Multiplication – As shown in Figure 3-1, as the flow is entered at MUL, the multiplicand is loaded, as is the Step Counter, and the C bit in the Extended Processor Status Register is cleared.

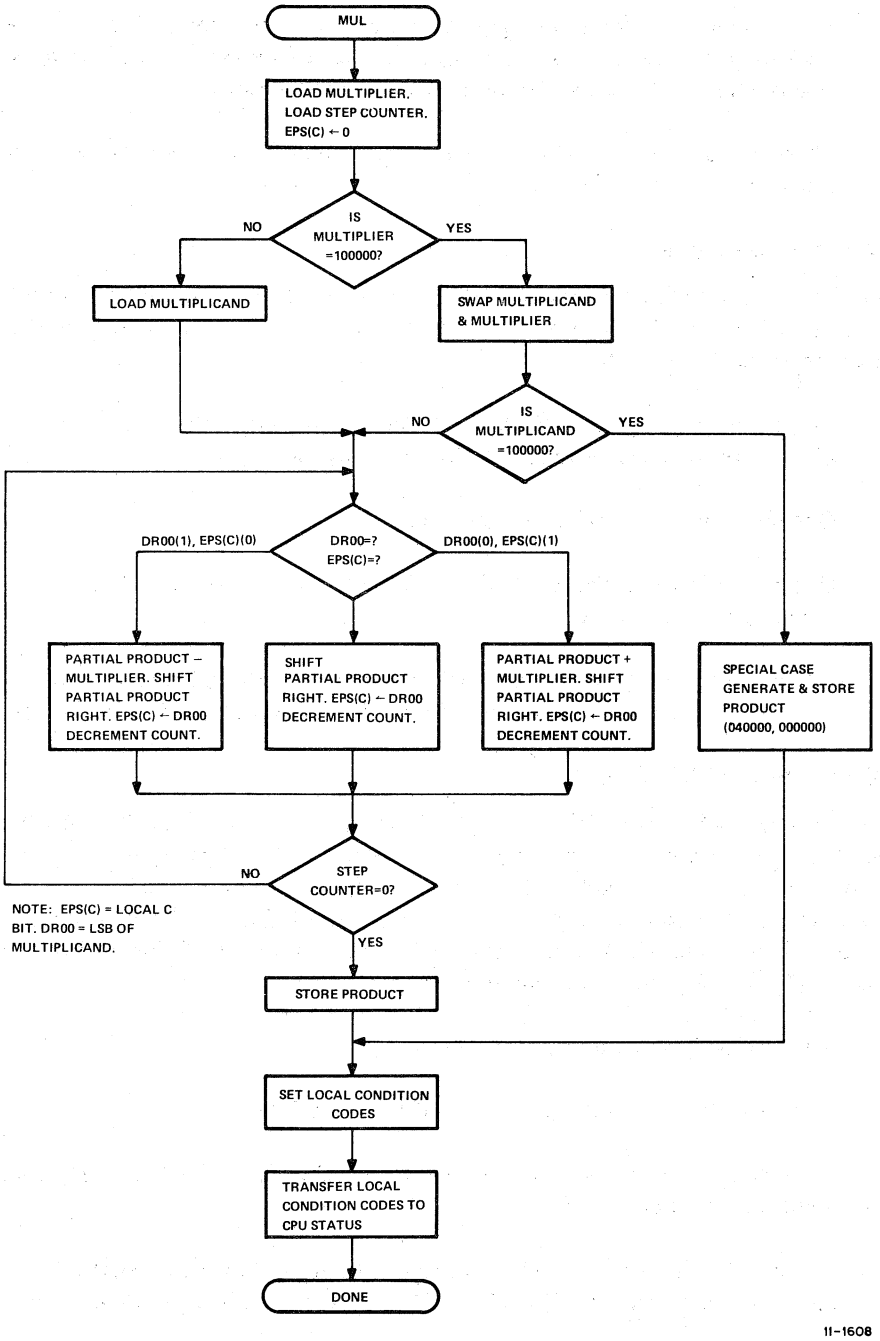


Figure 3-1 KE11-E MUL Algorithm

At this point, a test is made to determine if the multiplicand is equal to 100000 (the most negative number). If it is not, the multiplier is loaded and the multiply loop is entered; if it is equal to 100000, the multiplier and multiplicand are swapped and the multiplier is tested to see if *it* is the most negative number. If the multiplier proves to also be the most negative number, a special case exists in which the answer can be generated. The product is stored, local condition codes are set and transferred, and the operation is done.

If the multiplier is not the most negative number, the multiply loop is entered as above. At this point, the hardware looks at the two least significant bits (DR00 and EPS(C)) to decide whether to either add or subtract and then shift, or to just shift. In all cases, DR00 is sent to EPS(C) as the hardware executes the loop. This action continues, each time testing to see if the Step Counter is equal to 0. When it is, the operation in the loop is complete. To conclude the operation, the loop is left, the product is stored, and local condition codes are set before being transferred to the CPU Status Register.

In the KE11-E hardware, the multiplier (the contents of the calculated destination address) is in B and the multiplicand (from R(SF)) is in DR. BR and EPS(C) are clear.

If DR00 (the LSB of the multiplicand) is (1), B is subtracted from BR. The result, shifted one place to the right, is loaded back into BR with the LSB of the result shifted into DR15. The DR is shifted right so a bit of the multiplicand is shifted into EPS(C). The sign of the result is loaded into BR15 and BR14. If DR00 is (0), the BR and DR are shifted right one place, with BR00 being shifted into DR15. The BR is shifted by the DMUX right data port.

In each subsequent step, only the shift is performed if the bits in DR00 and EPS(C) are the same. If they are different, addition or subtraction is performed along with the shift. If DR00 = (0) and EPS(C) = (1), B is added to BR. If DR00 = (1) and EPS(C) = 0, B is subtracted from BR.

Consequently the low order bits of the running sum of the partial products are shifted into DR as the multiplicand is shifted out. At each step, the effect of the multiplier in B on the partial sum in BR15 is binarilly one order of magnitude greater than in the preceding step because the partial sum was shifted right. B can consequently be combined directly with BR. The first arithmetic operation will always be subtraction. If DR00 is initially (0), no subtraction will be performed until a (1) is shifted into it. Shifting will then continue until DR00 is (0) and EPS(C) is (1).

This process continues, subtracting when DR00 is (1) and EPS(C) is (0), adding when DR00 is (0) and EPS(C) is (1), and simply shifting when DR00 is the same as EPS(C).

After 16 steps, the DR holds the low half of the product and the BR holds the high half of the product.

The following example shows that the above procedure produces the correct product.

```
1 0 0 1 1 0 0 1 = Binary Integer
7 6 5 4 3 2 1 0 = Powers of 2 for each position
```

This number is equal to

```
 1 0 0 0 0 0 0 0
+     1 1 0 0 0
+           1
```

A string of 1s whose right-most bit corresponds to 2^k is equal to $2^{k+n}-2^k$ or equivalently $2^k(2^n-2^0)$, i.e., 2^n-2^0 is a string of n 1s and the 2^k shifts the string left k places. Therefore,

$$1\,0\,0\,0\,0\,0\,0\,0 = 2^{7+1}-2^7 = 2^8-2^7$$
$$1\,1\,0\,0\,0 = 2^{3+2}-2^3 = 2^5-2^3$$
$$1 = 2^{1+0}-2^0 = 2^1-2^0$$
$$2^8-2^7+2^5-2^3+2^1-2^0$$

In the last representation, each power of 2 that is subtracted corresponds to a transition from (0) to (1) (from right to left) whereas each power of 2 that is added corresponds to a (1) to (0) transition. The largest term corresponds to the transition to the sign bit, which is (0) for a positive number. The multiplication algorithm interprets the multiplicand in the above manner, alternatively subtracting and adding the multiplier to the partial sum in the order-of-magnitude positions corresponding to the transitions.

3.1.5.2 Division – As shown in Figure 3-2, as the flow is entered at DIV, the divisor, dividend, and step count are all loaded. At this point, a test is made to determine if the divisor is equal to 0. If it is, the local condition codes are set to indicate the error, they are transferred to the CPU Status Register, and the flow ends at that point.

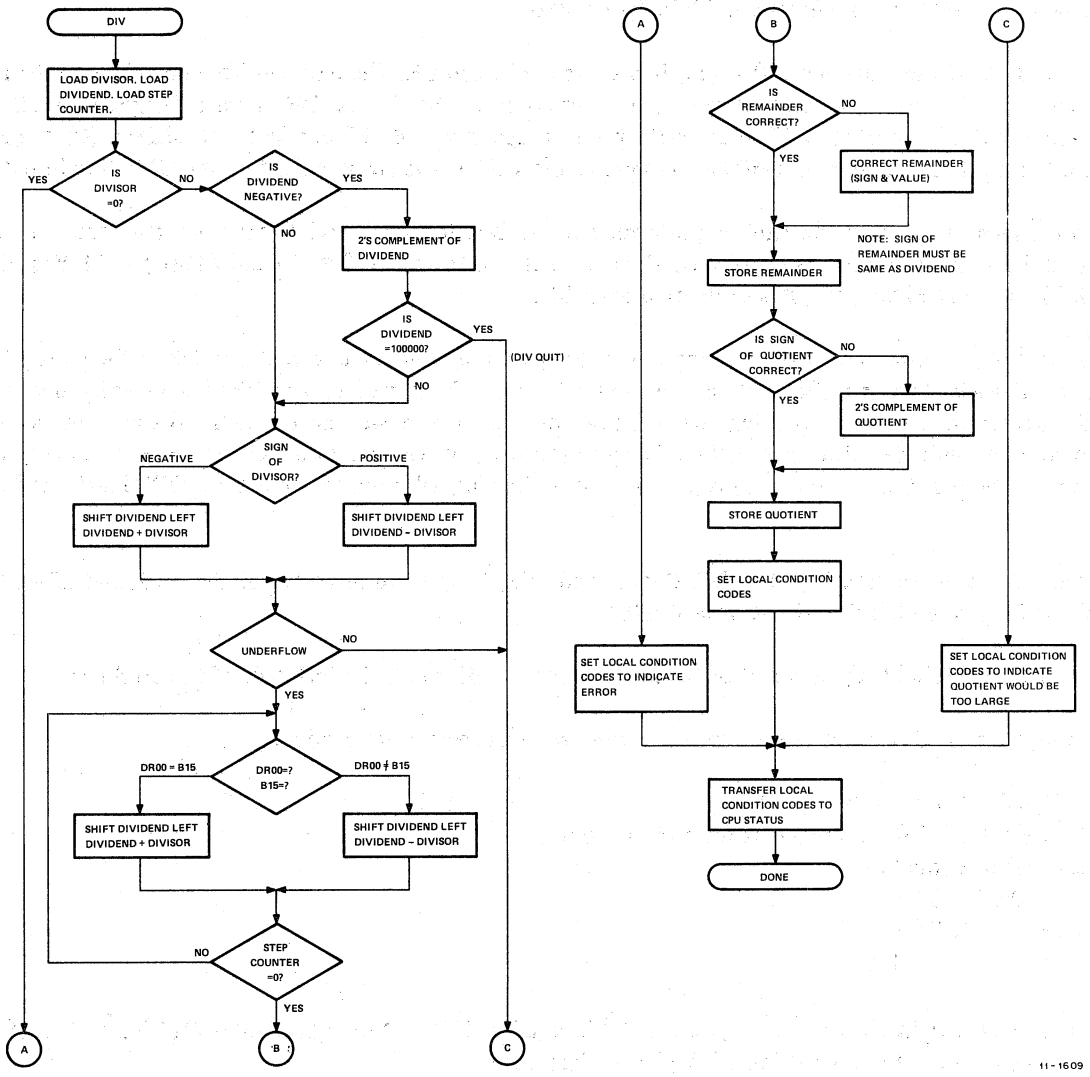


Figure 3-2 KE11-E DIV Algorithm

If the divisor is not equal to 0, a test is made to see if the dividend is negative. If it is not, the flow continues to test the sign of the divisor, but if it is a negative dividend, the 2's complement is taken of it to determine if it is the most negative number (=100000). If this condition exists, division is impossible and the DIV QUIT path is taken to set the local condition codes to indicate that the quotient would be too large. The codes are then transferred to the CPU and the flow is ended.

If, however, after 2's complementing the dividend, it proves not to be the most negative number, the flow continues to test the sign of the divisor as above. This is the first division step; and, if the divisor is negative, it is necessary to add it to the dividend since adding a negative number is equivalent to subtracting a positive number. But if the divisor is positive, it is simply subtracted. Before either the add or subtract operation, however, the dividend is shifted left.

At this point, the hardware is caused to look for the result of this initial add or subtract operation as indicated by the presence of or absence of a carry-out from the ALU. If a carry-out is not seen, this indicates that the scaling of the dividend vs the divisor will produce the proper number of bits in the quotient. If a carry-out is seen (underflow), however, it indicates that more than 16 bits are required to display the answer and the DIV QUIT path is taken to set local condition codes before they are transferred, and the flow is terminated.

If the first division step does not produce an underflow, the division loop is entered and the operation continues. In the divide loop, two conditions are monitored for each pass through the loop. These conditions are DR00 (which represents the carry-out result of the last pass through the loop), and B15 (the sign of the divisor). This test is made for each pass through the divide loop, up to the number set in the step count. Each pass, the two bits are sampled, and the appropriate action taken. If the divisor is indicated as being negative and a carry-out was detected for the last pass, the indication is that too much was either added or subtracted in that last step, causing the reverse action to take place in the currrent step. This continues until the step count reaches 0, at which time the loop is left and the flow continues.

At this point in the flow, a test is made to see if either the sign or value of the remainder is incorrect. This is the fix-up step in which the sign or value of the remainder is corrected. The sign of the remainder should match the sign of the dividend. Once done, the remainder is stored and the sign of the quotient is tested for correctness. If it is incorrect, the 2's complement of the quotient is taken and, in either case, the quotient is stored.

The flow ends in the usual manner with the local condition codes being set and transferred.

In the KE11-E hardware, the divisor (the contents of the calculated destination address) is in the B Register. The BR and DR hold the high and low dividend, respectively. If the dividend is negative, the 2's complement is taken and loaded into BR and DR.

Before the division process is initiated, a check for the divisor being 0 is made. If a 0 divisor is detected, the division is aborted with the condition codes (Z, V, C) set to indicate the error.

The first step of division is performed so that a test for underflow may be made to determine if the quotient is too large to be expressed in 16 bits. If underflow does not occur, the instruction is aborted. If not, the remaining 15 division steps are performed.

Note that the dividend is shifted left one place before each addition or subtraction, dropping the current MSB of the dividend. As the dividend is shifted out, the quotient is shifted in.

The test for underflow that determines whether the ALU should add or subtract is based on the following considerations:

a. If the divisor is negative and the dividend is positive, adding the divisor to the dividend should produce a result closer to 0 than the original dividend. If the result is negative, underflow has occurred and a 0 is shifted into the DR.

b. If the divisor is negative and the dividend is also negative, an underflow condition already exists. The divisor is subtracted from the dividend to return the dividend to a positive number. If the result is still negative, a 0 is shifted into the DR; if the result is positive, the underflow has been corrected and a 1 is shifted in.

c. For a positive divisor and dividend, a subtraction is performed. If the result is positive, a 1 is shifted into the DR, but if the result is negative, underflow has occurred and a 0 is shifted in.

d. If the divisor is positive and the dividend is negative, an addition is performed to correct an existing underflow. If the result is positive, the underflow has been corrected and a 1 is shifted into the DR, otherwise a 0 is shifted in.

As a result of these considerations, if the divisor is positive (B15 is 0) and there is no underflow (DR00 is 1), or if the divisor is negative (B15 is 1) and there is underflow (DR00 is 0), the KE11-E performs a subtract operation and shifts the carry-out of the ALU into DR00. A carry-out of the MSB of the ALU indicates that no underflow has occurred; if an uncorrected underflow existed, the carry indicates that it has been corrected.

If the opposite conditions exist (divisor is positive and DR00 is 0 or divisor is negative and DR00 is 1), an addition is performed, followed by a shift of the carry-out of the ALU into DR00. Note that the cases for which a carry-out of the MSB of the ALU exists are equivalent to the cases described above for which DR00 is set.

If, after the last division step, the LSB of the quotient is a 0, an underflow condition still exists. This condition can be corrected (unless an overflow condition also exists) by adding a positive divisor or subtracting a negative divisor to correct the remainder. If no remainder correction is needed and the remainder has the wrong sign or has the wrong sign after correction, the remainder is complemented and stored.

If EPS (N) is set, the original dividend was negative. The complemented remainder, which is negative because the corrected remainder is positive (if all underflow conditions are corrected), is stored as the final value of the remainder. If both the dividend and the divisor were positive, the quotient, which is also positive (the most significant bit of the quotient must be positive or an immediate overflow condition aborts the division), is written into the appropriate general register.

Similarly, if both dividend and divisor are negative, the quotient should be positive and is written in its present form. If the original signs of the dividend and divisor were different, the quotient should be negative. One special case in which the quotient is the most negative number is considered an error.

3.1.5.3 Arithmetic Shift – As shown in Figure 3-3, as the flow is entered at ASH, the Shift Counter is loaded with the number of bit positions to be moved (if any). A test is then made to determine the direction of shift, or whether no shift will occur at all.

If no shift is called for (no count set in the Shift Counter), the local condition codes are set and transferred to the CPU Status Register as the flow concludes.

If a shift left is called for, the operand is shifted left one bit position, stored and the count incremented. This repeats until the shift count is exhausted, at which time the local condition codes are set and transferred.

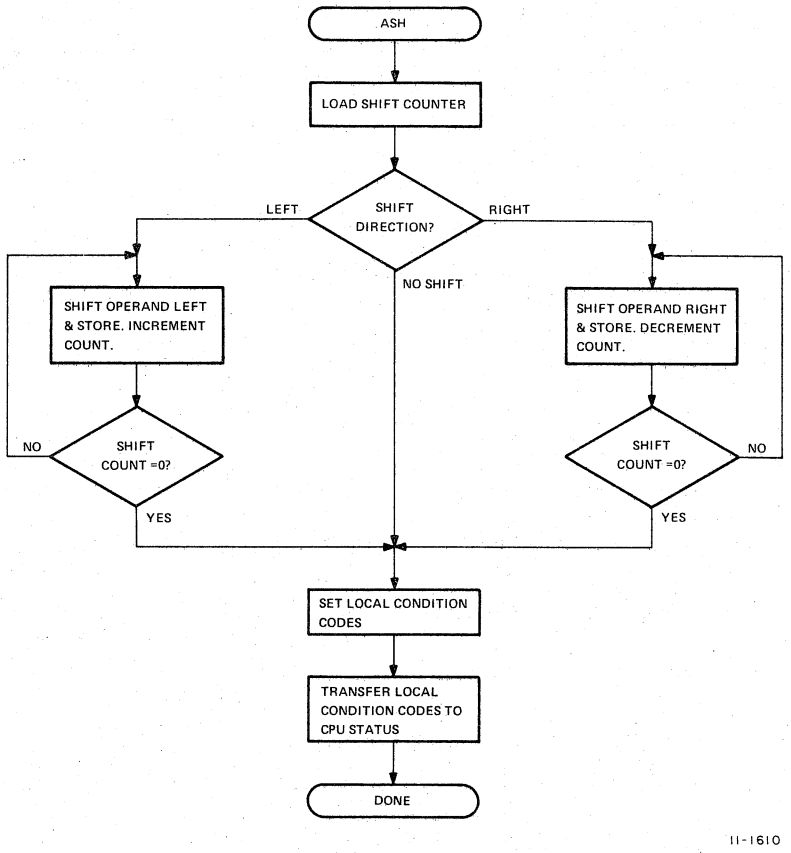


Figure 3-3 KE11-E ASH Algorithm

If a right shift is called for, the opposite occurs with the shift count being decremented on each pass through the loop.

In the KE11-E hardware, the operand to be shifted is in R(SF) and is sent first to the D Register and from there to both the BR and to a precleared B Register. The contents of BR (05:00) determines the direction of shift. If these contents are greater than 0, the shift will be to the left. If they are less than 0, a right shift is called for. If they are 0, no shifting will occur. Setting of condition codes for transferral is implemented identically to MUL and DIV.

3.1.5.4 Arithmetic Shift Combined – As shown in Figure 3-4, as the flow is entered at ASHC it can be seen that the flow is similar to the ASH flow except that in this case the hardware is dealing with 32 bits instead of 16. The Shift Counter is loaded from the least significant 6 bits of the source operand, and the low shift operand is sent to the shift register.

As in ASH, the value of the shift count with respect to 0 is tested to determine right or left shift. If a right shift is called for, the operand is shifted right one position and the partial high answer is stored while the count is decremented for each pass through the shift loop until the shift count is exhausted. For the left shift condition, the count is incremented; for the right shift, it is decremented for each pass through the loop.

When the shift count is exhausted for either loop, the low answer is stored (the high answer is already stored), the condition codes are set, transferred to the CPU Status, and the flow is terminated.

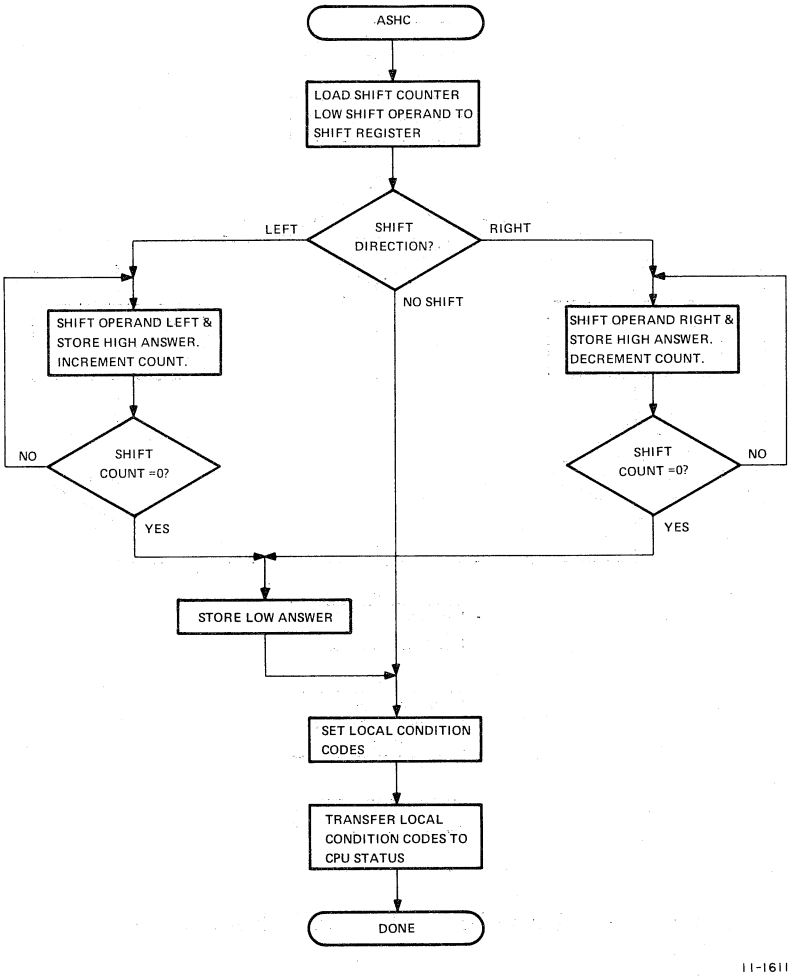


Figure 3-4 KE11-E ASHC Algorithm

In the KE11-E hardware, the contents of the register designated by the source field (R(SF)) and that register ORed with one (R(SFV1)) are loaded into the BR and DR Registers concatenated such that the high operand is in the BR and the low operand is in the DR. These are then shifted until the count is exhausted, at which time the low answer is stored in the odd register and the high answer is already stored in R(SF) by the shifting action within the loop.

Setting condition codes and transferral is identical to ASH.

3.2 KE11-F FLOATING INSTRUCTION SET

The KE11-F Option is used for floating-point operations in the KD11-A Centeral Processor. The principles involved in these instructions are given in the following paragraphs.

3.2.1 Polish Mode

In the KE11-F, floating operations take place on the top of a hardware stack by what is termed *Polish Accumulator Technique.* This technique is based on a form of mathematical notation developed by the Polish logician Lukasiewicz. The procedure allows complex logical expressions to be stated in a nonambiguous manner without the

necessity of relying on hierarchical delimiters such as parentheses. Its use in the KE11-F greatly simplifies scanning mechanisms. By determining interrelationships between operands in various mathematical operations, Lukasiewicz's technique allowed rearrangement of terms so that ordering of operands on the stack minimized the number of operations required to achieve the desired mathematical result. Operations could then be performed in sequence with intermediate results being stored automatically by the stack, then used in the next sequential operation until calculation was complete. Without this scheme, each equation term would have to be calculated separately, stored, then called for use in sequential steps.

Polish notation permits the writing of algebraic or logical expressions in a manner that eliminates the need for grouping symbols and conventions as to operator precedence.

In the expression X(Y+Z), the parentheses are necessary so that the reader (or interpreting device) can understand the grouping intended, and the precedence of operations to be performed.

Certain operators have precedence over others. For example in the expression X+Y/Z, the divide operator (/) is understood to have higher precedence than the add operator (+) so that it is understood that the operation is to be interpreted as X+(Y/Z) rather than (X+Y)/Z.

In right-hand Polish notation, the operators are located to the right of the operands. For example, the expression X+Y is written XY+. Similarly the expression X+Y+Z can be written either as XY+Z+ or XYZ++. In this latter example, the first add operator adds Y and Z while the second adds X to that sum.

In Polish notation, the following juxtapositions can be made to basic logical expressions:

Basic Expression	Polish Notation
X(Y+Z)	Either YZ+X×, or XYZ+X
X+Y/Z	Either YZ/X+, or XYZ/+
Q=X(Y-Z)/(R+S)	XYZ-×RS+/Q=

The last listing is an example of how Polish notation can be used to reorder many variables.

3.2.2 Floating-Point Arithmetic

Floating-point representation of a binary number consists of three parts; a sign bit, an exponent, and a mantissa. The mantissa is a fraction in magnitude format with the binary point positioned to the left of the most significant bit of the mantissa. All mantissas are assumed to be normalized; therefore, all leading 0s are eliminated from the binary representation. The most significant bit is thus a logical 1. Since all mantissas are assumed to be normalized, the MSB, which will always be a 1, is not stored because it is redundant. Leading 0s are removed by shifting the mantissa left; however, each left shift of the mantissa must be followed by a decrement of the exponent value to maintain the true value of the number. The exponent value represents the power of 2 by which the mantissa is multiplied to obtain the value to be used. Figure 3-5 shows an unnormalized number in floating-point notation and then the same number after it has been normalized.

3.2.2.1 Floating-Point Addition and Subtraction – For floating-point addition or subtraction operations, the exponents must be aligned or equal. If they are not aligned, the mantissa with the smaller exponent is shifted right until they are. Each shift to the right is accompanied by an incrementing of the exponent value. When the exponents are aligned or equal, the mantissas can be added or subtracted, whichever the case may be. The exponent value indicates the number of places the binary point is to be moved to obtain the actual representation of the number.

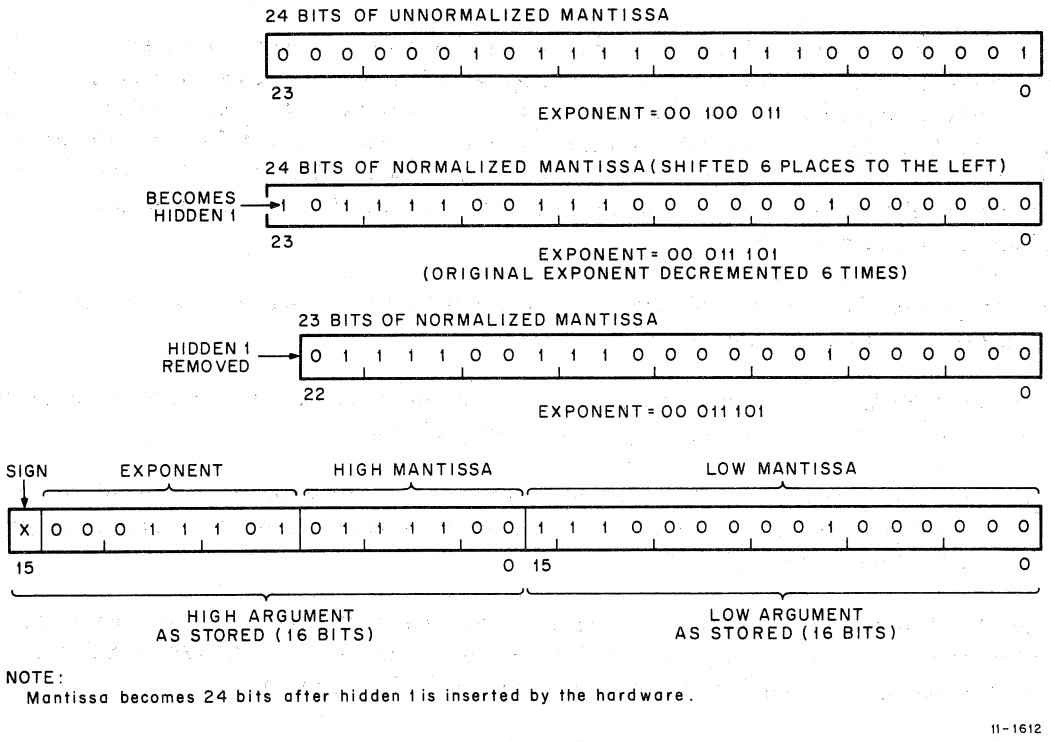


Figure 3-5 Floating-Point Representation

In the example below, the number 7_{10} is added to the number 40_{10}, using floating-point representation in binary-octal notation. Note that the exponents are first aligned and then the mantissas are added; the exponent value dictates the final location of the binary point.

$$\begin{array}{l} 40_{10} = 50_8 = 0.101\ 000 \times 2^6 \\ +\ 7_{10} = 7_8 = 0.111\ 000 \times 2^3 \end{array}$$

a. To align exponents, shift the mantissa with the smaller exponent three places to the right and increment the exponent by 3.

$$\begin{array}{l} 40_{10} = 50_8 = 0.101\ 000 \times 2^6 \\ +\ 7_{10} = 7_8 = 0.000\ 111 \times 2^6 \\ \hline 47_{10} = 57_8 = 0.101\ 111 \times 2^6 \end{array}$$

b. Move the binary point six places to the right.

$$\overbrace{0.\overbrace{101}^{5}\ \overbrace{111}^{7}.}$$

3.2.2.2 **Floating-Point Multiplication and Division** – In floating-point multiplication, the mantissas are multiplied and the exponents are added. For floating-point division, the mantissas are divided and the exponents are subtracted.

There is no requirement to align the binary point in the floating-point multiplication or division.

In the following example, the number 7_{10} is multiplied by the number 5_{10}. An 8-bit register is assumed for simplicity.

$$
\begin{array}{r}
7_{10} = 7_8 = 0.1\ 110\ 000 \times 2^3 \\
\times 5_{10} = 5_8 = 0.1\ 010\ 000 \times 2^3 \\
\hline
00000000 \\
1110000 \\
0 \\
1110000 \\
\hline
0.10001100000000 \times 2^6
\end{array}
$$

Move the binary point six places to the right.

$35_{10} = 43_8 = 0.100011,00000000$

3.2.3 Algorithms for KE11-F Operations

Figures 3-6 through 3-10 illustrate the sequences of operation for the floating-point operations. Note that to complete its function, the KE11-F hardware utilizes much of the KE11-E hardware.

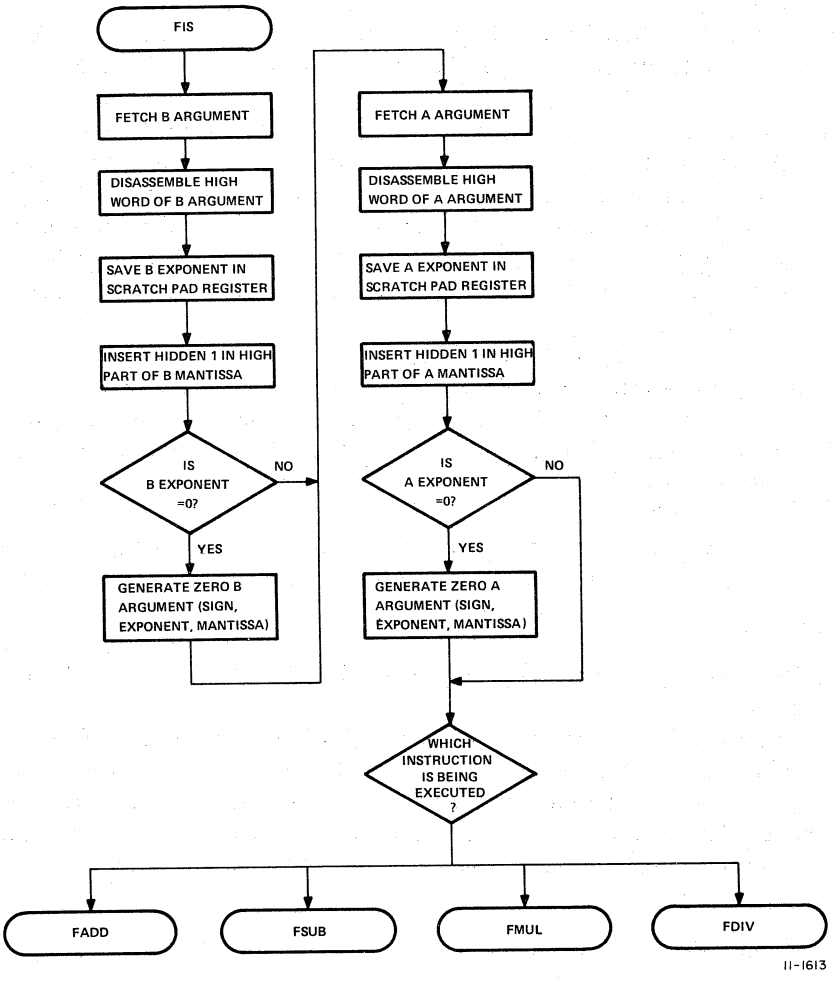


Figure 3-6 Floating Entry Algorithm

When a floating instruction is executed, the flow is entered at FIS (Figure 3-6). The B argument is fetched from core, high B first and then low B. The high B argument consisting of sign, exponent, and high mantissa is then disassembled separating the sign, exponent and mantissa. The exponent is saved in the low byte of one of the Scratch Pad Registers.

As previously stated, all mantissas are assumed to be normalized, meaning that the most significant bit of the mantissa is always a 1. Therefore, the MSB (referred to as the hidden 1) of the mantissa is not stored in core. This hidden 1 is now reinserted into the mantissa.

Next, a test to determine if the B exponent is equal to 0 is made. If it is 0, a clean 0 is generated as the B argument (sign, exponent, and mantissa).

The A argument is now fetched from core and the same procedure described above is followed. A test is now made to determine which floating instruction is to be executed.

3.2.3.1 Floating-Add and Floating-Subtract – As shown in Figure 3-7, the floating subtract flow enters at FSUB and immediately complements the sign of the subtrahend. From that point on, it proceeds as in floating-add. This is mathematically valid since adding the negative of a number is identical to subtracting it.

The floating-add flow is entered at FADD. A test is then made to see first if the B Addend is negative and then if the A Addend is negative. If either or both are, the 2's complement is taken of either or both.

The A exponent is subtracted from the B exponent and a test is made to see if the B exponent is equal to or greater than the A exponent. If it is not, then the A and B Addends and exponents must swap positions. The swap is made so that the mantissa with the smaller exponent is in position to be shifted later to align the binary points. Since there are only 30_8 bits of mantissa, an attempt to align binary points by shifting the smaller mantissa right more than 30_8 places would result in that mantissa being lost. Because of this, another test is made to see if the exponents are in range (difference $\leqslant 30_8$). If they are not in range, the argument with the larger exponent is taken as the answer.

If the exponents *are* in range (difference $\leqslant 30_8$), a check is made to see if they are equal to each other. If so, A Addend is added to B Addend next. If the exponents are unequal, then the Addend with the smaller exponent is shifted right. This corresponds mathematically to lining up the decimal (binary) points. The larger exponent then becomes the initial exponent of the answer. The term "initial" exponent is used because normalization of the answer has not as yet taken place.

At this point, the A Addend is added to the B Addend. The sign of the answer is checked, and the answer is complemented if the sign is negative. The answer is then normalized, rounded, and stored, as described in Paragraph 3.2.3.

As stated before, floating subtract is implemented by changing sign of the subtrahend and adding.

It should be noted that there are two extra bit positions on the low end of the mantissa for rounding purposes. These bits hold the last two bits shifted out of the mantissa when aligning binary points. One of these bits is dropped before going to the normalize round and store flow. It will be seen in Paragraph 3.2.3.4 that the rounding bit is added to the remaining extra bit. The second extra bit was maintained in case the answer was negative. When the answer was complemented, the lower extra bit could affect the upper extra bit.

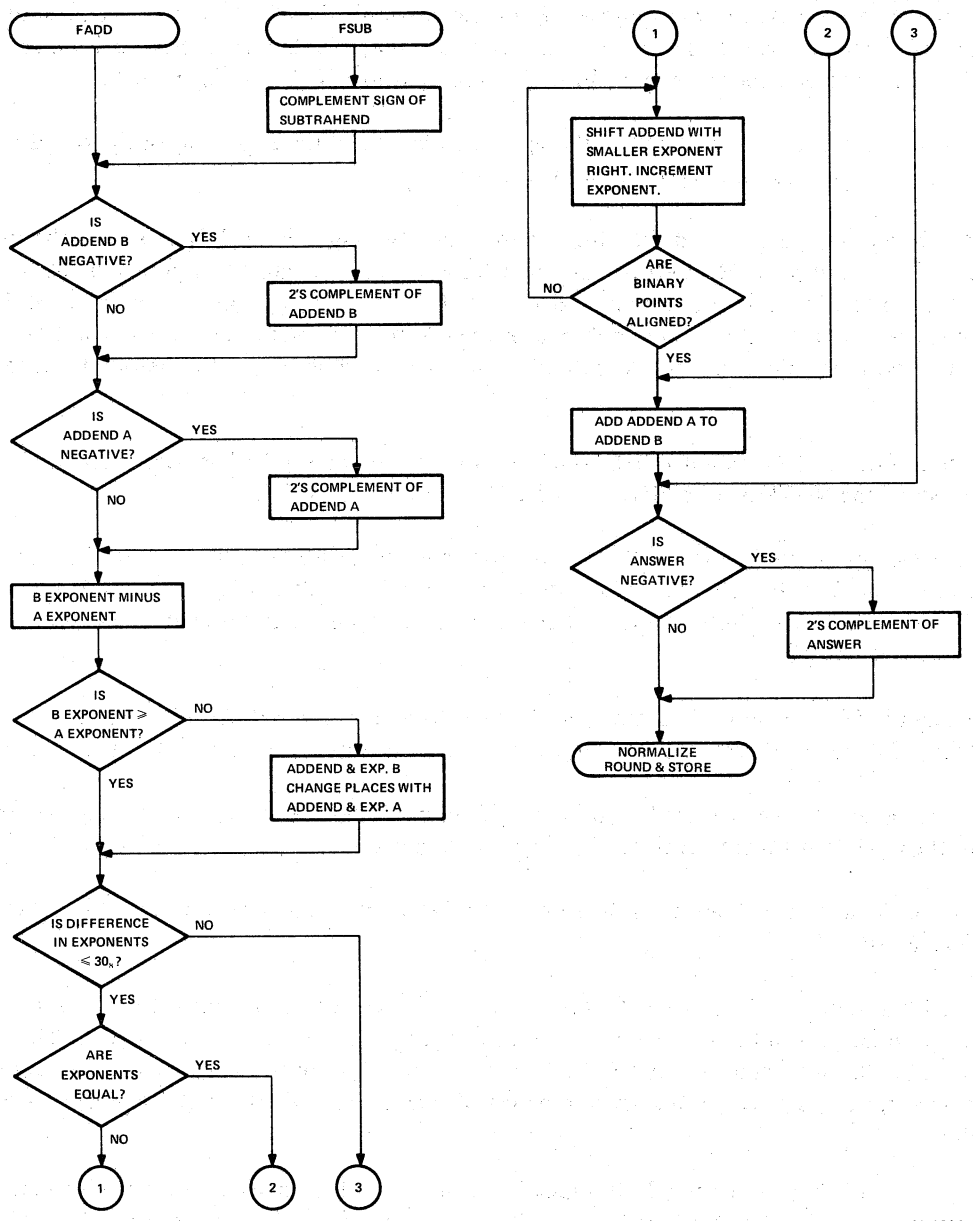


Figure 3-7 KE11-F FADD and FSUB Algorithm

For example:

	Mantissa	Extra Bits
Negative Answer (uncomplemented)	0 1 1 1	0 1
(complemented)	1 0 0 0	1 1

Rounding bit added to this position.

It can be seen from the above example that the second extra bit can have an effect on the mantissa when rounding takes place, i.e., cause a carry into the mantissa.

3.2.3.2 **Floating-Multiply** – As shown in Figure 3-8, the flow is entered at FMUL and a check is made to determine if either argument is equal to 0. If so, there is no need to go any farther with the operation and a 0 answer is generated. The local condition codes are set and the flow proceeds to STORE on Figure 3-10.

If neither argument is equal to 0, the XOR of the sign bits is saved and the exponents of the A and B arguments are added to produce the unnormalized exponent of the answer.

At this point, the step counter is loaded with 30_8 and the multiply loop is entered. In this loop, the state of the least significant bit of the multiplier (MSR00) is monitored. In any pass through the loop, if this bit is a (0), both the multiplier and the partial product are shifted right one position and the step count is decremented. If, however, on any pass through the loop the LSB of the multiplier is seen to be a (1), both the multiplier and partial product are shifted right one position and then the multiplicand and partial product are added before decrementing the Step Counter.

This process continues until the count is exhausted (each bit of the multiplier has been monitored), at which time the flow proceeds to the NORMALIZE, ROUND & STORE flow in Figure 3-10.

It can be seen from the above description that multiplication of the mantissas is identical to the way in which it was taught in Grade School. The multiplicand is added to the partial product each time a 1 is encountered in the multiplier, followed by a right shift of the partial product. For each 0 encountered in the multiplier, the partial product is simply shifted right.

3.2.3.3 **Floating-Divide** – As shown in Figure 3-9, the flow is entered at FDIV. At this point, a test is made to see if either the divisor or dividend are equal to 0. If they are, there is no reason to continue the computation.

If the divisor is equal to 0 (divide by 0), the local condition codes are set to indicate an underflow, then are transferred to the CPU Status Word. At this point the flow terminates.

If the dividend is equal to 0, a 0 answer is generated, the local condition codes are set, and the flow proceeds to STORE on Figure 3-10.

If, however, neither argument is equal to 0, the XOR of the signs of the arguments is saved, the exponents are subtracted to produce the initial exponent of the answer, and the Step Counter is loaded. The divisor is then subtracted from the dividend and the carry-out of the ALU is saved. Both the dividend and quotient are shifted left one bit position as the saved carry-out is shifted into MSR00, and as the LSB of the quotient and the Step Counter is decremented.

At this point the divide loop is entered. In this loop, the state of the least significant bit of the quotient (MSR00) is monitored. In any pass through this loop, if this bit is a (1), the divisor is subtracted from the dividend and the carry-out of the ALU is saved. Both the dividend and quotient are shifted left, the carry-out of the quotient is sent to MSR00, and the Step Counter is decremented. If, however, upon entering this loop the LSB of the quotient (MSR00) is seen to be a (0), the divisor is added back into the dividend and the carry-out of the ALU is saved. Once again the dividend and quotient are shifted left, the saved carry-out of the ALU is sent to MSR00, and the Step Counter is decremented.

This process continues until the step count is exhausted, at which time the flow proceeds to the NORMALIZE, ROUND & STORE operation described in Paragraph 3.2.3.4.

Note that division of the mantissas is also identical to the method taught in Grade School. The divisor is subtracted from the high part of the dividend and, if the divisor was smaller than or equal to the portion of the dividend being subtracted from, a 1 is shifted into the answer. The quotient and dividend are then shifted left. If the divisor was larger than the dividend, a 0 is shifted into the quotient. Since the hardware cannot look ahead to determine how many places the dividend must be shifted left before the next subtraction will be successful, the divisor will be added back into the dividend until the dividend is larger than or equal to the divisor and then a 1 will be shifted into the answer.

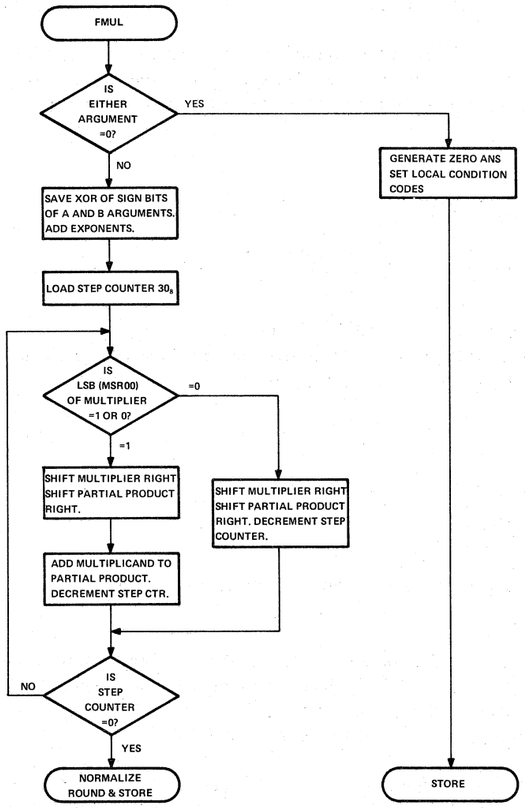


Figure 3-8 KE11-F FMUL Algorithm

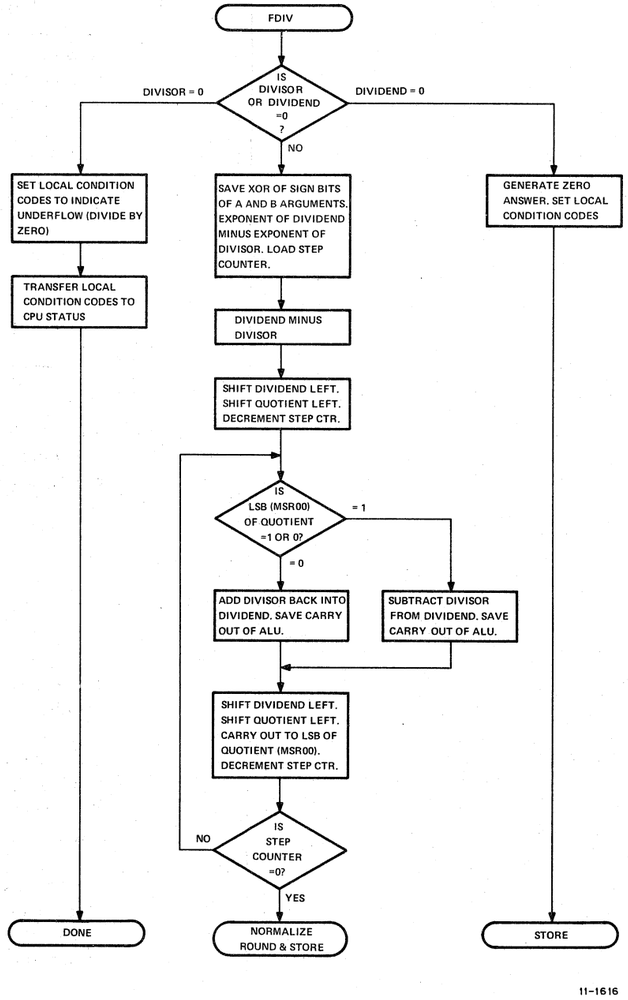


Figure 3-9 KE11-F FDIV Algorithm

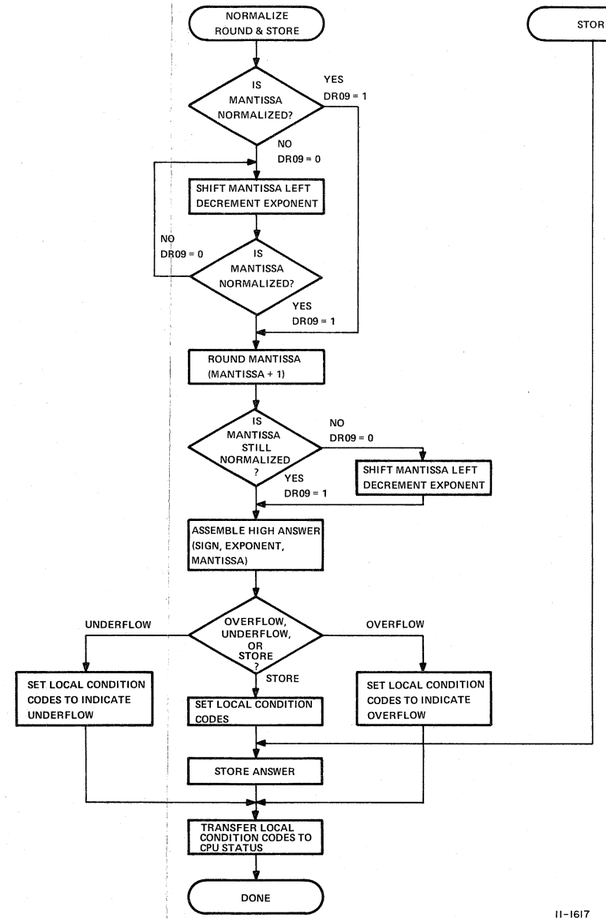


Figure 3-10 KE11-F Normalize, Round & Store Algorithm

3.2.3.4 Normalize, Round and Store – As shown in Figure 3-10, the flow can be entered either at NORMALIZE, ROUND & STORE, or at STORE. The STORE flow is entered from FDIV after determination that the dividend was equal to 0, or from FMUL after it has been determined that one of the arguments was equal to 0. In these flows (FDIV & FMUL), local condition codes were set and upon entering this flow the answer is stored, the local condition codes are transferred to the CPU Status Word, and the flow terminates.

The NORMALIZE, ROUND & STORE entry is made from all other flows if no unusual conditions are detected in the process. The first test is made to see if the mantissa is normalized. This is indicated by the state of DR09, the MSB of the mantissa. If this bit is not set, the mantissa is not normalized and, as a result, is shifted left one bit position while decrementing the exponent. This is formed into a loop until DR09 becomes set, at which time the mantissa is normalized and ready to be rounded.

As noted earlier, an extra bit position is carried on the low end of the mantissa. This bit is the position the rounding bit is added to. If the extra bit is a 1, then there will be a carry into the least significant bit of the mantissa, thus increasing the absolute value of the number. If the extra bit is a 0, there will be no carry-in.

For example:

Mantissa	Extra Bit	
1 0 0 1 0 0 1	1	
+	1	Rounding Bit
1 0 0 1 0 1 0	0	

The extra bit position is dropped before the mantissa is stored. After rounding, a test is made to see if the mantissa is still normalized. It is possible for the mantissa to become unnormalized as a result of rounding, i.e., carries could propagate all the way through and beyond the most significant bit of the mantissa.

For example:

1 1 1 1 1 1 1 1	1
+	1
10 0 0 0 0 0 0 0	0

If this happens, the mantissa must be shifted right one place and the exponent incremented to renormalize the mantissa.

The high answer (comprising the sign, the exponent, and the mantissa) is now assembled. Remember that the mantissa is always assumed to be normalized, which means that the most significant bit is always going to a (1). Because of this, there is no point in storing the most significant bit in core. This (1) in the MSB of the mantissa is termed the hidden 1 and is dropped when the high answer is assembled.

Next a check for either overflow or underflow is made. If underflow is indicated, the local condition codes are set appropriately, transferred to the PSW, and the flow terminates. Likewise if overflow is indicated, the same action occurs. If, however, neither is indicated, the local condition codes are set and the answer is stored in core. The flow terminates after transfer of condition codes to the CPU Status Word.

CHAPTER 4
LOGIC DESCRIPTION

4.1 SCOPE

This chapter describes the hardware associated with both the KE11-E and KE11-F. Because of their interdependence, these two options are not separated in this chapter as they are in other chapters but rather are described as one entity. The options are described at both a block diagram level and a logic level. In addition, the philosophy behind ROM programming is discussed together with a guide to reading the flows. Where necessary, interaction on a flow level with the KD11-A is also given. For convenience, logic descriptions are ordered as to their appearance in the Drawing Set.

4.2 FUNCTIONAL BLOCK DIAGRAM DISCUSSION

Figure 4-1 is a functional block diagram of the KE11-E and KE11-F showing the interconnections with the KD11-A Central Processor. Both options are shown. The dotted line separates the EIS on the left and the FIS on the right.

The KE11-E comprises one 16-bit input register (BR); a holding register that receives data from the KD11-A; a 16-bit left/right shift register (DR); an 8-bit up/down counter that receives data from the input register; a local condition code register that records the status of the KE11-E operations; a dual 4:1 multiplexer (RDMUX) that channels data via BUS RD drivers from the two KE11-E registers and status to the KD11-A; and a 256-word by 68-bit ROM that is used to control both the KD11-A and KE11-E during the execution of the EIS instruction. The two 16-bit registers (BR and DR) are simply an expansion of the basic KD11-A data path.

The KE11-F comprises all KE11-E hardware plus two 16-bit left/right shift registers (HSR and MSR) that function also as holding registers; a dual 4:1 multiplexer (FRD MUX) that assembles data for channeling from the KE11-F registers to the KD11-A via separate bus drivers; a constants generator that creates the offsets required in FIS computations; and a 256-word by 8-bit ROM used together with the EIS ROM to control both the KD11-A and KE11-F during the execution of an FIS instruction.

The input to the BR Register (DMUX(15:00)) is one of the buses in the KD11-A. All data to the KE11-E or KE11-F options is received over this bus. The BR Register is similar to the B Register in the processor in that it is clocked by the P1 and P3 timing pulses from the basic machine. Normally, without the KE11-E/F installed, data in the KD11-A might be moved from the scratch pad, over the BUS RD through the buffer and the ALU. From the ALU, it would move to the D Register, onto the DMUX, and into the B Register (this can be followed by referring to drawing KD11-A-BD). With the KE11-E/F installed, however, if one of the EIS or FIS instructions were issued, the data on the DMUX might not enter the B Register but might continue on and into the BR Register in the EIS option. The BR is merely a holding register and every register within the option is loaded from it.

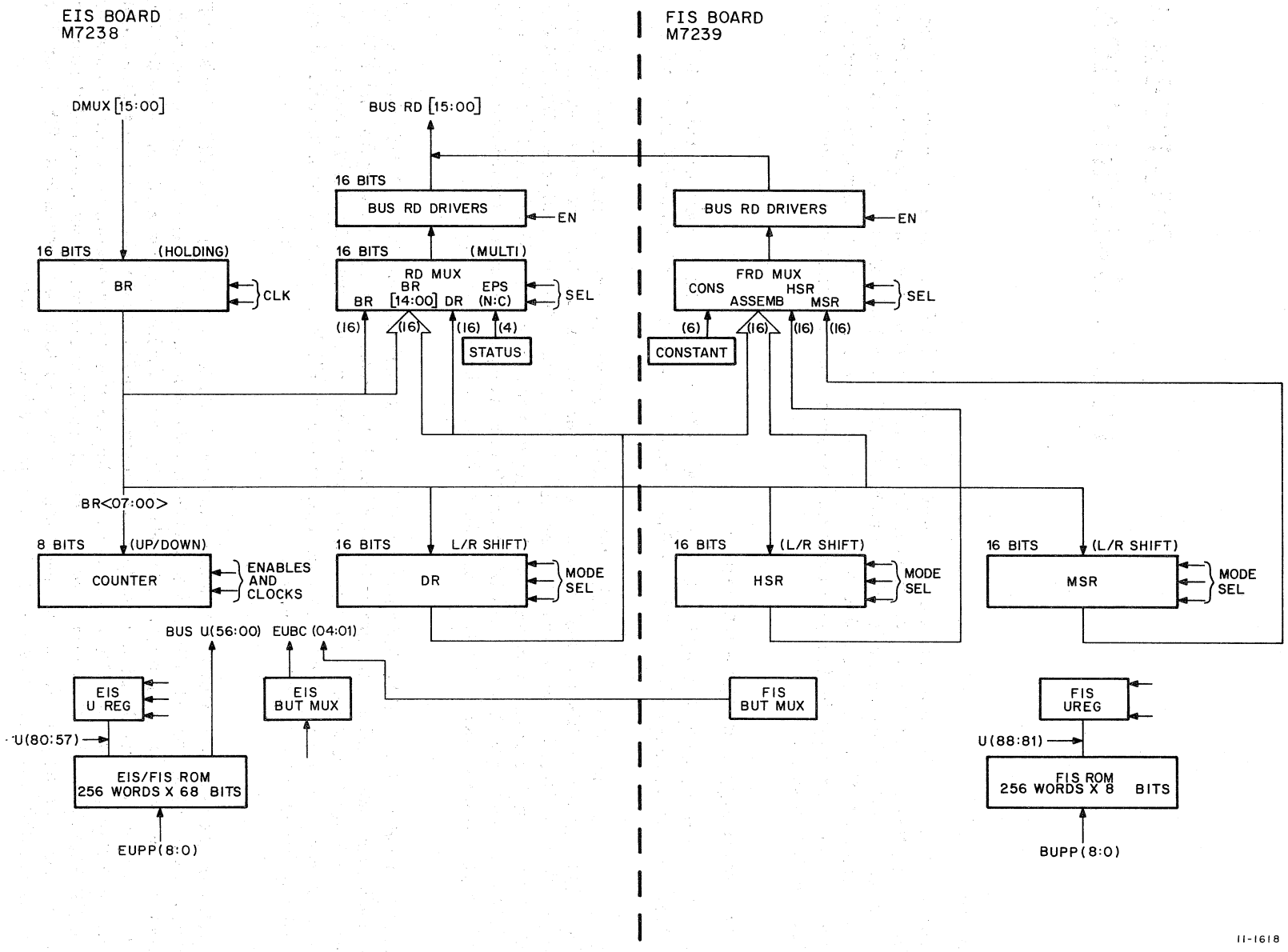


Figure 4-1 EIS/FIS Functional Block Diagram

The BR feeds the DR, the RD MUX concatenated with DR15, and the RD MUX straight through. It feeds the counter with bits (07:00) to keep track of the number of steps performed. When the FIS is installed, the BR is used to feed the "assemble" input of the FRD MUX and to load both the HSR and MSR Registers.

NOTE

For the EIS option, only bits (05:00) are required for the counter even though bits (07:00) are loaded. Bits (07:00) are required for the FIS option.

In multiply and divide operations, the counter keeps track of how many steps have been executed in the various loops. In arithmetic shift and arithmetic shift combined instructions, it holds the count or number of shifts that are to be made.

The DR Register is clocked by P1 + P2 and is fed directly from the BR Register. The mode selected determines whether it will shift right, shift left, or simply be loaded. At times it is used concatenated with the BR to hold the lower portion of the operand being shifted while the upper portion is shifted in the DMUX of the basic machine. The DR can feed either the RD MUX (EIS) or the FRD MUX (FIS), depending upon the mode of operation. The DR Register feeds the BUS RD via the RD MUX for transmission back to the basic machine.

The RD MUX is a multiplexer with four input ports. From right to left, depending upon the combination present at its select input, it is fed with the following:

1. Local status (EPS(N:C)). This input records the condition codes of the instruction as to whether it is negative or equal to 0, or whether there is any carry bit, or if there is overflow. This information is assembled here and transferred back to the basic machine as the last event when an instruction is complete. From here, it is loaded into the basic machine's Status Register.

2. The entire 16 bits of the DR Register are put on the RD BUS and fed either through the ALU or into the general registers, or any place in the basic machine accessible from BUS RD.

3. The BR shifted left one place concatenated with DR15 (BR(14:00),DR15). As in the case of Arithmetic Shift Combined, the BR and DR are concatenated and shifted left through that port with the DR being the lower register.

4. The entire 16 bits of the BR Register where it is fed, with no shifting, onto the BUS RD.

The EIS ROM word is physically 68 bits wide (i.e., the ROM bits that are actively being used), but the KD11-A ROM contains 56 bits while the KE11-E ROM has 24 bits of its own. Since the KE11-E must control the KD11-A data path, most of the 56 ROM bits in the basic machine must be duplicated. Not all need be duplicated, however, just those that are to be used by the option. The others are driven low by hardware in the option, thereby effectively loading 0s into those positions in the U Register. As a result the KE11-E effectively sends 56 ROM bits back to the basic machine, not all of which are active, and generates 24 bits of its own which it feeds to its own U Register. These bits are used to control the EIS data path, to clock the BR and DR Registers, to control which way the DR Register will shift, to load the counter, and to cause the counter to count. They are used further to control what port on the RD MUX is active, and to clock the status bits (see ROM U word descriptions in Paragraph 4.6.2).

The option also contains a branch microtest multiplexer (BUT MUX) which is used for testing conditions that determine selectable changes in microprogram flow. This BUT MUX is similar to the one in the KD11-A and is duplicated here to control conditions peculiar to the option. Bits in the ROM are used to control inputs to the multiplexer which, in turn, checks for the true or false state of some testable condition. The result of that test then is used to alter the "next address" residing in the present ROM word UPF field. The output of the BUT MUX goes back to the basic machine to an OR gate in front of the UPP Register where 1s may be inserted in appropriate positions to alter that base address to as many different addresses as there are branches called for.

In the KE11-F, the HSR and MSR Registers are both fed by the BR when an FIS instruction is called for. Their mode of operation is selected similarly to the DR in the EIS option except that here three bits are used to control two registers. In operation, normally the HSR can be loaded at any time unless conditions require that both the HSR and MSR be loaded. In this event, the MSR is loaded before the HSR. An examination of the mode selection for the HSR and MSR on print KF-2 illustrates why the latter statement is true.

The FRD MUX is similar to the RD MUX in the EIS option. In this case, a constants generator is also multiplexed to the BUS RD. Select bits control the inputs from the HSR or MSR Registers, the constants generator, or the assemble input from BR and DR in which the high 7 bits of the mantissa are in the DR Register, the 8-bit exponent is held in the BR Register, and the sign is taken from the EPS(N) bit.

The BUS RD drivers (74H01s) are identical to those used in the EIS option but are enabled only when an FIS instruction is called for.

The FIS ROM is a further horizontal extension of the microinstruction word. It supplies the extra control bits required for floating-point operation. It should be noted that the ROMs on the EIS board are also used to execute the FIS instructions.

Additional control logic is provided in this option to allow branch control of bit 1 of the ROM address from this hardware rather than from the EIS option. Provision is also made to enable DATO operations on the bus so that answers may be stored back in core. This feature is not needed in the EIS option.

4.3 DETAILED BLOCK DIAGRAM DISCUSSION

The descriptions in this paragraph are intended to supplement those in Paragraph 4.2. The detailed block diagrams for the KE11-E and KE11-F are shown on drawings KE11-E-BD and KE11-F-BD, respectively. These block diagrams have been arranged such that the inputs and outputs match the outputs and inputs of the KD11-A block diagram in drawing KD11-A-BD. Note that each block contains the drawing number on which the logic may be found, e.g., the BR Register is found on drawing KE-2 of the EIS schematics.

As shown on KE11-E-BD, the option is fed from the processor DMUX output. This is sent directly to the BR Register through which all data to both options is fed.

The output of the BR Register feeds the RD MUX and the DR Register. All 16 bits of the BR go to the DR while bits (07:00) go to the counter. As explained earlier, BR(14:00) can be shifted left one position and fed to the RD MUX with bit 15 of the DR sent to the low bit position. BR(15:00) is also sent off the page to the FIS block diagram, as is DR(15:00). The RD MUX is selected by combinations of SRDM1 and SRDM0, two bits in the EIS extension of the ROM word. Its output feeds a 74H01 driver which is in turn enabled by STRDM(1)H, the latter being the ERD field in the extension ROM word. When asserted, this data is enabled out onto BUS RD from the RD MUX.

The COUNT, fed by DR(07:00), is used in both the KE11-E and KE11-F. The counter is loaded by LD COUNT L and is clocked by CLK COUNT H. It is used to test whether or not the count is equal to 0, thereby keeping track of where the operation is in a loop.

The EPS(N,Z,V,C) block is used to compile the local condition codes (EPS means External Processor Status) for transmission back to the Processor Status Word via the RD MUX at the conclusion of each instruction. An inhibit signal (KE-5 INH PS CLK (1) L) is also generated at this time that prevents clocking any other bits in the basic machine status. In the case of an aborted instruction, these bits are not transferred but rather are stored here for information after servicing the abort. This leaves the basic machine's condition codes untouched as further information in servicing the abort.

The AUXILIARY ROM CONTROL block consists of some combinational logic that looks at a general purpose code (GPC=2) and the decoding of the MUL and DIV instructions to feed ESALU(3:0) back to the processor. These are select bits sent to the basic machine's ALU via a multiplexer into which the ROM bits are also sent. When selected, this auxiliary combinational logic replaces ROM word control of the ALU, causing the add, subtract, or straight through operations to be controlled by special conditions during the multiply and divide instructions.

The box CLOCK ENABLE GATES is shown to indicate that P1, P2, and P3 from the basic machine are used to gate internal conditions when generating the clocking signals for the various registers in the option.

The EUBF MUX box is a multiplexer that looks at 5 EUBF bits in the extended ROM word. These bits serve a similar function to the EUB bits in the basic machine ROM word. They are used for microbranch testing within the option. When just the EIS is installed, only 4 bits are used with the 5th bit pulled up. When the FIS option is installed, all 5 bits are used. Signals EUBC(4:1) are sent back to the basic machine for branch control.

The box marked U WORD CONTROL ROM stands for the KE11-E U Word Control ROM, comprising 256 words X 80 bits. Its output feeds a KE11-E U WORD REGister, which is 24 bits wide, with the EIS ROM bits (bits 57 through 88). The lower 56 bits are sent back to the KD11-A U Word Register (K2) over BUS UxxL. This is actually 44 bits since in the KE11-E U Word some of the bits are used to drive two bits back to the basic machine. The BUS U is a wired-OR of the CPU ROM output. Eight bits (BUS U(07:00)) are sent back to the Microprogram Pointer Register in the processor for feeding the KD11-A U Word Control ROM.

There are eight non-duplicated ROM bits that are hardware driven. They may be considered as wire-ORed with the ROM outputs. Note that they are *not* ROM bits but rather open-collector gates.

The FIS block diagram on drawing KE11-F-BD contains all that was contained on the EIS block diagram plus the hardware representations for the FIS option. The EIS descriptions will not be repeated here.

As shown, BR(15:00) are used to feed both the HSR Register and the MSR Register. Both are clocked by E(P1+P2)H. This clock comes from the EIS board and is always present at both registers. It is not effective, however, until a register is selected by select bits generated in the FIS ROM Register. The same is true of the MSR Register except that an additional select bit is required to enable this register.

Both the HSR and MSR Registers feed the FRD MUX which is similar in operation to the RD MUX in the EIS. BUS RD is fed with either HSR(15:00) straight-through; with the MSR(15:00) straight-through; with the input from the 9-bit constants generator; or it will assemble the EPS(N) bit (which is the sign bit) with the 8-bit exponent field (BR(07:00)) and with the high 7 bits of the mantissa (DR(06:00)), all of which constitute the high answer from a floating operation.

The output from the FRD MUX is fed through an enabled driver similar to the drivers for the EIS option. In this case, a special enable is provided (STFRDM(1)H) that is called up for only floating instructions. The FUB MUX box is an 8:1 branch multiplexer that controls EUBC1 for the FIS operations. When enabled, it disables its counterpart in the EIS option and allows this multiplexer to control bit 1 of the ROM address.

The KE11-F U Word Control ROM (256 words X 8 bits) provides the additional extension of the microinstruction word for the FIS option. It interfaces the processor data path and control in similar fashion to the ROM in the EIS option. This ROM is addressed by another buffered version of the Microprogram Pointer (BUPP(8:07)) rather than the (EUPP(8:0)) used for the EIS ROM. This ROM feeds an 8-bit wide U Word Register similar to the register used in the EIS.

4.4 INTERFACE

The KE11-E Option (M7238 module) and the KE11-F Option (M7239 module) both interface the KD11-A Central Processor via module slots in the KD11-A (A–F2 for the KE11-E and A–D1 for the KE11-F). In addition three (3) "over-the-back" cables from the M7238 module (EIS) connect the 40-pin Berg connectors on the M7232 (U Word) module at location A–D3. These cables wire-OR the outputs of the "main" KD11-A ROM with the "auxiliary" KE11-E and/or KE11-F ROMs.

The KE11-E receives data from the KD11-A via DMUX(15:00)H. The KE11-F, in turn, receives data from the KE11-E. Data is returned from both options over the wire-ORed bus BUS RD(15:00)L.

When the KE11-E is installed, J1 on the processor module M7233 (IR DECODE) at location A–F5 must be removed. When the KE11-F is installed, jumpers W1, W2, and W3 on the EIS option must also be removed. For more information on installation, refer to Paragraph 5.1.

When either the KE11-E or KE11-F are in operation, the KD11-A ROMs are disabled and both the KD11-A and the options are controlled by the auxiliary ROMs. The processor fetches instructions from core and decodes them. If the instruction contains a reserved code, the KD11-A ROM address bit UPP8 is set when BUT(INSTR I) in the basic PDP-11/40 ROM flow is executed. The setting of UPP8 disables the ROMs in the U Word (M7232) module and enables the auxiliary ROMs on the EIS module (M7238).

The KE11-E does another decode of the instruction, and if the instruction is in the EIS or FIS (if installed) group, it will branch to the specified address calculation. If the instruction does not fall within these groups, main ROM address UPP8 is cleared, thus disabling the option ROMs and re-enabling the KD11-A ROMs. The KD11-A will then execute a reserved instruction trap. This sequence of events can be followed by tracing words FET03, FET04, and FET05 on the KD11-A flows and word EI0 on the KE11-E flows. This flow sequence is described in more detail in Paragraphs 4.7.2 and 4.7.3.

The interfacing signals between the KE11-E/F and the KD11-A are shown in Figure 4-2 and listed with their definitions in Table 4-1.

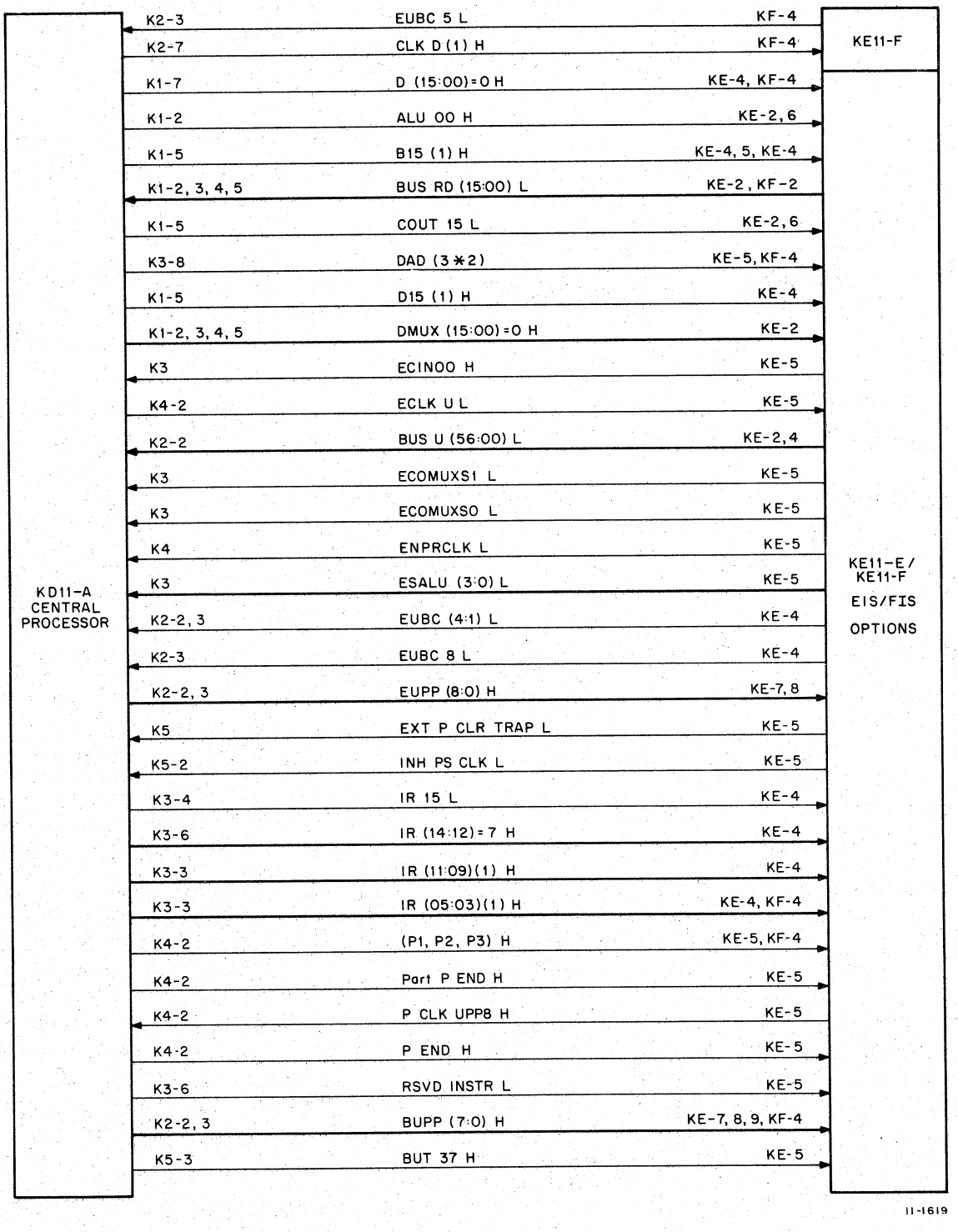


Figure 4-2 KE11-E/F/KD11-A Interfacing Signals

Table 4-1
KE11-E/F/KD11-A Interface

Signal	Definition
ALU00 H	Bit 0 output from the CPU ALU, used to shift into the BR Register.
B15 (1) H	Bit 15 from the CPU B Register.
BUS RD(15:00) L	Sixteen lines over which data is transferred from the KE11-E/F to the CPU. A wire-ORed bus.
COUT15 L	The carry-out from bit 15 of the CPU ALU.
DAD(3*2) L	Bits 3 and 2 of the CPU DAD code field. Allows option auxiliary control of the CPU ALU rather than direct (KD11-A) ROM control.
D15 (1) H	Bit 15 from the CPU D Register. Used in the branch table.
D(15:00)=0 H	A signal from the CPU which indicates when the CPU D Register is equal to 0. The resultant output from combinational logic (D(15:00)=0) is used on the branch BUT(D=0).
DMUX(15:00) H	Sixteen lines from the CPU over which data is transferred to the KE11-E BR Register. From here, it is sent to the KE11-F, when installed.
ECIN00 H	An external carry-in from the option to bit 0 of the CPU ALU. This is the signal line from which the jumper is removed on the M7233 module when the EIS is installed.
ECLK U L	A clock pulse from the CPU that is gated with an enable to generate the U Register clock for the EIS and FIS.
BUS U(56:00) L	Fifty-six ROM output lines that wire-OR the option ROM outputs with the KD11-A ROM outputs over three 40-pin Berg connectors on the back of the module. Note that the option ROM always controls the basic machine when activated. The basic machine ROM never controls the option.
ECOMUXS1 ECOMUXS0	"External Carry-Out Mux Select" –Two signals to the CPU that allow the option to control the carry-out multiplexer of the CPU data path.
ENPRCLK	"External NPR Clock" – A signal from the option that allows clocking of the CPU NPR flag and BR flag, and clears the CPU BBSY flag so that NPRs may occur during the EIS and FIS instructions.
ESALU(3:0) L	"External Select ALU" – Four signals to the CPU that allow the KE11-E auxiliary ALU control to specify what arithmetic function to perform. Used only in special situations such as loops during which external control is needed. Normally, the ALU is controlled by the CPU ROM word in which case the EIS feeds bits directly into the CPU U Register for ALU control.

Table 4-1 (Cont)
KE11-E/F/KD11-A Interface

Signal	Definition
EUBC(4:1) L	Four signals to the CPU that may modify the base ROM address when a branch test is executed. When an address is brought out of the ROM, i.e., 100, the 6 lower bits of this address would be 0s. The EIS would OR 1s into any of bits 04:01 to modify that base address. Note, bits 04:01 are the only bits available for modification by the EIS.
EUBC8 L	A signal to the CPU that is essentially the reserved instruction gated back to the CPU. When true, it is used as data, causing the ROM address bit UPP8 to be clocked set, thereby disabling the CPU ROM and enabling the KE11-E ROM. Whenever UPP8 is clocked after that, it clears, reversing the conditions.
EUPP(7:0) H	Eight signals to the option which specify the ROM address currently in the UPP Register. UPP8 controls which ROM is enabled while bits 7:0 control which address is in the ROM. In the CPU, addresses range from 0 to 377; in the auxiliary, they range from 400 to 777. This is because of bit 8. Address 400 is actually bit 0 inside the ROM, with bit 8 being the enable for the ROM.
EXT P CLR TRAP L	A signal to the CPU that allows the option to clear the CPU trap flag on a reserved instruction that the option has decoded as being either an EIS or FIS instruction.
INH PS CLK (1) L	A signal to the CPU that inhibits clocking of CPU status bits (07:04). This signal allows the modification of the N, Z, V, and C bits in that word by comparable bits in the option EPS Register but protects the priority bits already resident in the Processor Status Word. If the KT11-D Memory Management Unit is also installed, this signal also inhibits clocking of bits (15:12) in the CPU status as well. Permits clocking only of CPU status bits (03:00).
IR 15 L IR(14:12)=7 H IR(11:09)(1) H IR(05:03)(1) H	Selected bits and conditions of the IR Register that are used to decode whether or not the reserved instruction is really an EIS or FIS instruction. Bits 0, 1, and 2 are not essential for this decoding process.
(P1,P2,P3) H	Three clock pulses from the CPU that are gated with enabling signals from the ROM word U Register to generate the various clocking signals for the registers and flip-flops in the options.
Part P END H	A signal from the CPU generated as a function of the END pulse for cycle length 2 and cycle length 3. It is equal to P2 or P3.
P CLK UPP8 H	A signal from the option that clocks ROM address bit UPP8 in the CPU. This signal results from P END (described below) gated with CLOCK UPP8, bit 64 of the EIS ROM word. Once the option is active, by virtue of UPP8 having been set, asserting CLOCK UPP8 in the ROM will result in this signal which, in turn, will disable the option ROM and enable the CPU ROM.
P END H	A signal from the CPU that is equal to (P1+P2+P3). This is the end pulse in each cycle length. In a cycle length 1, it is P1; in a cycle length 2, it is P2; in a cycle length 3, it is P3, even though a P2 exists.

Table 4-1 (Cont)
KE11-E/F/KD11-A Interface

Signal	Definition
RSVD INSTR L	A signal to the option that indicates that the CPU has fetched a reserved instruction code and that the instruction may be an EIS or FIS. This signal is used as data gated with EUPP8 H to yield EUBC8 L described above.
BUT 37 H	A signal from the CPU which when gated with PART P END produces the signal P CLK UPP8 H.
EUBC5 L	A signal from FIS to CPU that is used to modify the base ROM address on branch tests.
CLK D (1) H	A signal from the CPU to FIS which is used to enable clocking the ARGA flip-flop.

4.5 ROM PROGRAMMING PHILOSOPHY

The PDP-11/40 System, and consequently the KE11-E and KE11-F Options, uses the principle of read-only-memory (ROM) microprogramming in their basic architecture. The use of this technique drastically reduces the requirements for discrete combinational logic and results in a system that is easier to understand and to maintain.

In hitherto conventional processor design, each control signal was the output of a combinational network that detected all the machine states and conditions for which the signal should be asserted. The machine state represented the contents of a number of storage elements (e.g., flip-flops) that had been loaded from signals that were, in turn, the outputs of other combinational networks. These outputs were based on such conditions as current state, sensed internal conditions, and sensed external conditions. Although many times the number of logical elements could be reduced by sharing outputs of networks, thereby reducing the size of the processor, this often increased the complexity of the machine and the difficulty in maintaining it.

In the PDP-11/40 System, however, the principle of microprogrammed control has been implemented in which the various control signals are stored in a self-contained ROM at time of manufacture. This storage is separate from the data storage element. Since each control signal can be completely defined if its value is known for each machine state, the ROM becomes the function generator divided into words. There is a word for each machine state and for each functional step of all operations. Each word contains a bit for every control signal. During each machine state, the contents of the corresponding word in the ROM are transmitted on the control lines. For most control signals, the output of the ROM *is* the control signal and no additional logic is required.

The two tasks of a sequence control section are to select the next machine state, and to provide information about the current machine state to the function generator. The only information that the function generator in a microprogrammed processor requires is which word to use as control signals. The sequence control then merely supplies an address that selects the correct word. The sequence control must also select the address of the *next* word to determine the machine state sequence.

Because the next machine state is determined in part by the current machine state, information is stored in the microprogram that aids in the selection of the next state. In a ROM programmed device, the microprogram word contains the control signal values and the address and sensing control information required by the microprogram address generation logic. Thus, this logic functions as the sequence control.

4.6 CONTROL ROM

The KE11-E control ROM word consists of 256 X 80 bit words. The KE11-F control ROM comprises 256 X 8 bit words. In the EIS, 24 ROM bits are used to control the KE11-E itself while 44 bits actively control 47 of the 56 bits in the CPU ROM word. The remaining 9 bits of the CPU ROM word are not utilized by the KE11-E or KE11-F and are driven low when the options are enabled. The outputs of the ROM and driver that duplicate the CPU ROM bits are wire-ORed to the outputs of the CPU ROMs.

The low eight bits of each ROM word specifies the next address of the microprogram. Occasionally there may be a desire to branch to one of several possible microroutines. Based upon certain conditions, the branch may be effected by executing a Branch Micro-Test (BUT) to test the desired condition. If a condition is met, the base address specified by the ROM will be modified by ORing a (1) into a (0) bit of the address and the microprogram will branch to the modified address. If the branch condition is not met, the next address will be the one specified by the control ROM.

Certain conditions can cause the microprogram to jam to a specific ROM address, thus aborting the normal microflow. The jam may be caused by a bus data timeout or an odd address error occurring on a bus data cycle. A red zone stack overflow error will also cause the jam.

4.6.1 KD11-A ROM Word

The KD11-A ROM microinstruction format is shown in Figure 4-3 and described in Table 4-2. Although much of this information is included in the *KD11-A Maintenance Manual,* it is repeated here for clarity and also to permit special reference as regards the KE11-E and KE11-F Options. Reference is also made to drawing KD11-A-BD, sheet 2 of 2 for more information and tabular data not included in Figure 4-3.

4.6.2 KE11-E/F ROM Word

The KE11-E/F ROM microinstruction format is shown in Figure 4-4 and described in Table 4-3. Note that this is an extension of the basic KD11-A format shown in Figure 4-3 and described in Table 4-2. The option ROMs duplicate all bits shown for the basic ROM in addition to generating these bits. Reference is also made to drawing KE11-E-BD, sheet 2 of 2 for more information and tabular data not included in Figure 4-4.

In both the figure and the drawing, the bits are represented identically to the representation for the processor. Note that in the top box of the drawing, the mnemonic for the field is given while the mnemonics below that are the bits in that field. For example, the field EUB contains bits EUBF(3:0). Their states represent an octal number appearing in the flow diagrams. Refer to word EI0 on the EIS flows, drawing KE11-E-F, sheet 1 of 5. Note that for a branch microtest of Extended INSTRuction I, EUB must equal 17_8.

A line printer printout showing all the octal values of each field of every ROM word in the flows has been made part of the print set. A portion of the first page of this printout is shown in Figures 4-5 and 4-6, with a legend to aid in their use. Arrangement is by U Word Address in numerical order. Note that the address is the basic address and does not include the 400_8 offset, e.g., in Figure 4-5, NOM14 on F6 lists the address as 140 but in the flow it is shown as 540. This holds for all addresses in this printout, as they represent only the states of ROM bits during an EIS or FIS instruction. Figure 4-6 gives the EIS/FIS ROM words while Figure 4-5 lists the KD11-A ROM words as generated by the EIS and FIS. Figure 4-6 is the extension left of Figure 4-5.

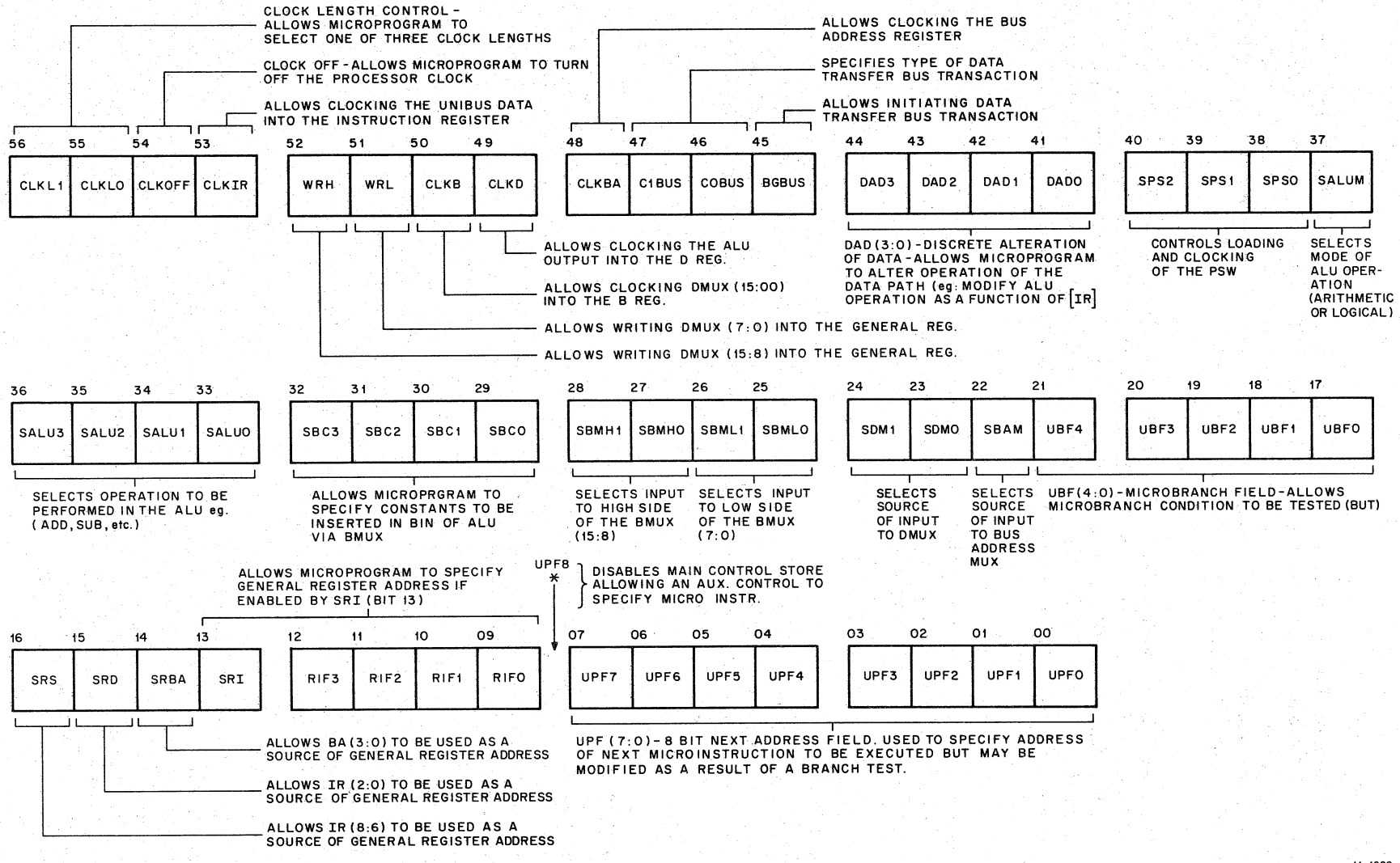


Figure 4-3 KD11-A ROM Format

Table 4-2
KD11-A ROM Word

Bit No. (Uxx(1)H)	Field Mnemonic	Bit Mnemonic	Definition
(07:00)	UPF	UPFx	Eight bits that yield the 256 basic locations. Note that they designate the *next* unmodified address of next microinstruction in the flow. Modified as a result of a branch test.
08		UPF8	Note, this is hardware, not in ROM. When clear, enables basic machine ROM. When set, disables basic machine ROM and enables the option ROMs. Controls which ROM will control the basic machine.
(12:09)	RIF	RIFx	Register Immediate Field. Provides the address of the internal registers. In conjunction with a bit in the SRX field, these 4 bits provide 16 possible addresses to select one of the general registers.
(16:13)	SRX	SRI	Select Register Immediate. When set, designates the RIF bits as the address of the scratch pad.
		SRBA	Allows BA(03:00) to be used as source of general register address.
		SRD	Select Register Destination. Uses the destination field of the instruction IR(02:00) to select the Scratch Pad Register in a destination address calculation.
		SRS	Select Register Source. Uses the source field of the instruction IR(08:06) to select the general register to be used.
			NOTE **The RIF field is 4 bits wide and the SRBA enables 4 bits in the BA. Therefore, these functions can access all 16 registers. The SRD and SRS bits, however, enable only 3 bits each; therefore, these functions can access only the lower 8 registers.**
(21:17)	UBF	UBFx	The microbranch field. Allows microbranch conditions to be tested so next address can be modified. This field is not used by the EIS or FIS. See EUB field in Table 4-3.
22	SBA	SBAM	Select BA Mux. Selects source of input to Bus Address Mux, either via BUS RD or via the ALU.
(24:23)	SDM	SDMx	Select DMUX. Selects DMUX input whether D Register straight through, the RD bus, D Register shifted right, or the Unibus. This function is used primarily in the basic machine.

Table 4-2 (Cont)
KD11-A ROM Word

Bit No. (Uxx(1)H)	Field Mnemonic	Bit Mnemonic	Definition
(28:25)	SBM	SBMHx and SBMLx	Selects input to high side of BMUX. Selects input to low side of BMUX. Controls the two 8-bit bytes of the BMUX independently. The BMUX can load the B Register straight through into the ALU, load a constant into the ALU, swap the two halves, or extend bit 7 of the B Register into the upper byte and load the lower byte with its sign extended into the upper byte. In addition, any combination of these can be used and are used by EIS and FIS operations.
(32:29)	SBC	SBCx	Select B Constant. Allows microprogram to specify one of 16 constants to be loaded into B IN of the ALU via the BMUX.
(37:33)	ALU	SALUM and SALUx	Select ALU mode (arithmetic or logical). Select ALU operation (add, subtract, OR, AND, etc). There are 16 operations for each mode.
(40:38)	SPS	SPSx	Select Processor Status. Controls loading and clocking of the PSW. Various combinations of these 3 bits perform separate operations on the PSW. See table on engineering drawing.
(44:41)	DAD	DADx	Discrete Alteration of Data. Allows microprogram to alter operation of the data path. For example, allows checking for stack overflow during a data cycle, or allows execution of an odd address, or control of the ALU by an auxiliary function rather than directly, etc.
(47:45)	BUS		Bus Control Bits.
		BGBUS	Begin Bus. When set, permits DATI, DATIP, DATO, or DATOB, depending upon setting of C1 BUS and C0 BUS. When cleared, sets AWBBY (Await Bus Busy) or restart on peripheral release, depending upon setting of C1 or C0 BUS.
		C1BUS and C0BUS	Unibus control bits which perform the standard PDP-11 functions in conjunction with BGBUS bit.
48	CBA	CLKBA	Clock Bus Address. Gated to clock the Bus Address.
49	CD	CLKD	Clock D Register. Allows clocking ALU into D.
50	CB	CLKB	Clock B Register. Allows clocking DMUX into B.
(52:51)	WR	WRL and WRH	Write enables for the general registers. 01 enables the low byte of the DMUX to be written, 10 enables the high byte to be written, and 11 enables both bytes to be written.

Table 4-2 (Cont)
KD11-A ROM Word

Bit No. (Uxx(1)H)	Field Mnemonic	Bit Mnemonic	Definition
53	CIR	CLKIR	Clock the Instruction Register. Allows Unibus data to be clocked into the IR.
(56:54)	CLK	CLKOFF	Clock Off. When asserted, allows microprogram to shut off processor clock.
		CLK1 and CLK0	Clock length control. 00 or 01 enables a cycle length 1. 10 enables a cycle length 2 and 11 a cycle length 3.

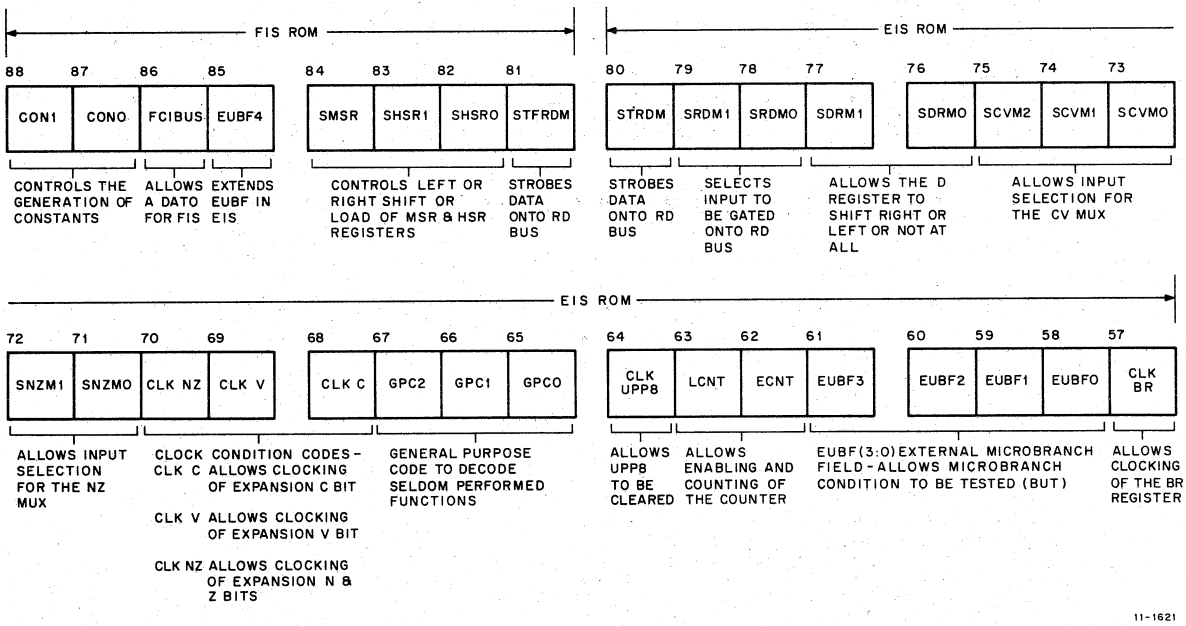


Figure 4-4 KE11-E/F ROM Format

FLOWS	STATE	ADR	CLK	CIR	WR	CB	CD	CBA	BUS	DAD	SPS	ALU	SBC	SBM	SDM	SBA	UBF	SRX	RIF	UPF
F6	NOM14	140	6	0	3	1	1	0	0	00	0	11	01	17	2	0	00	01	15	142
E3	MUL26	141	2	0	3	0	0	0	0	00	0	00	00	00	2	0	00	10	00	177
F6	EXI0	142	2	0	0	0	0	0	0	00	0	00	00	00	0	0	00	00	00	332
E3	MUL11	143	4	0	0	0	1	0	0	00	0	20	00	00	0	0	00	00	00	005
F1	FP12	144	2	0	3	0	1	0	0	00	0	00	00	00	2	0	00	01	11	244
E1	DST12	145	2	0	3	1	0	0	0	00	0	00	00	00	1	0	00	01	12	275
F1	FP10	146	6	0	3	1	1	0	0	00	0	23	00	00	2	0	00	01	13	252
E4	DIV14	147	2	0	0	0	0	0	0	00	0	00	00	00	0	0	00	10	00	062
F4	FML5	150	6	0	3	0	1	0	0	00	0	06	00	00	2	0	00	01	15	365
F1	FP4	151	6	0	3	0	1	0	0	00	0	11	00	00	2	0	00	01	13	135
		152	0	0	0	0	0	0	0	00	0	00	00	00	0	0	00	00	00	000
E4	DIV20	153	6	0	0	0	1	0	0	14	0	00	00	00	2	0	00	00	00	153
F6	NOM1	154	6	0	3	0	1	0	0	00	0	11	01	17	2	0	00	01	15	324
F1	FP6	155	6	0	3	1	1	0	0	00	0	11	00	00	2	0	00	01	11	175
F6	NOM8	156	6	0	0	0	1	0	0	00	0	11	01	17	2	0	00	00	00	174
E5	DIV21	157	4	0	0	0	1	0	0	00	0	00	00	00	0	0	00	00	00	107
F5	FDV6	160	4	0	0	0	1	0	0	00	0	11	00	00	0	0	00	00	00	344
F6	EXI4	161	6	0	0	1	0	0	0	00	0	00	00	00	0	0	00	00	00	256
F5	FDV5	162	6	0	3	1	1	0	0	00	0	23	00	00	2	0	00	01	15	176
F2	ADD13	163	4	0	0	0	1	0	0	10	0	06	00	00	0	0	00	00	00	340
F2	ADD5	164	2	0	0	1	0	0	0	00	0	00	00	00	0	0	00	01	11	165
F2	ADD6	165	4	0	0	0	1	0	0	00	0	06	00	00	0	0	00	00	00	202
F5	FDV3	166	4	0	0	0	1	0	0	00	0	00	00	00	0	0	00	00	00	350
E3	MUL20	167	2	0	3	0	0	0	0	00	0	00	00	00	2	0	00	10	00	143
E1	DST4	170	3	0	3	0	0	0	1	00	0	00	00	00	2	0	00	04	00	271
F5	FDV11	171	2	0	0	0	0	0	0	00	0	00	00	00	0	0	00	01	10	042
F4	FML8	172	2	0	0	0	0	0	0	00	0	00	00	00	0	0	00	01	10	210
F5	FDV13	173	6	0	0	0	1	0	0	14	0	00	00	00	2	0	00	00	00	331
F6	NOM9	174	2	0	0	0	0	0	0	00	0	00	00	00	0	0	00	00	00	273
F1	FP7	175	4	0	0	0	1	0	0	00	0	32	00	16	0	0	00	00	00	367
F6	NOM3	176	2	0	0	0	0	1	0	06	0	00	00	00	1	0	00	04	00	220
F2	ASH15	177	6	0	0	0	0	0	0	00	7	00	00	00	0	0	00	00	00	323

Figure 4-5 KD11-A ROM Words Generated by the Options (Sample)

FLOWS	STATE	ADR	CON	FC1	FUB	MHR	FRD	ERD	SRD	SDR	CVM	NZM	CCC	GPC	CEE	CNT	EUB	CBR
F6	NOM14	140	0	0	0	1	0	0	0	1	0	0	0	0	0	0	00	1
E3	MUL26	141	0	0	0	0	0	0	0	0	1	3	5	0	0	0	00	0
F6	EXI0	142	0	0	0	0	1	0	2	0	0	0	0	0	0	0	00	1
E3	MUL11	143	0	0	0	0	0	1	3	0	2	0	2	0	0	0	04	0
F1	FP12	144	0	0	0	0	0	0	1	0	0	0	0	0	0	0	00	0
E1	DST12	145	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	0
F1	FP10	146	0	0	1	0	0	0	0	0	0	0	0	0	0	0	00	1
E4	DIV14	147	0	0	0	0	0	0	0	3	0	0	0	0	0	0	00	1
F4	FML5	150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	0
F1	FP4	151	0	0	0	0	0	0	0	0	4	0	1	0	0	0	00	0
		152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	0
E4	DIV20	153	0	0	0	0	0	1	2	2	0	0	0	2	0	1	10	1
F6	NOM1	154	0	0	1	0	0	0	0	0	0	0	0	1	0	0	04	0
F1	FP6	155	0	0	0	0	0	0	0	0	0	0	0	4	0	0	10	0
F6	NOM8	156	0	0	0	0	1	0	1	0	4	0	1	0	0	0	00	1
E5	DIV21	157	0	0	0	0	0	1	3	0	0	0	0	0	0	0	12	0
F5	FDV6	160	0	0	0	0	1	0	3	0	0	0	0	6	0	0	00	0
F6	EXI4	161	2	0	0	0	1	0	3	0	1	3	6	0	0	0	00	0
F5	FDV5	162	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	1
F2	ADD13	163	0	0	0	0	0	0	0	0	0	2	4	0	0	2	00	0
F2	ADD5	164	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	0
F2	ADD6	165	0	0	0	0	0	0	0	0	0	0	0	4	0	0	00	0
F5	FDV3	166	1	0	0	0	1	0	3	0	1	0	1	0	0	0	00	0
E3	MUL20	167	0	0	0	0	0	0	3	0	0	0	0	0	0	0	05	0
E1	DST4	170	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0
F5	FDV11	171	0	0	0	7	0	0	0	0	0	0	0	0	0	0	00	1
F4	FML8	172	0	0	0	7	0	0	0	0	0	0	0	0	0	0	00	1
F5	FDV13	173	0	0	1	0	0	1	1	3	4	0	1	2	0	0	05	1
F6	NOM9	174	0	0	0	3	0	0	1	0	0	0	0	0	0	0	00	0
F1	FP7	175	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0
F6	NOM3	176	0	1	0	3	0	0	1	3	0	2	4	0	0	0	00	0
F2	ASH15	177	0	0	0	0	0	1	0	0	0	0	0	0	1	0	00	0

Figure 4-6 Comparable EIS/FIS ROM Words (Sample)

Table 4-3
KE11-E/F ROM Word

Bit No. (Uxx(1)H)	Field Mnemonic	Bit Mnemonic	Definition
57	CBR	CLK BR	Enabling signal for the BR Register clock.
(61:58)	EUB	EUBFx	External Microbranch Field – Allows extended microbranch condition to be tested.
(63:62)	CNT	LCNT and ECNT	Load Counter. Enable Counter – Allows loading and enabling of counter so that it may count up or down.
64	CEE	CLK UPP8	Clock Expansion Enable. Enables clocking of UPP8 to disable expansion ROM and enable basic ROM.
(67:65)	GPC	GPCx	General Purpose Code – Decodes seldom performed functions (see table on engineering drawing).
(70:68)	CCC	CLK NZ	Clock the N and Z bits in the External Processor Status Register. In the option, both are clocked simultaneously.
		CLK V	Clock the V bit in the EPS.
		CLK C	Clock the C bit in the EPS.
(72:71)	NZM	SNZMx	N Z Multiplexer Select – Control the source of data for these two EPS bits (N,Z).
(75:73)	CVM	SCVMx	C V Multiplexer Select – Control the source of data for these two EPS bits (C,V).
(77:76)	SDR	SDRMx	Select DR Register – Control whether the DR Register will load, shift right, shift left, or do nothing.
(79:78)	SRD	SRDMx	Select RD Multiplexer – Selects which source will be gated out onto the RD bus.
80	ERD	STRDM	Strobe RD Multiplexer – While the SRD determines what data will be put on BUS RD, this bit enables the drivers to that bus and actually gates the data to BUS RD.
	NOTE **The following signals are available only when the KE11-F is installed.**		
81	FRD	STFRDM	Strobe Floating RD Multiplexer – Performs the same function as the ERD does in the EIS option.

Table 4-3 (Cont)
KE11-E/F ROM Word

Bit No. (Uxx(1)H)	Field Mnemonic	Bit Mnemonic	Definition
(84:82)	MHR	SMSR and SHSRx	Select MSR Register – Is gated with the Shift HSR Register bits to enable the same function in the MSR Register as selected for the HSR Register. The x bits determine whether the HSR (and MSR when selected) will shift left, shift right, load, or no op.
85	FUB	EUBF4	Extension of EUB field in EIS, providing the additional branch tests for FIS option. When asserted, it disables the low bit multiplexer in the EIS and enables a similar multiplexer in the FIS to control the low bit of ROM address modification. There are six tests with this function, see table on engineering drawing.
86	FC1	FC1 BUS	This replaces the FC1 bit of the basic machine. Note that in the EIS no DATOs are required since nothing is stored back into core. This enables the FIS to do a DATO for writing answers back into core.
(88:87)	CON	CONx	Constants decoding bits. See table on engineering drawing.

4.7 FLOW DIAGRAM DISCUSSION

The flow diagram in conventional computer design has always played a major role in the understanding of the operation of the equipment. It is in a sense a road map of operation guiding the reader from one event to the next based upon sets of intervening conditions. In the PDP-11/40 System, however, the flow diagram plays a much more important role than before since it ties the operations to the major sequencing device in the machine, the ROM. Indeed, understanding the flow is a major prerequisite for the understanding of the KE11-E and KE11-F Options.

Because of the added responsibility intrinsic to this portion of the documentation, some changes have been made to the conventional flow diagram symbology to accommodate the added functions it serves. In this paragraph, these new conventions are discussed and explained. In addition, representative operations are followed through the flows so that, once familiar with the procedure, the reader can follow any operation through from its initiation to completion.

4.7.1 Symbology of the Flows

Figure 4-7 illustrates the common drawing conventions used in both the KD11-A and KE11-E/F flow diagrams. General flow is from top to bottom unless further continuation is required, in which case it is carried to the top of the page before continuing.

Horizontal flow is indicated and limited by the direction of the arrow on the line. Branching flow is dictated by the prevailing conditions that result from the branch test. The conditions for the branch are indicated to the right of each branch flow. For the most part, to the degree possible, the branches have a priority with the highest priority to the left and the lowest to the right. A double squiggley line in any flow line indicates that a time delay is experienced before continuing.

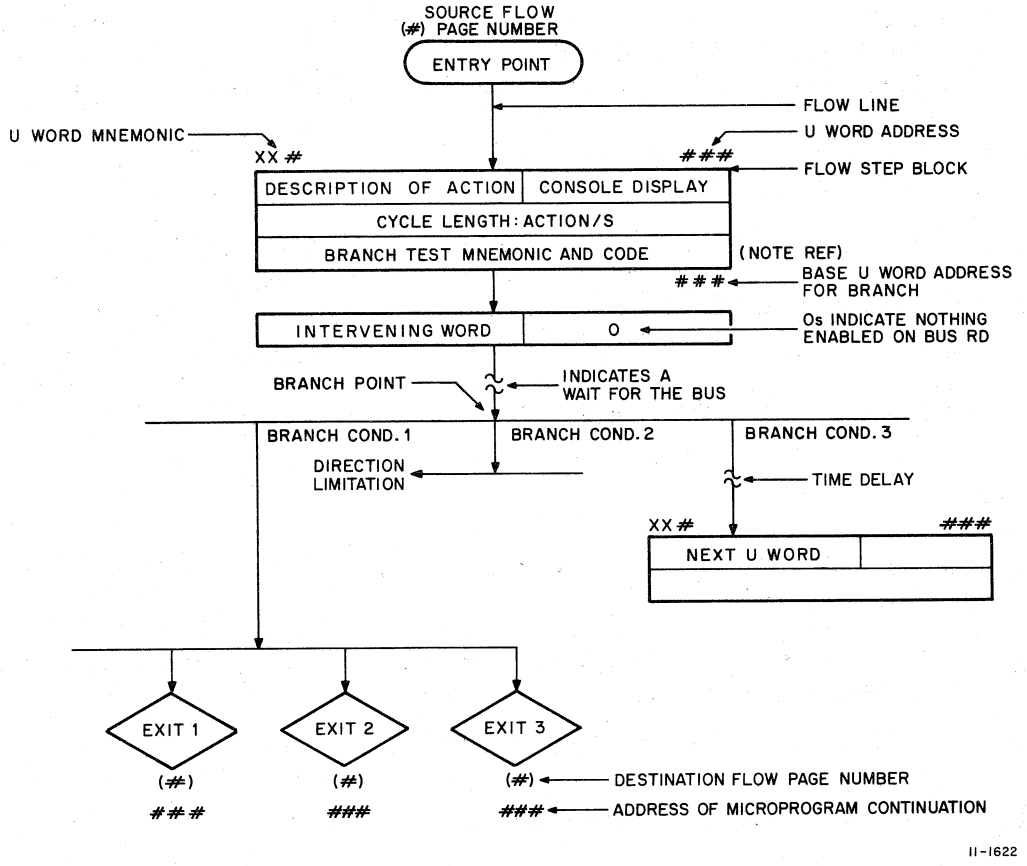


Figure 4-7 Flow Diagram Conventions

Entry into the flow is indicated by a lozenge (ellipse) containing the name of the operation to be performed in the flow. A decimal number in parentheses above this indicator refers to the number of the page from which the flow enters. In some instances, this is accompanied by a description of the flows if they are for a different piece of equipment than that described by the flow page.

Each step of the sequence is designated by the Flow Step Block, which contains either two or three divisions. The top division contains a general description of the action taken by the step, with a description of what is displayed on the console within a further subdivision of that area. This data is visible only in maintenance clock mode.

The second division of the Flow Step Block contains a description of the actions taken by that step in ISP notation. This is preceded by the cycle length for the action expressed by either P1, P2, or P3.

The third division of the block does not always appear. It is set aside to indicate that a branch test is to be made and expresses that test in mnemonic form. This block occurs in all cases two words prior to the branching point.

The octal number at the upper right-hand corner of the block indicates the Micro-Word Address of that word in the ROM.

The number at the lower right-hand corner of the block is given only when branch tests are made. It indicates the base ROM address for the branch before modification by the branching conditions.

Exit from the flow is indicated by a diamond containing the mnemonic used in the entry point of the destination. A decimal number within parentheses below the diamond indicates the page number of that continuation. An octal number appearing below this page number designates the address of the microprogram continuation.

References to notes on the flow sheet are either given within the general description subdivision of the Step Block or are located to the right of the block opposite the subdivision that they illuminate.

The symbology used in the operator division of each block follows the convention of ISP notation as defined in the Appendix of the *PDP-11/40 Processor Handbook,* 1972. To supplement this information, a few examples of the more complex statements are given and described here. From these sources, the reader can decipher any statement in the flows.

The use of the back arrow (←), or data transmission operator, is shown at word EI2 on page 1 of the EIS flow diagram.

UPP8←0

Basically, this means that the UPP8 bit in the basic machine is cleared. There are several ways of stating this such as "UPP8 gets 0" or "0 is sent to UPP8."

Brackets are used to further define a quantity in the statement. For example, at word ADD20 on page 2 of the FIS flows, the following statement is used:

P2:D←R[14]

This means that at time P2, the contents of register 14 are sent to the D Register.

The definition, however, may be indirect as shown in the following example at word DST2 on page 1 of the EIS flows:

P2:BA←R[DF]

In this case, DF means "Destination Field." This statement then is saying, "The contents of the register designated by the destination field of the IR is sent to the BA Register." The P2 preceding that statement means that the BA Register will be loaded upon the occurrence of P2. During the cycle length of that word, the data path is steering the data to the BA Register. Once it is set up, the pulse does the loading.

The bracket can be used on the left-hand side of the transmission operator. This is illustrated in word DST1 on page 1 of the EIS flows. The following statement appears:

P1:BR,B,R[DEST]←D

This statement means that at P1, the contents of register D are sent to three registers: 1) the BR, 2) the B, and 3) the register specified by the contents of the destination field of the IR.

In some instances, the use of the bracket can produce confusion on the part of the reader, if the specific use is not clearly defined. An example of this is shown at word DST2 on sheet 1 of the EIS flows. That statement reads as follows:

D←R[DF]PLUS 2

In this instance, the bracket pertains to "the contents of." This statement says that the *contents* of a register designated by the destination field of the IR will be incremented by two then sent to the D Register. Note that it does not say that the D Register will receive the contents of a register two locations away from the register designated by DF.

An example of the use of two types of brackets is seen at word MUL9 on sheet 3 of the EIS flows. This also illustrates that conditional transfer can be a function of more than one variable.

P2:D←f(DR00 & EPS(C) BR & B ;

This statement says that the BR and B Registers are to be sent to the D Register in a cycle length 2; but they will be acted upon in that transfer as a function of bit DR00 and bit EPS(C). On that same sheet, off to the right, a table is given for the four possible sets of conditions for these two function bits. From this, it can be seen that for two conditions of these bits, the contents of BR will be put into D; but for the other two sets, B will be either added to BR or subtracted from BR and that result will be put into D.

The semicolon is used to designate separate action(s) to occur at the same time pulse:

P2:D,BA←R[DF] MINUS 2;DATI
P3:CLKOFF

This is shown at DST9 of sheet 1 of the EIS flows. DATI follows the semicolon for P2, indicating that a DATI will be performed also on P2. In that same word, the clock will be turned off (CLKOFF) at pulse P3.

Note also at DST9 the notation SBC=1 in the lower right-hand corner of the block. This refers to the SBC (or Set B Constant), indicated by ROM bits U(32:29) in the basic machine (see table on KD11-A-BD). Although this refers to the basic machine, it should be remembered that these bits are being driven by the duplicating bits of the ROM in the option. In this case, it brings in a constant for the MINUS 2 condition. The ALU is performing the operation A-B-1. This brings in a 1, causing the ALU to perform an A-1-1 or A-2.

A comma to the left of the back arrow separates the blocks that receive data simultaneously. A comma to the right of the back arrow separates the sets of data to be transferred.

At word EI2 on sheet 1 of the EIS flows, the following statement is made:

D←f(SBC=00(STPM))

STPM refers to what the signal is eventually called in the hardware. SBC=00 is sent to the D Register and becomes signal STPM. The SBC=00 looks at discrete logic in the processor and could get any one of several values. The SBC=00 does not always select the same value to be sent to the D Register. This word is actually forming a trap vector, with the error that is set at the time determining what that vector shall be.

In many places in the flows, a general statement is often made before a branch and then is further defined after the proper branch is entered. An example is shown in ASH1 on sheet 2 of the EIS flows.

; CLOCK COUNT

Note that this indicates that the clock will be caused to count, but that it does not indicate which way. If the flow enters ASH3, the following notation is given:

P1:COUNT←PLUS 1

This indicates an incrementing count. If the flow enters ASH5, the following notation is given:

P1:COUNT←MINUS1

This indicates a decrementing count.

At ASH4 on that sheet there is an example of a number of things occurring at the same time, yet not necessarily as a result of each other, stated as follows:

P2:D←R[SF];D(C)←ALU15;EPS(C)←ALU00

These are separated by semicolons, and a transmission operator is given for each. This says: at P2, the contents of the register specified by the source field of the IR is sent to D; at the same time, ALU bit 15 is sent to the D(C) flop; and simultaneously, ALU bit 00 is sent to the EPS(C) bit.

The comma indicates inclusion as illustrated by the P3 operation of that same word:

P3:BF,B,R[SF]←D(C),D[15:01]

In this case, all the information on the right-hand side of the arrow is being sent to all the destinations on the left-hand side of the arrow. Only bits D(15:00) are being sent from the D Register. The 16th bit is the D(C) bit. By referring to the ROM output, it can be seen that the right data port of the DMUX is being selected and that data is being shifted right. Bit 1 of the D Register becomes the new bit 00, and everything else is shifted right.

In ASH7 of this page, the following notation is given which indicates another use of the bracket in specifying a register.

BR←R[SFV1]

This states that the BR receives the contents of the register specified by the source field of the IR ORed with 1. This is of course the odd register being specified.

Many times in the flow, it is not always obvious what is contained in a specified register unless the flow is traced back a few steps to see what was last put into that register. A case in point is at word ASH17 of this same page.

P2:D←R[SF]PLUS B

This says: at P2, the contents of the register specified by the IR source field plus the contents of B are sent to D. This means little unless the contents of B at that moment are known. Looking back in the flow to word ASH8 will show that at P2, R[SF] went to D, and at P3 of the same word, it went from D to B. This means that in ASH17 what is really happening is that the contents of R[SF] are being added to itself. This is equivalent to shifting it left one place.

This same word (ASH17) also illustrates the use of the "IF" statement in the next line.

EPS(V)←1 IF BR15≠BR14

This means that EPS(V), the overflow bit in the status word, will be set if a difference in BR15:14 is noticed, thus sensing an impending transition in the bit stream.

The next line in that word's statement deserves mention also:

DR←DR[14:00],0

This indicates that the DR is being shifted left and (0) is being shifted into the low bit.

In contrast to this, the notation at word MUL24 on sheet 3 of the EIS flows is as follows:

DR←0,DR(15:01)

In this notation, the DR Register is being shifted right with a (0) being shifted into the high bit.

A functional condition can be implied without the use of the "f" operator. A case in point is shown in ASH20.

EPS(Z)←D=0

Back in ASH19, both the BR and R[SF] were ORed on the RD BUS to determine the zeroness of the answer. (P2:D←R[SF],BR). In this word then the condition set in the Z bit is determined by the result of that test. If the D Register was 0, then the Z bit is set. If it was not 0, the Z bit remains cleared.

An example of the role played by the arrows on flow lines, to reduce confusion as to which direction to take, is illustrated on sheet 3 of the EIS flows at the output of word MUL3. The horizontal flow line entering the output of MUL8 shows that the MUL8 flow cannot go to MUL4 whereas the output of MUL3 can go to either MUL4 or MUL19. Likewise the decision notes on these lines pertain to the output conditions of MUL3 and not of MUL8. This is further indicated by the fact that the D15 BUT was made back in MUL2.

In word MUL11, the following notation is made:

P2:D←-BR

The use of -BR indicates that the 1's complement of the contents of BR is sent to D. To indicate 2's complement, 0 MINUS 1 is used as illustrated on sheet 4 of the EIS flows at word DIV6:

P2:D←0 MINUS B

An instance of multiple branches in sequence is given also on sheet 3 of the EIS flows at MUL20. In this word, a BUT for DR15 is made, and in the next word MUL11, a BUT for D=0 is made. This results in four branches: two for DR15 and two each for D=0. The BUT(D=0) in MUL11 tests for the branch at MUL16 while the BUT(D=0) in MUL12 tests for the branch at MUL13.

For customers with the FIS option as well, certain terminology in those flows is illuminating to the understanding of the flows. These examples are separated here to avoid confusion on the part of those customers with only the EIS option.

The bracket can be used to designate a choice of registers, based upon other variable conditions defined by the symbol "f" which means "as a function of." This is illustrated at word FP10 on page 1 of the FIS flows.

P3:BR,B,fARGA(R[12+13])←D

In this case, the contents of register D are sent to the BR Register, to the B Register, and to either register 12 or register 13 as determined by the condition of the ARGA flip-flop. The ROM will always select the odd register in the floating hardware if the ARGA flip-flop is clear. If it is set, the even register will be selected.

The XOR function is illustrated in word FP15 on page 1 of the FIS flows. The statement is as follows:

P2:D←B∀MSR

In floating multiply or floating divide, the XOR of the sign is used to give a negative sign to the answer if the signs of the two operands are unlike.

An example of a parenthetical statement used as a description is seen in word FP7:

P2:D(15:08)←fSBC00; (ZERO)
D(07:00)←B(15:08)

This says: at P2, bits D(15:08) will get 0s as a function of SBC00 while bits D(07:00) get B(15:08). In effect the high bits of the constant called for by SBC00 are 0, thereby putting 0s into the upper byte of the D Register; while loading the low byte of the D Register with the high byte of the B Register.

The insertion of the hidden 1 is illustrated in word FP9.

P2:D←400 PLUS(000,B(07:00))

This means the high byte on the B leg of the ALU is equal to 0 and the low byte will be B(07:00). The constant 400 is added to insert the hidden 1.

Concatenation is illustrated in word FML12 on sheet 4 of the FIS flows. The statement is as follows:

P1:MSR←HSR00,MSR(15:01)
HSR←DR00,HSR(15:01)
DR←BR00,DR(15:01)
BR←D(C),D(15:01)

From this it can be seen how the four registers are concatenated with MSR the LSB and BR the MSB. This is a SHIFT EVERYTHING RIGHT operation. A comparison can be made with the notations listed in FDV16 on sheet 5 of the FIS flows for a SHIFT EVERYTHING LEFT operation.

4.7.2 KD11-A Flow Discussion

A complete discussion of the KD11-A is contained in both the *PDP-11/40, PDP-11/35 System Manual (21" Chassis)* (EK-11040-TM-002) and in the *KD11-A Maintenance Manual* (EK-KD11A-MM-001). The discussion here is general, containing merely that information necessary for an understanding of the KE11-E/F Options and the ways that they interact with the processor.

Referring to the KD11-A block diagram on drawing KD11-A-BD, the Unibus is shown on the left with its 16 data lines received and driven, and its 18 address lines that are driven onto the Unibus. In addition, the Unibus is driven by the Switch Register KY11-D (which can oftentimes be addressed to retrieve data, e.g., in diagnostics) and by processor status from the PS Register (K5-5).

The heart of the processor data path is the arithmetic logic unit (ALU). This unit has two inputs: the AIN fed by BUS RD(15:00) through a buffer and the BIN fed by the BMUX. Note that the BUS RD is also the bus on which data from the EIS and FIS options are fed back into the processor.

The BMUX is fed by 1) the B Register straight through, 2) the B Register with bit 7 extended into the high byte (a sign extension), 3) the B Register with the two bytes swapped, and 4) the B constants of which there are many including address increments, switch register address, and masks.

There are five ALU control bits originating in the control ROM. The SALUM determines the mode (arithmetic or logical). The other four (SALUx) select one of 32 functions that the ALU can perform (16 in idle mode). These include (among others) carry-in logic from the EIS (only if the ALU is in arithmetic mode) and carry-out multiplexing of four inputs to the D(C) flip-flop. The latter is used in ASH right in which the data is loaded into the D Register through the ALU and the D(C) gets bit 15 of the D Register so that the sign can be extended down through the DMUX during shifting.

NOTE

This is used on word operations. In byte operations, carry-out 7 is used for this purpose.

The COUT MUX also receives the C bit of the processor status for use in rotates.

During any of these operations, the B Register, which feeds the BMUX, functions as a storage register. Note that this is the only holding register on the BIN side of the ALU. The general registers are on the AIN side via the BUS RD, although they *can* feed the BIN side through the BMUX and B Register by virtue of a feed back through the DMUX. Note also that the DMUX can also feed the options. The DMUX is not wire-ORed, it is TTL and is used to feed data to the EIS and FIS.

BUS RD also feeds the bus address multiplexer (BA MUX). An address can be brought out of the general purpose registers and fed directly into the BA through the BA MUX, or can be fed through the AIN of the ALU, added to, subtracted from, or operated on in the many ways possible, and *then* fed to the BA via the BA MUX.

The processor status is also fed to the BUS RD for internal use, e.g., for a condition code instruction, and does not have to be fed out onto the Unibus. It can be operated on in similar fashion as above without interrupting the Unibus.

The BA Register contains logic on its output for decoding processor status address, stack limit register address (an option), register address (internal registers are being addressed), and switch register address. There is also logic on the D Register output to determine whether or not the D Register is equal to 0. The latter is used in the EIS option as described later.

The instruction register feeds logic to decode all the discrete instruction Op codes. These 16 bits are converted into many signals which are then encoded back down into the U Branch Control which is 6 bits wide. The IRD code then is used in the first FETCH branch so that the proper U Word can be accessed to perform the proper instruction.

The ALU control block is actually an *auxiliary* ALU control fed by the ALU field in the ROM and by some discrete logic. This is functionally a multiplexer used in common routines in which the discrete logic is used to control the ALU rather than a separate ROM word for each instruction. In this case, a common ROM word is used to point to auxiliary control.

The U Word Control ROM contains 256 words, 56 bits wide, 8 bits of which comprise an address which doubles back to a NOT OR into a pointer register (UPP). This register functions as a PC for the ROM with these 8 bits specifying the next ROM address to be looked up. The rest of the ROM bits (48) control the internal machine and the data paths, clock the BA and D Registers, and determine which multiplexer port will be active. Some bits are gated with the basic clock pulses to control cycle length, others enable different registers to be clocked such as CLKD which is gated with P2 to clock the D Register.

The JAMUPP logic is used to jam the UPP Register to specific addresses for specific error conditions.

The PUPP Register holds the previous microprogram pointer or the address of the word that is in the U Register. This is used to feed the maintenance console display.

The condition codes input block is used to set up the condition code bits in the PS Register to indicate the results of the last operation or instruction.

The Data Display on the console is fed by the DMUX. Since the DMUX can contain either Unibus data, the D Register straight through, the D Register shifted right, or the contents of the BUS RD, there are basically four sources for display; and since the BUS RD can have many things on it such as the general registers, status, or option data, the range of data displayed is very wide. In single instruction mode, processor status is displayed at the end of the instruction.

NOTE

On a HALT instruction, the contents of R0 are displayed and not PS, even if in single instruction mode.

By referring to sheet 1 of the KD11-A flows (drawing KD11-A-F), the way in which the EIS/FIS options are enabled can be followed. Entry is at FETCH A and proceeds to word 013 (FET00). This is the "Fetch Next Instruction" word in which, during a cycle length P1, the contents of the program counter (R[PC]) are loaded into the BA, a DATI is performed, the clock is turned off (CLKOFF), and the R[PC] is displayed.

NOTE
FETCH OVERLAP does not apply to EIS or FIS instructions.

A wait is indicated in the flow followed by entry into word 001 (FET03) to "store the instruction." Here, in P1, the Unibus data is sent to the B Register, to the general register, to R(IR) which holds a copy of what is to be loaded into the IR, and to the instruction register. Bus data (Instruction Word) is displayed.

NOTE
The R[IR] copy provides a convenient way to access this information when the IR is not accessible.

In the next word (FET04), the PC is incremented (R[PC]+2) and put into the BA and D Registers. In addition a branch test is made (BUT(INSTR I)), numerically BUT 37, to determine which flow exit indicator to take. Once again the R[PC] is displayed.

The flow then goes to FET05 to store the modified R[PC]. This is done by transferring the R[PC]+2, which was put in D in FET04, from D back into R[PC] in FET05.

Normally, without the options installed, an EIS or FIS instruction would not be recognized by the processor. The machine would branch off to address 100, the base U Word address for the branch, and take the TRAP B exit. But when the options are installed, the BUT(INSTR I) signal (BUT 37) is sent to the option where it is gated with a pulse and sent back to the KD11-A to UPP bit 8 as the clock for bit 8 of the ROM address. The signal RSVD INSTR, gated with EUPP8, is sent back to the KD11-A as the data for UPP8. Setting bit 8 modifies the next address to be looked up from 100 to 500. Note that now the flow exits through the expansion diamond and enters the EIS flows at word EI0 (location 500). Bit 8 remains set to enable the option ROM and disable the basic ROM.

If the option is installed and a reserved instruction is issued, the flow still follows this path and exits to TRAP D.

4.7.3 KE11-E Flow Diagram Discussion

The KE11-E flows are shown on drawing KE11-E-FD, sheets 1 through 5. The format of these flows is identical to that of the KD11-A and the conventions follow those described in preceding paragraphs.

As described in Paragraph 4.7.2, entry into the expansion flows is at EI0 after the CPU has stored the EIS/FIS or Reserved Instruction in the IR, decoded the IR, made a branch microtest, set ROM bit UPP8, modified the address to 500_8, and set the CPU Trap flag.

At this point the KE11-E ROMs are enabled and EIS ROM word EI0 is present on the wire-ORed BUS U(56:00) lines at the input of the CPU U Register, as are the outputs of the KE11-E Control ROMs U(80:57) at the input of the EIS U Register. The KE11-E performs another decode of the CPU IR and does a branch microtest (BUT(EINSTR I)) to determine what address calculation flow to enter. The CPU Trap flag is cleared.

NOTE
If the instruction was not one of the EIS instructions, the CPU Trap flag would not be cleared here in order that a RSVD INSTR trap would occur upon exit back into the CPU Trap flow.

The above sequence occurs for each EIS instruction. Upon completion of the EIS instruction, the microflow returns to the basic KD11-A microprogram at microword SER_{02} at ROM address 17_8.

Note that if it had been a reserved instruction and not an EIS/FIS instruction, the flow would be from EI1 to address 640 (EI2) to form the vector 10 and clear the UPP8 bit. Then at EI3, the Special Trap Pointer Marker (STPM) previously sent to D is stored by transfer to Scratch Pad Register R[VECT] and B. Exit is to TRAP D at microprogram address 7 on sheet 6 of the KD11-A flows. The entry point of TRAP D into traps differs from that of the TRAP B, used by the basic machine. TRAP B forms the vector and stores it, whereas at TRAP D this has already been done in the option.

There is an overlap in this transition back to the basic machine that is similar to the transition from the basic machine to the option. At the end of EI2, UPP8 was cleared and at that instant the expansion ROM was disabled and the basic ROM enabled. In word EI3, while the data path is setting up to store the vector, a new ROM word is being looked up. That new word comes from the basic ROM so that the last expansion word is being executed at the same time that the basic ROM word is being fetched. The same thing happened when the CPU vectored for the expansion. Bit UPP8 was set at the end of FET04 and the contents of FET05 were loaded into the U Word Register. While the FET05 was being executed in the data path, the next word being looked up was in the expansion ROM.

4.7.3.1 Destination Calculation – Sheet 1 of the EIS flows describes the destination calculation operations required to perform the four fixed-point instructions.

NOTE
The FIS exit is directly from word EI1.

The BUT(EINSTR I):EUB=17 in EI0 tests the path to be taken in terms of destination mode of addressing (DM0 through DM7) as determined from the destination mode bits in the instruction. Each path performs the functions necessary to calculate the destinations as a function of these modes. In all paths, two words prior to the instruction exit, the branch test BUT(EINSTR II):EUB=16 is made to determine which instruction is called for. This is determined from the Op code of the instruction.

The operation of this flow is almost identical to the basic DEST flow for the processor as described in the *KD11-A Maintenance Manual* (EK-KD11A-MM-001). One exception is word DST8, through which all other paths flow before branching. In this word, the Unibus data (which is the data from the final destination address) is sent to R[DEST] (the register specified by the destination field of the instruction). It is also sent to B and BR. The exception is word DST1 which does not go through DST8. Note that both words accomplish the same operation except that DST8 gets data from the Unibus and DST1 gets it from an internal register. In DM0, there is no need to go to the bus for data since the register contains the operand.

At the conclusion of this flow, the data retrieved from the calculated destination resides in BR, B, and R(DEST).

4.7.3.2 Arithmetic Shift and Arithmetic Shift Combined – The shift flows are shown on sheet 2 of the EIS flows and in Figure 4-8. In this operation, the data fetched from the calculated destination is a shift count that also indicates the direction in which to shift. The bits to be shifted are in a register designated by the source field of the instruction. If the instruction is ASH, 16 bits are to be shifted and the result will be stored in BR, B, and R(SF). If the instruction is ASHC, 32 bits are to be shifted, with the high 16 bits taken from an even register specified by the source field and the low 16 bits from that register ORed with 1. The results are stored in BR, B, and R(SF) (high answer); and in R(SFV1) (low answer).

NOTE

In ASHC, if an odd register is specified by the source field of the IR, the ORing process in the hardware will result in two duplicate operands being fetched. If an odd Source Register is specified, the low 16 bits of the answer will overlay the high 16 bits of the answer.

ASH is entered at word ASH0 at location 605. At P1, the 8 bits of data in the BR(07:00) are loaded into the counter and the BR is cleared. Prior to that pulse, however, bits(05:00) of the BR are tested (EUB=13) to determine the branch after ASH1.

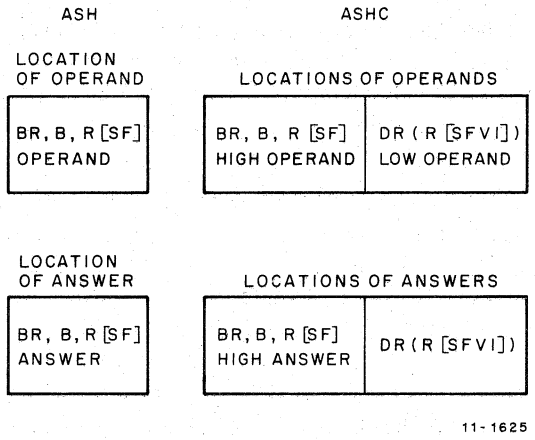


Figure 4-8 ASH and ASHC Locations of Operands and Answers

In ASH1 at P2, the register specified by the source field (R[SF]) is sent to the D Register, the V bit in the local status is cleared, and the count is clocked one time. At P3, the contents of the D Register (R[SF]) are loaded into the BR and B Registers. This is the data to be shifted.

If, back in ASH0, the result of that BUT indicated that bit 05 of the BR was set, a shift right operation was indicated, causing the flow to proceed now to ASH3 where the count is incremented and the branch microtest (EUB=10) is set up for the shift loop in ASH4. The count is clocked at this point so that the shift loop in ASH4 will not be executed more than the specified amount. Note that in one word loops, the word is always executed one time more than the count on initial entry would indicate. This is because of the overlap of executing the present word while looking up the next word, and by the fact that branch tests are made two words ahead of the actual branch point.

In ASH4, the right shift is implemented by putting the general register specified by the source field through a buffer and through the AIN of the ALU into the D Register. At the same time, ALU bit 15 is sent to the D(C) flop via the COUT MUX and ALU bit 00 is sent to EPS(C). This occurs at P2 of a cycle length 3. At P3 of that word, D(C) and D(15:01) are fed through the DMUX right shift data port back into the general register R(SF). Remember, in this operation D(C) is equal to D15. In addition to this, data is sent to the B and BR Registers and the count is incremented. This operation continues, shifting right one bit position for each pass through the loop, extending the sign down and putting the low bit of the ALU into the EPS(C) bit until the count is equal to 0. At this point, the answer is in three places: R(SF), B, and D. Note that the counter is disabled from counting when count bits 5 through 0 are equal to 0.

In ASH14, the contents of B are sent to D for the zeroness test at P2, and the flow progresses to ASH20 where BR15 (the sign of the answer) is sent to the N bit of the EPS and the result of the zeroness is sent to EPS(Z). If the contents of D were 0, EPS(Z) will be set.

This word continues on to ASH15 in which the status bits EPS(N) through EPS(C) are transferred to the processor status in the basic machine. Note that in ASH20 there is no need to set either the V bit (set in ASH1) or the C bit (set in ASH4). In ASH15, the UPP8 bit is also cleared to transfer ROM control back to the basic machine.

ASH21 is a No-Op word that loads 0s into the upper 24 bits of the ROM Register. This is done for housekeeping reasons so that upon reentry to the option, no extraneous bits will be left in that register.

From here the flow exits to SERVICE C (location 17_8) on sheet 10 of the KD11-A flows. Everytime the option is exited, the flow is through this route to test if an interrupt or trap is pending. If one is pending, it is serviced before continuing to the next instruction.

If, back in ASH0, bit 05 of the BR was cleared and BR(04:00)≠0, a shift left operation was indicated, causing the flow to proceed from ASH1 to ASH5 where the count is decremented instead of being incremented as in ASH3. The same BUT is made (EUB=10) and ASH6 is entered for a shift left operation.

In this case, shifting does not take place in the DMUX but rather in the ALU by the function A PLUS A. In this case, BR15 goes to C rather than ALU00. The statement EPS(V) gets 1 if BR15≠BR14 refers to the sensing for sign change. If when shifting, these bits are different, an impending change in sign is indicated and the EPS(V) bit will be set. This bit will then remain set even if more shifting is necessary so that at the end of shifting the programmer has an indication of sign change. This shifting continues with the count being decremented for each pass until the count is exhausted. The sequence from that point on is identical to a right shift.

In the instance of no shift (BR(05:00)=0), the EPS(V) and EPS(C) bits are both cleared, BR15 is sent to the EPS(N) bit, and the EPS(Z) bit is conditioned by the zeroness test of D. This progresses to ASH15 for transfer, ASH21 for cleanup, and out to SERVICE C.

The ASHC flow is similar to the ASH flows except that now 32 bits are involved instead of 16. Entry is at ASHC to word ASH7 at location 607. At P1, the low 8 bits of the BR are loaded into the counter and the low operand (R[SFV1]) is put in the BR. At the same time the EPS(V) bit is cleared. Prior to that, before the count has been transferred out of the BR, a BUT is made to determine what branch to take (shift right, shift left, or no shift).

In ASH8, the low 16 bits that are now in the BR are sent to the DR Register and the high operand is taken from the even register (R[SF]) and put into the D Register. These operations occur on P2. On P3 of that word, the high 16 bits, now in D, are transferred to the BR and B Registers. At the same time the count is clocked. At this point, the data to be shifted comprises 16 bits of low operand in the DR and 16 bits of high operand in the BR and B Registers. Note that the low operand went through the BR and was moved out to the DR before the high operand was put into it.

If the result of the BUT indicates no shift, the flow is to ASH9 where the R(SFV1) and BR are simultaneously sent to D, effectively ORing the high and low operand on the BUS RD to determine zeroness of the full 32-bit operand. In ASH2, the local status is set and moved to the processor in ASH15 as before.

If a right shift is indicated, the count is tested for zeroness in ASH10, the BUT is set up for the loop and the right shift loop is entered (ASH11). Here the lower 16 bits are shifted in the DR Register while the high 16 bits are sent through the ALU and shifted in the right data port of the DMUX. This operation is identical to that in ASH. Since the BR and DR are concatenated, BR00 goes into the high bit position of the DR while the low bit of the DR is put into the EPS(C) bit. All this occurs on P2. At P3, the right data port of the DMUX is fed back into the source (R[SF]), the B, and the BR Registers, and the count is incremented. This loop continues until the count equals 0 (EUB=10) at which time the loop is left and the flow continues to ASH12.

The low answer in the DR is sent to D in ASH12 and stored into the odd Source Register in ASH13. In ASH19, the high answer (BR) and the low answer (R(SFV1)) are sent to D so that the zeroness test of the 32-bit answer can be made by ORing on the BUS RD; and in ASH20, the EPS(N) and EPS(Z) bits are loaded prior to transfer in ASH15.

Operation of ASHC left is similar to ASHC right except that the count is decremented and the operation of the shift loop differs in the same way that it did in the ASH instruction. In shift left ASHC, the low half is shifted in place in the DR Register, with 0 being put in the low bit (DR00). The high half is shifted by the A PLUS A function of the ALU. At P3 of ASH17, the D is put back in the BR, B, and Source Registers. In ASH18, the low answer is put into D (same as ASH12) and the flow continues as in ASHC right.

Note that in ASHC operation, the high part of the answer does not have to be stored as that was accomplished in the loop.

4.7.3.3 Multiply – The multiply flow is shown on sheet 3 of the EIS flows and in Figure 4-9. In this operation, the data fetched from the calculated destination is the multiplier, and the data from the register specified by the source field (R(SF)) of the instruction is the multiplicand. The two 16-bit numbers are multiplied and the 32-bit result is stored in R(SF) (high product) and in R(SFV1) (low product).

NOTE

In MUL, if an odd R(SF) is specified in the instruction, the low product will overlay the high product.

Entry is at MUL to MUL0 at location 601. The multiplier from the destination address had been loaded into the BR, B, and R(DEST) Registers during DST1 or DST8. The count (17_8) is generated and put into D at P2 and into BR from D at P3. A BUT is set up for the sign of the multiplier (EUB=3).

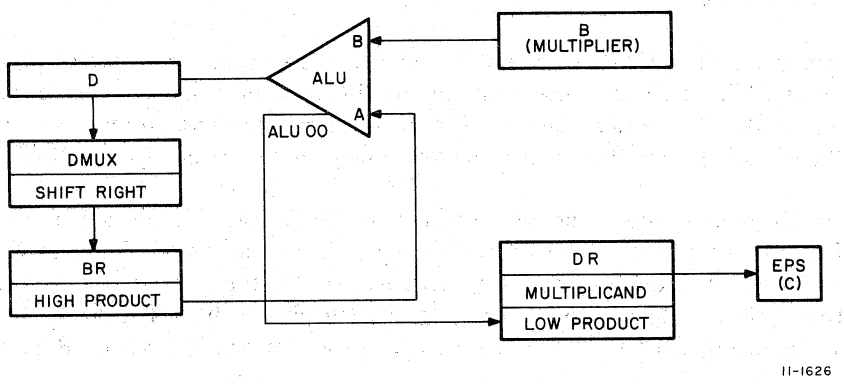


Figure 4-9 MUL Flow, Block Diagram

In MUL1, the 2's complement of the multiplier is taken. This is done at this point in case the negative multiplier branch is called for. In this word, the EPS(C) bit is cleared and the count is transferred from the BR to the counter.

If BR15 was set on the BUT, the next word is MUL2 in which the multiplicand is put into the BR from the source field designated register and a BUT is made for the state of D15 (EUB=1).

In MUL3, the multiplicand is put into the DR and the contents of the Source Field Register is put into D. If as a result of the BUT in MUL2 D15 was 1 (note, this was what was in D at that time or the 2's complement of the multiplier), then the multiplier is determined as being the most negative number since that is the only number that can be 2's complemented and remain negative. In this event, the flow goes to MUL4 where the copy of the multiplier in R[DEST] is now sent to the BR.

NOTE

This is the uncomplemented multiplier. The complementation done in MUL1 was for testing purposes.

Another test for D15 is made in this word, and in MUL5 the multiplier (in BR) is sent to DR and BR is cleared.

NOTE

Normally the multiplier would be put in B when the loop was started, but the most negative multiplier must be put in the normal position of the multiplicand to produce the correct result.

If the multiplier had proven to be positive in MUL0, flow would have gone to MUL7 where the Source Register would have been loaded into the BR Register, and from BR into DR in MUL8, and then the count would have been decremented in MUL19 to take care of the one-word loop discrepancy described in ASH.

In MUL5, returning once again to the flow for the most negative multiplier, the flow branches again as to the sign of the multiplicand. This is indicated by the state of D15 since the multiplicand (R[SF]) was loaded into D back in MUL3. In this flow, the sign of the multiplicand must be tested also since a most negative number can never be placed in the B Register. Since one operand has been determined to be most negative, the other operand must be tested for that characteristic as well.

If D15 is clear at MUL5, no problem exists and the multiplicand is put into B as the count is decremented. If D15 is set, however, the flow goes to MUL21 where *it* (the multiplicand) is tested for being the most negative number. This time the 2's complement is taken by subtracting the Source Field Register from the B Register (already established as being the most negative number).

NOTE

BUT(COUNT=0):EUB=10 is done here to clock the NPR and Bus Request flags and to clear the Bus Busy flag in the KD11-A (see Note 2 on sheet 3 of EIS flows).

If the result of the comparison just done in MUL21 is 0, the indication is that the multiplicand is also the most negative number and the answer can be generated at this point in the flow without continuing with the multiplication.

After zeroing the low answer in MUL23, the flow branches to MUL24 and the high product is shifted right one place.

NOTE

The most negative number times the most negative number is equal to that number shifted right one place as the high product and with 0 as the low product.

In MUL25, the answer in DR is put into D and stored in the Source Field Register in MUL26. The local condition codes are set and the flow exits to the MOVE EPS point in the ASH/ASHC flows. This is ASH15, the common transfer point to the Processor Status Word before returning to the service routine.

If, back in MUL23, the multiplicand proved *not* to be the most negative number (-D=0), it is put into the B Register at MUL27 and the count is decremented.

This brings the flow to the common point of all flows described for multiply so far. This is the entry point to the multiply loop at MUL9.

The multiply operation is essentially a right shift operation through the right data port of the DMUX. It is a functional operation of the ALU, determined by the instantaneous conditions of bit 00 of the DR Register and the EPS(C) bit. (See the table to the right of this block.) The contents of BR and B will be either subtracted or added in the ALU before being shifted one bit postion to the right through the DMUX, or the ALU will put the data straight through before it is shifted.

One of these operations will occur prior to each shift for each pass through the loop, depending upon the states of the two conditioning bits for the ALU. At P3 of each pass, the high product is being assembled in the BR and the count is decremented. This continues for 16 passes, at which time the count equals 0 and the loop is left.

NOTE

The notation GPC=2; DAD=14 at P3 of MUL9 pertains to generation of auxiliary ROM control. The DAD code is for the basic machine auxiliary ROM control enable and the GPC code is for the option auxiliary ROM control enable.

In MUL10, the high product that has been assembled in BR is sent to D; in MUL20, it is stored in the even Source Field Register. At this word, a test is made of the sign of the low product (BUT(DR15):EUB=5) in preparation for setting the C bit in the EPS. If the result is less than -2^{15} or is greater than or equal to $2^{15}-1$, this bit must be set, otherwise it is cleared. The result is represented in BR (high product) concatenated with DR (low product); and to determine the proper setting of the EPS(C) bit, the high bit of the DR (bit 15) must be compared with the entire contents of the BR. IF BR contains all 1s and DR15 is also a 1, the answer can be represented by just the low 16 bits. Similarly, if DR15 is 0 and the BR is all 0s, the high 16 bits are still an extension of the MSB of the low 16 bits and the answer can still be expressed in one word.

In MUL11, the high product is complemented and sent to D. The EPS(V) bit in the local status is cleared, and the BUT for D equal to 0 (EUB=4) is made. This is done because if D is equal to 0 after complementing, then it was all 1s.

Coming out of MUL11, the branch is made. If DR15 was 0, the low product was positive and the flow proceeds to MUL16 where the EPS(C) bit is set and the DR is put into D. At the branch coming out of MUL16, the result of the test for the state of D in MUL11 determines the flow. If it had not been 0, the EPS(C) bit is left set, the other status bits are set, and the low product now in D is sent to the odd Source Field Register. This occurs in MUL18 from which the flow exits to MOVE EPS on sheet 2 of these flows.

If D, in MUL11, had been 0, the flow is to MUL17 where the EPS(C) bit is cleared once again, the other status bits are set, and the low product is sent to R(SFV1) as in MUL18.

If, back in MUL20, DR15 was seen to be 1, similar actions take place in MUL12, MUL13, and MUL14 or 15. The status bits are set accordingly with the EPS(N) bit set if the product is less than 0 or cleared if it is more than 0. The EPS(Z) bit is set if the product is equal to 0 and is cleared otherwise. The EPS(V) bit is cleared and the EPS(C) bit is set or cleared according to the already stated criteria.

From all of these flows, the resultant status is transferred to the Processor Status Word (MOVE EPS), the high product was stored in the even register in MUL20, and the low product in the odd register in the steps just described. The flow then proceeds to SERVICE as in other operations.

4.7.3.4 Divide – The divide flow is shown on sheets 4 and 5 of the EIS flows and in Figure 4-10. In this operation, the data fetched from the calculated destination is the divisor, and the data from both the register specified by the source field of the instruction and that register ORed with 1 is the dividend. The high dividend is taken from the even register and the low dividend from the odd register. The 32-bit dividend is divided by the 16-bit divisor and the results are stored in R(SFV1) (remainder) and R(SF) (quotient).

NOTE
In DIV, an even R(SF) *must* be specified.

Entry is at DIV to DIV0 at location 603. In this word, the count is generated from the SBC code of 12. This generates an octal 17 or decimal 15, providing the 16 passes through the divide loop. This is loaded into D at P2 and from there to the BR in P3.

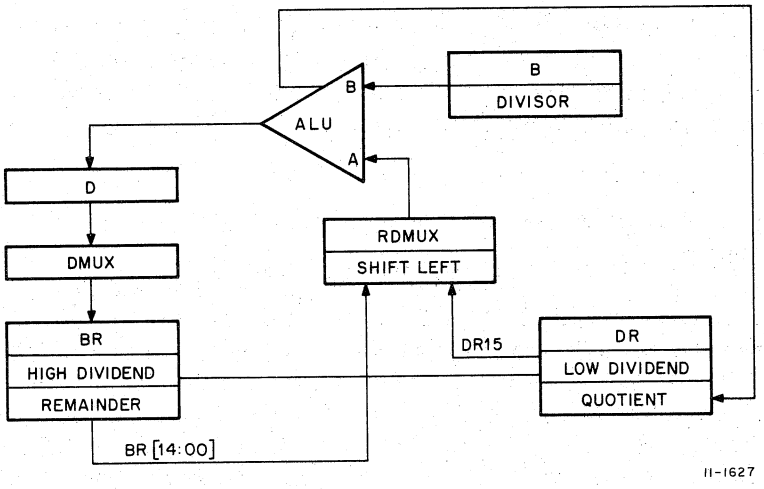


Figure 4-10 DIV Flow Block Diagram

At DIV1, the divisor in B is sent to D; and in DIV2, the count is loaded from BR. At the same time (P2), the high dividend in the even Source Register is loaded into BR and a test is made of the D Register which holds the divisor. If it is 0, then the divisor is 0 and the flow will go through DIV3 (a No-Op) to DIV4 where the EPS(Z,V,C) bits are set before exiting to MOVE EPS. Divide by 0 is undefined and is not executed.

If the divisor is not 0, DIV5 is entered where the low dividend (R(SFV1)) is put into B and the EPS(N) bit is set to the state of BR15. This is the sign of the dividend. The test BUT(BR15):EUB=3 is set to test the sign of the dividend for the branch from DIV6.

In DIV6, the 2's complement of the low dividend is taken. This is done in the event the negative dividend path at DIV7 is *to be* taken, and it affords a chance to make the dividend positive before operating on it. If the negative path is *not* taken, nothing is destroyed by this complementing. At the same time, the carry-out of bit 15 of the ALU (COUT15) is stored in EPS(C) for future use.

At P3 of DIV6, the complemented dividend is put into the BR and the BUT(COUNT=0) is made to clear any hinderances to NPRs.

If BR15 was (1) in word DIV5, the flow is to DIV7 where the high dividend is sent to B so that a 2's complement of it can be taken in DIV8. Note that the carry-out stored in EPS(C) in DIV6 is now used as a carry-in to effect the 2's complement. This arrangement provides the 2's complement of the total 32-bit dividend. A BUT is made here to determine whether or not the dividend is the most negative number (EUB=3) which tests bit 15 of the BR.

In DIV10, the divisor in R(DEST) is sent to B for use in DIV16; but if BR15 proved to be a (1) in DIV9, its not needed and the DIV QUIT path is taken to DIV11 (a No-Op) and DIV12 where the condition codes are set appropriately and the flow exits to MOVE EPS.

NOTE

DIV QUIT is taken because division into the most negative number results in more than 15 bits of answer.

If BR15 proved to be a (0) in DIV9, the indication is that the dividend is not the most negative number and DIV16 is entered for the first division step.

Before entering DIV16 in this discussion, the flow is taken back to DIV13 which would be entered if the dividend was determined, in DIV5, to be positive. In this case, the low dividend is moved to BR and from BR to DR in DIV14. At the same time, the high dividend is put into BR. Then in DIV15, the divisor is put into the B Register. From either path, the entrance into DIV16 sees the divisor in B and the full dividend in BR and DR, with BR holding the high 16 bits.

The first division step is done at DIV16. The operation in this step, in DIV19 and in the loop of DIV20, is to shift then add, or shift then subtract.

Note from the statement in DIV16 that the D Register will be loaded with BR(14:00), DR15 and B as a function of B15. This process can be followed on the block diagram to the top right of the sheet. There it can be seen that the BR concatenated with DR15, shifted left one bit position through the RDMUX, and fed to the AIN of the ALU, while the B Register (the divisor) is fed to the B input of the ALU. Depending upon the state of B15, the ALU will add B to or subtract B from the AIN data of the ALU and feed it to D, through the DMUX to BR. The carry-out, bit 15 of the ALU, is shifted into the low end of DR as the partial quotient, while a remainder (if there is any) is being formed in the BR.

For divide, there are two possible criteria that determine auxiliary ALU functions. One has already been mentioned (B15) and this is the determinant whenever DAD=14 in the basic ROM is asserted. The ALU functions for the two states of B15 are given in a table to the right of word DIV16. The other criteria are given in the table to the left of DIV20 in which the dual conditions of both B15 and DR00 are the determinants. These are used upon the simultaneous assertion of GPC=2 in the option ROM and DAD=14 in the basic ROM. Note DR00 is the result of the last add or subtract.

At DIV16, just the state of B15 is used (DAD=14 is asserted). Since the 2's complement of the divisor has already been taken, and since if after complementing the high bit was still a 1, the most negative number would have been indicated; it is known at this point that B15 is 0. Thus, the first shift will take place, followed by a subtract operation. The count is decremented, making the count equal to 14.

The flow then progresses to DIV17 where the size of the quotient is determined with respect to the ability of the hardware to express it. If it is greater than 16 bits, it exceeds the capability of the machine and the DIV QUIT path is initiated. As such, DIV17 is a No-Op merely to establish the test BUT (DIV QUIT):EUB=7. This test looks for the three possible sets of conditions expressed in the formula to the right of this word. This is really a test for overflow.

In DIV18, the count is decremented and the C bit is cleared to indicate that divide by 0 was not attempted. This reduces the count to 13.

At this point , if DIV QUIT was not indicated back in DIV17, the flow is to DIV19 which is the second division step. Here another shift is accomplished as described above except that now both B15 and DR00 are used as criteria for adding or subtracting after the shift (GPC=2 is asserted). The count is decremented, putting the count at 12.

Word DIV20 is the divide loop in which the arithmetic operations are performed according to the same determinants (BR15 and DR00) until the remaining 14 passes are completed. The BUT for count equal to 0 is also set in this word.

Upon completion of the count, the flow exits to DIV A on sheet 5 of the EIS flows for the completion of the divide operation.

The object of the steps on this sheet is to ensure that the sign of the remainder is the same as that of the dividend. If the dividend is negative, the remainder must be stored as a negative number and vice versa. Further, the sign of the quotient must follow the algebraic rule that division of two negative number (or of two positive numbers) produces a positive quotient, whereas one of each produces a negative quotient. Furthermore, since the division process subtracts ascending orders of the divisor, an excess operation may occur. Therefore, the remainder might have to be corrected away from 0 before it is stored.

Entering the flow on this page shows the remainder stored in D (DIV21) and then stored in the odd register (DIV22). The test in DIV21 (EUB=12) is made to determine which path to take at the branch from DIV22. This test looks at the states of B15 (the sign of the divisor) and DR00 (the low bit of the quotient). The latter indicates whether or not a correction must be made. If DR00=1, no correction is required, but if DR00=0, a correction *is* required. If B15=0, a positive divisor is indicated; if B15=1, a negative divisor is indicated.

Note that the extreme left-hand path to DIV23 is taken if the divisor is positive and no remainder correction is needed. The next path is taken if the divisor is positive and remainder correction is required. Note here (DIV27) that the correction adds the divisor (B) to the remainder (BR), correcting away from 0. The next path to DIV31 is taken if remainder correction is required for a negative divisor in which correction away from 0 results in the divisor being subtracted from the remainder. And the last path to the right is taken to DIV23 if the divisor is negative and no remainder correction is called for.

The second word in these two central correction paths stores both the corrected remainder in D in the odd Source Register and in the BR Register. The copy in BR is used for 1's complementing and is transferred to D in case the dividend is deemed to be negative by the BUT (EUB=2) in either words DIV23 or DIV33. In words DIV24 or DIV34, the complemented remainder is returned from D to B and BR.

At this point in either DIV23 or DIV33, the results of the BUT(SDIV) are used to guide the flow either to the positive dividend paths (-SDIVD) or the negative dividend paths (SDIVD). Note that at this point the sign of the divisor is already implicit in the path.

The path from DIV24 to DIV25 is taken if both the dividend and divisor are positive. This indicates that the quotient will also be positive. In DIV25, therefore, the quotient is sent to D and BR from DR; in DIV26, it is stored in the even register from D, while the status bits are appropriately set before exit to MOVE EPS.

The path from DIV34 to DIV35 is taken if both the dividend and divisor are positive. Once again the quotient will be positive and the same exit path as above is taken through DIV26, after the remainder is 2's complemented in DIV35 and stored in DIV36.

The other two paths (from DIV24 to DIV29 and from DIV34 to DIV37) are taken if the signs of dividend and divisor are different, yielding a negative quotient. From DIV34 to DIV37, no complementing of the remainder is

required before complementing the quotient. The corrected remainder was already stored back in DIV32; however, from DIV24 to DIV37 the remainder must be complemented before complementing the quotient. This is done in DIV29 and DIV30 where the remainder is stored in the odd register.

Entering DIV37, the remainder has been properly stored and the flow is concerned with storing the quotient as a negative number, together with determining if that quotient is or is not the *most* negative number (100000).

At DIV37, the quotient is 1's complemented (in the event that it needs to be at the branch out of DIV39); in DIV38, it is stored in the even register (as well as in B). DIV38 also sets up the BUT for D15 (EUB=1), the high bit of the complemented quotient. This is done to see whether or not the quotient is negative after 1's complementing.

In DIV39, at P2, 1 is added to the 1's complement of the quotient to see if it is a most negative number after 2's complementing. This test occurs between P2 and P3 of that word.

NOTE

This is one of the few cases in which a BUT is made in the same word with the modification of that data. The data is modified on P2 and tested from P2 to P3.

At P3 the 2's complemented quotient is stored in the even register. If the result of the BUT in DIV38 found D15 set, the quotient is not the most negative number and the quotient is stored in DIV26 where the status bits are also set before exiting to MOVE EPS. If, however, the result of the BUT in DIV38 found D15 cleared, the quotient was indicated to be a negative number, and in DIV40, the local condition codes are set to indicate that.

Coming out of DIV40, the results of the BUT in DIV39 come into play and if D15 tested to be clear, the quotient is deemed to be negative but *not* the *most* negative. This flows through a No-Op at DIV42 to MOVE EPS; but if D15 tested to be set in DIV39, the quotient *is* the most negative number and in DIV41 the EPS(V) bit is altered to (1). The flow exits to DIV QUIT.

4.7.4 KE11-F Flow Diagram Discussion

The KE11-F flows are shown on drawings D-FD-KE11-F-FD, sheets 1 through 6. The format of these flows is identical to that of the KD11-A and the KE11-E, and the conventions follow those described in preceding paragraphs.

Entry into the FIS flows is through the KE11-E, initialized in a similar manner to that described for the EIS option in which the flow follows through FETCH and then BUT 37 sets bit 8 of the ROM address. This sends the flow over into the expansion ROM entering the EIS flows as described before. When BUT(EINSTR I) is raised in the EIS, decoding takes place to recognize whether it is an EIS or FIS instruction; and if the outputs of that decoding yield IR=75xxxx, that signal is sent to the FIS hardware where it is gated with the proper IR bits for an FIS instruction. If they compare, a signal is sent back to the EIS branching logic, forcing a branch to the FIS EXIT and from there to the FIS entrance on sheet 1 of the FIS flows.

4.7.4.1 FIS Entry – Sheet 1 of the FIS flows describes the FIS entry operation. Entry to this flow is at FIS to FP0 at location 642. Here the floating stack pointer (R(DF)), pointed to by the destination field of the instruction, is put into the BA. Also the DATI is initiated and the clock is shut off to await memory response. The contents of R(DF) point to the high B argument on the stack containing the sign, the exponent, and the high part of the mantissa.

There are two passes through this flow in fetching and storing the arguments. The first from FP0 through FP13 fetches and adjusts the B arguments. The second from FP1 through FP14 fetches and adjusts the A arguments (Figure 4-11).

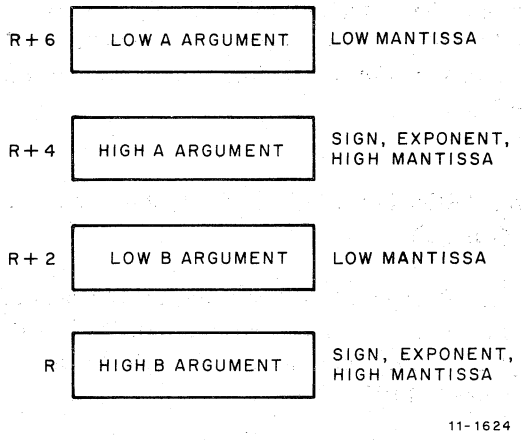


Figure 4-11 Floating-Point Arguments Order on the Stack

FP1 strobes the argument in off the bus and places it in three places: 1) in BR, 2) in B, and 3) in R(11), the odd register. Note that the general purpose register that is used for storage (10 or 11) is a function of ARGA. The ROM always specifies the odd register in this flow, but when ARGA is set (only during the second pass through the flow), its setting diverts the argument fetched to the even register.

In FP2, at P2, the high B argument in the BR is put into the DR, the pointer is incremented by 2, and a DATI is initiated in preparation for getting the low B argument. At P3 of FP2, the incremented pointer is returned to R(DF) and the clock is turned off. After a wait for memory response, FP3 strobes the low B argument into R(13) and into B.

The low B argument is shifted left in FP4 by adding it to itself. Note that in FP3 the low B argument is sent to B, and in FP4 those contents are added to the same data in R(13), then sent to D as the carry-out 15 is saved in EPS(C). At FP3 then, the shifted data is returned to R(13).

FP5 puts the high B argument, which was stored in R(11) in FP1, in B; and FP6 shifts it left. This single bit-shift to the left puts the entire exponent that was partially in the low byte into the high byte. In so doing, the sign bit is shifted out (not lost, still in BR) and the EPS(C) bit is inserted at the low end. The carry-in to the ALU is enabled by GPC=4. This double-precision shift operation makes it appear that the full 32 bits were shifted at once and provides an extra bit position on the low end of the low B argument for future rounding purposes. At P3 of that word, the shifted high B argument is returned to B and R(11).

Word FP7 is used to separate the exponent and high mantissa and to put the exponent in the low byte of a testable word. To do this, the high byte of D is forced to 0 (SBC00) and the exponent (B(15:08)) is sent to the low byte of D. This is done for future overflow or underflow testing. These conditions will be indicated by what happens to the high byte of D (1s from the right = overflow, 1s from the left = underflow).

In FP8, the separated exponent is sent from D to R(15) and if D was equal to 0 (0 exponent) the EPS(Z) bit is set. Note that ZB is not relevant on the first pass through this flow. The zeroness of the exponent is also tested for branching at FP10 (BUT(D=0):EUB=4) so that if it *is* 0, a 0 argument will then be generated.

At FP9, the hidden 1 is inserted (all numbers are assumed to be normalized). A CON field=0 is asserted, generating a constant of 400_8 which is gated onto the BUS RD to AIN of the ALU and then added to 0s for the high byte, and to B(07:00) for the low byte of the BIN port of the ALU.

NOTE

At this point, the B Register holds the high mantissa (shifted left in FP6). The previously separated exponent is in R15, and the sign is in the copy of the high B argument stored in BR back in FP1.

Word FP9 also BUTs the state of the ARGA flop which for this first pass is cleared. Coming out of FP9, the results of the test for exponent (D=0) in FP8 are felt.

If the exponent was 0 at that time, the flow goes to FP10 where a 0 is generated for the whole argument. The D Register is first zeroed and that is then used to 0 every appearance of the low B argument (BR, B, and R(13)). At FP11, the D is used to zero the high B argument in R(11) and the ARGA flop is set. Note that the ARGA BUT was in FP10, however, directing the flow now to FP13.

If, back in FP8, the exponent proved to be not equal to 0, the flow from FP9 is to FP12 where the pure high B mantissa in D is stored in R(11) and the ARGA flop is clocked before proceeding to FP13.

NOTE

The hidden 1, inserted at FP9, is used only if the exponent ≠ 0. If exponent *is* 0, the 1 is destroyed in FP10 and FP11.

In FP13, the sign of the B argument, still in BR, is saved in MSR and in EPS(N). The stack pointer is updated in the D Register to point to the high A argument, and then put back in R(DF) at P3 of this word. Once again the DATI is initiated and the clock is turned off to wait for memory to respond.

From FP13, the flow is to FP1 to fetch the A argument. The fact that ARGA flop is now set overrides the low bit of the register address and causes the even general registers to be selected. Now fARGA will select R(10) for storage of the high A argument, R(12) for the low A argument, and R(14) for storage of the A exponent. All other operations are identical to the fetching of the B arguments previously discussed, except that this time in FP8 the zeroness of the B exponent in EPS(Z) is transferred to ZB, and the zeroness of the A exponent is put into EPS(Z). Now the status of both exponents can be used in FMUL and FDIV to determine automatic generation of a 0 answer. Also, this time the ARGA flop is clocked to the clear state so that odd *or* even register selection can be determined by the ROM or the instruction.

Coming out of FP11 or FP12 in this pass, flow is to FP14 since ARGA was set at FP9. In this word, the high A argument (stored in DR at FP2) is sent to B and the decoding of the instruction is made (BUT(FINSTR I):EUB=15).

At FP15, the high A argument in B and the high B argument in MSR (stored in FP13) are sent through the ALU to be XORed. The MSR is also sent to the BR via the DMUX. The XOR is taken at this point to determine the sign for FMUL and FDIV.

From here, the flow exits to the appropriate page as determined by the BUT in FP14.

4.7.4.2 FADD and FSUB – Entrance to this flow is either at FADD or FSUB. As shown in the diagram, the operations are identical except for one extra step in floating subtract instructions. This is SUB0 at location 522 in which the high B argument in MSR (subtrahend) is complemented in D (merely to change sign), and put into BR. Flow is then diverted to the add flow at ADD0.

In a floating-add instruction, entrance through FADD is to ADD0 at location 520. Here the low B argument in R(13) is sent to B and the sign of the B argument (BR15) is tested for the branch out of ADD1. For FADD instructions, BR is loaded with the high B argument at FP15 and for FSUB instructions it is reloaded at SUB0.

In ADD1, the 2's complement of the low B argument is taken, DAD=10 inserts the carry into the ALU, and the carry-out is saved. This whole operation is done in case the B argument is negative. In that case, the branch is to ADD4 where the 2's complement of the low B argument is stored in R(13).

At ADD5, the high B argument (R(11)) is put into B and complemented in ADD6 with the carry-out of the previous 2's complement inserted to yield a 32-bit 2's complement. In ADD7, this complemented high B argument is stored back in R(11) and flow proceeds to ADD2.

If BR15 was (0) in ADD0, the B argument would have been positive, flow would have gone directly to ADD2, and complementing of the high B argument would not have occurred.

At ADD2, the low A argument (R(12)) is put in B and BR, and the high A argument sign is tested (DR15). The DR was loaded with the high A argument at FP2. The low A argument is then put in HSR in ADD3, the 2's complement is also taken (in case the argument is negative) and put into BR, and BUT(COUNT=0) is done for NPRs.

If the A argument is positive, the flow proceeds to ADD8 where the high A argument, uncomplemented, is put into the BR. If it is negative, ADD9 is entered instead where the complemented low A argument is stored in HSR and the high A argument (R(10)) is put into B. The 2's complement is then taken in ADD10 and put into BR. Note that either flow (DR15(0) or DR15(1)) results in the high A argument being stored in BR.

Now the exponents are considered (they were separated back in the fetching of the arguments), and in ADD11, the high A argument is put into the DR while the A exponent in R(14) is put into B. At the same time, the EPS(Z) bit is cleared for later use.

In ADD12, the A exponent in B is subtracted from the B exponent in R(15) and that difference (which is used as a shift count for lining up binary points) is put into B and BR. Note that between P2 and P3, D15, which indicates the relationship of argument exponents, is tested for the branch out of ADD13.

If, in ADD12, D15 was set, it indicated that the A exponent (R(14)) was greater than the B exponent (R(15)), and in that case, positions of all arguments and exponents must be swapped. This is necessary later on when binary points are lined up so that the proper argument is in position to be shifted.

ADD13 performs a 2's complement in case A>B, and the flow proceeds to either FADDA exit if B≥A, or to ADD14 where the general swapping operation begins. Note that if the exponents are equal (D=0), the setting of EPS(Z) in this word (ADD13) indicates that fact.

If A and B arguments must change places, ADD14 moves the A exponent in D (actual count) to BR, ADD15 moves it from BR to the count, and moves the low A (in HSR) to D. In ADD16, the low B argument is put into the BR, and in ADD39 the low A is put in R(13) from D.

NOTE
At this point, low A is where low B was.

At ADD17, low B in BR is put in HSR so that low B is now where low A used to be, and at the same time the high A in DR is sent to D.

In ADD18, the high B in R(11) is put in BR; in ADD19, it is put in the DR while the high A in D is put in R(11). Now both arguments are swapped and in ADD20 and ADD21 the A exponent is put where the B exponent was. This

completes the swap of arguments and exponents. Only the A exponent is moved in this operation. The B exponent is lost since their difference was determined in ADD12 and A has been determined to be the larger.

NOTE

If B was ⩾ A, R(15) would still contain the larger exponent (B).

Flow then proceeds through the exit FADDA to the next sheet of this flow.

The floating add and floating subtract flows continue on sheet 3 of the FIS flows, then on through FADDA to ADD22 at location 740. Here the low B argument is shifted left to provide an additional bit position on the low end for rounding. This is done by adding R(13) to itself, saving the carry-out, and putting the result in B and BR. This makes two extra rounding positions, as the first was gained in fetching the arguments.

NOTE

Although arguments are identified in the discussion from this point on, it must be remembered that due to swapping, what is called the low B here *may* be the low A. What is termed the low B should be thought of as the *larger* of the two arguments to avoid confusion.

Since a shift is performed, the counter is checked for a count greater than 30_8. This condition would exceed the range of the machine. The mantissa consists of 24 bits including the hidden 1 ($24_{10}=30_8$). If the difference in exponents exceeds this, the hardware will not perform any shifting but will take the argument with the larger exponent as the answer before rounding and normalization.

The BUT (COUNT>30):FUB1, EUB3 is made at P2 of ADD22, and at P3 of that word, the count is decremented.

ADD23 performs the same shift of the high B argument by adding R(11) to itself with the carry-in inserted from the previous word, yielding a resultant 32-bit left shift of which only 24 bits are looked at. The high bits are considered sign extension.

Coming out of ADD22, if the count was sensed greater than 30_8, the path to ADD29 is taken, thereby bypassing the binary point alignment procedure and putting the low answer in HSR and the high answer in BR. BUT COUNT=0 accommodates NPR servicing and the flow proceeds to ADD33. Note that this enters the flow after the steps that would have been taken if the exponents had been in range. If this were the case, flow would be to ADD24 from ADD23.

At that point in the flow, the DR (high mantissa) is concatenated with HSR (low mantissa). When these bits are shifted right, the sign of the DR must be duplicated. The statement at P1 (ADD24) performs that function by placing DR15 (sign) into BR00. As shifting continues, the state of DR15 is duplicated. The BUT in this word is testing for the equality of exponents. EPS(Z) and ZB have previously been set to indicate this equality. If the exponents *are* equal, EPS(Z) will be set, no shifting due to exponent difference is required, and the flow is to ADD28 through ADD25 where the count is decremented again. Note that no shifting due to *exponent difference* is required, but that binary points still require lining up due to the two extra rounding bits already added to the low end of the B argument. Thus, the A argument (that picked up one extra bit in fetching) must pick up an additional extra bit to be aligned with the B argument. This is done in ADD28 where the high A argument in DR and the low A argument in HSR are shifted left one place. HSR00 picks up a 0 from MSR15. This word then carries off to ADD30 to begin the add operation.

If as a result of the BUT in ADD24 the exponents were determined not to be equal, the indication is that shifting is required to align the binary points. In this event, the flow would proceed through a decrement in ADD25 to ADD26. The count, however, had also been decremented in ADD22 so that the original difference in exponents was reduced by one. Thus, if the BUT(COUNT=0) in ADD25 is true, an original difference of one in exponents existed and no alignment is required; i.e., an additional bit position was picked up on the B argument in ADD22 and, by not shifting the A argument right one place to exhaust that difference, an extra place on the A argument has effectively been gained. In this event, the flow out of ADD26 will not be to ADD27 but rather to ADD30.

In the event that the result of the BUT(COUNT=0) in ADD25 was $\neq 0$ (note: this BUS is after the decrement in ADD22 but before the decrement in ADD25), the decrement in ADD25 will cause the BUT in ADD26 to indicate that an original difference of two existed and just one pass is required through ADD27, thereby shifting the A argument one additional position to account for the extra bit on the low end.

In ADD27, the number of passes will always be one less than the original count for the reasons just stated, and in the example just given, the decrement in ADD26 will carry the flow out of ADD27 to ADD30.

An add of the low A and low B arguments is performed in ADD30 and the results (low answer) are stored in BR. The low answer is then sent to HSR in ADD31 and the high B argument (R(11)) is put in B.

An add of the high A and high B arguments is performed in ADD32 and the results (high answer) are stored in BR. The high answer is sent to DR in ADD33 and the low answer in HSR is put in B. The sign of the answer (BR15) is put in EPS(N).

A 2's complement of the low answer in B is taken at ADD34 and stored in the BR. This is done in case the answer is negative. At the same time, BUT(BR15) is made to test the sign in the high answer for the branch out of ADD35.

In ADD35, P2, the high answer in DR is ORed with the low answer in B through the ALU to D. This is done to accommodate the BUT(D=0) test in the first word in the normalize flow. At P3, the high answer in DR is sent through the DMUX to B.

At this point, the effects of the sign of the answer are felt. If the answer is positive, it is shifted right in ADD38 to get rid of the second extra bit on the low end and the flow exits to NORMALIZE. If the answer is negative, the 2's complement of the high answer is taken in ADD36, and in ADD37, the answer is put in DR and B for the ORing operation again in ADD35. This time the BUT (BR15) will find a positive answer (because of the complementation) and exit through ADD38 to NORMALIZE.

There are exits in this flow for bus requests. These tests, made by hardware assertion of GPC=7, are not made on the BUT MUX. This code will assert bit 5 in the UPP and cause the ROM to branch to BRQ if a bus request has been clocked in. The bus data cycle master sync would have clocked the BR request flags in the processor. The BUT(COUNT=0) in the option also clocks that flag and throughout these flows that BUT appears periodically whenever the length of time for an operation has taken a considerable period.

If a bus request had been flagged at ADD25 when the flow was to proceed to ADD28 for example, the GPC=7 would have been asserted in ADD25, causing bit 5 of the UPP to be asserted thereby changing the next address from 717 (ADD30) to 757 (BRQ1).

If the flow was from ADD25 through ADD26 to ADD27, rather than go to 713 (ADD27), the flow would proceed to 753 (BRQ0). Also in ADD27, this code is asserted for each pass through the loop so that anytime a bus request is present, the flow will immediately branch to BRQ0.

When a bus request is sensed by a GPC=7 code, the instruction is aborted, the stack pointer is backed up to the high B argument, and the PC is backed up to the floating instruction. This allows return to a restart of the floating instruction after servicing the request.

Both BRQ0 and BRQ1 perform this same operation by generating a constant of 6_8 (fCON=2) to decrement the pointer. It exits through BRQ to BRQ4 at location 755 where the constant in B is subtracted from R(DF), the stack pointer. UPP8 is cleared to switch ROMs and word BRQ5 decrements the PC by 2 to point to the floating instruction.

The final exit is to SERVICE C in the KD11-A flows.

FADD Example

5000/FADD 3	R [3] = 7000
	7000/040000 Hi B 7002/000000 Lo B 7004/040000 Hi A 7006/000000 Lo A
Consider hidden bit:	Bop fraction = 1/2 exp = 0 Aop fraction = 1/2 exp = 0
Arithmetic:	A + B = 1/2 X 2^0 + 1/2 X 2^0 = 1 X 2^0 = 1
Since hidden bit is still implied in result:	1 = 2^1 X 1/2 Therefore the result stored must be: 7004/040200 Hi ans 7006/000000 Lo ans

The following chart lists the contents of the various registers for each flow diagram step of this example.

Step	B	D	BR	DR	HSR	MSR	Registers & Notes
FP1	040000		040000				R [11] = 040000
FP2		007002		040000			R [3] = 7002
FP3	000000						R [13] = 000000
FP4		000000					R [13] = 000000 EPS(C) = 0
FP5	040000						
FP6	100000	100000					R [11] = 100000
FP7		000200					
FP8							R [15] = 200; EPS(Z) = 0, ZB = 0
FP9		000400					Insert hidden bit in D
FP12							R [11] = 400 ARGA ← 1
FP13		007004				040000	EPS(N) ← 0 R [3] = 7004
FP1	040000		040000				R [10] = 040000
FP2				040000			R [3] = 7006
FP3	000000						R [12] = 000000
FP4		000000					R [12] = 000000, EPS(C) = 0
FP5	040000						
FP6	100000	100000					R [10] = 100000
FP7		000200					
FP8							R [14] = 200; EPS(Z) = 0, ZB = 0
FP9		000400					Insert hidden bit
FP12							R [10] = 000400 ARGA ← R
FP14	040000	000000	040000				

Step	B	D	BR	DR	HSR	MSR		Registers & Notes
ADD0	000000							
ADD1		000000						EPS(C) = 0
ADD2	000000		000000					
ADD3		000000	000000		000000			EPS(C) = 0
ADD8			000400					
ADD11	000200			000400				EPS(Z) ← 0
ADD12	000000	000000	000000					
ADD13		000000		000400	000000	count=0		EPS(Z) = 1 DR = HI A HSR = LO A FRACT
ADD22	000000	000000	000000					EPS(C) = 0
ADD23		001000						R [11] = 001000
ADD24			000000					
ADD25								
ADD28				001000	000000			
ADD30		000000	000000					EPS(C) = 0
ADD31	001000				000000			
ADD32		002000	002000					
ADD33	000000			002000				EPS(N) = 0
ADD34		000000	000000					EPS(C) = 0
ADD35	002000	002000						
ADD38				001000	000000			Go to normalize
NOM0	000000		000000					
NOM1		000201						R [15] = 201
NOM4								DR9(1) so do not normalize
NOM7				000400	000000			
NOM8		000001	000001					Round up EPS(C) = 0
NOM9					000001			
NOM10		000400	000400					
NOM11				000400				
NOM12				000200	000000			
NOM13	000201	000201	000201					
EXI0		000000	040200					EPS(C) ← 0, EPS(Z) = 1
EXI1		000244						
EXI2								R [14] = 244
EXI7		000000					*	DATO, Lo Ans to 7006
EXI8	040200							
EXI9		007004						R [3] = 7004
EXI10		040200					*	DATO, Lo Ans to 7006
EXI11								EPS(N) = 0, EPS(Z) = 0, EPS(V,C) = 0
ASH15								PS(N:C) ← EPS(N:C)
ASH21								NO-OP
SERVICE C								

*Here we agree with original calculation from sheet 1 7004/040200
7006/000000.

4.7.4.3 FMUL – The floating multiply flow is shown on sheet 4 of the FIS flows and in Figure 4-12. Entry to this flow is through FMUL to FML0 at location 524. In this word, the contents of the D Register is put into the BR. This is the result of the XOR of B and MSR performed on sheet 1 at FP15 prior to entry and represents the sign of the answer.

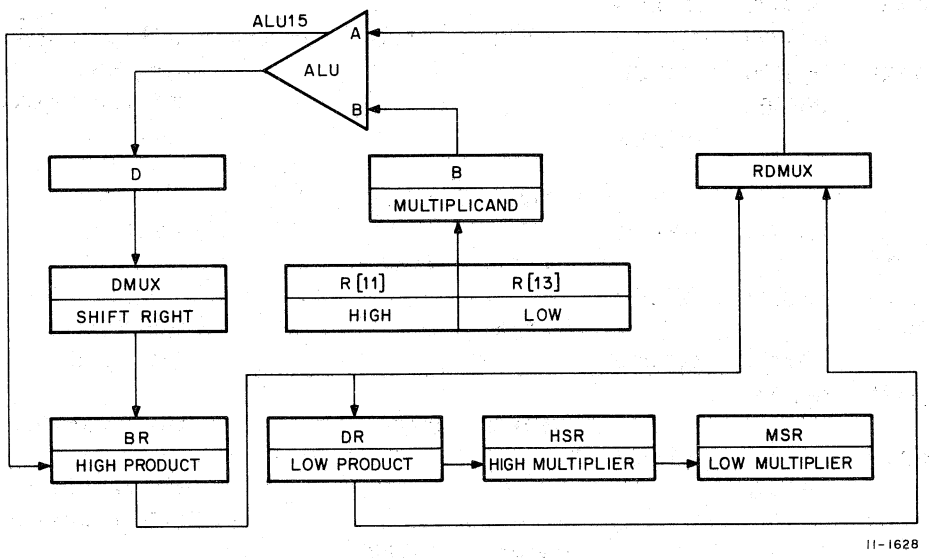


Figure 4-12 FMUL Flow, Block Diagram

In FML1, the B exponent (R(14)) is put in B and the sign of the answer (BR15) is put in EPS(N). The zeroness of the A and B exponents is tested with BUT(ZB+EPS(Z)). These bits were set while fetching the exponents in FP8.

Word FML2 adds the A and B exponents and then proceeds to either FML3 or FML4. Note that adding two exponents, each of which is expressed in excess 200 notation, yields a result that is in excess 400 notation. This will be corrected in subsequent steps.

If the result of the BUT in FML1 indicated that one or both exponents (arguments) was equal to 0, the flow is taken to FML3 where the answer is zeroed before exiting to ZERO A on the NORMALIZE flow. In this case, the fact that the addition of exponents produced excess 400 notation has no meaning and is ignored.

If neither argument was determined in FML1 to be equal to 0, the flow is to FML4 where the exponent is corrected to excess 200 notation by forming a constant of 200. That value is then subtracted in FML5. Note that the subtraction includes a "MINUS 1". Subtracting this additional 1 is done to accommodate the entry to normalize.

The multiply loop count is formed in FML6 as generated by fCON=3 (30_8 or 24_{10}) so that once the multiply loop is entered at FMUL11, the hardware will keep track of the number of passes through the loop until all 24 bits of significant mantissa have been monitored.

FML7 loads the count and puts the low multiplier (R(12)) into BR. This proceeds to FML8 where the low multiplier is transferred from BR to MSR to make room for the high multiplier (R(10)) from which it is put in HSR in FML9, where the BR is also cleared. At this point, the full multiplier is in MSR concatenated with HSR.

At FML10, the zeroed condition of BR is used to clear the DR, and the low multiplicand (R(13)) is put into B. This sets up the full 64-bit concatenation with the BR and DR cleared ready to receive the product, and with the multiplier in HSR and MSR. The flow is then to the entry of the multiply loop at FML11.

In this word, a shift right through the DMUX is done with D(C) getting ALU15. The D Register gets the low partial product (BR) and the count is decremented. Note that the state of MSR01 is tested in this word rather than MSR00. This is because MSR00 is actually the extra bit picked up by shifting during the fetching of the arguments on sheet 1 of these flows and is consequently not significant at this time.

On each pass through the multiply loop, this bit position always contains the current least significant bit of the multiplier. It is this bit that is used to determine whether to add and then shift or to just shift without adding. Whenever MSR01 is a (1), the multiplicand must be added to the partial product before shifting. If it is (0), a simple shift of the partial product and multiplier is executed.

In this multiply loop, the flow is from FML12 to FML14 through FML17 and back to FML11 whenever MSR01 is a (1); or whenever MSR01 is a (0), the route is FML12 to FML13 and then back to FML11. In each pass the count is tested, and when COUNT=0, the loop is exited and flow is to FML18 to store the products for normalization.

There are two bus request escapes. These are BRQ2 (653) and BRQ3 (657). BRQ2 is used for breakouts during any pass through the loop except the last. On the last time through, however, coming out of FML13, the base address 613 can be modified from two sources at the same time. The fact that COUNT=0 modifies 613 to 617 (FML18) and a bus request can modify *that* address to 657 or BRQ3. Once entered, the BRQ routine is identical to that described for FADD and FSUB.

During the multiply loop, data is swapped from register to register to accommodate the instantaneous needs of the operation. The low multiplicand is always being added to the partial product held in the DR and the BR. As the operation progresses, the low multiplicand is added to the DR, the carry is saved, that sum is loaded into the D Register, and the BR is loaded into the DR. The high part is brought up where it can be operated on, the two halves are added, and then everything is moved around again.

In FML14 and FML15, the low multiplicand (B←R(13)) is added to the partial product (D←DR PLUS B:EPS(C)←COUT 15). The contents of BR are saved in the DR to make room for the low partial product in D.

At FML16, the high multiplicand (R(12)) is put into the B Register, and at FML17, the rest of the double-precision add is done with the previous carry-out (EPS(C)) being used as the carry-in. This is sent to D. The low partial product in BR is put back in DR and the high partial product in D is put back in BR. The flow then goes back to FML11 to look at the next LSB of the multiplier and continues for a count of 30_8, unless interrupted by a bus request.

When the count is exhausted, the flow proceeds to FML18 where at P2 the low product in DR is put into the D Register and the high product is put into the DR Register. At P3 of that word, the low product is transferred from D to BR.

In FML19, the low product in BR is sent to the HSR while the B and BR Registers are cleared. This results in the final assembly of the product in the DR and HSR Registers concatenated where it needs to be for the normalization process.

4.7.4.4 FDIV – The floating divide flow is shown on sheet 5 of the FIS flows and in Figure 4-13. Entrance is through FDIV to FDV0 at location 526. As in multiply, this word takes the XOR of the high A and B arguments put in D at FP15 and loads that result (sign of the answer) into BR.

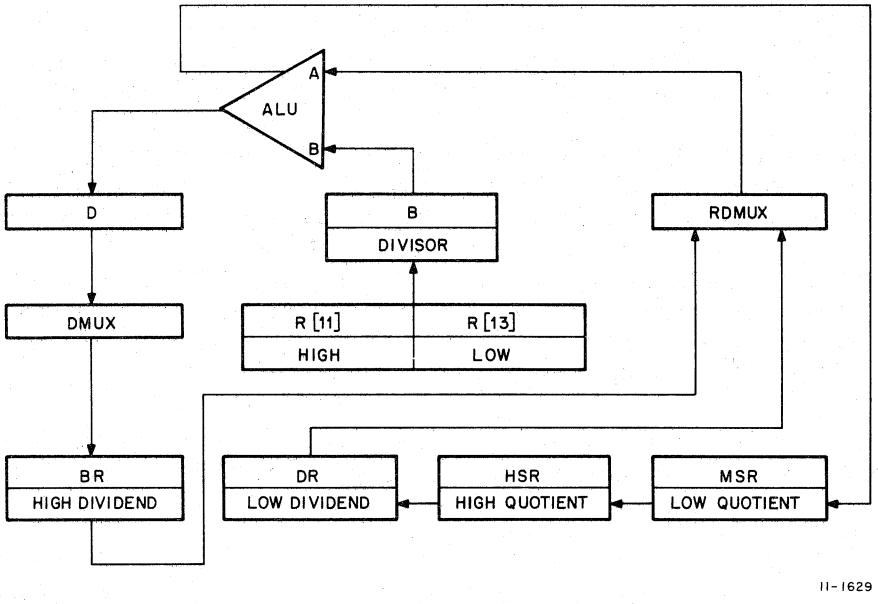


Figure 4-13 FDIV Flow Block Diagram

Word FDV1 sends the B exponent R(15) to B, and the sign of the mantissa (BR15) is put in EPS(N). The BUT in this word differs from the similar test in FMUL in that this is the ANDing of ZB and EPS(Z) whereas in multiply it was an OR function. This test of the A and B exponents determines whether either the divisor or the dividend, or neither is equal to 0. If the divisor is 0, an underflow is indicated; and if the dividend is 0, a 0 answer will be generated without going through the divide loop.

In FDV2, the B exponent (divisor) put in the B Register in FDV1 is subtracted from the A exponent (dividend) in R(14) and the difference is put in B. This will be the "initial" exponent, so called because it has not as yet been modified by normalization. This subtraction removes the excess 200_8 factor in the notation and will be corrected later.

If, at FDV1, the ZB flop was set, the indication is that the divisor (B argument) is 0 and flow is to FDV3 where the trap vector for the floating point (244) is formed by fCON=1 and the EPS(C) bit is set to indicate division by 0. The vector is stored in FDV4 and exit is to the underflow routine on sheet 6 of these flows.

If, at FDV1, the indication was that the dividend (A argument) was 0 (ZB=0 and EPS(Z)=1), flow is to FDV5 where a 0 answer is generated by zeroing D, BR, B, and R(15), and then exiting to ZERO A on sheet 6.

If, at FDV1, both ZB and EPS(Z) were found to be clear, this would indicate that neither the dividend nor the divisor were 0 and flow would proceed to FDV6. Here the exponent is returned to excess 200_8 notation by adding 200_8 to the difference in exponents. GPC=6 generates the constant 200_8.

Word FDV7 stores the corrected exponent and FDV8 forms the constant 32_8 or 26_8, (the number of times the divide loop must be executed).

At FDV9, the count is loaded from the BR, and BR gets the low dividend (R(12)). The low dividend is then put in the DR at FDV10 and the BR is cleared. The fact that BR is all 0s is then used in FDV11 to clear the HSR and MSR after which the high dividend (R(10)) is put into BR. FDV23 then puts the low divisor (R(13)) into B. The data is now set up for the first division step in FDV24.

The first division step is always a subtract operation, and in FDV24, the low half of a double-precision subtract of the low arguments is performed. The high dividend (BR) goes to DR, the low divisor (B) is subtracted from the low dividend (DR), the carry-out is saved in EPS(C), and the result is put in BR.

In FDV21, the high divisor (R(11)) is put in B while the count is decremented before proceeding to FDV22. Here the result of the low subtraction (BR) is put in DR. At the same time, the high divisor (B) is subtracted from the high dividend (DR), and the carry-in is inserted from the previous carry-out. At the same time, the carry-out 15 from this result is stored in EPS(C) which will now become the MSB of the quotient. Later on this bit will shift into the MSR Register as part of the quotient. At P3 of FDV22, the result of this subtraction (high result) is put into BR. The BUT(COUNT=0) is done both for NPRs and to see if this is the last time through the loop.

The flow here is to FDV16 where everything is shifted left. Note that these registers are concatenated and that EPS(C) goes into MSR00. That is the carry-out from the subtraction that indicates whether or not the division step was successful. If it is, a carry-out is seen and a (1) is shifted into the answer. If the step is not successful, no carry-out occurs and a 0 is shifted into the answer.

If this is not the last pass through the loop (COUNT≠0), flow is to FDV12 where the low divisor (R(13)) is put in B and the BUT for BRQ is made (GPC=7).

At FDV13, the division step for the low dividend is done. The BR is put into DR, then DR and B are processed according to the function of MSR00 (see table at bottom of sheet). MSR00, remember, is the result of the last subtraction or addition and is an indication of whether division was successful or not.

On each pass through the divide loop, the hardware determines whether the last subtraction was successful. If it was, a subtraction at FDV13 and FDV22 will occur on the next pass through the loop. If it was not, an addition at FDV13 and FDV15 occurs on the next pass. If a carry-out occurs, a subtraction is tried on the next pass and a 1 is inserted in the answer. If no carry-out occurred, a 0 is inserted.

Each pass decrements the count and the loop continues until the count is exhausted. Note that a BUT for COUNT=0 is done in either FDV22 or FDV15 since in the last pass the flow could be through either word.

When COUNT=0, the flow is to FDV17 to set up the quotients for normalization. In FDV17 and FDV18, the high quotient in HSR is put in DR. In FDV19 and FDV20, the low quotient in MSR is put in HSR. The last operation before exit to NORMALIZE is to zero the B and BR Registers so that they may be used in the rounding routine on sheet 6 of these flows.

The one BRQ breakout facility in this flow is tested at FDV12 where, if a bus request has been clocked in, the UPP Register address is modified from 731 to 771 (BRQ6).

4.7.4.5 Normalize, Round and Store – Sheet 6 of the FIS flows contains the routines for normalization, rounding, and storing; the flows for FADD, FSUB, FMUL, and FDIV all exit to this flow before storing their answers. Their points of entry differ, however, depending upon the instruction being performed.

Entrance to ZERO A is from FMUL or FDIV (the only instructions that detect whether one or the other argument equals 0 so that a 0 answer may be stored).

Upon entering at ZERO A, the 0 answer has already been generated and flow is to NOM3 at location 576. Here the BR, already cleared, is used to 0 the HSR and DR. BR15 is used to 0 the sign (EPS(N)). The FC1BUS is set to enable a DATO. Note that it is enabled one word early because it is double buffered by the FIS U Register and CPU U Register.

The statement BA←R(DF);DAD=6 is done to check overflow. R(DF) is the stack pointer and if it should happen to be register 6, as specified by the destination field of the IR, a decrement into a protected area could occur later on

when answers are stored. The BA is loaded here so that the CLOCK BA signal will be generated which, with DAD=6, will set the Check Overflow flag and check for overflow if R(DF) is register 6. Flow then is to the STORE routine described later at the end of this discussion.

Entry at NORMALIZE is from the FADD or FSUB flow and enters word NOM0 at location 727. In this word, the BR and B Registers are zeroed for use later in rounding. BUT(D=0) tests for a 0 answer as set up previously in ADD35. If D=0, it indicates that a 0 answer should be generated and the flow is through NOM1 and NOM2 where a 0 answer is generated.

Entry at NORMALIZE A is from the FMUL or FDIV flows and enters word NOM1 at location 554. This entry bypasses the zeroing action in NOM0 because this has already been done in FMUL or FDIV.

In NOM1, the exponent (R(15)) is adjusted by incrementation in D and restored to R(15). In that same word, the BUT(NORMALIZED) test is made with GPC=1 also asserted. This combination looks at DR09 for the branch out of NOM4. DR09 is the MSB of the mantissa and, if it is set, it indicates that the mantissa is normalized. If it is 0, the mantissa is not normalized.

NOTE

Most BUTs in this option look for a condition to be asserted to OR a 1 into the ROM address. In this case, assertion does not modify (705) whereas non-assertion does (707).

Flow is then directly to NOM4 and not to NOM2 because the BUT(D=0) is not felt by the NORMALIZE A entry. NOM4 sets up the R6 overflow check as described for NOM3 and takes the appropriate normalization branch.

If the BUT in NOM1 indicated that the mantissa was *not* normalized, NOM5 is next where the exponent is decremented because the mantissa is shifted left. GPC=5 allows HSR15 to be concatenated with DR00. As HSR is shifted left one place, a 0 is brought in. Note that the BUT(NORMALIZED) does not assert GPC=1 in this word. This causes the hardware to check normalization before the shift by looking at DR08. If DR08 is set at this time, DR09 will be set after the shift and, as a result, the flow will be to NOM7 after the decremented exponent is stored in NOM6. Of course it could take several passes to normalize the exponent (up to 31_8 places), so the BUT(COUNT=0) in NOM6 serves to clock NPRs. Note also that the exponent is decremented for each pass.

At NOM7, the answer is shifted right one place to put the extra bit on the low end in position for rounding. NOM8 rounds the low part of the answer by adding 1 and the carry-out is saved in EPS(C). In NOM9, the rounded low answer is returned to HSR.

At NOM10, the high answer is rounded by adding 0 to the DR, and bringing in the previous carry-out (EPS(C)) as the carry-in. In NOM11, the result is sent via the BR to DR. By adding 0 to the high part and bringing in the carry, the effect of adding the 1 may ripple up from the low answer to the high answer via the carry. At this point, the rounded high answer is in the DR and the rounded low answer is in the HSR.

Now that rounding has been done, the extra bit on the low end is no longer needed so in NOM12 it is dropped by shifting everything right one place. The BR is also cleared. At the same time, normalization is checked by looking at DR09 (at this point DR08 is the MSB). This is to be sure that it is still normalized after rounding. If, as a result of rounding, the added 1 had rippled all the way across, the answer would have become unnormalized. To become renormalized, an additional shift is required along with an increment of the exponent. NOM13 sends the exponent (R(13)) to the D, BR, and B Registers in case adjustment is not required.

Coming out of NOM13, the effects of the BUT in NOM12 are felt and, if the answer was not still normalized (DR09(1)), NOM14 is entered to effect renormalization and to increment the exponent. The adjusted exponent is then stored in R(15), B and BR.

If, however, the answer *was* normalized (DR09(0)), the ROM address is modified to 542 and the flow exits to EXI0.

NOTE

At this point, the answer has been rounded and normalized, the exponent has been adjusted to the correct value, and the hardware is ready to assemble the answer and store it.

EXI0 assembles the high answer by putting the sign bit (EPS(N)) in BR15, the exponent (BR(07:00)) in BR(14:07), and the high mantissa minus the hidden bit (DR(06:00)) in BR(06:00).

Word EXI14 sets up the D Register for the BUT to be made in EXI1 for underflow, overflow, or store. This is done by zeroing the upper 8 bits of the D Register with SBC=00. The EPS(Z) bit is loaded with whether or not the exponent is equal to 0 as set up at P2 of NOM13. This establishes one of the conditions for underflow. The low 8 bits of the D Register are loaded with the high byte of the register that held the exponent (R(15)) (previously loaded into the B Register at P3 of NOM13 or NOM14). In addition EPS(C) is zeroed.

At EXI1, the trap vector 244 is formed (fCON=2) in case underflow or overflow are indicated by the BUT in that word, and in EXI2 that vector is stored along with setting the DATO control bit (FC1BUS).

If, at EXI1, the D Register contains any data, overflow exists indicating that what was the high byte of the exponent had some carryover from the low byte of the exponent. In this event, flow will be to EXI12. Underflow is indicated if either EPS(Z) is set, indicating a 0 exponent; or B15 is set, indicating that decrementation of the exponent produced a negative number. In this case, the flow will be to EXI3. If neither underflow or overflow are indicated, the answer is determined to be legal and flow is to EXI7 for a store operation.

NOTE

If overflow or underflow are indicated, the FC1BIT set in EXI2 will have no effect since on the next bus cycle the basic ROM is used and the FIS U Register is cleared.

If overflow is indicated, EXI12 and EXI14 load the EPS(N) bit via BR with a 0 (top bit of 244_8 in D is a 0). If underflow is indicated, EXI4 generates the stack pointer adjustment constant of 6, zeros the EPS(Z) bit, and sets the EPS(V) bit. The condition codes for overflow and underflow are as follows:

Condition Codes for Overflow and Underflow

	OVFL	UNFL	FDIV By Zero
EPS (N)	0	1	1
EPS (Z)	0	0	0
EPS (V)	1	1	1
EPS (C)	0	0	1

By referring to the condition codes in the table above, it can be seen that all codes are properly set by these word combinations, including the divide by 0 combination in which the EPS(C) bit was set prior to entry at UNFL A. In this same path then, EXI15 moves EPS to the Processor Status Word and clears the extension ROM enable bit UPP8 while EXI16 adjusts the stack pointer and exits to KD11-A Trap D flow.

If the STORE path is taken, flow is to EXI7 where the low answer in HSR is sent to D, a DATO is initiated, and the clock is shut off to wait for the memory response.

At EXI18, the high answer assembled in BR in word EXI0 is brought into the B Register, and at EXI19 the stack pointer is decremented by 2 and put into the BA. The DAD=6 checks for a register 6 stack overflow, the modified stack pointer is put back in R(DF), and the FC1BUS is set.

At EXI10, the assembled high answer (sign, exponent, and high mantissa) in B is stored in D, the DATO is initiated, and the clock is turned off. Then at EXI11 the local condition codes are set before exiting to MOVE EPS.

4.8 LOGIC DESCRIPTIONS

The KE11-E logic diagrams are shown in drawing D-CS-M7238-0-1, sheets 2 through 9. The sheets are designated KE-2 through KE-9, as follows:

KE-2	BR(15:00),DR(15:00)
KE-3	RDMUX(15:00)
KE-4	EUBC MUX AND CONTROL
KE-5	CLOCKING AND CONTROL
KE-6	EPS AND COUNTER
KE-7	KE ROM AND U WORD REGISTER
KE-8	KD ROM EXPANSION
KE-9	KD ROM EXPANSION AND CONNECTORS
KE-10, 17	EIS ROM LISTING

The KE11-F logic diagrams are shown in drawing D-CS-M7239-0-1, sheets 2 through 4. The drawings are designated KF-2 through KF-4, as follows:

KF-2	HSR & MSR
KF-3	FRDMUX(15:00)
KF-4	ROM & CONTROL
KF-5, 12	FIS ROM LISTING

In these paragraphs, the logic is described in this sequence for convenience only. The sequence bears no relationship to their logical arrangement. These descriptions, together with the Glossary in Appendix A, provide an adequate description of the logic diagrams associated with the KE11-E and F Options.

4.8.1 Basic CPU Timing

As the KE11-E and KE11-F operate on the basic timing of the KD11-A, a brief description of this timing is given at this point.

There are three basic timing cycle lengths used in the KD11-A. For a description of their generation refer to the *KD11-A Maintenance Manual* EK-KD11A-MM-001.

The three cycle lengths are designated CL1, CL2, and CL3. Cycles CL1 and CL2 have one clocking pulse which occurs at the end of the cycle. Cycle CL3 has two pulses associated with it. The time relationship of the cycle lengths and their respective pulses are shown in Figure 4-14. All clocking action takes place on the trailing edge of the clock pulse.

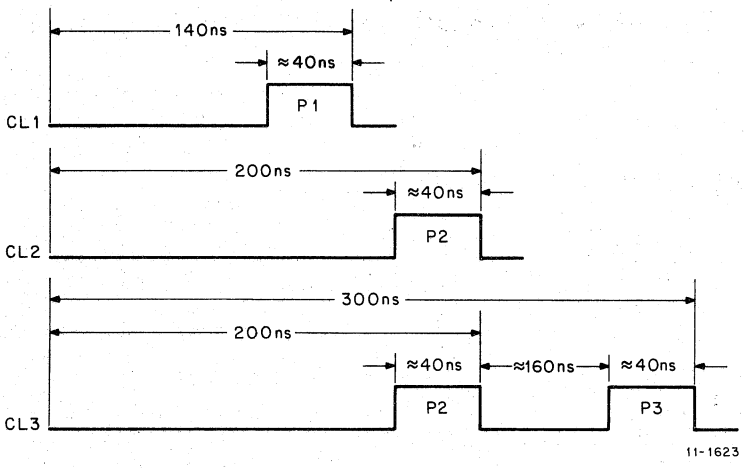


Figure 4-14 Basic KD11-A Timing

4.8.2 BR and DR Registers (Dwg KE-2)

The BR Register is a 16-bit holding register which receives data from the CPU via DMUX(15:00) H. The BR is used to hold data during the EIS and FIS instructions. This register is loaded on the trailing edge of P1 or P3 pulses.

The DR Register is a 16-bit left-right shift register which receives data from the BR. The DR is also used as a holding register. During the multiply instruction, the low product is shifted into the DR via DR15 as the multiplier is shifted out at DR00. During a divide, the quotient is shifted into the DR via DR00 while the low dividend is shifted out at DR15. Output signals are listed in Table 4-4.

Table 4-4
KE-2 Output Signals

Mnemonic	Description
BR(15:00) H	Output from the BR Register, fed to DR(15:00) and through the EIS option. Also to pins for distribution to EIS and FIS options.
DR(15:00) H	Output from the DR Register. Used in ASHC, MUL, and DIV operations in the EIS and in all FIS instructions.

On the left-hand side of this sheet are shown the three hex flip-flop registers, 74174s that comprise the BR Register. Inputs to this register are DMUX(15:00) coming from the basic machine. All input data from the KD11-A comes into this register from which it is distributed to the rest of the EIS and FIS options. It functions as a holding register and buffer.

The outputs of the BR Register feed four 74194s that comprise the DR Register. These are left-right shift registers having the ability to also be loaded in parallel. The DR is used extensively in ASHC, MUL, and DIV to shift and store data. Likewise, it is used in FIS instructions to temporarily store and shift data.

The shift input to bit 00 of the DR Register has several sources, some from the EIS and others from the FIS; one is enabled during an ASHC instruction.

The bit 00 input to the DR Register is important only if a left shift operation is being performed. In a right shift, this line is insignificant.

In ASHC, while shifting left, 0s must be shifted into the LSB. The DR holds the lower portion of the operand being shifted, and as the data is being shifted left, the ASHC L on the gate at E48 keeps a constant 0 asserted at the shift input so that each succeeding shift brings in another 0.

Another source for this shift input is at gate E42. The jumper W1 is present if just the EIS is installed and is removed if the FIS is installed as well. When inserted, it disables the upper input to the gate. When removed, a floating divide instruction in combination with GPC=5 L will cause a 0 to be shifted into the DR on a left shift operation.

This action can be referred to the flow diagrams in sheet 5 of the FIS flows at word FDV16. Here the high quotient is being assembled in the HSR as the concatenated low dividend is being shifted left in the DR. Everything is being shifted left after the add or subtract. A 0 is brought into the 0 shift-in of the DR Register. If GPC=5 is not true, 0s will be enabled through. If GPC=5 *is* true, HSR15 is enabled to the shift input of bit 0. Since the use of HSR15 as a shift input is not dependent upon any particular instruction, but rather is supplied by the GPC code, it can be used at anytime that particular combination is set in the ROM word.

The DR Register is clocked by E(P1+P2) H, generated on drawing KE-5 at location D-6. There is no signal in the basic machine called P1+P2, these two signals are ORed here for this function.

The BR Register is clocked by CLK BR H, generated on KE-5 at location C-6. It is a function of P1 or P3 and CLK BR(1) H generated on KE-7.

DR Register bit 15 shift input is used for shifting the register right. Normally BR00 is enabled into DR15. This is when the BR and DR are concatenated with the BR holding the higher bits. In this case, everything may be shifted right with BR00 going into DR15. During multiply instructions, however, ALU00 is used as the shift-in with ALU00 providing the partial answer being shifted into the DR.

There are two select lines on the DR Register, KE-7 SDR0 and KE-7 SDR1. The truth table for these two signals is given on the right-hand edge of sheet KE-2. If both select signals are low, nothing will occur when the register is clocked (No-Op). Note that the clock pulse is always present on any P1 or P2. A binary 1 combination produces a shift right, a binary 2 combination produces a shift left, and a binary 3 permits parallel loading of the register. These two bits are generated in the EIS ROM.

Both the DR and BR Register outputs are brought out to pins for use elsewhere in the options and for testing purposes.

The BR can be used for buffering the loading into either the count or DR in the EIS, or into either the HSR or MSR in the FIS option.

4.8.3 RDMUX (Dwg KE-3)

The RDMUX is a 16-bit-wide 4:1 multiplexer which selects one of four sets of 16-bit inputs to be fed to the CPU via the 16-bit wire-ORed BUS RD(15:00). Output signals are listed in Table 4-5.

Table 4-5
KE-3 Output Signals

Mnemonic	Description
BUS RD(15:00) L	Output of the RDMUX. Feeds data directly back to the processor.

This is the hardware that enables data onto the BUS RD to be sent back to the basic machine. These are 74153 dual 4:1 multiplexers controlled by combinations of SRDM1 and SRDM0 as generated in the Expansion U Word (see truth table on this sheet).

A binary 0 combination sends the EIS status (EPS(C,V,Z, and N)) back to the basic machine. This operation is called out in the flows as MOVE EPS. Note that in this mode the upper 12 bits of the multiplexer inputs are grounded.

A binary 1 sends the contents of the DR Register straight through and back to the basic machine.

A binary 2 concatenates the BR and DR Registers, essentially shifting both registers left, and either losing the high bit of the BR or using it elsewhere.

A binary 3 sends the BR Register straight through and back to the processor.

Note that these enables are set up one word before the enable for the 74H01 drivers where possible. This is the AND of EUPP8 (set if the option is enabled) and STRDM(1) H, another bit set in the ROM word. This allows putting data out to the RDMUX as quickly as possible without any timing problems. Note that the multiplexer strobe inputs (STB) are grounded (always enabled).

The outputs of the 74H01s feed BUS RD(15:00) of the CPU.

4.8.4 EUBC Control (Dwg KE-4)

This is the External Microbranch Control that serves as a supplement to the branch control logic in the basic machine. The EUB field of the KE11-E ROM is used to select various inputs to the EUBC multiplexer for testing in order to enable the microprogram to branch to alternate paths of flow. Output signals for this sheet are listed in Table 4-6.

Table 4-6
KE-4 Output Signals

Mnemonic	Description
EUBC(3*4)ENB L	Enables EUBC(4:3) MUX. Used as a partial enable on KE-5 8-D 7402 gate at E19.
D(15:00)=0 L	Inversion of D(15:00)=0 H from the basic machine. Is used by FIS board as part of OVFL/UNFL test on that board.
EUBC(4:1) L	Four bits that modify a base address on a microbranch test. They are sent to OR gates on the M7232 module in the basic machine.
IR=075xxxL	The Op code that indicates a floating instruction. It is sent to FIS board where it is gated and is returned as FIS INSTR L at location D-6. If true, allow the branch to the FIS flow.

In the upper left-hand corner of this sheet is the 8251 decoder which is enabled by a 07 condition in the upper two octal digits of the IR (IR(15:12)). It then decodes IR bits 11:09 to detect whether an EIS (070 through 074) or FIS (075) instruction is called for. Any other code is not recognized and results in a reserved instruction trap. The IR=075xxxL signal goes out to the FIS module decoder logic.

The four decoded instructions are fed to a 7420 NOT OR gate that yields EIS INSTR H used to feed the conditioning inputs of the input gates on the EUBC multiplexers.

The EUBC MUX comprises four multiplexers. The 74153 dual 4:1 multiplexer controls EUBC(4:3) while a single 8:1 74151 controls EUBC2. EUBC1 is controlled by two 74151s operated in tandem. When EUBC1 MUX B is operating, EUBC1 MUX A is disabled and vice versa. EUBC1 MUX A at E37 provides testing of octal combinations 0 through 7 of the EUBC field and EUBC MUX B at E24 tests combination 10 through 17 (see table at right-hand side of this sheet, KE-4). This provides 16 possible conditions that may be used to control EUBC1. Note that when the FIS option is installed, the EUBF consists of 5 bits. When it is *not* installed, the 74H10 AND gate at location C-3 on the drawing has pin 03 (EUBF4(0)H) held high via R2 (1K). There is an additional 8:1 multiplexer in the FIS to be described later that provides 8 more branch conditions for EUBC1 control. If EUBF4 from the FIS board is asserted, the EUBC1 MUX A is disabled.

The outputs of these multiplexers (EUBC(4:1)L) are sent to OR gates on the M7232 module in the basic machine where they are used to modify the base address on a branch.

Conditioning of the inputs to these multiplexers is effected by combinations of EUBF(4:0) as set in the Expansion U Word. These are fed to the multiplexers as selection signals. The operation performed by their combination may be derived by reference to the truth tables at the bottom of the page and the BUT chart at the right of the sheet (KE-4).

4.8.5 Control (Dwg KE-5)

This sheet contains much of the discrete logic used to generate the many control signals used throughout the EIS logic. It utilizes inputs from the ROM, together with clock pulses generated by the basic machine, to generate these control signals. Most of the input signals on this drawing are fairly obvious and as such are not described. Some are enables from the ROM word, others are covered elsewhere as interfacing signals. Output signals are described in Table 4-7.

In the center of the sheet is the 8251 GPC decoder with inputs GPC(2:0). These bits are set in the Expansion U Word.

The input logic in the upper right-hand corner of this sheet at location C-4 pertains to auxiliary ALU control where several conditions are monitored to determine what function the CPU ALU will perform. The jumper W2 is normally inserted unless the FIS option is installed. When removed, it allows the FIS option to control auxiliary ALU control also.

Table 4-7
KE-5 Output Signals

Mnemonic	Description
EXT P CLR TRAP L	Clears the Trap flag in the processor which was set as a result of EIS or FIS instruction being sensed during a FETCH.
CLK EPS(N,Z) H	Clocks the EPS N and Z bits. Made up of the enable bit of the EIS microregister and P1 or P2 from the CPU.
CLK UPP8 H	Is sent back to basic machine to clock UPP bit 8. Gated with an enable bit from the EIS ROM word and a clock pulse. Once the option has been enabled, bringing up the enable bit allows UPP bit 8 to be clocked clear.
CLK EU(88:57) H	This is the clock pulse for the external U Register bits 88 through 57. (Bits 88:81 are on the FIS board, M7239.)

Table 4-7 (Cont)
KE-5 Output Signals

Mnemonic	Description
ESALU(3:0) L	External Select ALU. These are the select bits for the ALU when operated in auxiliary control mode. Discrete ALU control is provided by bits in the ROM word where it is known ahead of time what function the ALU should perform. Auxiliary ALU control is provided in instances where the operation to be performed by the ALU is dependent upon incidental conditions of the operation. These signals then are made dependent upon combinational logic that decodes conditions within the operation. The functions performed by the ALU for the various states of these signals are given in a table to the right of the print (KE-5). Note that the four signals are really two signals, each of which are brought out to two separate pins.
ECIN00 H	External Carry-In to the CPU ALU. Used in auxiliary control mode of the ALU to insert a carry-in to correct the A MINUS B MINUS 1 function during subtraction.
GPC=2 L 1 L 7 L 6 L 5 L	Output combinations of the GPC decoder. These are used for special case applications such as operations to be performed only once or twice in a flow. Some of these signals go to the FIS board.
INH PS CLK L	Used to inhibit clocking all bits in the Processor Status Word except PS(N,Z,V,C). This signal is also used in the memory management option (KT11-D).
EUPP8 B H	External microprogram pointer bit 8. Used in the EIS and FIS options as an enable signal.
EUPP8 L	Same as above.
EUBC8 L	External microbranch code bit 8. Used as data in the basic machine to set or clear UPP8 when clocked by P CLK UPP8.
ECOMUXS0 L 1 L	External Carry-Out Mux. (Same source to two pins.) Selection signal to the 4:1 multiplexer on the output of the CPU ALU, causing ALU bit 15 to be selected into the Carry-Out Mux.
EUPP8 A L	Same as EUPP8 B H above. Duplication for loading purposes.
ENPRCLK L	External NPR Clock. Provides the clock for the NPR and Bus Request flags in the basic machine while the EIS and FIS options are active. Made up of the branch microtest BUT(COUNT=0) performed periodically in the flow, particularly in loops where long periods of time could be consumed.
E(P1+P2) H	The P1 and P2 from the basic machine used to clock the DR and the counter. Also clocks the MSR and HSR Registers in the FIS board M7239.
CLK EPS(V) H	A clock for the EPS(V) bit. Made up of CLK(V) (1) H from the EIS U Word gated with P1 or P2 from the CPU.
CLK BR H	Clock for the BR Register, enabled from the EIS U Word and P1 or P3 from the CPU.

Table 4-7 (Cont)
KE-5 Output Signals

Mnemonic	Description
LD COUNT L	Load Count. When true, loads the counter with BR(07:00).
CLK EPS(C) H	A clock for the EPS(C) bit. Gated with an enable by a bit from the EIS U Register and P1 or P2 from the CPU.
B EUPP8 A H	Buffered external microprogram pointer A. Another source for bit 8 of the ROM address to enable various points in the logic.
CLK COUNT H	This is essentially the end pulse in each cycle length. Generated by an enable from the EIS U Register and with the fact that the counter is not equal to 0. When the count goes to 0, this clock is disabled preventing any further counting during testing of that condition.

4.8.6 EPS and Count (Dwg KE-6)

This drawing contains the external processor status, which records the condition codes of the EIS and FIS instructions; and the count, an 8-bit up-down counter used to count the number of shifts for the ASH and ASHC instructions and to keep track of the number of steps in the MUL and DIV instructions. The count is loaded from the BR. The logic contains multiplexers for gating information into the external status. Output signals are listed in Table 4-8.

Table 4-8
KE-6 Output Signals

Mnemonic	Description
EPS(N) (1) H	The output of the external processor status N bit. Used to store the sign of operands.
EPS(Z) (1) H	The output of the external processor status Z bit. Used to store whether an operand is equal to 0 or not.
EPS(V) (1) H	The output of the external processor status V bit. Used to record overflow conditions.
EPS(C) (1) H	The output of the external processor status C bit. Used to store carry data.
COUNT(7:0) (1) H	The outputs of the counter used to keep track of the number of steps executed in the EIS and FIS instructions. These signals also go to the FIS board (M7239).
COUNT=0 H	This signal is asserted when COUNT(5:0) is equal to zero.

The counter consists of two 74191s at E64 and E56, operated in tandem and fed by BR(07:00) H. It is an up/down counter that can be loaded by LD COUNT L. This is not a clock type load but rather a write type load in which the counter is loaded without a clock whenever pin 11 is low. Pin 05 is used to determine direction of count. Note that this signal is derived from bit 5 of the count through a 74H04 inverter at E52. When bit 5 is set, the output of the inverter is low, causing the counter to count up. When bit 5 is clear, the count is down. This is used in ASH so that a right shift will count up and a left shift will count down (always toward 0).

The counter always counts if it is clocked (CLK COUNT at pin 14). If the load input is low it will override the clock input. The clock is enabled by COUNT=0 L inverted. This way whenever the count has reached 0, counting ceases.

The 74H50 at E46, operated as an XOR of BR15 and BR14, is used during an ASH or ASHC left operation to forecast an impending change in sign. The output of this gate feeds an OR (7402) which is also fed by the EPS(V) bit. This provides a latch path for the EPS(V) flip-flop so that once it is set it will remain set regardless of what happens at the XOR from that point on.

Each output of the counter is fed to the FIS board (M7239) for use in branch tests during floating instructions.

There are three multiplexers on this sheet each used to set control flip-flops for each external status bit in the option. The flip-flop outputs go to the RDMUX and the EPS(Z) is also fed to the FIS board. The dual 4:1 multiplexers are used to control the bits. Because there were more than four conditions required to control the C bit, an additional half 74153 is ORed in to control the (EPS(C)) bit flip-flop. In the CV MUX at E53, the V portion of that multiplexer is always enabled, but the C portion is disabled by SDVM2(1) H when the C MUX at E66 is enabled.

The combinations of the basic select bits (SCVM(2:0) and SNZM(1:0)) required to perform multiplexing operations are listed with their results in the truth tables provided on this drawing.

4.8.7 KE ROM Word (Dwg KE-7)

This logic contains the basic KE ROM word (U80:57), 24 bits of ROM that control the KE11-E. All outputs from the ROMs are fed into 74174 hex registers except bit CLK UPP8 which is a discrete flip-flop. Output signals are listed in Table 4-9.

This sheet contains six 4-bit ROMs (23-BXXA2), feeding 24 bits into four 74174 hex register gates. One exception, EUPP8 at coordinate C-2, is fed to a discrete flip-flop. The enable for the ROMs is EUPP8 L. The signal CLR EU H is just a pull up for the registers. Registers are clocked by CLK EU (33:57) H generated on KE-5.

The boxes labeled 13-11003-02 on the outputs of each ROM bit are resistor divider networks which are part of a 16-pin IC package (see table on this drawing (KE-7) for schematic and values).

Table 4-9
KE-7 Output Signals

Mnemonic	Description
CLK BR (1) H	An enable for the clocking of the BR Register. Gated with P3 on KE-5.
EUBF0(1) H 1 2 3	External Microbranch Field. Used on KE-4 for the EUBC MUX. Also used on FIS board for the one EUBC MUX on that board.
ECNT(1) H	Enable Count. An enable to clock the count gated with P END H on KE-5.
CLK UPP8(1) H	Enable for clocking bit 8 of the UPP. Gated with ECLK U L on KE-5.
LCNT(1) H	Load Count. An enable for the signal that is gated to load the counter. Gated with P1 and P2 on KE-5.

Table 4-9 (Cont)
KE-7 Output Signals

Mnemonic	Description
GPC0(1) H 1 2	The three bits in the GPC field of the U Word used on KE-5 as an input to the GPC decoder.
CLKC(1) H	Clock C. Used on KE-5 gated with P1 and P2 to generate the clocking signal for the EPS(C).
CLKV(1) H	Clock V. Enable for the EPS(V) bit clock.
CKLNZ(1) H	Clock NZ. Enable for the EPS(N,Z) bits clock.
SNZM0(1) H 1	Select bits for the N,Z multiplexer used on KE-6.
SCVM0(1) H 1 2	Same as above for the C, V multiplexer.
SDR0(1) H 1	Select bits for the DR Register that determine a shift left, a shift right, a load or No-Op. Used on KE-6.
SRDM0(1) H 1	Select bits for the RD MUX. Determines which MUX input is sent back to the basic machine via the BUS RD(15:00). Used on KE-3 and KF-3.
STRDM(1) H	Strobe RD Mux. Used on KE-3. Enables the 74H01 drivers to the BUS RD(15:00).

4.8.8 KD ROM Word (Dwgs KE-8 and KE-9)

This logic (2 sheets) contains only those ROM bits in the basic machine that are actively duplicated. Those that are not actively duplicated are driven low by the KE11-E during the time that the option is enabled. These outputs are wire-ORed with the outputs of the basic machine ROMs; and when the option is enabled (by UPP8 being set), these bits control the inputs to the basic machine U Register rather than those in the KD11-A ROM which is disabled by the setting of UPP8. Output signals are listed in Table 4-10.

Table 4-10
KE-8 and KE-9 Output Signals

Mnemonic	Description
BUS U(43:00) L	Microbus output signals. All go to the three Berg 40-pin connectors on the back of the module. There are three cables to the basic machine. The connectors are shown on KE-9.

An example of one ROM bit feeding two U Register bits is seen at coordinates C-6. Here one bit from the ROM is feeding two bits back to the basic machine through a pair of 74H01 gates (BUS U40 L and BUS U39 L). It is not necessary to actively duplicate all the bits in the basic machine, but they do have to be driven low if they are not used. As long as the option is enabled, these bits are driven low and always register 0s. These bits are the SPS field (a 3-bit field) in the basic machine U word from which only 2 codes are required in the EIS. The terminators for the BUS U bits are not shown on this sheet because they are located in the basic machine. There *is* a terminator, however, for the signal feeding the 74H01-1s on this drawing because that signal is not sent directly back to the basic machine. The 74H01-1 is an open collector gate and *its* terminator is back in the processor.

On sheet 9 at coordinates D-6, the jumper W3 must be removed when the FIS option is installed. This is used when fetching the floating arguments. A common flow is provided for fetching both arguments, and the first time through, the ROM always specifies an odd general register address in which to store the argument. The first time the odd address is used and the second time through the even address is used. This is done by input signal ARGA at D-7. It enables that gate when ANDed with an active option to drive BUS U9 low. This is the low bit of the register address that is being negated to yield an even register address.

4.8.9 HSR and MSR (Dwg KF-2)

This is the first drawing for the M7239 module constituting the FIS option. The same timing prevails for this option as that described for the FIS option (Paragraph 4.8.1). These paragraphs pertain to only those customers utilizing the Floating-Point Option.

Drawing KF-2 contains the HSR and MSR Registers. Both are left/right shift registers and both are fed from the BR in the EIS option. They are used as either holding registers or shift registers. Output signals are listed in Table 4-11.

Table 4-11
KF-2 Output Signals

Mnemonic	Description
MSR(15:00)(1) H	The outputs from the MSR left/right shift registers.
MSR00 L	The inverted output of MSR00. Used on KE-5 to determine auxiliary ALU functions for floating divide instructions.
HSR15 L	The inverted output of HSR15. Used on KE-2 as DR00 shift input data.
HSR(15:00)(1) H	The outputs from the HSR left/right shift registers.

These registers consist of 74194s fed in parallel by BR(15:00)(1) H. Both registers are clocked by E(P1+P2) H generated on the EIS board. This clock is always present at both registers but no action will occur until the proper select bits are present on the register. Selection is accomplished by a 3-bit combination of SHSR0(1) H, SHSR1(1) H, and SMSR(1) H. The HSR select bits are fed directly to the HSR Register, and are also gated into the MSR Register by the common signal SMSR(1) H. When only the SHSR signals are asserted, only the HSR is loaded. When SMSR is asserted as well, the MSR is also loaded. This is why in the flows whenever both registers are to be used, the MSR is loaded first and then the HSR is loaded without affecting the MSR.

The clear inputs to these registers are tied to pull ups. There are shift inputs to the high bit and low bit of both registers.

Generally, these registers are used to shift data. In FMUL, they are concatenated with the MSR being the lower register and the HSR the higher. In this application, HSR00(1) H enters the shift input of MSR15 at DS3, coordinates D-7. The low shift input to the MSR is EPS(C) at coordinates C-3 used in shifting left.

The high bit shift input to the HSR is DR00(1) H at coordinates B-7. In FMUL, the low DR bit will enter the high shift input of HSR. The low shift input of the HSR is either MSR15 gated with the code $\overline{GPC=5}$, or 0s. When GPC=5 is not present, 0s are shifted into the HSR.

4.8.10 FRDMUX(15:00) (Dwg KF-3)

The FRDMUX is a 16-bit-wide 4:1 multiplexer that selects one of four sets of 16-bit inputs to be fed to the CPU via the BUS RD(15:00). This operates similarly to the RD MUX in the EIS option, using the same select bits as used in that logic with a separate strobe (STFRDM) to enable this set of drivers instead of the EIS drivers. Output signals are listed in Table 4-12.

Table 4-12
KF-3 Output Signals

Mnemonic	Description
BUS RD(15:00) L	The outputs of the 74H01-1 drivers to the BUS RD over which data from the FIS option is transferred to the CPU.

Outputs of the dual 4:1 multiplexers (74153) are fed to open collector NAND gates (74H01-1). These then drive the BUS RD where the data is transferred back to the processor. As in the RDMUX of the EIS, the FRDMUX is selected by combinations of SRDM1(1) H and SRDM0(1) H but information is not transferred to the BUS RD until the drivers are enabled by STFRDM(1) H. A similar enable is provided on the FIS counterpart, whose input to the multiplexers is transferred to the bus and is listed in the truth table on this sheet as a function of the select bits.

A 00 combination selects the MSR Register while a 01 selects the HSR Register. If the combination is 10 the C input is selected. Here the high argument of the floating point is assembled into one 16-bit word. A 11 combination selects the constants as generated on KF-4. Note that not all bit postions are needed for this transfer. Those that are not required are tied to ground.

4.8.11 ROM and Control (Dwg KF-4)

This logic provides the extra control needed for the FIS instructions. It comprises combinational logic and two ROMs that supply the extra ROM bits required by the FIS logic. The logic controls the registers in the FIS and generates the constants. Output signals are given in Table 4-13.

Table 4-13
KF-4 Output Signals

Mnemonic	Description
STFRDM(1) H	Strobe floating RD multiplexer bit, used on KF-3 to enable the FRDMUX to the BUS RD.
SHSR0(1) H 1	Select bits for the HSR Register.
SMSR(1) H	Select bit for the MSR Register. Enables the SHSR(1:0) (1) H bits to control the MSR Register.

Table 4-13 (Cont)
KF-4 Output Signals

Mnemonic	Description
EUBF4	Provides additional branch test conditions for the FIS. Disables the EUBC1 branch multiplexer in the EIS and enables FUB MUX on this print (KF-4). Allows the FIS to control bit 1 (EUBC1) of the ROM address rather than its counterpart in the EIS.
FC1BUS(0) H	Sets the appropriate bit (BUS C1) on the C lines to initiate a DATO operation for floating-point operations.
CON0(1) H 1	These are the two bits that select the constants generated by the FIS option. These together with GPC=6 combine in the logic to generate CONxx signals that are used to generate the octal constants 400, 244, 6, 30, and 200.
FAUX ALU H	Floating auxiliary ALU control. Used in the floating divide loop to enable auxiliary ALU control on KE-5 of the EIS prints.
FDIV H	Floating divide. Used here to enable the generation of FAUX ALU H, and on KE-2 of the EIS board as one of the shift input enables for the DR Register.
FIS INSTR L	Allows a floating instruction to enable the option ROM and to branch to the FIS flow.
EUBC5 L	Inputs to the M7232 module U word in the basic machine. When low, ORs into the ROM address to modify the address. Used only to abort the floating instruction in event of a bus request.
FUBC1	Controls the lowest modifiable bit (1) of the ROM address to provide additional branch tests for the FIS option.
ZB+EPS(Z) H	The OR of ZB (previous) and EPS(Z) present Z status information.
AB(1) H	Output of the ZB flop (see above).
ARGA(1) H	Sent to the EIS, gated with bit 8 of the ROM address. When set, forces selection of the even register for storage of floating arguments even though the ROM is selecting the odd register.
UNFL H	Underflow, indicated by either the EPS(Z) bit being set or B15(1).
OVFL H	Overflow, indicated by both B15 being clear and the D Register not equal to 0.

The two ROMs (23-BXXA2) are fed by BUPP(7:0) H and are enabled by EUPP8 H. Their outputs feed two 74175 quad registers that supply the resultant outputs to the logic. The clear inputs to these registers are tied to a pull up resistor and they are clocked by CLK EU(88:57) H, generated on KE-5 of the EIS board.

The 74151 (FUB MUX) is an 8:1 multiplexer that controls bit 1 (EUBC1) of the ROM address, thereby providing extra branch tests not available with the EIS logic. This MUX is strobed by EUBF4(1) H ANDed with bit 8 of the ROM address. When EUBF4 is asserted, the 0 side of that bit disables the multiplexer EUBC MUX A on KE-4 of the EIS board and enables the FUB MUX. Inputs to this MUX are selected by combinations of EUBF(2:0)(1) H. The table on this sheet (KF-4) gives the results of these combinations.

When D0 is selected, the state of ARGA flip-flop is tested during floating argument fetching to determine if the A argument has been fetched. When D1 is selected, MSR01 is tested. This is used in the floating multiply loop to determine the need for addition.

D2 tests the OR of ZB and EPS(Z). Note that the ZB flip-flop gets its input from the EPS(Z) flop and is clocked at the same time as EPS(Z). D3 looks at counter bits 7:0 to determine when the count exceeds 30_8.

D4 tests to see if the answer is normalized. There are two tests here: 1) is the answer now normalized? (DR09(1) H), or 2) will the answer be normalized after the shift? (DR08(1) H). The 74H50 is a NOT OR AND gate in which one of the inputs on each OR gate must be satisfied. If GPC=1 is asserted, pin 02 of E09 will be enabled causing the logic to look at DR09 on pin 05. If GPC=1 is not asserted, the inverter output at pin 04 will be low causing DR08 at pin 03 to be examined.

If D5 is selected, MSR00(1) H is tested. This is used to determine whether to add or subtract the high divisor in the FDIV loop.

CHAPTER 5
INSTALLATION AND MAINTENANCE REFERENCE INFORMATION

5.1 INSTALLATION

When the KE11-E is included as part of the initial PDP-11/40 System, the M7238 module is installed prior to shipment. If it is being added to an existing system, proceed as follows:

a. Insert the M7238 module in 2(A–F).

b. Remove the jumper (W1) on processor module M7233 (IR DECODE) at location 5(A–F).

c. Install the three "over the back" cables from J1, J2, and J3 of the M7238 module to J1, J2, and J3 respectively of the M7232 (U Word) module at location 3(A–D).

When the KE11-F is to be added to a system, the KE11-E must also be added. Proceed as follows:

a. Perform steps a. through c. above.

b. Insert the M7239 module in 1(A–D).

c. On the M7238 module, remove the following jumpers:

1. W1 from C02F2 to ground.

2. W2 from A02B1 to ground.

3. W3 from D02L1 to ground.

NOTE
If these jumpers are not removed, the KE11-E Option will still execute EIS instructions but will not execute FIS instructions.

When the above steps are performed, the KE11-E and KE11-F Options are ready to be checked out using the diagnostic programs supplied with the options.

5.2 MAINTENANCE

The design, construction, and implementation of the M7238 and M7239 modules used in these options are similar to those used in the KD11-A Central Processor and other options. Maintenance procedures for the entire system are described in the *PDP-11/40 System Manual,* Chapter 7. There are no special maintenance procedures for these options.

5.2.1 Diagnostic Programs

Table 5-1 lists the KE11-E and KE11-F diagnostic programs. These programs are part of a complete package of basic processor and option diagnostics. The sequence of running the diagnostics is set up to completely test the KD11-A Central Processor before attempting to run the diagnostic programs for these options, thus eliminating the KD11-A as a possible cause of failure.

Table 5-1
KE11-E and KE11-F Diagnostic Programs

MAINDEC No.	Function
KE11-E EIS Option	
MAINDEC-11-DCKBL	Divide instruction
MAINDEC-11-DCKBK-A-PB	Multiply instruction
MAINDEC-11-DCKBJ	Arithmetic shift combined instruction
MAINDEC-11-DCKBI	Arithmetic shift instruction
MAINDEC-11-DCQA	MUL/DIV Exerciser
KE11-F FIS Option	
MAINDEC-11-DBKEA	Basic instruction tests
MAINDEC-11-DBKEB	Exerciser
MAINDEC-11-DBKE0	GTP overlay

The MAINDEC (maintenance descriptions) for each diagnostic program indicates how the program is to be loaded and run. The program listing indicates the functional logic that is being tested by each routine. The diagnostic programs are written along functional lines to test and exercise all of the KE11-E and KE11-F logic.

5.2.2 Troubleshooting Test Procedures

The KM11-A Maintenance Module (also referred to as the maintenance console) provides the user with a means of manually operating the system and monitoring status during maintenance operations.

The maintenance module is a W130/W131 board containing 4 switches and 28 indicators that monitor various signals within the processor. When an indicator is lit, it means that the associated logic level is high. An overlay can be attached to the module to indicate what signals are being monitored. This overlay is necessary because the module is designed as a general-purpose device and can be used, without modification other than using different overlays, in many PDP-11 devices. The specific functions monitored by the module depend on the logic signals wired to the device receptacle that receives the module.

When the module is used for monitoring operation of the KE11-E Extended Instruction Set and KE11-F Floating Instruction Set Options, addition of the KE11-E/F overlay (Figure 5-1) is necessary. In this application, the module is inserted into processor slot E1 and the 12 indicators on the right of the overlay are used for the KE11-E/F functions. Note that none of the switches are operational when the module is used for this purpose. The functions monitored by the indicators are listed in Table 5-2.

NOTE

The functions described in Table 5-2 indicate the general purpose of the indicator. At times, a single indicator may show a number of functions, depending on the current state of the processor and option. This is why in order to use the maintenance module properly, the flow diagrams should be followed to determine the significance of an indication at any one time.

Table 5-2
KE11-E/F Maintenance Module Indicators

Indicator	Indication	Print
B15	Bit 15 of CPU B register. In divide, used with DR00 to determine the ALU function to be performed in division loop.	K1–5
ECIN 00	An external carry-in to the ALU.	KE-5
EXP UNFL	Indicates exponential underflow during EXI1 of floating point flows.	KF-4
EXP OVFL	Indicates exponential overflow during EXI1 of floating point flows.	KF-4
DR00	Used in conjunction with other bits to indicate various conditions, e.g., with B15 in divide to determine ALU functions and to determine need for divisor correction. See EPS(C) for other use.	KE-2
DR09	Used as test for normalization (see floating point flows page 6).	KE-2
MSR00	Bit 00 of MSR register. Indicates ALU function in FDIV.	KF-2
MSR01	Bit 01 of MSR register. Indicates ALU function in FMUL.	KF-2
EPS(C)	C bit of extended processor status. In MUL, used with DR00 to determine ALU function in multiply loop.	
EPS (V)	Overflow bit of the extended processor status	KE-6
EPS (Z)	Zero bit of the extended processor status	KE-6
EPS (N)	Negative bit of the extended processor status	KE-6

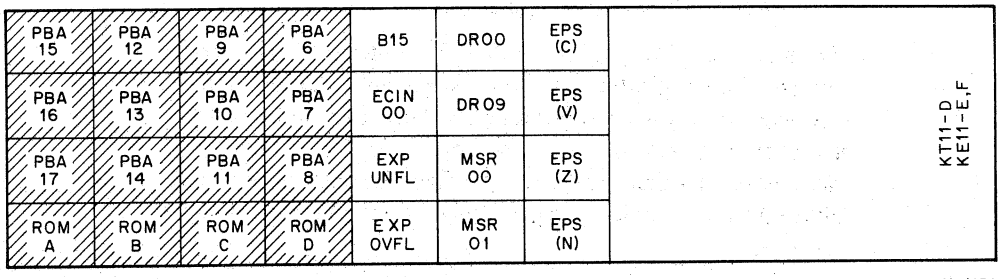


Figure 5-1 KE11-E/F Maintenance Module Overlay

APPENDIX A
GLOSSARY OF TERMS

A.1 GENERAL

Table A-1 contains a collection of some of the terms used in this manual that may need defining. It does not include all terms, only those that it is thought might be confusing. Listing is in alphabetical order.

Table A-1
Glossary of Terms

Term	Definition
ADD	Add (instruction)
ADR	Address
ALU	Arithmetic Logic Unit
ALUM	Arithmetic Logic Unit Mode
ARGA	Argument A (f/f)
ASH	Arithmetic shift (instruction)
ASHC	Arithmetic shift combined (instruction)
BBSY	Bus busy
BRQ	Bus request
BUS	Unibus
BUS U	Bus microprogram
BUSY	Busy
BUT	Branch microprogram test
CIN	Carry-in (ALU)
CLK	Clock
CLKB	Clock B Register
CLKBA	Clock BA Register
CLKD	Clock D Register
CLKOFF	Clock off
CLR	Clear C,V,N,Z (instruction)
CON	Constant
COUT MUX	Carry-out multiplexer (ALU)
C1 BUS	C1 of Unibus
DAD	Discrete alteration of data
DEST	Destination
DIV	Divide (instruction)
DMUX	Data multiplexer
EINSTR	Extended Instruction
EIS	Extended arithmetic instruction set

Table A-1 (Cont)
Glossary of Terms

Term	Definition
EPS	Extended Processor Status
EUB	Extended microprogram bus
EUPP	Extended microprogram pointer
EXP	Exponent
f	Function of
FADD	Floating add (instruction)
FC1BUS	Floating C1 Bus
FDIV	Floating divide (instruction)
FETCH	Fetch (Processor State)
FINSTR	Floating Instruction
FIS	Floating instruction set
FMUL	Floating multiply (instruction)
FSUB	Floating subtract (instruction)
FUB	Floating microprogram bus
IR	Instruction register
ISP	Instruction set processor
JAMUPP	Jam microprogram pointer
MUL	Multiply (instruction)
MUX	Multiplexer
NO-OP	No operation
OVFL	Overflow
PC	Program Counter
PS	Processor Status Register
R(x)	Scratch Pad Register
RSVD INSTR	Reserved instruction
SALU	Select arithmetic logic unit
SALUM	Select arithmetic logic unit mode
SBC	Select B constant
SERVICE	Service
SET COND CODES	Set condition codes
SF	Source field
SFV1	Source field ORed with 1
SRC	Source (processor major state)
STPM	Special Trap Pointer Marker
TRAP	User call
U	Microprogram
UBF	Microprogram branch field
UNFL	Underflow
UPP	Microprogram pointer
U WORD	Microprogram word
VECT	Vector
XOR	Exclusive OR (V)
ZB	"Z" bit previous state (flip-flop)

KE11-E and KE11-F INSTRUCTION SET OPTIONS MANUAL
EK-KE11E-TM-002

Reader's Comments

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications.

What is your general reaction to this manual? In your judgment is it complete, accurate, well organized, well written, etc.? Is it easy to use? _____

_____

_____

_____

What features are most useful? _____

_____

_____

_____

What faults do you find with the manual? _____

_____

_____

_____

Does this manual satisfy the need you think it was intended to satisfy? _____

Does it satisfy *your* needs? _____ Why? _____

_____

_____

_____

Would you please indicate any factual errors you have found. _____

_____

_____

_____

Please describe your position. _____

Name _____ Organization _____

Street _____ Department _____

City _____ State _____ Zip or Country _____

CUT OUT ON DOTTED LINE

— — — — — — — — — — — — Fold Here — — — — — — — — — — —

— — — — — — — — — — — — Do Not Tear - Fold Here and Staple — — — — — — — — —

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

Digital Equipment Corporation
Technical Documentation Department
146 Main Street
Maynard, Massachusetts 01754

digital equipment corporation

Printed in U.S.A.