FP11 floating-point processor maintenance manual

pdp11

digital

DEC-11-HFPAA-C-D

FP11 floating-point processor maintenance manual

digital equipment corporation · maynard, massachusetts

1st Edition, March 1972
2nd Printing (Rev), October 1972
3rd Printing, December 1972
4th Printing, April 1973
5th Printing (Rev), August 1973
6th Printing, January 1974
7th Printing, November 1974

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

Printed in U.S.A.

CONTENTS

Page

CHAPTER 1 INTRODUCTION

CHAPTER 2 INTERFACE

CHAPTER 3 DATA AND DATA FORMATS

CHAPTER 4 CONTROL ROM

CHAPTER 5 ARITHMETIC ALGORITHMS

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

ILLUSTRATIONS

ILLUSTRATIONS (Cont)

TABLES

FOREWORD

The PDP-11 Floating-Point Processor is an optional arithmetic processor used with the PDP-11/45. This processor eliminates the necessity of writing complex software routines to implement arithmetic operations. This manual is divided into the following seven chapters and two appendices:

Chapter 1 is both a system description and a physical description of the FP11.

Chapter 2 is a description of the PDP-11/45 and FP11 interface.

Chapter 3 is a description of the data and instruction formats and describes the FP11 instruction set.

Chapter 4 is a description of the control ROM used to microprogram the FP11. A description of the FP11 flow diagrams is also included in this chapter.

Chapter 5 is a conceptual description of the add, subtract, multiply and divide algorithms.

Chapter 6 is a detailed description of the FP11 logic diagrams.

Chapter 7 provides maintenance information on the Maintenance Module, 11/45 console, FP11 maintenance instructions, and diagnostic programming.

Appendix A is a brief description of the integrated circuits in the FP11.

Appendix B is a signal glossary of the FP11.

The following list of documents supplement the information contained in this manual.

PDP-11/45 Maintenance Manual	DEC-11-H45A-D
KB11 Central Processor Maintenance Manual	DEC-11-HKBA-D
PDP-11/45 Processor Handbook	
PDP-11/45 Unibus Interface Manual (2nd edition)	DEC-11-HIAB-D
TTL Integrated Circuits Catalog from Texas Instruments	CC-201-R
TTL Catalog Supplement from Texas Instruments	Catalog Supplement CC-301
MSI/TTL Integrated Circuits from Texas Instruments	Bulletin CB-125
INTEL LSI Product Guide	
The Integrated Circuits Catalog for Design Engineers	

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The FP11 Floating-Point Processor is a hardware option used with the PDP-11/45 Central Processor. The FP11 enables the PDP-11 Central Processor to perform arithmetic and logic operations using floating-point arithmetic. The prime advantage is increased speed without the necessity of writing complex floating-point software routines. The FP11 has single- and double-precision floating-point capability. Prior to describing the FP11 Floating-Point Unit, several fundamentals of floating-point arithmetic are presented.

1.2 FLOATING-POINT ARITHMETIC

Floating-point representation of a binary number consists of two parts, an *exponent* and a *mantissa*. The mantissa is a fraction in sign and magnitude format with the binary point positioned between the sign bit and the most significant bit. If the mantissa is normalized, all leading 0s are eliminated from the binary representation; the most significant bit is thus a logical 1. 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 1-1 shows an unnormalized number in floating-point notation and then the same number after it has been normalized.

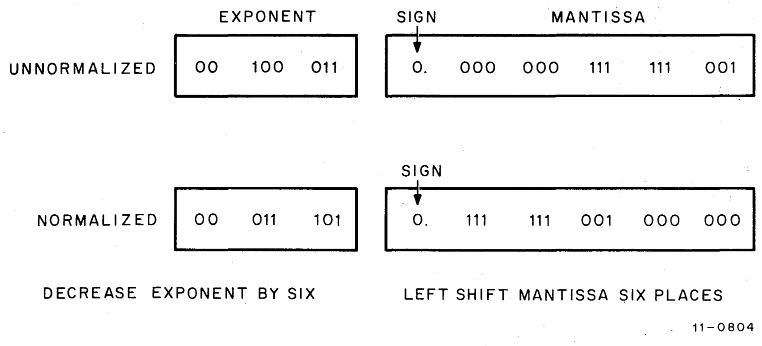


Figure 1-1 Floating-Point Representation

1.2.1 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 mantissa 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.

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

$$\begin{array}{llllll} 0. & 101 & 000 & 000 & 000 & 000\ \ \text{x}2^6 = 50_8 = 40_{10} \\ \underline{0.} & \underline{111} & \underline{000} & \underline{000} & \underline{000} & \underline{000\ \ \text{x}2^3} = \ 7_8 = \ 7_{10} \end{array}$$

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

$$\begin{array}{llllll} 0. & 101 & 000 & 000 & 000 & 000\ \ \text{x}2^6 = 50_8 = 40_{10} \\ \underline{0.} & \underline{000} & \underline{111} & \underline{000} & \underline{000} & \underline{000\ \ \text{x}2^6} = \ 7_8 = \ 7_{10} \\ 0. & 101 & 111 & 000 & 000 & 000\ \ \text{x}2^6 = 57_8 = 47_{10} \end{array}$$

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

$$0.\ \overbrace{101}^{5}\ \ \overbrace{111}^{7}\ \ .000\ \ 000\ \ 000$$

1.2.2 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 eight-bit register is assumed for simplicity.

$$\begin{array}{r} 0.1\ \ 110\ \ 000 \text{ x } 2^3 = 7_8 = 7_{10} \\ \underline{\text{x } 0.1\ \ 010\ \ 000 \text{ x } 2^3 = 5_8 = 5_{10}} \\ 00000000 \\ 1110000 \\ 0 \\ \underline{1110000} \\ .10001100000000\text{x}2^6 \end{array}$$

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

$$.100011.00000000 = 43_8 = 35_{10}$$

1.3 FLOATING-POINT FEATURES

The Floating-Point Processor is an integral part of the central processor. It uses the same memory management facilities provided by the Memory Segmentation option and similar addressing modes. Floating-point instructions can reference any core location, the CPU general registers, and any of the floating-point accumulators discussed in this chapter. Some of the notable features of the FP11 Floating-Point Unit are listed as follows:

- Performs arithmetic operations on 32- or 64-bit floating-point numbers.
- Includes special instructions to optimize input/output routines and mathematical subroutines.
- Utilizes microprogramming techniques for reduced cost.
- Compatible with existing PDP-11 address modes.
- Overlap processing, i.e., CPU and FP11 can run simultaneously.
- Allow execution of in-line code, i.e., CPU and floating-point instructions can be interspersed as desired.
- Employs multiple accumulators for ease of data handling.
- Is capable of converting 16- or 32-bit integers to 32- or 64-bit floating-point numbers during the load class of instructions, if desired.
- Is capable of converting 32- or 64-bit floating-point numbers to 16- or 32-bit integers during the Store class of instructions, if desired.
- Is capable of converting single-precision floating point to double-precision floating point and vice versa during the Store class of instructions, if desired.
- Average single-precision multiply time is 6 μs.
- Average double-precision multiply time is 9.5 μs.
- Average single-precision divide time is 7.5 μs.
- Average double-precision divide time is 12.5 μs.
- Contains floating-point condition codes that can be copied into the CPU status register to provide the CPU with the capability of branching on results of floating-point operations.
- Contains built-in maintenance instructions for ease of maintenance.
- Hardware provides for flexible handling of error conditions.

1.4 SIMPLIFIED BLOCK DIAGRAM DESCRIPTION

Figure 1-2 shows a simplified block diagram of the Floating Point Processor. The major elements of the FP11 are the exponent calculation logic, the accumulators, and the fraction calculation logic.

The exponent calculation logic connects to a 16-bit wide data path that processes exponent or data information; the fraction calculation logic consists of a 60-bit wide data path that processes the fractional part of the operands. The fraction calculation logic sends or receives data to or from the 32-bit scratchpad accumulator.

The accumulators (ACs) are general-purpose read/write scratchpad memories with nondestructive readout. Accumulators 5 through 0 are used for storage of general-purpose data and for register-to-register transfers. Accumulator 6 is used as internal storage and is not accessible by the programmer.

Accumulator 7 is used for internal temporary storage of the following status information:

1. FEC *Floating Exception Code* – a number that identifies the cause of the interrupt.
2. FEA *Floating Exception Address* – the address of the instruction that caused an error.

Accumulator 7 is also used for temporary storage of the address of the current instruction, the program status (FPS), and the exception code.

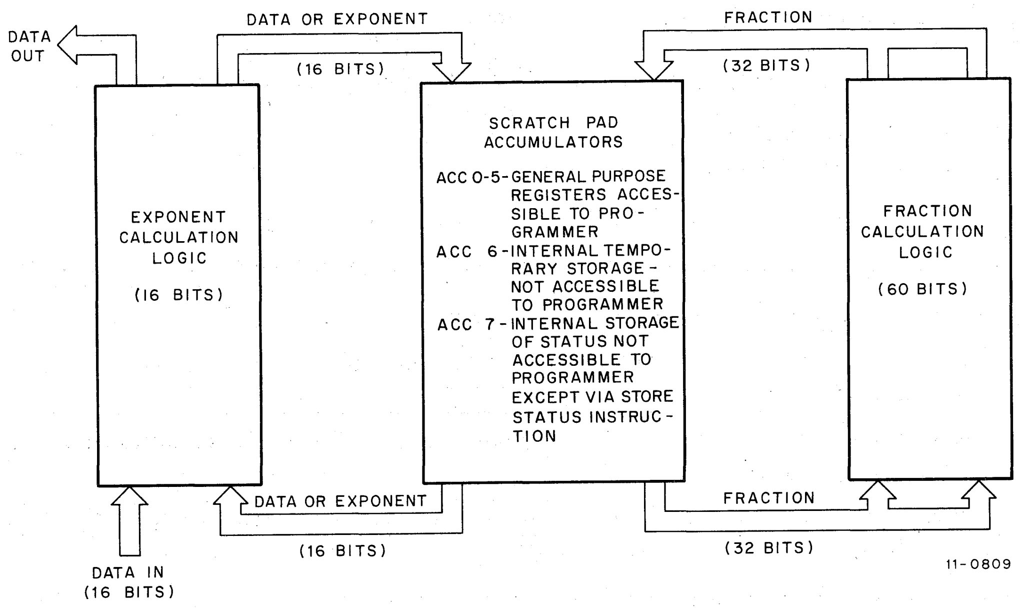

DATA OUT
DATA OR EXPONENT
(16 BITS)
FRACTION
(32 BITS)
EXPONENT CALCULATION LOGIC
(16 BITS)
SCRATCH PAD ACCUMULATORS
ACC 0-5-GENERAL PURPOSE REGISTERS ACCESSIBLE TO PROGRAMMER
ACC 6-INTERNAL TEMPORARY STORAGE - NOT ACCESSIBLE TO PROGRAMMER
ACC 7-INTERNAL STORAGE OF STATUS NOT ACCESSIBLE TO PROGRAMMER EXCEPT VIA STORE STATUS INSTRUCTION
FRACTION CALCULATION LOGIC
(60 BITS)
DATA OR EXPONENT
(16 BITS)
FRACTION
(32 BITS)
DATA IN (16 BITS)
11-0809

Figure 1-2 FP11 Simplified Block Diagram

The ACs are interpreted as 32- or 64-bits long depending on the data formats (refer to Chapter 3). For a single-precision floating-point format, a 32-bit AC is specified (the left-most 32 bits as shown in Figure 1-3). For double-precision floating-point format, a 64-bit AC is specified. The ACs are accessible in 32-bit words. The designated AC and the length of the word contained therein is specified as follows:

AC 5 [3:2]   AC 3 [3:2] [1:0]

The number following the AC designates one of 8 accumulators, and each number in the bracket denotes a 16-bit word. In the first example, AC 5 contains a 32-bit word; in the second case, AC 3 contains a 64-bit word [3:2] [1:0]. The [3] represents the most significant 16 bits, and the [0] represents the least-significant 16 bits. This notation is carried throughout this manual and also in the associated flow diagrams.

1.5 FP11/MEMORY WORD RELATIONSHIPS

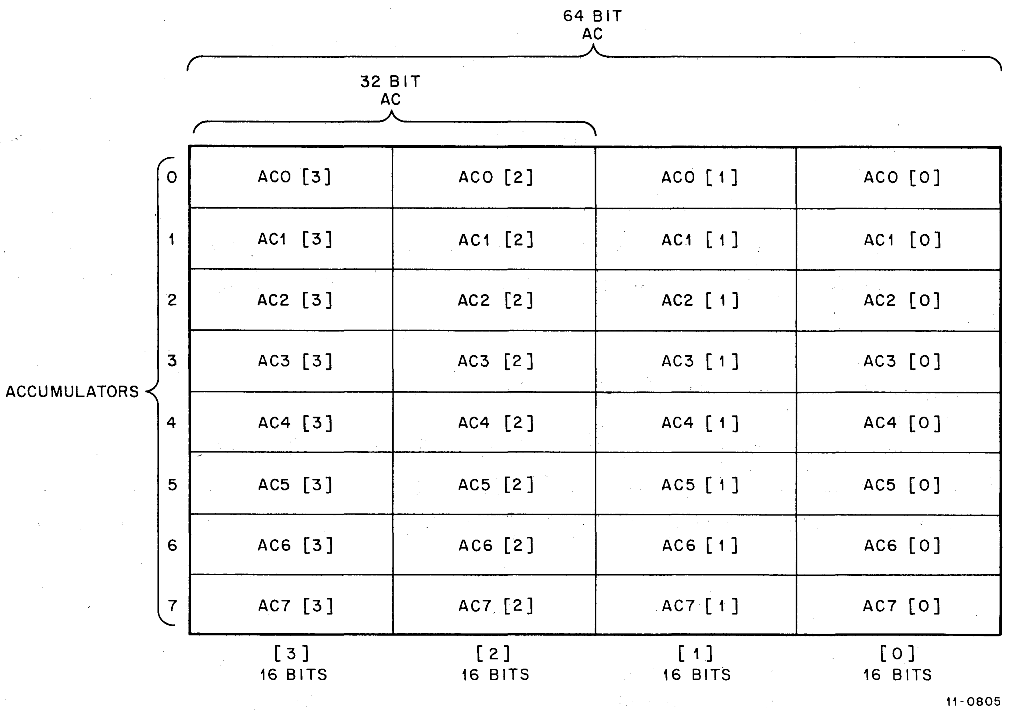
Words stored in memory are either integers or floating-point numbers. Integers are stored in 2's complement format and are converted to sign and magnitude format when transferred to the FP11. Floating-point numbers are already in sign and magnitude format and are transferred directly to the FP11 without being converted. When the FP11 finishes processing the numbers, they can be transferred back to memory as two's complement integers or sign and magnitude floating-point numbers. Floating-point numbers are normalized before being transferred back to memory.

All positive numbers are represented the same in two's complement or in sign and magnitude format. An example is the positive number 2 shown below.

$$\begin{array}{lll} +2 & 0\ 0\ 0\ 0\ 1\ 0 & \text{two's complement} \\ & \underset{\text{sign}}{0}\ \underbrace{0\ 0\ 0\ 1\ 0}_{\text{magnitude}} & \text{sign and magnitude} \end{array}$$

For a negative 2, the number is represented as shown below.

```
        0 1 1 1 1 0     two's complement of -2
        1 0 0 0 1 0     sign and magnitude
sign ───┘ └magnitude┘
```



0	AC0 [3]	AC0 [2]	AC0 [1]	AC0 [0]
1	AC1 [3]	AC1 [2]	AC1 [1]	AC1 [0]
2	AC2 [3]	AC2 [2]	AC2 [1]	AC2 [0]
3	AC3 [3]	AC3 [2]	AC3 [1]	AC3 [0]
4	AC4 [3]	AC4 [2]	AC4 [1]	AC4 [0]
5	AC5 [3]	AC5 [2]	AC5 [1]	AC5 [0]
6	AC6 [3]	AC6 [2]	AC6 [1]	AC6 [0]
7	AC7 [3]	AC7 [2]	AC7 [1]	AC7 [0]
	[3] 16 BITS	[2] 16 BITS	[1] 16 BITS	[0] 16 BITS

Figure 1-3 Accumulator Configuration

NOTE

AC 7 [1] contains address of instruction
AC 7 [0] contains FPS (temporary storage of FEC)

1.5.1 FP11 Hidden Bit

All numbers (fractions) transferred to the fraction calculation logic are transferred as positive fractions of the form 0.1 xxxx. Since the most significant bit to the right of the binary point is always a 1, this bit is referred to as the "hidden bit" and is dropped when the word is stored in memory. This provides another bit of significance in the FP11. Words transferred from memory to the FP11 are represented in the FP11 by a sign bit, eight bits of exponent and 23 bits (single-precision) or 55 bits (double-precision) of fraction. For example, consider the number minus 1/2. The sign is 1, the exponent is 200 which is equal to 2^0 power, and the fraction is .10 . . . . .

Figure 1-4 shows the word as it appears in memory and how it appears when stored internally in the FP11.

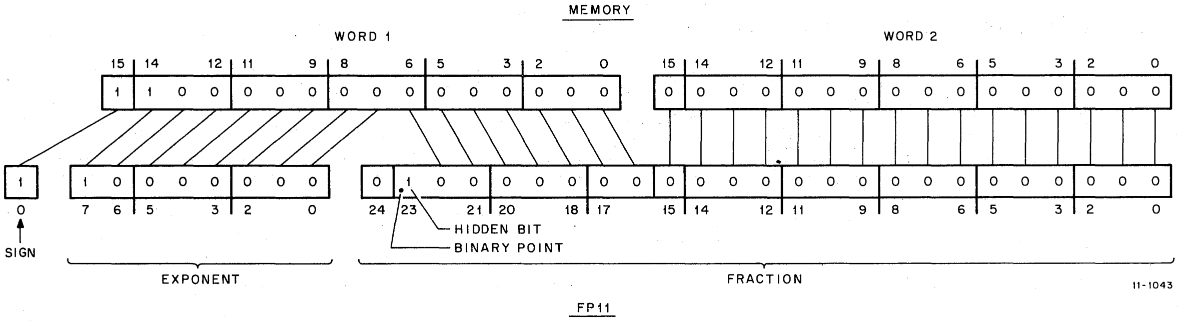


Figure 1-4 FP11

Note that even with a negative number as shown above the fraction is treated as a positive normalized fraction (bit 24 = 0, bit 23 = 1). Bits 24 and 23 are dropped when the floating-point word is reassembled and stored back in memory.

1.6 FP11-B PHYSICAL DESCRIPTION (PDP-11/45)

The FP11-B Floating-Point Processor is used with the PDP-11/45 CPU. The FP11-B consists of four multi-layer hex modules that are plugged into the pre-wired KB11A Main Frame. The four modules plug into slots 2, 3, 4, and 5 and take-up rows A through F (see Figure 1-5). The chart below shows the slots associated with each module.

Module	Slot	Row
M8113 – FXP	5	A through F
M8112 – FRM	4	A through F
M8115 – FRL	3	A through F
M8114 – FRH	2	A through F

A +5V regulator card is included and is plugged into the upper power supply in slot A. The –15V needed for the time state generator on the FRH module is supplied by regulator E, which is included as part of the Central Processor Regulator Set.

Slot E1 on the KB11A Main Frame is reserved for the Floating-Point Maintenance Module (refer to Chapter 7 for additional information on this module). The +8 Vdc required for this module is obtained from the upper bulk supply (PS H742A).

Slot No.	Row F	Row E	Row D	Row C	Row B	Row A		
1	CPU MAINT	FPP MAINT		KW11 LINE CLOCK	UNIBUS A	TERM		Pin Side
2	FRH (M8114)						FLOATING POINT	
3	FRL (M8115)							
4	FRM (M8112)							
5	FXP (M8113)							
6	DAP (M8100)						CENTRAL PROCESSOR	
7	GRA (M8101)							
8	IRC (M8102)							
9	RAC (M8103)							
10	PDR (M8104)							
11	TMC (M8105)							
12	UBC (M8106)							
13	SSR (M8108)							
14	SAP (M8107) or SJB (M8116)							
15	TIG (M8109)				PHK ( )			
16	MEM CTRL (M8110)						SEMICONDUCTOR MEMORY	
17	MTRX (Bipolar=M8111 & MOS=G401)							
18	MTRX (Bipolar=M8111 & MOS=G401)							
19	MTRX (Bipolar=M8111 & MOS=G401)							
20	MTRX (Bipolar=M8111 & MOS=G401)							
21	MEM CTRL (M8110)							
22	MTRX (Bipolar=M8111 & MOS=G401)							
23	MTRX (Bipolar=M8111 & MOS=G401)							
24	MTRX (Bipolar=M8111 & MOS=G401)							
25	MTRX (Bipolar=M8111 & MOS=G401)							
26	DEVICE 1				UNI A CABLE			
27	DEVICE 2				UNI B CABLE			
28	DEVICE 3				UNIBUS B TERM			

← Slot No.

Figure 1-5 FP11-B Module Layout

CHAPTER 2
INTERFACE

2.1 INTRODUCTION

The Floating Point Processor connects directly to the 11/45 Central Processor (see Figure 2-1) and not to the Unibus. This is to allow for proper operation of the segmentation option and to increase the speed of instruction execution.

The 11/45 CPU fetches instructions from memory and decodes them. If the instruction contains a 17_8 op code, it is a floating-point instruction and the CPU branches to the CPU ROM states associated with floating-point instructions. At this point, the CPU/FP11-B interaction is initiated (refer to Paragraph 2.3).

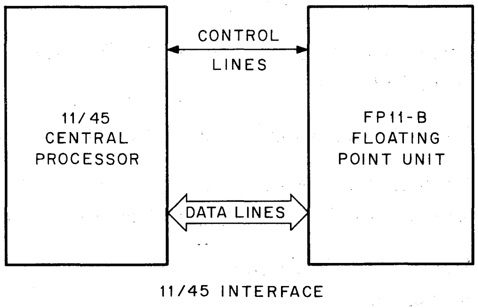


Figure 2-1 11/45 Simplified Interface Diagram

2.2 INTERFACE SIGNALS

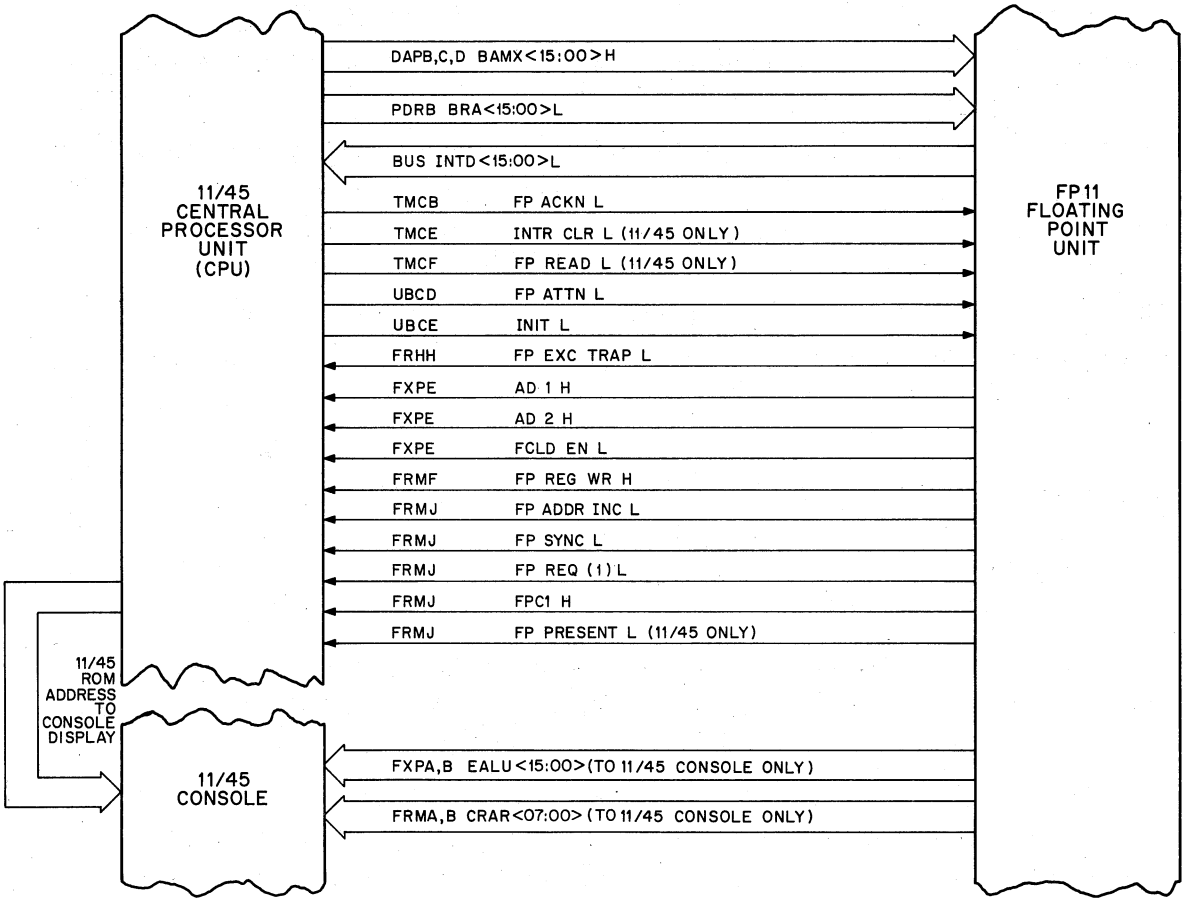
The signals that interface the 11/45 CPU to the FP11 are described below (see Figure 2-2).

Signal	Description
BAMX (00:15) H	Sixteen lines from the CPU that contain the address of the instruction.
BR (00:15) B L	Sixteen data lines that provide transfer of data from the CPU to FP11.
BUS INTD (00:15) L	Sixteen lines used to send data from the FP11 to the CPU. These lines are also used by the segmentation option.
FP ACKN L	A signal from the CPU indicating that an FP TRAP was received from the FP11.

Signal	Description
INTR CLR L	A signal from the 11/45 that indicates that the 11/45 CPU is in its interrupt service routine.
FP READ L	A signal from the 11/45 that indicates that the BUS INTD lines can be used by the FP11.
FP ATTN L	A signal from the CPU to the FP11 that accompanies information sent to or from the FP11.
INIT L	An initialize pulse used to reset major registers in the FP.
FP EXC TRAP L	This signal, when low, causes the CPU to trap to vector address 244_8 (Trap Vector).
AD 1, AD 2 H	Represent constants that are added to or subtracted from the general registers in the CPU for address calculation. The constants are:

AD2	AD1	
0	0	constant of 8
0	1	constant of 4
1	0	constant of 2
1	1	constant of 0

FCLD EN L	This signal causes the FP11 floating-point condition codes to be written into the CPU condition codes.
FP REG WR H	When high, this signal causes BUS INTD data to be loaded into general registers in the CPU.
FP ADDR INC L	A signal to the CPU indicating that the address is to be incremented by 2.
FP SYNC L	A signal from the FP11 in response to FP ATTN indicating that the data has been accepted or that the FP11 is ready to send or receive data.
FP REQ (1) L	A signal used in conjunction with FP SYNC to indicate that more data words are desired.
FPC1 H	Indicates a DATO operation. When this signal goes low, it indicates a DATI operation.
FP PRESENT L	Indicates the FP11 is present.
EALU (00:15)	Sixteen lines to console that allow the contents of EALU to be displayed (used with 11/45 CPU).
CRAR (00:07)	Eight lines to console that allow the next ROM address to be displayed.



NOTE:
For 11/20 CPU, an Interface Unit is inserted between the CPU and FP11 to connect 11/20 Unibus signals to FP11 Compatible signals.The signals shown above are used with the 11/20 also, except as noted.

11-0807

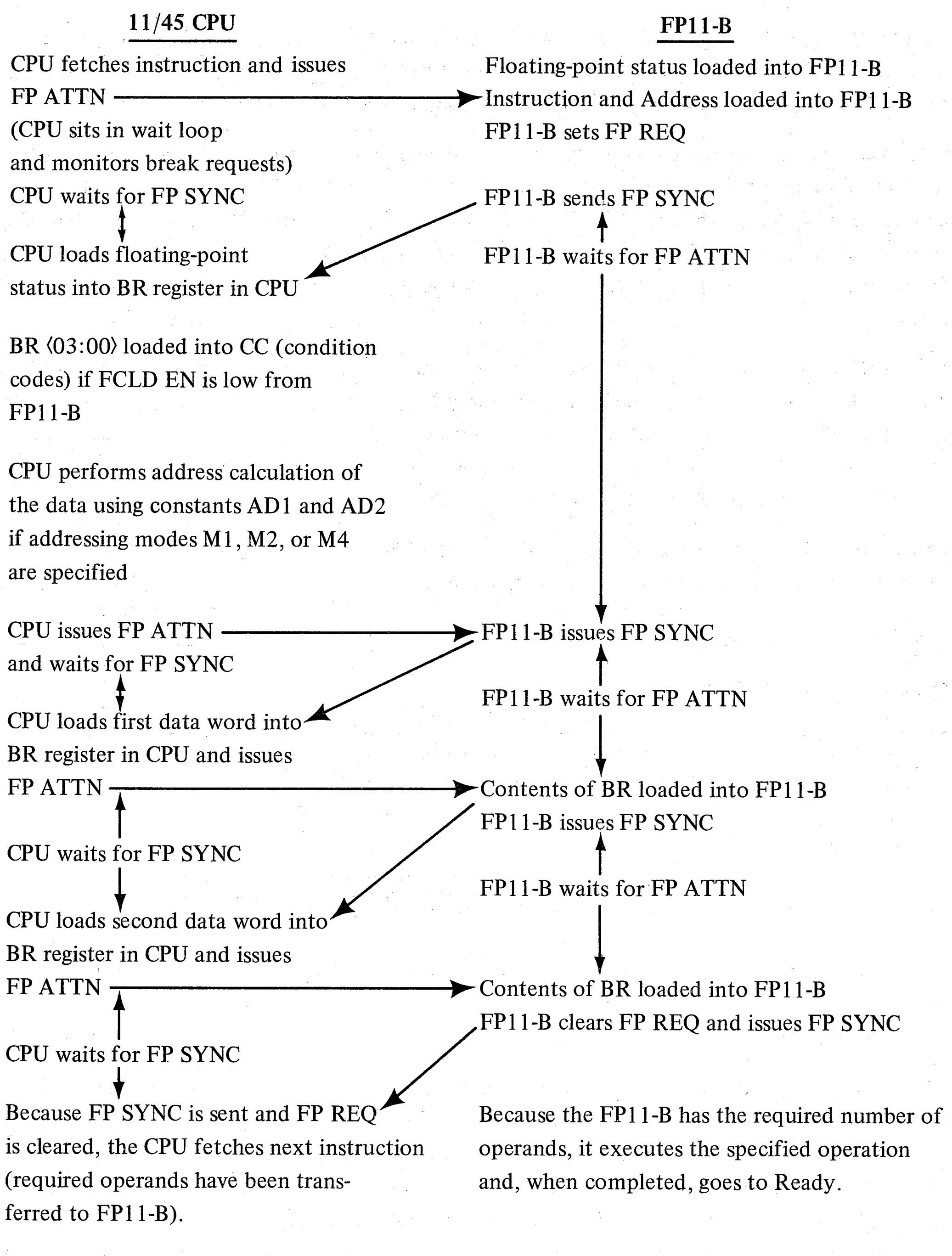
Figure 2-2 CPU/FPP Interface Diagram

2.3 11/45 INTERFACE

Figure 2-3 shows the interaction involved between the 11/45 CPU and the FP11-B for a floating-point Load instruction. The sequence of events for other instructions can be found in the FP11 flow diagrams. The CPU puts the instruction on the BRA lines accompanied by FP ATTN. The CPU also decrements the PC and puts the decremented PC on the BAMX lines (see Figure 2-2). It is necessary to transfer the contents of the PC to the FP11-B because the CPU and FP11 can execute instructions simultaneously. The CPU may jump or trap to a new location while the FP11 instruction is being executed, and if a floating-point error occurs the programmer can then determine the address of the FP11 instruction that caused the error condition.

After the instruction and address are on the lines, the CPU goes into a wait loop, monitoring break requests and also waiting for FP SYNC from the FP11. If a break request occurs, the CPU branches to its break service routine and issues an Abort signal (INTR CLR) to the FP11. This signal aborts the floating-point instruction in process. On return from the break service routine, the CPU fetches the next instruction; however, this instruction is the same instruction that was aborted since the PC was previously decremented.

If FP SYNC occurs before a break request, the CPU loads the status from the FP11 into the CPU (see Figure 2-3). If FCLD EN is low, the floating-condition codes (BR ⟨3:07⟩) from the FP11 are inserted in the status word;



11/45 CPU

CPU fetches instruction and issues FP ATTN

(CPU sits in wait loop and monitors break requests)

CPU waits for FP SYNC

CPU loads floating-point status into BR register in CPU

BR ⟨03:00⟩ loaded into CC (condition codes) if FCLD EN is low from FP11-B

CPU performs address calculation of the data using constants AD1 and AD2 if addressing modes M1, M2, or M4 are specified

CPU issues FP ATTN and waits for FP SYNC

CPU loads first data word into BR register in CPU and issues FP ATTN

CPU waits for FP SYNC

CPU loads second data word into BR register in CPU and issues FP ATTN

CPU waits for FP SYNC

Because FP SYNC is sent and FP REQ is cleared, the CPU fetches next instruction (required operands have been transferred to FP11-B).

FP11-B

Floating-point status loaded into FP11-B

Instruction and Address loaded into FP11-B

FP11-B sets FP REQ

FP11-B sends FP SYNC

FP11-B waits for FP ATTN

FP11-B issues FP SYNC

FP11-B waits for FP ATTN

Contents of BR loaded into FP11-B

FP11-B issues FP SYNC

FP11-B waits for FP ATTN

Contents of BR loaded into FP11-B

FP11-B clears FP REQ and issues FP SYNC

Because the FP11-B has the required number of operands, it executes the specified operation and, when completed, goes to Ready.

Figure 2-3 Sequence of Events for Load Instruction

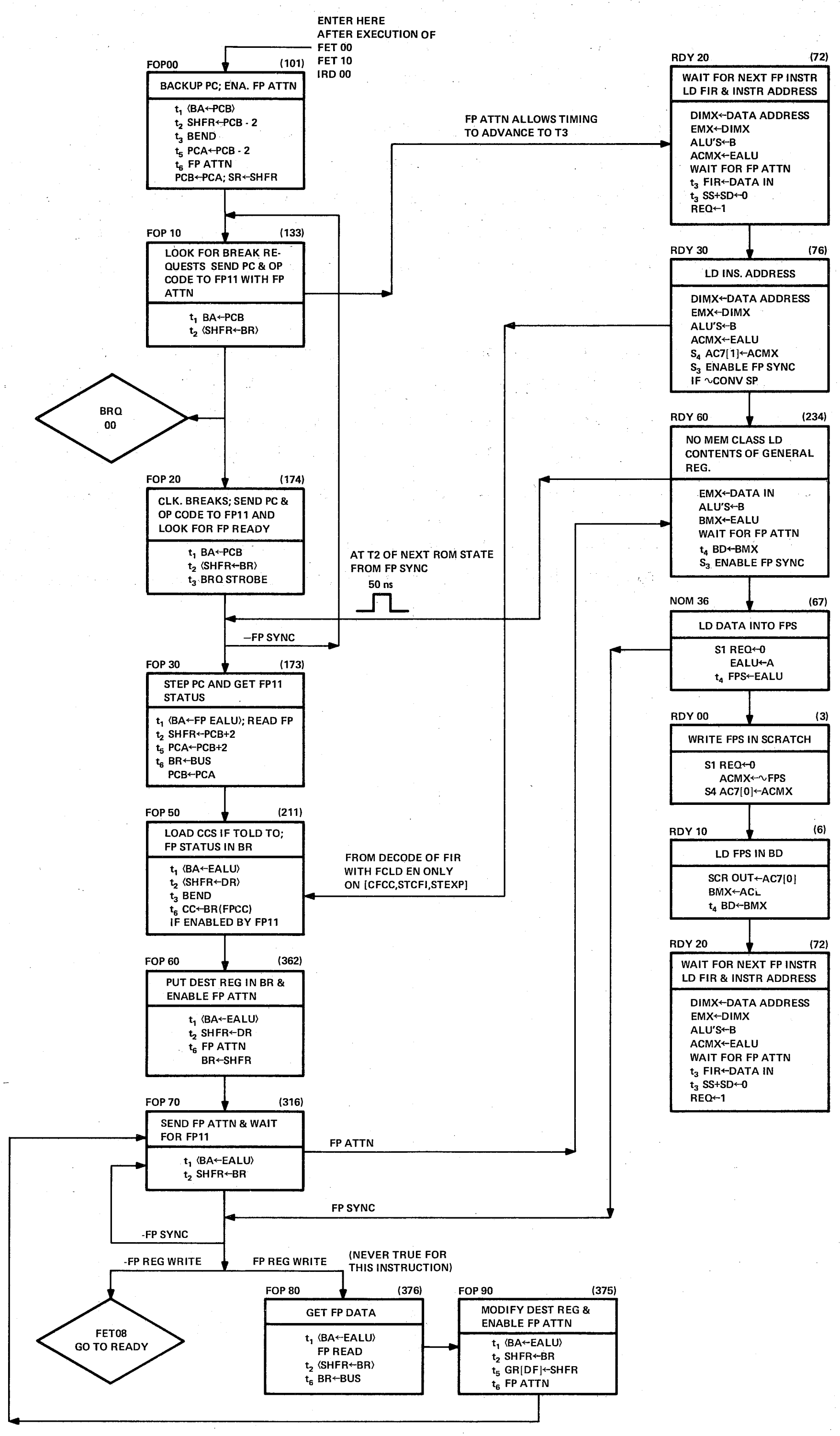
otherwise, the word is unmodified. In addition, the CPU starts to calculate the address of the data. If addressing modes M1, M2, or M4 are specified, the address is calculated using constants AD1 and AD2. If another mode is specified, the address is calculated like any other destination address. On completion of the address calculation, the CPU issues FP ATTN to the FP11, and the FP11 responds with FP SYNC.

On receipt of FP SYNC from the FP11, the CPU loads the first data word into the BR register, raises FP ATTN, and waits for FP SYNC. The data word is loaded in the FP11-B, and the FP11-B raises FP SYNC, acknowledges receipt of this word, and awaits the next data word accompanied by FP ATTN.

The CPU loads the second data word into the BR register and raises FP ATTN. This word is loaded in the FP11-B and the FP11-B raises FP SYNC; however, because this is the last data word desired (single-precision floating-point format requires two data words), FP REQ is cleared. When the CPU receives FP SYNC with FP REQ cleared, it fetches the next instruction. While the CPU is fetching the instruction, the FP11-B proceeds to execute the operations specified. When this is completed, the FP11-B goes to the Ready state to await the next floating-point instruction.

When the FP11 is executing an instruction and the CPU fetches another FP11 instruction, the FP11 continues execution of the instruction and the CPU hangs in a wait loop. If a break request occurs while the CPU is in the wait loop, the CPU branches to its service routine and issues an Abort (INTR CLR) as previously described; however, because the FP11 is busy in this case, the Abort is not honored and the FP11 proceeds to complete execution of the instruction. The CPU subsequently refetches the instruction so it can be executed.

To further clarify the interaction between the CPU and the FP11, two examples are provided. The first (Figure 2-4) shows the interaction for address mode 0 and the second (Figure 2-5) shows the interaction for address mode 2.



11-1443

Figure 2-4 LD FPS Instruction Interaction – Mode 0

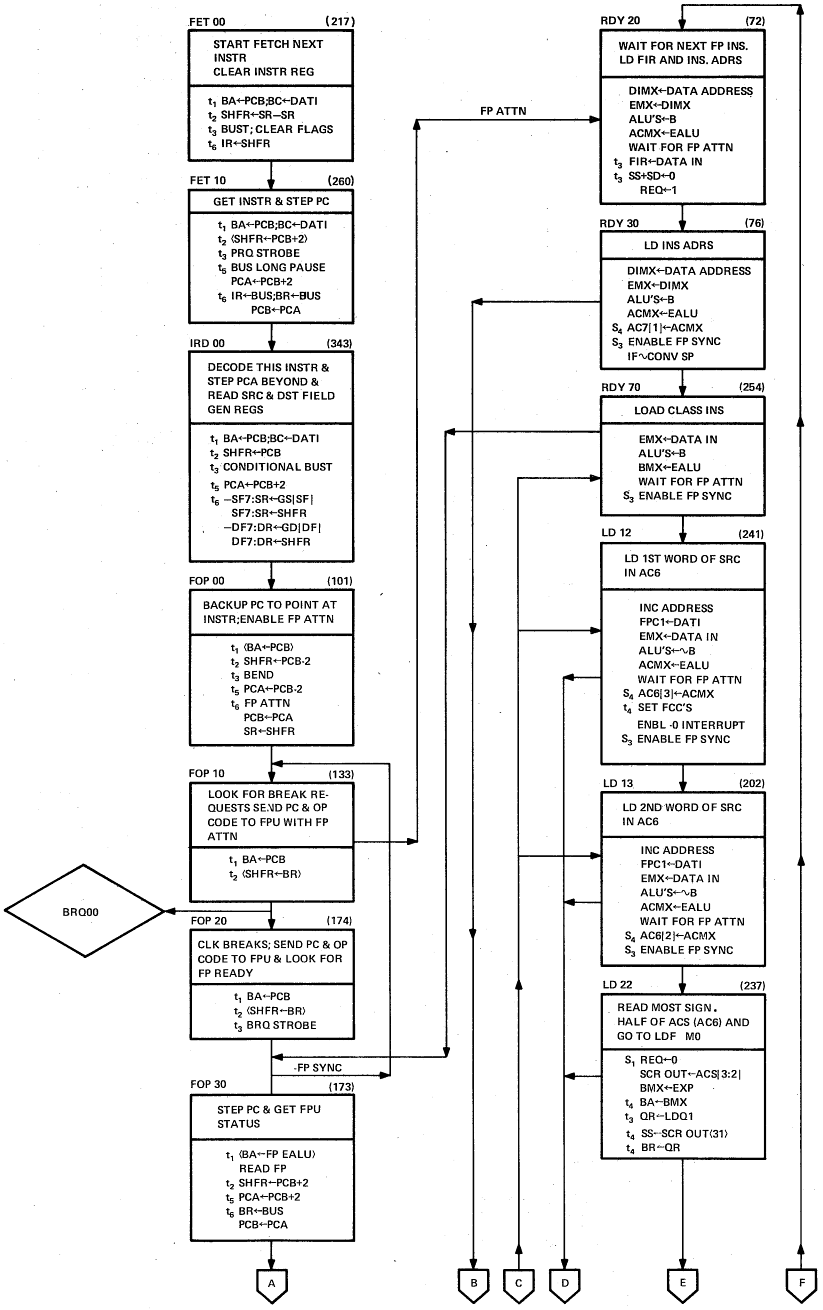


Figure 2-5 LDF Instruction Interaction – Mode 2 (Sheet 1 of 2)

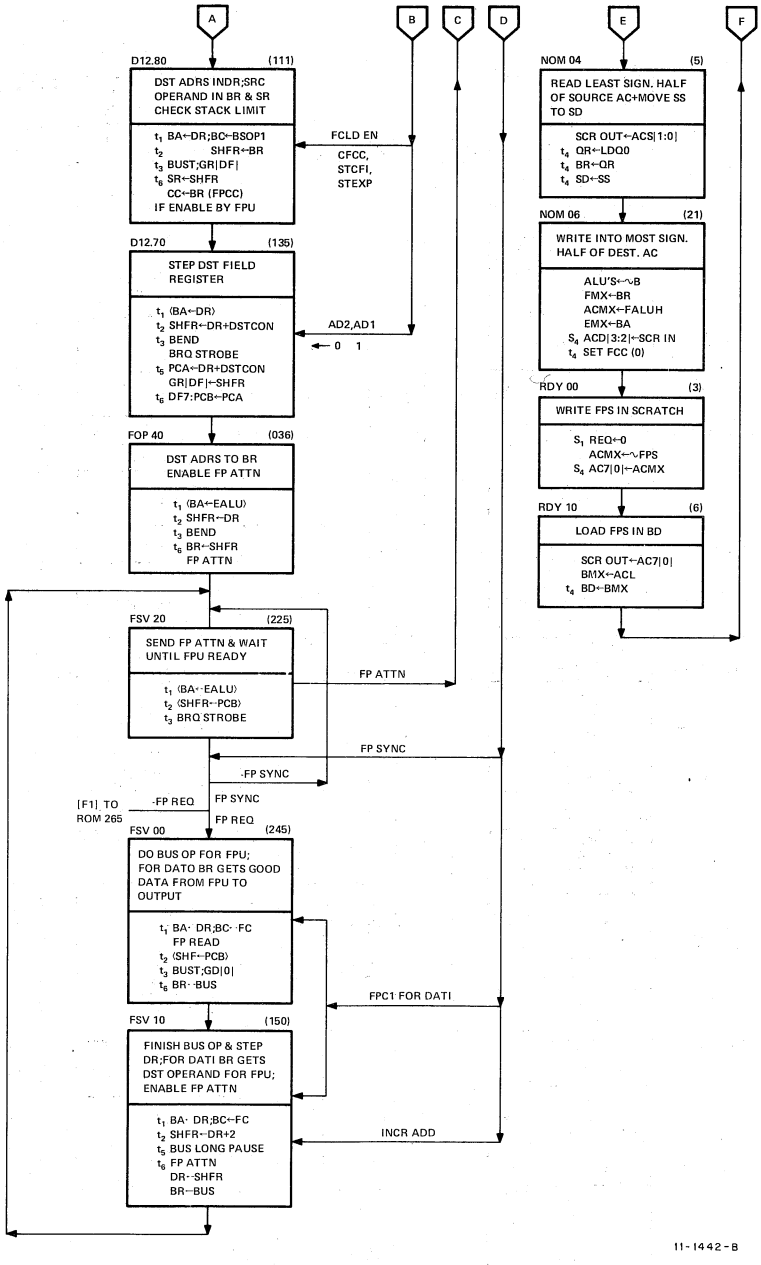


Figure 2-5 LDF Instruction Interaction – Mode 2 (Sheet 2 of 2)

CHAPTER 3
DATA AND DATA FORMATS

3.1 FP11 DATA FORMATS

The FP11 utilizes short (I) and long (L) integer format in addition to single- (F) and double-precision (D) floating-point format. The following paragraphs briefly define the integer formats followed by a description of the floating-point formats.

3.1.1 FP11 Integer Format

Integer format is represented in 2's complement notation in the FP11. The short-integer format is 16 bits long; the long-integer format is 32 bits long. In both instances the most significant bit represents the sign bit. Figure 3-1 shows the integer 5 in both formats followed by the integer minus 5 in both formats.

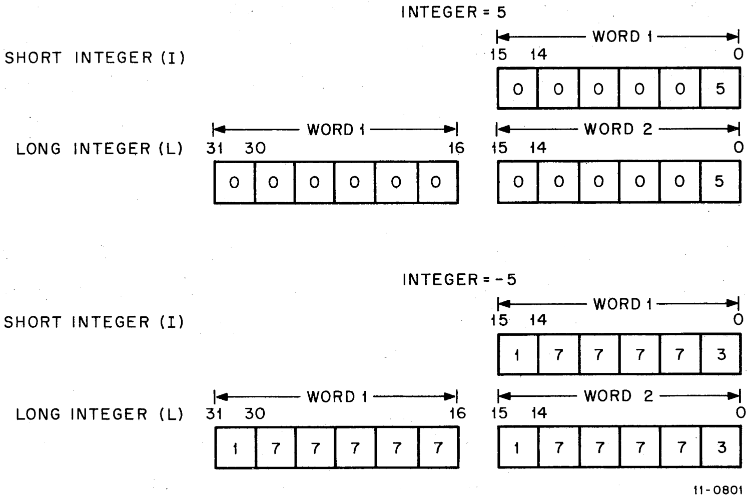
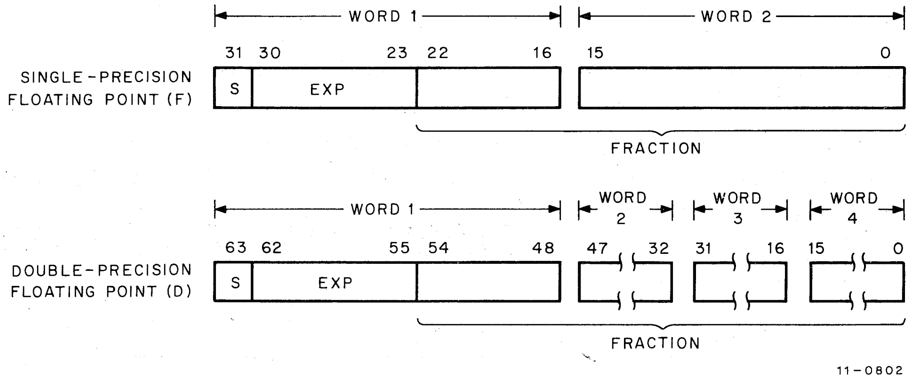


Figure 3-1 Integer Formats

3.1.2 FP11 Floating-Point Formats

Single-precision floating-point format is 32 bits long and is designated by F; double-precision (extended) format is 64 bits long and is designated by D. All floating-point numbers are assumed to be normalized. The mantissa or fraction is represented in sign and magnitude format with the sign bit extended to the most significant bit position, as shown in Figure 3-2. Note that the 8-bit exponent separates the fraction from its associated sign.



S = Sign

EXP = Exponent in excess 200_8 notation (refer to Paragraph 3.1.4.)

Fraction = 23 or 55 bit fraction in sign and magnitude format. Binary point between bits 22 and 23 for F format or between bits 54 and 55 for D format.

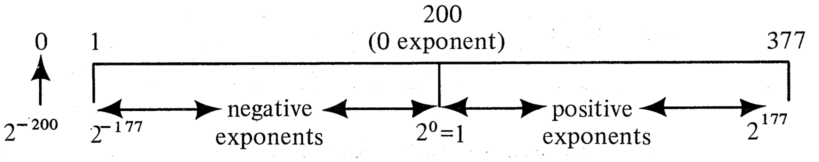
Figure 3-2 Floating-Point Data Formats

3.1.3 Floating-Point Mantissa

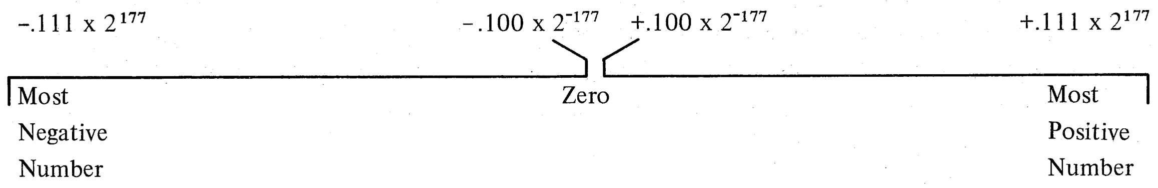
All floating-point numbers are normalized; thus, in sign and magnitude format, the mantissa has a range from 0.10000. . . . to 0.1111. . . . for positive operands and a range from 1.10000. . . . to 1.11111. . . . for negative operands. All operands transferred between the CPU and FP11 are in sign and magnitude format and are converted internally to 2's complement format to perform arithmetic operations. Because, in sign and magnitude format, the bit immediately to the right of the binary point is always a 1, it is not stored in memory or in the scratchpad accumulators. This *hidden bit* provides another bit of significance in the results of arithmetic operations. However, when data is loaded into the fractional calculation logic data path, the hardware inserts the hidden bit; this point must be kept in mind when examining results during maintenance procedures.

3.1.4 Floating-Point Exponent

The exponent in the FP11 is specified by eight bits, providing a range from 0 to 377_8. Excess 200 notation is used, which means that 200 is added to the exponent. Thus, an exponent of -177 is represented by 001_8, an exponent of 000_8 is represented by 200_8, and an exponent of 177 is represented by 377_8.



For example, the number 0.1_2 is actually 0.1×2^0, and the exponent is represented as $10\ 000\ 000_2$ because 200_8 represents an exponent of zero. The following chart shows the range of floating-point numbers that can be handled by the FP11. Only three bits are shown for simplicity, but they can be extended to any number.



3.2 FP11 PROGRAM STATUS REGISTER

The FP11 contains a program status register; this register contains FP11 condition codes (carry, overflow, zero, and negative) that can be copied into the Central Processor. In other words, FC, FV, FZ, and FN can be copied into the CPU's C, V, Z, and N condition codes, respectively. The program status register also contains four mode bits and additional bits used to enable various interrupt conditions. Figure 3-3 shows the layout of the program status register. Each bit shown in the figure is described in the following paragraphs:

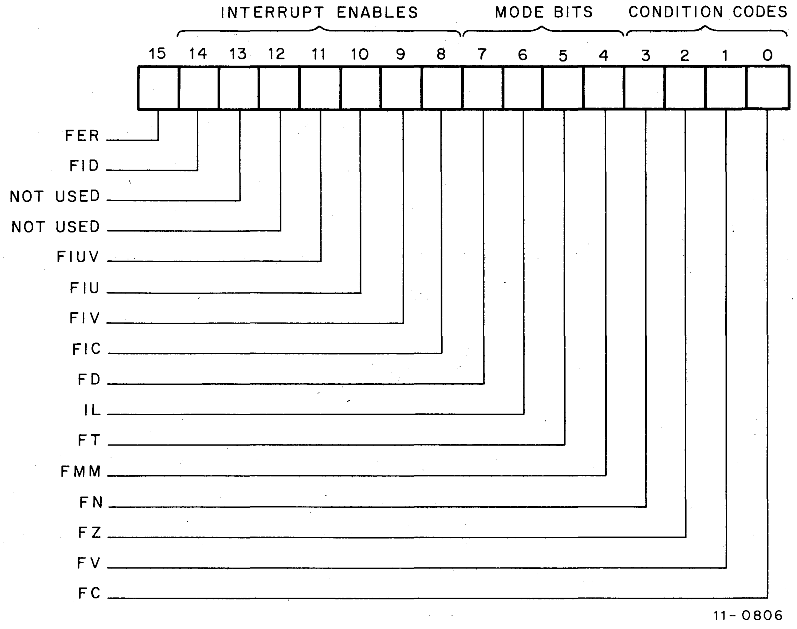


Figure 3-3 Status Register Format

FER – This bit indicates an error condition of the FP11.

FID (Floating Interrupt Disable) – All interrupts by the FP11 are disabled when this bit is on.

FIUV (Floating Interrupt on Undefined Variable) – When this bit is set and a minus 0 is obtained from memory, an interrupt occurs. If the bit is not set, minus 0 can be loaded and stored; however, any arithmetic operation is treated as if it were a positive 0.

FIU (Floating Interrupt on Underflow) – When this bit is set, an underflow condition causes a floating underflow interrupt. The result of the operation causing the interrupt is correct except for the exponent, which is off by 400_8. If the FIU bit is not set and underflow occurs, the result is set to zero.

FIV (Floating Interrupt on Overflow) – When this bit is set, floating overflow causes an interrupt. The result of the operation causing the interrupt is correct except for the exponent, which is off by 400_8. If the FIV bit is not set, the result of the operation is the same; the only difference is that the interrupt does not occur.

FIC (Floating Interrupt on Integer Conversion Error) – When this bit is set, and the Store Convert Floating to Integer instruction causes FC to be set (indicating a conversion error), an interrupt occurs. When a conversion error occurs, the destination register is cleared and the source register is untouched. When FIC is reset, the result of the operation is the same; however, no interrupt occurs.

FD (Double-Precision Mode Bit) – This bit, when set, specifies double-precision format and, when reset, specifies single-precision format.

IL (Long-Precision Integer Mode Bit) – This bit is employed during conversion between integer and floating-point format. If set, double-precision, 2's complement integer format of 32 bits is specified, and, if reset, single-precision 2's complement integer of 16 bits is specified.

FT (Truncate Bit) – This bit, when set, causes the result of any floating-point operation to be truncated rather than rounded.

FMM (Maintenance Mode Bit) – This bit is used to enable special maintenance logic and is described in Chapter 7.

FC, FV, FZ, and FN – These bits are the four floating-point condition codes, which can be loaded in the CPU's C, V, Z, and N condition codes, respectively. This is accomplished by the Copy Floating Condition Codes (CFCC) instruction. To determine how each instruction affects the condition codes, refer to the instruction description in the *PDP-11 Handbook.*

For the Store Convert Floating to Integer instruction (which converts a floating-point number to an integer), the FC bit is set if the resulting integer is too large to be stored in the specified register.

3.3 PROCESSING OF FLOATING-POINT EXCEPTIONS

The interrupt vector used to handle all floating-point interrupts is in location 244_8. A total of seven possible interrupts can occur. These seven possible interrupt exceptions are encoded in the FP11 Exception Code Register (FEC). The interrupt exception codes represent an offset into a dispatch table, which routes the program to the right error handling routine. The dispatch table is a function of the software. The offset for each exception code is shown below followed by a brief description.

FP11 Exception Code	Definition
2	Floating Op Code Error – The FP11 causes an interrupt for an erroneous op code if the FID bit is not set.
4	Floating Divide by Zero – Division by zero causes an interrupt if the FID bit is not set.
6	Floating Integer Conversion Error
10	Floating Overflow
12	Floating Underflow
14	Floating Undefined Variable
16	Micro Break Trap

NOTE
The traps for exception codes 6, 10, 12, and 14 can be enabled in the FPU's Program Status Register.

In addition to the FEC register, the FP11 contains a 16-bit Floating Exception Address register (FEA), which stores the address of the last floating-point instruction that caused a floating-point exception.

3.4 FP11 INSTRUCTION FORMATS

The FP11 instruction set is divided into five formats as shown in Figure 3-4.

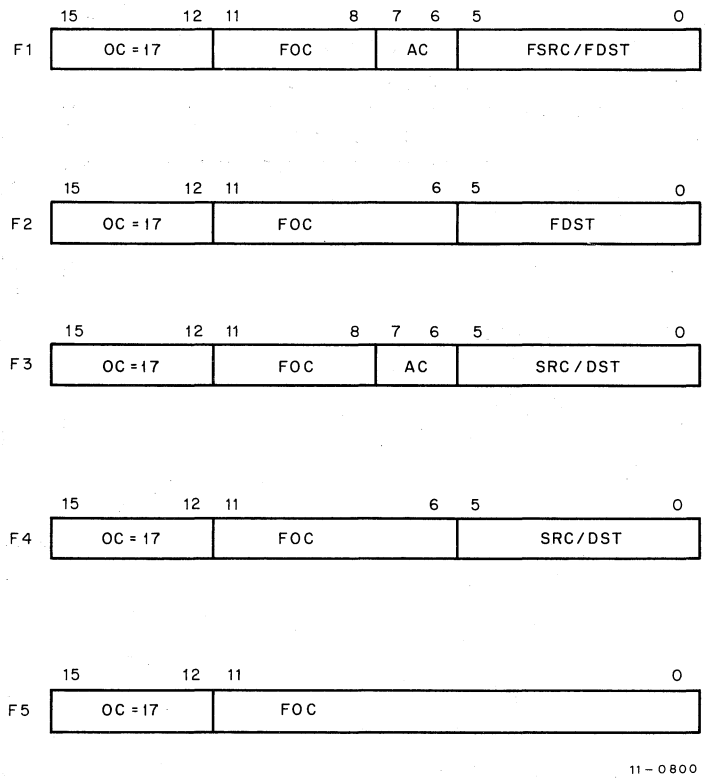


Figure 3-4 Instruction Formats

The 2-bit AC field (bits 6 and 7) allows selection of scratchpad accumulators 0 through 3 only. If address mode 0 is specified with formats F1 or F2, bits 2 through 0 are used to select the floating-point accumulator. Only accumulators 5 through 0 can be accessed in this manner. If accumulators 6 or 7 are specified, the FP11 traps if the interrupt is enabled.

The fields of the various instruction formats (refer to Table 3-1) are interpreted as follows:

Mnemonic	**Description**
OC	Operation Code – All floating-point instructions are designated by a 4-bit op code of 17_8.
FOC	Floating Operation Code – The number of bits in this field varies with the format and is used to specify the actual floating-point operation.
SRC	Source – A 6-bit source field identical to that in a PDP-11 instruction.
DST	Destination – A 6-bit destination field identical to that in a PDP-11 instruction.
FSRC	Floating Source – A 6-bit field used only in format F1. It is identical to SRC, except in mode 0 when it references a floating-point accumulator rather than a CPU general register.
FDST	Floating Destination – A 6-bit field used in formats F1 and F2. It is identical to DST, except in mode 0 when it references a floating-point accumulator instead of a CPU general register.
AC	Accumulator – A 2-bit field used in formats F1 and F3 to specify accumulators 0 through 3.

Table 3-1
Format of FP11 Instructions

Instruction Format	Instruction	Mnemonic
F1	ADD	ADDF FSRC, AC ADDD FSRC, AC
	LOAD	LDF FSRC, AC LDD FSRC, AC
	SUBTRACT	SUBF FSRC, AC SUBD FSRC, AC
	COMPARE	CMPF AC, FDST CMPD AC, FDST
	MULTIPLY	MULF FSRC, AC MULD FSRC, AC
	MODULO	MODF FSRC, AC MODD FSRC, AC
	STORE	STF AC, FDST STD AC, FDST
	DIVIDE	DIVF FSRC, AC DIVD FSRC, AC
	LOAD CONVERT	LDCFD FSRC, AC LDCDF FSRC, AC
F1	STORE CONVERT	STCFD AC, FDST STCDF AC, FDST
F2	CLEAR	CLRF FDST CLRD FDST
	TEST	TSTF FDST TSTD FDST
	ABSOLUTE	ABSF FDST ABSD FDST
F2	NEGATE	NEGF FDST NEGD FDST
F3	LOAD EXPONENT	LDEXP SRC, AC
	LOAD CONVERT INTEGER TO FLOATING	LDCIF SRC, AC LDCID SRC, AC LDCLF SRC, AC LDCLD SRC, AC
	STORE EXPONENT	STEXP AC, DST
F3	STORE CONVERT FLOATING TO INTEGER	STCFI AC, DST STCFL AC, DST STCDI AC, DST STCDL AC, DST
F4	LOAD FP11's PROGRAM STATUS	LDFPS SRC
	STORE FP11's PROGRAM STATUS	STFPS DST
F4	STORE FP11's STATUS	STST DST

(continued on next page)

Table 3-1 (Cont)
Format of FP11 Instructions

Instruction Format	Instruction	Mnemonic
F5	COPY FLOATING CONDITION CODES	CFCC
↑	SET FLOATING MODE	SETF
│	SET INTEGER MODE	SETI
│	LOAD UBREAK REGISTER	LDUB
│	LOAD SHIFT COUNTER	LDSC
│	STORE AR REGISTER IN AC0	STA0
│	MAINTENANCE RIGHT SHIFT	MRS
│	STORE QR REGISTER IN AC0	STQ0
↓	SET DOUBLE MODE	SET D
F5	SET LONG INTEGER MODE	SET L

3.5 INSTRUCTION SET

Table 3-2 contains the instruction set of the FP11. Some of the symbology may not be readily apparent; therefore, a brief description is given in the following paragraphs:

a. A floating-point flip-flop, designated FD, determines whether single- or double-precision floating-point format is specified. If the flip-flop is reset, single-precision is specified and is designated by F. If the flip-flop is set, double precision is specified and is designed by D. Examples are NEG F, NEG D, SUB D, etc.

b. An integer flip-flop, designated IL, determines whether short-integer or long-integer format is specified. If the flip-flop is reset, short-integer format is specified and is designated by I. If the flip-flop is set, long-integer format is specified and is designated by L.

c. Several convert type instructions use the symbology below and are defined as follows:

$C_{IL,FD}$ – convert integer to floating

$C_{FD,IL}$ – convert floating to integer

$C_{F,D}$ or $C_{D,F}$ – convert single-float to double-float or double-float to single-float

d. Numbers in angle brackets indicate bit positions; an example is AR ⟨57:0⟩, which indicates AR bits 57 through 0.

e. UPLIM is defined as the largest possible number that can be represented in floating-point format. This number has an exponent of 377 and a fraction of all 1s. Note that UPLIM is dependent on the format specified. LOLIM is defined as the smallest possible number that is not identically zero. This number has an exponent of 001 and a fraction of all 0s except for the hidden bit.

f. Some of the octal codes listed in Table 3-2 are in the form of mathematical expressions. These octal codes can be calculated as shown in the following examples.

Example:

LD FPS instruction   Mode 3, Register 7 specified
170100+SRC
SRC field is equal to 37
Basic op code is 170100

SRC + basic op code is added to yield 170137.

(continued on next page)

Example:

LDF instruction AC2, Mode 2, and Register 6 specified

172400+AC*100+FSRC

AC = 2

2*100 = 200

172400 + 200 = 172600

FSRC is equal to 26

172600 + 26 = 172626

3.6 FP11 PROGRAMMING EXAMPLES

This paragraph shows two programming examples using the FP11 instruction set. In program example 1, A is added to B, D is subtracted from C, the quantity (A+B) is multiplied by (C-D), and the product of this multiplication is divided by X and the result stored. Example 2 calculates $DX^3 + CX^2 + BX + A$. This involves a three-pass loop, whereby each loop does the calculation indicated below.

$$AC0 = [(\underbrace{\overbrace{\underbrace{D*X}_{\text{Loop 1}}+C)*X}^{\text{Loop 2}}+B]}_{\text{Loop 3}}*X+A$$

$$AC0 = [DX^2 + CX + B] * X + A$$

$$AC0 = DX^3 + CX^2 + BX + A$$

Example 1:

```
ØØØØØØ   172467   ØØØ122   LDF    A,ACØ       ;LCAD ACØ FROM A
ØØØØØ4   172Ø67   ØØØ122   ADDF   B,ACØ       ;ACØ HAS (A+B)
ØØØØ1Ø   172567   ØØØ122   LDF    C,AC1       ;LOAD AC1 FROM C
ØØØØ14   173167   ØØØ122   SUBF   D,AC1       ;AC1 HAS (C-D)
ØØØØ2Ø   171ØØ1            MULF   AC1,ACØ     ;ACØ HAS (A+D)*(C-D)
ØØØØ22   174467   ØØØ752   DIVF   X,ACØ       ;ACØ HAS (A+D)*(C-D)/X
ØØØØ26   174Ø67   ØØØ752   STF    ACØ,Y       ;STORE (A+D)*(C-D)/X IN Y
```

Example 2:

```
ØØØ1ØØ   Ø127ØØ   ØØØØØ3           MOV    #3,%Ø       ;SET UP LOOP COUNTER
ØØØ1Ø4   Ø127Ø1   ØØØ146           MOV    #D+4,%1     ;SET UP POINTER TO
                                                       ;COEFFICIENTS
ØØØ11Ø   172526                    LDF    (6)+,AC1    ;POP X FROM STACK
ØØØ112   17Ø4ØØ                    CLRF   ACØ         ;CLEAR OUT ACØ
ØØØ114   172Ø44            LOOP;   ADDF   -(4),ACØ    ;ADD NEXT COEFFICIENT
                                                       ;TO PARTIAL RESULT
ØØØ116   171ØØ1                    MULF   AC1,ACØ     ;MULTIPLY PARTIAL RESULT
                                                       ;BY X
ØØØ12Ø   Ø77ØØ3                    SOB    %Ø,LOOP     ;DO LOOP 3 TIMES
ØØØ122   172Ø44                    ADDF   -(4),ACØ    ;ADD X TO GET RESULT
ØØØ124   174Ø46                    STF    ACØ,-(6)    ;PUSH RESULT ON STACK
```

Table 3-2
Instruction Set

Mnemonic	Instruction Description	Octal Code

CFCC	Copy Floating Condition Codes
	C ← FC
	V ← FV
	Z ← FZ
	N ← FN
		170000
		F5 Format

SETF	Set Floating Mode FD ← 0	170001 F5 Format
SETI	Set Integer Mode FL ← 0	170002 F5 Format
LDUB	Load Microbreak Register This instruction is a maintenance instruction in which the content of register R3 is gated into the UB register. When the control ROM address register matches the contents of the UB register, a scope sync is generated. If the FP11 is in maintenance mode (FMM=1), an interrupt is also generated and the FPU traps to the Ready state. A UB interrupt cannot be generated by the Ready state or by the states that are used to generate the U Break interrupt.	170003 F5 Format
LDSC	Load Step Counter This is a maintenance instruction in which the content of register R4 is gated into the step counter, if the FP11 is in maintenance mode (FMM=1). Whenever the step counter is loaded by an LDSC, normal loading via the microprogram is inhibited until the step counter is incremented to zero. This allows partial quotients and products to be formed for diagnostic purposes. If FMM=0, the LDSC acts as a NOP.	170004 F5 Format

Table 3-2 (Cont)
Instruction Set

Mnemonic	Instruction Description	Octal Code
STA0	Store AR in AC0 AC0 ⟨54:32⟩ ← AR ⟨57:35⟩ if FD = 0 AC0 ⟨54:0⟩ ← AR ⟨57:3⟩ if FD = 1	170005 F5 Format
MRS	Maintenance Right Shift AR ← AR/2; QR ← QR/2	170006 F5 Format
STQ0	Store QR in AC0 BR ← QR; AC ⟨54:32⟩ ← BR ⟨57:35⟩ if FD = 0 AC0 ⟨54:0⟩ ← BR ⟨57:3⟩ if FD = 1	170007 F5 Format
SETD	Set Floating Double Mode FD ← 1	170011 F5 Format
SETL	Set Long Integer Mode FL ← 1	170012 F5 Format
LDFPS SRC	Load FP11's Program Status Word FPS ← (SRC)	170100 + SRC F4 Format
STFPS DST	Store FP11's Program Status Word DST ← (FPS)	170200 + DST F4 Format
STST DST	Store FP11's Status DST ← (FEC) DST + 2 ← (FEA) if not mode 0 or not immediate mode	170300 + DST F4 Format

CLRF FDST
CLRD FDST
	Clear
	FDST ← 0
	FC ← 0
	FV ← 0
	FZ ← 1
	FN ← 0
		170400 + FDST
		F2 Format

(continued on next page)

Table 3-2 (Cont)
Instruction Set

Mnemonic	Instruction Description	Octal Code
TSTF FDST	Test	170500 + FDST
TSTD FDST	FDST ← (FDST)	F2 Format
	FC ← 0	
	FV ← 0	
	FZ ← 1 if (FDST) = 0, else FZ ← 0	
	FN ← 1 if (FDST) < 0, else FN ← 0	
ABSF FDST	Absolute	170600 + FDST
ABSD FDST	FDST ← –(FDST) if (FDST) < 0; else FDST ← (FDST)	F2 Format
	FC ← 0	
	FV ← 0	
	FZ ← 1 if (FDST) = 0; else FZ ← 0	
	FN ← 0	
NEGF FDST	Negate	170700 + FDST
NEGD FDST	FDST ← –(FDST)	F2 Format
	FC ← 0	
	FV ← 0	
	FZ ← 1 if (FDST) = 0, else FZ ← 0	
	FN ← 1 if (FDST) < 0, else FN ← 0	
LDEXP SRC, AC	Load Exponent	176400 + AC * 100 + SRC
	AC SIGN ← (AC SIGN)	F3 Format
	AC EXP ← (SRC) + 200	
	AC FRACTION ← (AC FRACTION)	
	FC ← 0	
	FV ← 1 if | AC |> UPLIM; else FV ← 0	
	FZ ← 1 if (AC) = 0, else FZ = 0, else FZ ← 0	
	FN ← 1 if (AC) < 0, else FN = 0, else FN ← 0	

(continued on next page)

Table 3-2 (Cont)
Instruction Set

Mnemonic	Instruction Description	Octal Code

LDCIF SRC, AC
LDCID SRC, AC
LDCLF SRC, AC or
LDCLD SRC, AC
LDCIF – single integer to single float
LDCID – single integer to double float
LDCLF – long integer to single float
LDCLD – long integer to double float
	Load and convert from integer to floating
	AC ← $C_{FL,FD}$ (SRC)
	FC ← 0
	FV ← 0
	FZ ← 1 if (AC) = 0; else FZ ← 0
	FN ← 1 if (AC) < 0; else FN ← 0
	$C_{FL,FD}$ specifies conversion from a 2's complement integer with precision I or L to a floating-point number of precision F or D. If integer flip-flop IL = 0, a 16-bit integer (I) is specified, and if IL = 1, a 32-bit integer (L) is specified. If floating-point flip-flop FD = 0, a 32-bit floating-point number (F) is specified, and if FD = 1, a 64-bit floating-point number (D) is specified. If a 32-bit integer is specified and addressing mode 0 or immediate mode is used, the 16-bits of the source register are left justified, and the remaining 16-bits are zeroed before the conversion.
		177000 + AC * 100 + SRC
		F3 Format

STEXP AC, DST	Store Exponent
	DST ←AC EXPONENT -200
	FC ← 0
	FV ← 0
	FZ ← 1 if (DST) = 0; else FZ ← 0
	FN ← 1 if (DST) < 0; else FN ← 0
	C ← FC
	V ← FV
	Z ← FZ
	N ← FN
		175000 + AC * 100 + DST
		F3 Format

(continued on next page)

Table 3-2 (Cont)
Instruction Set

Mnemonic	Instruction Description	Octal Code

STCFI AC, DST
STCFL AC, DST
STCDI AC, DST or
STCDL AC, DST
 	
STCFI = Single float to single integer
STCFL = Single float to long integer
STCDI = Double float to single integer
STCDL = Double float to long integer
	Store Convert from Floating to Integer
	Destination receives converted AC if the resulting integer number can be represented in 16 bits (short integer) or 32 bits (long integer). Otherwise, destination is zeroed and C bit is set.
	FV ← 0
	FZ ← 1 if (DST) = 0; else FZ ← 0
	FN ← 1 if (DST) < 0; else FN ← 0
	C ← FC
	V ← FV
	Z ← FZ
	N ← FN
	When the conversion is to long integer (32 bits) and address mode 0 or immediate mode is specified, only the most significant 16 bits are stored in the destination register.
		175400 + AC * 100 + DST
		F3 Format

STF AC, FDST
STD AC, FDST
	Floating Store
	FDST ← (AC)
	FC ← FC
	FV ← FV
	FZ ← FZ
	FN ← FN
		174000 + AC * 100 + FDST
		F1 Format

DIVF FSRC, AC
DIVD FSRC, AC
	Floating Divide
	AC ← (AC)/(FSRC) if | (AC)/(FSRC) |> LOLIM; else AC ← 0
	FC ← 0
	FV ← 1 if | (AC) |> UPLIM
	FZ ← 1 if (AC) = 0; else FZ ← 0
	FN ← 1 if (AC) < 0; else FN ← 0
		174400 + AC * 100 + FSRC
		F1 Format

(continued on next page)

Table 3-2 (Cont)
Instruction Set

Mnemonic	Instruction Description	Octal Code

LDCDF FSRC, AC
LDCFD FSRC, AC
	Load Convert Double to Floating or Floating to Double
	AC ← $C_{F,D\ v\ D,F}$(FSRC)
	FC ← 0
	FV ← 1 if $\lvert(AC)\rvert > UPLIM$; else FV ← 0
	FZ ← 1 if (AC) = 0; else FZ ← 0
	FN ← 1 if (AC) < 0; else FN ← 0
	If the current format is single-precision floating-point (FD = 0), the source is assumed to be a double-precision number and is converted to single precision. If the floating truncate bit is set the number is truncated; otherwise, it is rounded. If the current format is double-precision (FD = 1), the source is assumed to be a single-precision number and is loaded left justified in the AC. The lower half of the AC is cleared.
		177400 + AC * 100 + FSRC
		F1 Format
		F,D – single-precision to double-precision floating
		D,F – double-precision to single-precision floating

ADDF FSRC, AC
ADDD FSRC, AC
	Floating Add
	AC ← (AC) + (FSRC) if $\lvert(AC) + (FSRC) \leqslant LOLIM$
	else AC ← 0
	FC ← 0
	FV ← 1 if $\lvert(AC)\rvert > UPLIM$; else FV ← 0
	FZ ← 1 if (AC) = 0; else FZ ← 0
	FN ← 1 if (AC) < 0; else FN ← 0
		172000 + AC * 100 + FSRC
		F1 Format

LDF FSRC, AC or
LDD FSRC, AC
	Floating Load
	AC ← (FSRC)
	FC ← 0
	FV ← 0
	FZ ← 1 if (AC) = 0; else FZ ← 0
	FN ← 1 if (AC) < 0; else FN ← 0
		172400 + AC * 100 + FSRC
		F1 Format

(continued on next page)

Table 3-2 (Cont)
Instruction Set

Mnemonic	Instruction Description	Octal Code

SUBF FSRC, AC or
SUBD FSRC, AC
	Floating Subtract
	AC ← (AC) – (FSRC) if |(AC) – (FSRC) |≥ LOLIM
	else AC ← 0
	FC ← 0
	FV ← 1 if |(AC) |> UPLIM; else FV ← 0
	FZ ← 1 if (AC) = 0; else FZ ← 0
	FN ← 1 if (AC) < 0; else FN ← 0
		173000 + AC * 100 + FSRC
		F1 Format

CMPF FSRC, AC
CMPD FSRC, AC
	Floating Compare
	FC ← 0
	FV ← 0
	FZ ← 1 if (FSRC) – (AC) = 0; else FZ ← 0
	FN ← 1 if (FSRC) – (AC) < 0; else FN ← 0
		173400 + AC * 100 + FSRC
		F1 Format

MULF FSRC, AC
MULD FSRC, AC
	Floating Multiply
	AC ← (AC) * (FSRC) if |(AC) * (FSRC) |≥ LOLIM
	else AC ← 0
	FC ← 0
	FV ← 1 if |(AC) |> UPLIM; else FV ← 0
	FZ ← 1 if (AC) = 0; else FZ ← 0
	FN ← 1 if (AC) < 0; else FN ← 0
		171000 + AC * 100 FSRC
		F1 Format

MODF FSRC, AC
MODD FSRC, AC
	Floating Modulo
	AC V 1 ← integer part of [(AC) * (FSRC)]
	AC ← fractional part of (AC) * (FSRC) – (AC V 1) if
	|(AC) * (FSRC) |≥ LOLIM or FIU = 1; else AC ← 0
	FC ← 0
	FV ← 1 if |(AC) |> UPLIM; else FV ← 0
	FZ ← 1 if (AC) = 0; else FZ ← 0
	FN ← 1 if (AC) < 0; else FN ← 0
		171400 + AC * 100 + FSRC
		F1 Format

(continued on next page)

Table 3-2 (Cont)
Instruction Set

Mnemonic	Instruction Description	Octal Code
(cont)	The product of (AC) and FSRC) is 48 bits in single-precision floating-point format or 59 bits in double-precision floating-point format. The integer part of the product [(AC) * (FSRC)] is found and stored in AC V 1. The fractional part is then obtained and stored in AC. Note that multiplication by 10 can be done with zero error, allowing decimal digits to be stripped off with no loss in precision.	

STCFD AC, FDST
STCDF AC, FDST
	Store Convert from Floating to Double or Double to Floating
	FDST ← $C_{F,D \vee D,F}$ (AC)
	FC ← 0
	FV ← 1 if |(AC)| > UPLIM; else FV ← 0
	FZ ← 1 if (AC) = 0; else FZ ← 0
	FN ← 1 if (AC) < 0; else FN ← 0
		176000 + AC * 100 + FDST
		F1 Format
		F,D – single-precision to double-precision floating
		D,F – double-precision to single-precision floating

CHAPTER 4
CONTROL ROM

4.1 INTRODUCTION

Figure 1-2 shows a simplified block diagram of the Floating Point Processor, which consists of the fraction calculation logic, exponent calculation logic, and scratchpad accumulators. Figure 4-1 expands this block diagram to show the various data paths of the FP11 and also to show each of the multiplexers and major registers. These registers and multiplexers are described below.

FIR ⟨11:0⟩ Instruction Register – The most significant four bits represent the 17_8 op code for floating point and should be 1s. When the other bits are loaded into the instruction register, these four bits are checked; if they are not all 1s, an illegal instruction trap occurs.

DIMX – An input multiplexer that selects data in or address in from the CPU. In the Ready state, the address is automatically selected so that the address of the floating-point instruction can be temporarily stored in the FP11 at the same time that data is clocked into the FP11 FIR.

EMX – The EMX selects one of four input sources to the B side of the EALU. The four inputs are:

a. BA register

b. DIMX output

c. CNST (constant)

d. Step Counter (when the step counter is selected, bits 15 through 6 are 0s).

BA Register – A 16-bit Temporary Storage register that feeds the B side of EALU via the EMX.

BD Register – A 16-bit storage register used to send data to the CPU and to the A side of the EALU.

EALU – An exponent arithmetic logic unit capable of performing both arithmetic and logical functions between the A and B inputs. The EALU is 16 bits wide.

Step Counter – A 6-bit up counter used to count the number of shifts required for normalization of the fraction and used to count the number of steps performed in multiplication or division or long shift subroutines.

Ubreak Register – An 8-bit register used to set up break points in the microprogram for diagnostic purposes.

FPS – The floating-point status register contains the current status of the FP11 including floating condition codes and interrupt enable status.

BMX – A multiplexer that selects one of four sources as inputs to the BA and BD registers. The four inputs are:

a. EALU

b. ACH – Selects most significant 16 bits of the 32-bit accumulator specified.

c. ACL – selects least significant 16 bits of the 32-bit accumulator specified.

d. EXP – strips off exponent portion of word (8 bits) contained in accumulator and right justifies it. Remaining bits are zeroed.

AC_i⟨63:0⟩, i = 0 through 7 – There are eight 64-bit wide accumulators in the FP11. Each accumulator is divided into four 16-bit segments (3, 2, 1, and 0 as described in Chapter 1). The high-order 32-bits, the low-order 32-bits or a 16-bit segment can be accessed. Data written into the scratchpad accumulator is inverted when read out of the scratchpad. This is compensated for by writing the 1's complement of the data into the scratchpad.

ACMX – The ACMX is 32 bits wide and selects one of four 32-bit input sources for writing into the accumulators. The ACMX allows the floating-point status to be written into the accumulator, allows the exponent and fraction to be assembled from the EALU and FALU into floating-point format, and allows the least significant bits ⟨34:3⟩ of FALU to be written into the accumulators.

QR – A 60-bit wide left-right shift register, which is loaded from the scratchpad in two segments. This is accomplished by LDQ1 and LDQ0 load signals.

BR – A 60-bit holding register, which receives inputs via the QR. The BR cannot be shifted.

AR – A 60-bit left-right shift register. In all arithmetic operations where the result is to be normalized, normalization occurs in the AR.

FMX – Allows the appropriate bit of the AR to be loaded into the B side of the FALU for rounding operations. The FMX also allows insertion of 1 in the appropriate position of the FALU to provide for the incrementing of integer numbers.

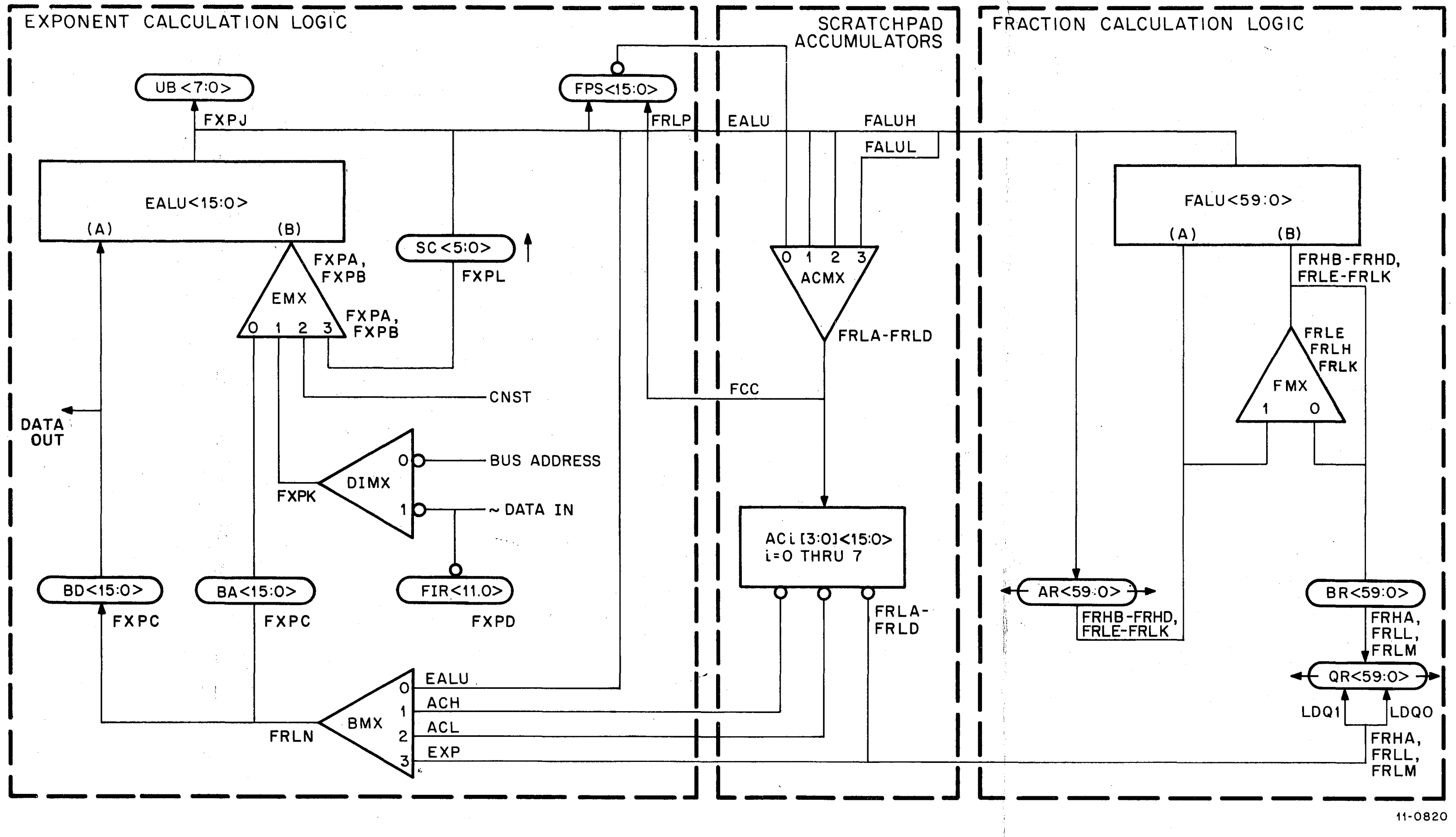
FALU – A 60-bit wide fractional arithmetic logic unit that has the capability of performing arithmetic and logical operations between the A and B inputs. Two levels of carry look-ahead are provided. The controls of the EALU and FALU are ganged together.

4.2 CONTROL ROM

The FP11 utilizes a control ROM (read-only memory) to implement microprogramming techniques. A microprogram is a sequence of control operations. Control operations, for example, might involve a sequence of information transfers from one register to another, which may take place directly or through an adder or other logical network as determined by the outputs of the read-only memory.

The control ROM in the FP11 is comprised of 256 64-bit words. Eight bits of each word represents the next address of the microprogram. If certain branch conditions are satisfied, the control ROM causes the next address to be modified and the microprogram, instead of branching to the next address, branches to the modified address. This action is shown in Figure 4-2. Note that the CRAR (Control ROM Address Register) specifies the next address. The instruction in this address is executed and, if the branch conditions are not satisfied, the 8-bit address in this instruction represents the next address of the microprogram. The following paragraphs introduce the ROM flow diagrams and associated symbology.

Asynchronous conditions can cause the microprogram to trap to specific microaddresses rather than continue in the normal sequence. These traps can be caused by initialization and 11/45 abort conditions, by a microbreak (which occurs when a control ROM address compares with a presettable address in maintenance mode), and by the floating minus zero trap, which occurs when a minus zero is detected.



DATA PATH DEFINITION

ACMX0 ⟨31⟩ ← ~ BN; ACMX0 ⟨30⟩ ← BZ; ACMX0 ⟨29:16⟩ ← 37777; ACMX0 ⟨15:0⟩ ← FPS
ACMX1 ⟨31:16⟩ ← EALU ⟨15:00⟩; ACMX1 ⟨15:00⟩ ← EALU ⟨15:00⟩
ACMX2 ⟨31⟩ ← ~ SD; ACMX2 ⟨30:23⟩ ← EALU ⟨07:00⟩; ACMX2 ⟨22:00⟩ ← FALU ⟨57:35⟩
ACMX3 ⟨31:00⟩ ← FALU ⟨34:03⟩

BMX0 ⟨15:00⟩ ← EALU ⟨15:00⟩
BMX1 ⟨15:00⟩ ← AC_i [3] ⟨15:00⟩ or AC_i [1] ⟨15:00⟩
BMX2 ⟨15:00⟩ ← AC_i [2] ⟨15:00⟩ or AC_i [0] ⟨15:00⟩
BMX3 ⟨15:08⟩ ← 0 ; BMX3 ⟨07:00⟩ ← AC_i [3:2] ⟨30:23⟩ or AC_i [01:0] ⟨30:23⟩

EMX0 ⟨15:00⟩ ← BA ⟨15:00⟩
EMX1 ⟨15:00⟩ ← DIMX ⟨15:00⟩
EMX2 ⟨15:00⟩ ← CNST ⟨15:00⟩
EMX3 ⟨15:06⟩ ← 0 ; EMX3 ⟨05:00⟩ ← SC ⟨05:00⟩

FMX0 ⟨02⟩ ← BR ⟨35⟩ ; FMX0 ⟨01⟩ ← BR ⟨19⟩ ; FMX0 ⟨00⟩ ← BR ⟨3⟩
FMX1 ⟨02⟩ ← AR ⟨34⟩ ; FMX1 ⟨01⟩ ← 1; FMX0 ⟨00⟩ ← AR ⟨02⟩

LDQ1 = QR ⟨59⟩ ← 0 ; QR ⟨58⟩ ← 1 if AC_i [3:2] ⟨30:24⟩ ≠ 0 else QR ⟨58⟩ ← 0
QR ⟨57:35⟩ ← AC_i [3:2] ⟨22:0⟩
LDQ0 = QR ⟨34:3⟩ ← AC_i [1:0] ⟨31:0⟩ ; QR ⟨2:0⟩ ← 0

Figure 4-1 FP11 Data Paths

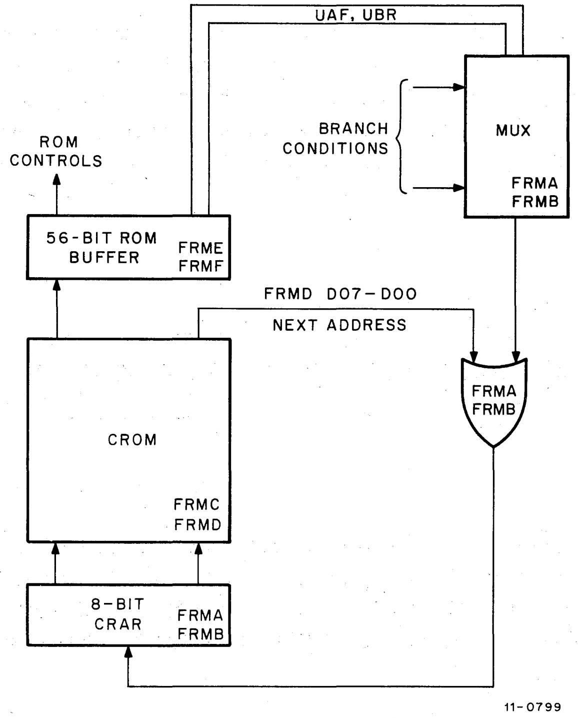
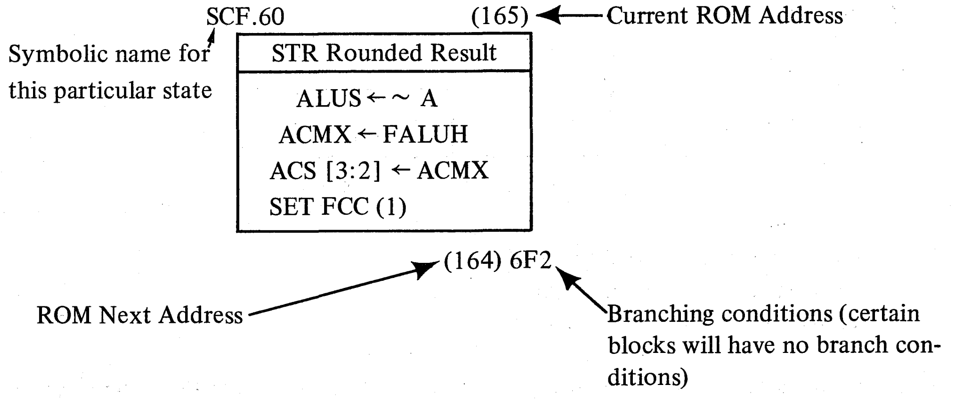


Figure 4-2 Control ROM Simplified Block Diagram

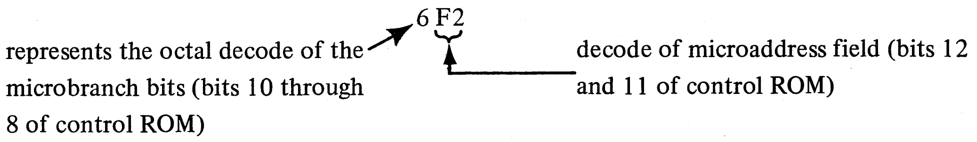
4.2.1 Control ROM Flow Diagram

This section describes the flow diagrams associated with the FP11. General points concerning the flow diagram symbology are described first, followed by Table 4-1 which lists and defines each of the statements found in the flow diagram.

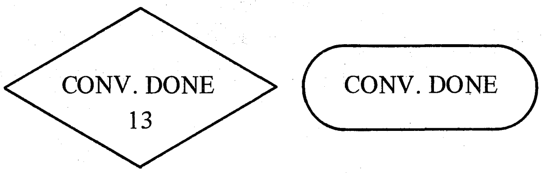
1. The flow diagram contains blocks with designators above the upper left and right corners of each block and below the right corner of each block. These are defined as shown in the sample block reproduced from sheet 13.



The branching conditions are designated as follows:


6 F2
represents the octal decode of the microbranch bits (bits 10 through 8 of control ROM)
decode of microaddress field (bits 12 and 11 of control ROM)

2. The flow diagram contains diamond shaped symbols with connector names listed inside. Below the connector name is the sheet reference. Normally, the diamond is connected to an oval shaped symbol of the same connector name. For example, the following symbol is reproduced from sheet 13 of the flow diagram. This indicates that the flow is connected to an oval symbol with the designation CONV DONE. This oval symbol is on sheet 13 as referenced by the number in the bottom of the diamond.



3. Certain connector names have numbers following them, which are used to differentiate between connectors of the same category. For example, on sheet 11 of the flow diagram there are diamond symbols designated NORM 10, NORM 20, NORM 30, and NORM 40. These symbols are connected to oval symbols designated NORM 10, NORM 20, NORM 30, and NORM 40, respectively, on sheet 12.

4. Several statements of the following forms:

   AC 7 [0] ← . . . .
   ACS [3:2] ← . . . .
   ACD [3:2] ← . . . .
   ACD V1 [3:2] ← . . . .

   These statements refer to the accumulator and the specific words referenced.

The 7 after the AC in the first statement references Accumulator 7 – one of the eight accumulators available to the microprogram. The S following the AC in the second statement specifies the source accumulator designated by FIR bits 2 through 0, while the D following the AC in the third statement specifies the destination accumulator designated by bits 7 and 6 of the FIR if address mode 0 is used; otherwise, AC6 is the destination accumulator. The number or numbers in brackets in each statement designate the portion of the accumulator word, as shown in the following example:

3	2	1	0
63 48	47 32	31 16	15 0

[3:2] specifies bits 63 through 32
[3:0] specifies bits 63 through 0
[1:0] specifies bits 31 through 0

The last statement specifies a logical OR function of (ACD) OR 1 and is used in the MODF instruction. The truth table for this statement is as follows:

FIR 7	FIR 6	ACD	ACD V 1
0	0	0	1
0	1	1	1
1	0	2	3
1	1	3	3

In the MODF instruction, the integer portion of the number is stored first followed by the fractional part. If an odd accumulator is specified, the integer and fraction are stored in the same accumulator; however, the integer part is stored first and is destroyed by the storing of the fraction. If an even accumulator is specified, the integer is stored in an odd numbered accumulator and the fraction is stored in an even numbered accumulator, which is one less than the odd numbered accumulator.

Table 4-1
Flow Diagram Statements

Statement	Description
ABSF	A branch specified if the current instruction is an ABS instruction with single- or double-precision floating point specified.
ACMX ← EALU	The output of the EALU (bits 15 through 0) is gated through the ACMX.
ACMX ← FALUH	The high-order bits of the FALU (bits 57 through 35), bits 7 through 0 of the EALU, and ~SD are gated through ACMX.
ACMX ← FALUL	The 32 bits (bits 34 – 3) of FALU are gated through ACMX.
ACMX ← FPS	The floating-point status word is gated through the ACMX. BN and BZ can also be gated through ACMX to set FZ and FN, respectively.
ADDF	A branch path specified for an add, subtract, or compare instruction using floating-point format.
ALU'S ← A PLUS B PLUS 1	The output of the EALU and FALU contains the sum of the data on the A and B inputs plus one.
ALU'S ← A Λ ~ B	The output of the EALU and FALU contains the data on the A input ANDed with the complement of the data on the B input.
ALU'S ← A MINUS B	The output of the EALU and FALU contains the data on the A input minus the data on the B input.
ALU'S ← A PLUS B	The output of the EALU and FALU contains the sum of the data on the A and B inputs.
ALU'S ← A	The output of the EALU and FALU contains the data on the A input.
ALU'S ← A MINUS 1	The output of the EALU and FALU contains the data on the A input minus 1.
ALU'S ← ~ A	The output of the EALU and FALU has the complement of the data on the A input.
ALU'S ← ~ (AVB)	The output of the EALU and FALU contains the complement of the logical OR function of the A and B inputs.
ALU'S ← B	The output of the EALU and FALU contains the data on the B input.

(continued on next page)

Table 4-1 (Cont)
Flow Diagram Statements

Statement	Description
ALU'S ← ~ B	The output of the ALU contains the complement of the data on the B input.
ALU'S ← 1	The output of the EALU and FALU contains all 1s.
ALU'S ← 0	The output of the EALU and FALU contains all 0s.
AR ⟨59⟩ (0)	Indicates a positive sign bit.
AR ⟨59⟩ (1)	Indicates a negative sign bit.
AR ⟨59:58⟩(0)	Indicates an unnormalized number (0.0) with bits 59 and 58 on a 0.
AR ⟨59:58⟩ (1)	Denotes the number is a normalized number (0.1).
AR ⟨59:58⟩ (2)	Denotes the number is an unnormalized number (1.0). A right shift of 1 causes this number to become normalized.
AR ⟨59:58⟩ (3)	Indicates an unnormalized number (1.1) with bits 59 and 58 on a 1.
AR ← FALU	The contents of FALU is loaded in the AR.
BA ← BMX	The BA register is loaded from the BMX.
BB1Z (B Registers Byte 1 Zero)	Indicates the upper eight bits of the last data word loaded in either the BA or BD register are zeros.
BD ← BMX	The BD register is loaded from the BMX.
BMX ← ACH	The high-order 16 bits of the 32-bit wide scratchpad output are gated through the BMX.
BMX ← ACL	The low-order 16 bits of the 32-bit wide scratchpad output are gated through the BMX.
BMX ← EALU	The output of the EALU is gated through BMX.
BMX ← EXP	The 8 bits of exponent from the AC are gated through the least significant 8 bits (bits 7 through 0) of the BMX. Upper 8 bits (bits 15 through 8) of BMX are zeroed.
BN – (B Registers Negative)	Indicates that the last data word loaded in the BA or BD register is negative.
BR ← QR	The contents of the QR are transferred to the BR.
BR ← 0	The BR register is zeroed.
BZ – (B Registers Zero)	Indicates the last data word loaded in the BA or BD register is zero.
CERR INT	Conversion error interrupt.
CFCC	Copy floating condition codes instruction.
CLRF	A clear instruction specifying single- or double-precision floating point.
CMPF	A compare instruction specifying single- or double-precision floating point.
CONV SP	A group of instructions which include STEXP, STCFI, and STCFD.

(continued on next page)

Table 4-1 (Cont)
Flow Diagram Statements

Statement	Description
DIMX ← DATA ADDRESS	The data address is gated through DIMX.
DIVF	A divide instruction specifying single- or double-precision floating point.
EMX ← BA	The output of the BA is gated through EMX.
EMX ← CNST 1	The output of EMX contains a constant of 1.
EMX ← CNST.2	The output of EMX contains a constant of 2.
EMX ← CNST.4	The output of EMX contains a constant of 4.
EMX ← CNST.10	The output of EMX contains a constant of 10.
EMX ← CNST.12	The output of EMX contains a constant of 12.
EMX ← CNST.17	The output of EMX contains a constant of 17.
EMX ← CNST.21	The output of EMX contains a constant of 21.
EMX ← CNST.31	The output of EMX contains a constant of 31.
EMX ← CNST.35	The output of EMX contains a constant of 35.
EMX ← CNST.71	The output of EMX contains a constant of 71.
EMX ← CNST.75	The output of EMX contains a constant of 75.
EMX ← CNST.200	The output of EMX contains a constant of 200.
EMX ← CNST.220	The output of EMX contains a constant of 220.
EMX ← CNST.100000	The output of EMX contains a constant of 100000.
EMX ← DATA IN	The output of EMX contains the input data from the CPU or the IU.
EMX ← DIMX	The output of DIMX is applied to EMX.
ENABLE FM0 INTERRUPT	Enables microtrap if 1's complemented floating minus zero is present at output of ACMX.
ENABLE FP REG WR	Indicates that the CPU is to copy data from the FP11 into a general register.
ENABLE FP SYNC	Enables FP SYNC to be generated at TS2 of next ROM state.
EQ	Equal branch (indicates that the exponents of the operands are equal or the exponent of the MODF instruction is zero).
FD ← 1 IF SET D	If the SET D instruction is specified, the FD flip-flop is set.
FD ← 0 IF SET F	If the SET F instruction is specified, the FD flip-flop is zeroed.
FD (1) (Bit 7 of FPS)	When the flip-flop is set, double-precision floating point is specified and, when reset, single-precision floating point is specified.
FINT	All errors branch to floating interrupt (FINT) ROM location.
FIR ← DATA IN	The data input is transferred to the floating-point instruction register.

(continued on next page)

Table 4-1 (Cont)
Flow Diagram Statements

Statement	Description
FIU (Bit 10 of FPS)	Floating Interrupt on Underflow. This bit, if set, causes a floating interrupt on underflow to occur if an underflow condition is detected.
FIV (Bit 9 of FPS)	Floating Interrupt on Overflow. With this bit set, an overflow causes an interrupt.
FL ← 1 IF SET L	If the SET L instruction is specified, the FL flip-flop is set.
FL ← 0 IF SET I	If the SET I instruction is specified, the FL flip-flop is zeroed.
FMX ← F. RND	If single-precision floating-point format, AR bit 34 is fed to bit 35 on the B input to FALU via FMX. If double-precision floating-point format, AR bit 2 is fed to bit 3 on the B input to FALU via FMX.
FMX ← I. INC	Inserts a 1 in bit 35 of the B input to FALU if short-integer format is specified, or inserts a 1 in bit 19 of the B input to FALU if long-integer format is specified.
FPCI ← DATI	Informs the CPU that a DATI cycle is requested.
FPCI ← DATO	Informs the CPU that a DATO cycle is requested.
FPS ← EALU	The output of EALU is transferred to the FPS register.
FRAC DIV	Initiates a divide subroutine and causes the ROM to pause until completion of the subroutine.
FRAC MUL	Initiates a multiply subroutine and causes the ROM to pause until completion of the subroutine.
FT (1) (Bit 5 of FPS)	This bit, when set, causes the result to be truncated and, when reset, causes the result to be rounded.
FV – Floating Overflow (Bit 1 of FPS)	A condition code indicating an overflow condition.
GT	Greater than branch. Indicates the exponent in the BD register is greater than the exponent in the BA register or the MODF exponent is greater than 200.
ILL. OP. CODE	An undefined op code.
IMMEDIATE	Specifies address mode 2 and register 7.
INC ADDRESS	Indicates to the CPU that the current address of the data is to be incremented by 2.
INIT V 1145 ABORT	If the 11/45 sends an 1145 ABORT or an INIT signal, the FP11 traps to the Ready state.
LD AC6	A branching path taken by instructions which require that one operand be fetched from memory.
LDCIF	A load instruction which loads and converts a number from integer to floating-point format.

(continued on next page)

Table 4-1 (Cont)
Flow Diagram Statements

Statement	Description
LD FPS	The instruction that causes the floating-point status to be loaded in the FP11 floating-point status register.
LDSC	An instruction that causes the step counter to be loaded from an external source.
LD UB	An instruction that loads the microbreak register from an external source.
LOAD CL	Indicates a class of instructions that require operands from memory.
LS. AR. 1	Left shifts the AR one bit position and inserts a 0 in AR00.
LS.QR.SC	Left-shift the QR by the number contained in the step counter. The step counter contains the 1's complement of the number of shifts desired.
LT	Less than branch. Indicates that the exponent in the BD register is less than the exponent in the BA register, or the exponent in the MODF instruction is less than 200.
MGT	Much greater than braneh. The number cannot be aligned within the boundaries of the AR and BR registers.
MLT	Much less than branch. The number cannot be aligned within the boundaries of the AR and BR registers.
NEGF	The negate instruction specified with single- or double-precision floating-point format.
NO. MEM. CL.	Indicates a nonmemory reference instruction.
NRM. AR	Initiates a hardware subroutine that normalizes the number in the AR. The number of shifts required to normalize is contained in the step counter.
QR ← LDQ1	The QR is loaded as follows: QR ⟨59⟩ ← 0; QR ⟨58⟩ ← 1 if exponent of word is not zero (hidden bit), else QR ⟨58⟩ ← 0; QR ⟨57:35⟩ ← AC_i [0:1] ⟨22:0⟩.
QR ← LDQ0	QR ⟨34:3⟩ ← AC_i ⟨31:0⟩; QR ⟨2:0⟩ ← 0.
QR ← 0	QR register is cleared.
REQ ← 1	Sets the REQ (request) flip-flop.
REQ ← 0	REQ flip-flop is cleared.
RS.AR. 1	Right shift the AR one place. A 0 is shifted into AR59.
RS.AR. SC	Right shift the AR by the number contained in the step counter (1's complement). Zeros are shifted into the AR.
RS.QR. 1	Right shift QR one bit position and shift in a 0 into QR bit 59.
RS.QR. SC (0 IN)	Right shift the QR by the number contained in the step counter (1's complement). A 0 is shifted into QR bit 59.
RS.QR.SC (1 IN)	Right shift the QR by the number contained in the step counter (1's complement). Shift a 1 into QR bit 58. QR bit 59 is cleared.

(continued on next page)

Table 4-1 (Cont)
Flow Diagram Statements

Statement	Description
SC ← EALU	Step counter is loaded with number contained in EALU.
SD ← SCR OUT ⟨31⟩	Bit 31 from the scratchpad accumulator is transferred to SD.
SD ← SS	Sign of source is loaded into sign of destination.

SD ← ~ SS IF SUB
ELSE SD ← SS
	If subtract instruction is specified, sign of destination is loaded with *complement* of sign of source; otherwise, sign of destination is loaded with sign of source.

SD ← SS ∀ SD	The exclusive OR of SS and SD.
SEND FP EXC TRAP	Signals the CPU to trap through the floating-point trap vector.
SET FCC (0)	FN is set by ACMX ⟨31⟩ (0); FZ is set by ACMX ⟨30:23⟩ (377); FV and FC are cleared.
SET FCC (1)	FN is set by ACMX ⟨31⟩ (0); FZ is set by ACMX ⟨30:23⟩ (377); FV is set by EALU ⟨8⟩ (1); FC is cleared.
SET FCC (2)	FN is set by ACMX ⟨31⟩ (0); FZ is set by ACMX ⟨30:23⟩ (377); FC is set to 1; FV is cleared.
SET MODES	A branch that the SET F, SET D, SET I, or SET L instructions follow.
SS ← 1	A 1 is loaded in the sign of the source.
SS Λ SD ← 0	Sign of source and sign of destination are zeroed.
STORE. CL	Indicates store class of instructions.
UB ← EALU	The μbreak register is loaded with the output of EALU.
WAIT FOR FP ACKN	The FP11 goes in the Wait state and waits for FP ACKN from the CPU or IU. FP ACKN is sent when the FP EXC TRAP is acknowledged.
WAIT FOR FP ATTN	The FP11 goes into the Wait state and waits for FP ATTN from the CPU or from the IU.
– 0 TRAP	Floating minus zero trap.

4.2.2 ROM Field Descriptions

Each block on the set of flow diagrams represents a specific ROM word. The number of ROM words necessary to execute a floating-point instruction are dependent on the instruction. Table 4-2 shows how each ROM word is subdivided into fields and briefly defines the purpose of each field. Several fields are unique and require further explanation. One is bit ⟨58⟩, the redefined constant bit. If this bit is a 0, bits ⟨57:53⟩ of the constant field are not affected. If this bit is a 1, the constants specified by bits ⟨57:53⟩ of the ROM word are redefined. For example, if bit ⟨58⟩ is a 0, bits ⟨57:54⟩ are 1s and bit ⟨53⟩ is a 0, a constant of 74 is specified. If bit ⟨58⟩ now becomes a 1, bit 53 takes on a new meaning whereby the FP11 issues FP TRAP and waits for FP ACKN (see ⟨58⟩ in Table 4-2). These bits can be microcoded also: for example, if bit 57 were also a 0, detection of minus 0 would also be enabled.

Table 4-2
ROM Fields

Bits	Field	Field Setting	Definition
⟨63⟩	DISBL 1	0	Clears FP REQ
		1	NOP
⟨62⟩	DISBL 0	0	Clears ICLR, 20 ABORT, INITF and ABORTF
		1	NOP
⟨61:59⟩	CONTROL SEL 2- CONTROL SEL 0	0	LOAD FPSC
		1	LOAD UBC
		2	FP REG WR
		3	DISABLE SYNC
		4	DISABLE SC
		5	FIR CLK
		6	Not used
⟨58⟩	RDFN CNSTF (Redefined Constant Field)	0	NOP
		1	Constant field (bits ⟨57:53⟩) redefined as follows: BIT 57 = CNT 4(0); enables detection of minus 0 BIT 56 = CNT 3(0); enables DATI BIT 55 = CNT 2(0); not used BIT 54 = CNT 1(0); wait for FP ATTN BIT 53 = CNT 0(0) issue FP TRAP and wait for FP ACKN
⟨57:53⟩	CNST F4–CNST F0 (constant field)	0	200 This field specifies the following
		1	1 list of constants
		2	2
		3	3
		4	4
		5	5
		6	6
		7	7
		10	10
		11	100000
		12	12
		13	13
		14	14
		15	100004
		16	16
		17	17
		20	220
		21	21
		22	22
		23	23
		24	24

(continued on next page)

Table 4-2 (Cont)
ROM Fields

Bits	Field	Field Setting	Definition
⟨57:53⟩ (cont)		25	25
		26	26
		27	27
		30	30
		31	31
		32	70
		33	71
		34	34
		35	35
		36	74
		37	75
⟨52⟩	SYNC	0	Enable FP SYNC
		1	NOP
⟨51⟩	D SEL (Data Select)	0	Select address
		1	Select data
⟨50⟩	SCC (Step Counter Control)	0	Load step counter
		1	NOP
⟨49⟩	BDC (BD control)	0	Load BD register
		1	NOP
⟨48⟩	ADDR INCR (Address Increment)	0	Increment address of data by 2
		1	NOP
⟨47⟩	BAC (BA Control)	0	Load BA register
		1	NOP
⟨46:45⟩	EMXC1, EMXC0 (EMX Control)	0	EMX ← BA
		1	EMX ← DATA IN or ADDRESS
		2	EMX ← CNST
		3	EMX ← SC
⟨44:43⟩	FCC1, FCC0 (Floating Condition Codes)	0	FN ← ACMX ⟨31⟩ (0); FZ ← ACMX ⟨30:23⟩ (377); FV ← 0; FC ←0
		1	FN ← ACMX ⟨31⟩ (0); FZ ← ACMX ⟨30:23⟩ (377); FV ← EALU ⟨8⟩ (1); FC ← 0
		2	FN ← ACMX ⟨31⟩ (0); FZ ← ACMX ⟨30:23⟩ (377); FV ← 0; FC ← 1
		3	NOP
⟨42:41⟩	SIGNC1, SIGNC0 (Sign Control)	0	SD ← ~ SS if subtract; otherwise, SD ← SS
		1	SD ← SS ∀ SD
		2	SS ← 1
		3	NOP

(continued on next page)

Table 4-2 (Cont)
ROM Fields

Bits	Field	Field Setting	Definition
⟨40:39⟩	BMXC1, BMXC0	0	BMX ← EALU
	(BMX Control)	1	BMX ← ACH
		2	BMX ← ACL
		3	BMX ← EXP
⟨38⟩	ACRE	0	Write enable
	(AC Read)	1	Read enable
⟨37:35⟩	ACC2–ACC0	0	[3:2] This field selects the following
	(AC Control)	1	[3] ACs or combinations of ACs.
		2	[2] One ACs specify 16 bits and
		3	None two ACs specify 32 bits.
		4	[1:0]
		5	[1]
		6	[0]
		7	
⟨34:32⟩	ACF2–ACF0	0	ACS (AC source) Bits ⟨2:0⟩ of instruction word specify ACS
		1	ACS or 1 (Selects odd AC)
		2	ACD
		3	ACD or 1 (Selects odd AC)
		4	AC6
		5	AC7
		6	Not used
		7	Not used
⟨31:30⟩	ACMXC1, ACMXC0	0	ACMX ← BN, BZ and FPS
	(ACMX Control)	1	ACMX ← EALU
		2	ACMX ← FALU H
		3	ACMX ← FALU L
⟨29:27⟩	CSB2–CSB0	0	Multiply BR and QR; leave result in AR.
	(Call hardware subroutine)	1	Divide AR by BR; leave result in QR.
		2	Shift AR right by the number in SC; shift in 0s.
		3	Shift AR left until normalized and count number of shifts in SC.
		4	Shift QR right by the number in SC. Shift in 0s.
		5	Shift QR left by the number in SC. Shift in 0s.
		6	Shift QR right by the number in SC. Shift in 1s (sign bit remains 0).
		7	NOP

(continued on next page)

Table 4-2 (Cont)
ROM Fields

Bits	Field	Field Setting	Definition	
⟨26:25⟩	ARC1, ARC0	0	Load AR	
	(AR Control)	1	Shift AR left. Shift 0s in.	
		2	Shift AR right. Shift 0s in.	
		3	NOP	
⟨24:23⟩	BRC1, BRC0	0	Clear and Load BR	
	(BR Control)	1	Clear BR	
		2	Load BR	
		3	NOP	
⟨22:21⟩	QRC1, QRC0	0	Load QR ⟨59:35⟩ if ACC ⟨2⟩ (0); otherwise load QR ⟨34:3⟩. QR59 is loaded with 0. QR58 is loaded with 1 if exponent is nonzero. QR2 through 0 are loaded with 0s. Load SS from SCR out 31 if ACF = 0 or 1. Load SD from SCR out 31 if ACF = 2 or 3.	
		1	Shift QR left; shift 0s in.	
		2	Shift QR right; shift 0s in.	
		3	NOP	
⟨20⟩	QRC2	0	Zero QR	
		1	NOP	
⟨19:16⟩	ALUC3–ALUC0	0	ALU ← ~ A	A = A side of ALU; B = B
	(ALU Control)	1	ALU ← ~ (A or B)	side of ALU
		2	ALU ← A – B	ALU = EALU and FALU,
		3	ALU ← 0	which are ganged together
		4	ALU ← ~ (A and B)	
		5	ALU ← ~ B	
		6	ALU ← A – B – 1	
		7	ALU ← A and ~ B	
		10	ALU ← A + B + 1	
		11	ALU ← A + B	
		12	ALU ← B	
		13	ALU ← A and B	
		14	ALU ← 1	
		15	ALU ← A – 1	
		16	ALU ← A or B	
		17	ALU ← A	

(continued on next page)

Table 4-2 (Cont)
ROM Fields

Bits	Field	Field Setting	Definition
⟨15:14⟩	FMXC1, FMXC0	0	Not used
		2	Round. Normally, BR is cleared so that AR ⟨34⟩ is added to AR ⟨35⟩ if FD = 0 or AR2 is added to AR3 if FD = 1.
		1	Normally, BR is cleared allowing AR to be incremented at bit 35 if IL = 0 or at bit 19 if IL = 1.
		3	NOP
⟨13⟩	UJP (Microjump)	0	Jump to READY if bits ⟨1:0⟩ of modified address are set.
		1	NOP
⟨12:11⟩	UAF1, UAF0 (Microaddress field–used in conjunction with UBR field (bits 10 through 8) to specify branch modification.)	0	OR function with UAD ⟨5:0⟩ if UBR ⟨0⟩ 0; otherwise OR with UAD ⟨5:2⟩ UBR O ⟨0⟩ specifies even rows, UBR O ⟨1⟩ specifies odd rows
		1	OR function with UAD ⟨0⟩ (only bit 0 can be modified)
		2	OR function with UAD ⟨1⟩ (only bit 1 can be modified)
		3	OR function with UAD ⟨1:0⟩ (bits 0 or 1 can be modified)

NOTE

The remainder of this table defines the microbranching conditions (bits 10 through 0).

(continued on next page)

Table 4-2 (Cont)
ROM Fields

Bits	Field	Field Setting	Definition					
			UAD 5	UAD 4	UAD 3	UAD 2	UAD 1	UAD 0
⟨10:8⟩	UBR2–UBR0 (Microbranch field – used in conjunction with UAF field to specify branch modification.)	0	SUB FRAC	FIRD 4	FIRD 3	FIRD 2	FIRD 1	FIRD 0
		1	FIR ⟨7⟩ (1)	FIR ⟨6⟩ (1)	FIR ⟨11⟩ (1)	FIR ⟨10⟩ (1)	AR ⟨50⟩ (0)	SD (1)
		2	RNG 2	RNG 1	RNG 0	0	BB1Z (1)	BN (0)
		3	0	0	0	FIU (1)	IL (0)	Immediate
		4	0	0	0	FT (1)	~ (FC and FIC)	FD (0)
		5	FIRD 6	FIRD 5	0	-CN V SP	~ (FV and FIV)	M0
		6	0	0	FIR ⟨8⟩ (0)	AR 58 (1)	AR ⟨59⟩ (0)	BZ (1)
		7	0	0	0	0	0	0
⟨7:0⟩	UAD 7 = UAD 0		8-bit address of next instruction. This is the address which may or may not be modified by bits ⟨12:8⟩.					

A second ROM group requiring further explanation is the microbranching fields. UAD bits 7 through 0 of the ROM word are used to define the next ROM address to be sequenced. This next address may or may not be modified under certain conditions. The UAF field (ROM bits 12 and 11) in conjunction with the UBR field (ROM bits 10 through 8 determine which bits of the next address can be modified, and which branch conditions will be used to cause the bits to be modified.

Several conditions must be met before microbranching can occur.

1. Only bits 5 through 0 of the UAD (next address) can be modified; bits 6 and 7 cannot be changed. The exception to this is the UJP and UTrap described in a subsequent paragraph.
2. Only UAD (next address) bits on a 0 can be modified, i.e., UAD bits on a zero can be modified to 1s, but UAD bits on a 1 cannot be modified.
3. The branch condition(s) being used must be true. See the chart in the definition column of the UBR field for the branch conditions. Note that some of the conditions are true when they are in the 0 state such as AR59, IL, etc.

The UBR field (ROM bits 10 through 8) is decoded to determine which conditions will be used to modify the next address, i.e., if the UBR = 6_8 we can use FIR bit 8 (0) to modify UAD bit 3, AR58 (1) to modify UAD bit 2, etc. These modifications are, of course, contingent on the prior listed conditions and also on the decoding of the UAF field (refer to Table 4-2).

The UAF field (ROM bits 12 and 11) is decoded to determine which bits of the UAD (next address) can be modified. See the definition column of the UAD field for this octal decoding. Note that a decode of 0 in the UAF field (i.e., ROM bits 12 and 11 both 0s) is further modified by the condition of UBR bit 0 (ROM bit 8). If UBR bit 0 is a 0 and the UAF field decode is a 0, then bits 5 through 0 of the next address can be modified. UBR bit 0 is a 0 for UBR octal decodes of 0, 2,4, and 6. If UBR bit 0 is a 1 (octal decodes of 1, 3, 5, and 7 of the UBR field) and the UAF field decode is a 0, then only bits 5 through 2 of the next address can be modified.

The UAD field (ROM bits 0–7) gives the next ROM address to be sequenced, subject to modification if selected.

As an example of microbranching refer to Block NRM.00 on sheet 12 of the flow diagrams. From block NRM.00 one of four different ROM addresses can be selected subject to the conditions of AR bits 58 and 59. The contents of the UAD (next address) field is octal 11 as indicated by the number 11 in parenthesis under the lower right-hand corner of the block. The term 6F0, following the next address of (11), refers to the UBR and UAF fields of the ROM word in location 273. The 6 is the octal decode of the UBR bits, and the F0 is the decode of the UAF bits. Because the UAF field is decoded to be 0, the state of UBR bit 0 must be examined to determine which bits of the next address can be modified. Because the octal decode of the UBR field is 6_8 or binary 110, UBR bit 0 is 0.

By definition, if the UAF field is 0 and UBR field is 0, bits 5 through 0 of the next address can be modified (see UAF field 0 in Table 4-2).

With an octal decode of 6 in the UBR field, only bits 3 through 0 have any conditional branches (see UBR field 6 in Table 4-2). Bits 4 and 5 cannot be modified (0 designates no change to the UAD bit). The next address in the UAD field is 11_8 thus, bits 0 and 3 are 1s, as shown below:

		UAD	Field
bits	7 6	5 4 3	2 1 0
next address = 11_8	0 0	0 0 1	0 0 1

As previously mentioned, bits on a 1 in the UAD field cannot be changed. Consequently, only bits 1 and 2 of the UAD field can be modified. These bits can be modified by AR59 and AR58, respectively, which agrees with the statements on the flow diagram.

AR⟨58⟩ modifies UAD bit 2 to a 1 and AR⟨59⟩ (0) modifies UAD bit 1 to a 1 (refer to UBR field 6 in Table 4-2), which yield the following branch possibilities:

UAD	Possible AR Conditions		
next address will be	if AR59	AR58	AR decoded
$11_8 = 00\ 001\ 001_2$	1	0	= (2)
$13_8 = 00\ 001\ 011_2$	0	0	= (0)
$15_8 = 00\ 001\ 101_2$	1	1	= (3)
$17_8 = 00\ 001\ 111_2$	0	1	= (1)

Examination of the branch conditions on sheet 12 of the flow diagram under block (273) verifies that the above microbranch decodes are correct.

The UJP field (ROM bit 13) is a special form of microbranching. This field is used to return the ROM program to the Ready state (ROM location 3 – see sheet 1 of the flow diagram).

The UJP bit, when cleared, causes the next address (UAD field) to be set to 3 (bits 7 through 2 of the UAD field being cleared) if bits 0 and 1 of this address are both 1s. This occurs if bits 0 and 1 are 1s either prior to or after address modification by the UBR and UAF fields.

For example, examine flow block NOM. 18 on sheet 2 of the flow diagram. Note the letter J following the UBR and UAF field designators below the lower right-hand corner of the block, i.e., (22) 4F1J. The J indicates that the UJP is a 0 in this ROM word, and that the next address is ROM location 3, provided bits 0 and 1 of the next address are 1s. For a next address of 22, UAD bit 1 is a 1 but UAD bit 0 is a 0. If the branch condition specified by 4F1 is not satisfied, the next address is 22. However, if FD is on a 0, UAD 0 is modified to a 1 and the next address is 23. Because UAD bits 0 and 1 are now both 1s and because the J bit is cleared, the next address is forced from state 23 to the Ready state (state 3 where bits 0 and 1 are set and bits 2 through 7 are cleared).

4.2.3 Detailed Analysis of ROM Word

Each ROM word is shown as a block on the flow diagram. As previously mentioned, a series of ROM words is necessary to execute a particular instruction. One such block is described in detail to illustrate how the ROM is implemented. The selected ROM word is block LD.12 (designated above the upper left corner of block) shown on sheet 4 of the flow diagrams. The ROM word selected is associated with the Load class of instructions, with some mode other than mode 0 (register-to-register) specified. The current address of this word is 241_8 shown above the upper-right corner of the block; the next address is 202_8 shown below the lower right corner of the block and followed by a 3F1. The first number (3) designates the UBR bit and the F1 designates the UAF bit.

Functionally, this ROM word takes a data word from the CPU, writes it into the scratch accumulator, and monitors the data for a minus 0; this procedure is done at the output of the ACMX. In order to see how these functions are accomplished, it is necessary to examine each step in the block. First, the INC ADDRESS indicates that the address of the data is to be incremented by 2. This is accomplished by making bit 48 of the ROM word a 0.

A FPCI signal generated by the FP11, specifies that data is to be transferred from the CPU to the FP11. The data is gated into the EMX by making bits 46 and 45 of the ROM word a 0 and 1, respectively. The data is then gated into the ALU where it is complemented. The reason for complementing the data is that the scratch accumulator hardware inverts the data and, therefore, a second inversion is necessary to have the true data available. From the ALU, the data is gated into the ACMX by making bits 31 and 30 of the ROM word a 0 and 1, respectively. The next element in the block specifies that the FP11 is to wait for an FP ATTN from the CPU, which accompanies a transfer of data. In order to accomplish this, bit 58 must be a 0 to redefine the CNT (constant) field, and bit 54 must be a 0 specifying that the FP11 wait for FP ATTN. The ACMX is loaded into AC6 [3]. This is accomplished by bits 37 through 35 of the ROM word on a 1_8. AC6 is used to temporarily store the data so that if a floating minus 0 occurs, the contents of the destination accumulator will not be destroyed.

The next statement specifies that the floating-point condition codes be set. This is accomplished by bits 44 and 43 of the ROM word. Because no overflow or carry occurs during a load, bits 44 and 43 should both be 0s.

The ENBL –0 INTERRUPT statement causes the hardware to examine the output of ACMX for the 1's complement of a floating minus 0.

Finally, the last statement in the block is FP SYNC, which is specified by bit 60 set to a 0. Consequently, all elements contained in this block have been specified by designated bits of the ROM word. All bits not discussed are set to the NOP or default condition and are subsequently not used at this time. A similar analysis can be followed by tracing through any of the ROM blocks in the diagram.

CHAPTER 5
ARITHMETIC ALGORITHMS

5.1 INTRODUCTION

This chapter describes the arithmetic algorithms associated with the FP11. Addition and subtraction are first described followed by multiplication and division. Several basic concepts are described before multiplication and division to familiarize the reader with the more complex concepts utilized in the FP11. State diagrams and examples of the multiply and divide algorithms are provided.

5.2 FLOATING-POINT ADDITION AND SUBTRACTION

Floating-point addition and subtraction is performed in the ALU. The exponents of the operands are processed in the EALU, and the fractions are processed in the FALU. The operands are designated source and destination operands. The following chart lists the register associated with the exponent, fraction, and sign of each operand.

Operation	**Exponent**	**Fraction**	**Sign**
Destination	BD	AR	SD
Source	BA	BR & QR	SS
Result	BD	AR	SD

For example, the exponent of the result of an addition or subtraction is found in the BD, the fraction is found in the AR, and the sign is found in SD.

The source operand is located in an AC if mode 0 is specified or located in memory if mode 0 is not specified. In the latter case, the operand in memory is transferred to AC6.

5.2.1 Description of Fraction Processing

To understand how the hardware implements the fractional part of the operand floating-point addition and subtraction, refer to Table 5-1. SS represents the sign of the operand in ACS, and SD represents the sign of the operand in ACD. The sign of the result is stored in SD. Note that the table contains four possible combinations of SS and SD for the add instruction and a similar number for the subtraction instruction. Further note that the sign that precedes the quantity in parenthesis corresponds to the sign of the destination. The sign of the result is the sign of the destination (SD) if the quantity in the parenthesis is positive, which is the case for combinations 1, 4, 6, and 7. In each of these cases, the quantities are actually added by the hardware because (|ACD| + |ACS|) is specified in each of these cases.

There are four possible combinations where the quantity in parenthesis can produce a negative result: combinations 2 and 3 for the add instruction, and combinations 5 and 8 for the subtract instruction. Note in combinations

2 and 3 that the sign of the source is the complement of the sign of the destination. If the quantity in parenthesis in combination 2 is negative, the final result is positive and the sign of the source is the sign of the result. Similarly, in combination 3, if the quantity in parenthesis is negative, the final result is negative and the sign of the source represents the sign of the result. In these two cases then the sign of the source is transferred to the sign of the destination where the sign of the result is stored. If the quantity in parenthesis is positive in either case, SD is the sign of the result. In combinations 5 and 8, listed under the subtract instruction, the sign of the source and the sign of the destination are the same and both are the opposite of the sign of the result. In combination 5, if the quantity in parenthesis is negative, the sign of the result should be negative, while SS and SD are both positive. The hardware circumvents this by complementing the sign of the source and transferring it to the sign of the destination. In combination 8, if the quantity in parenthesis is negative, the sign of the result should be positive, while SS and SD are both negative. Again, the sign of the source is complemented and transferred to the sign of the destination. If the quantity in parenthesis in combination 5 or 8 is positive, SD is the sign of the result.

Table 5-1
Add and Subtract Implementation

Combination	SS	SD	Add Instruction	Hardware Performs	Sign of Result	
					Positive Parenthesis	Negative Parenthesis
1	0	0	$ACD \leftarrow +(\lvert ACD\rvert + \lvert ACS\rvert)$	Add	$SD \leftarrow SD$	–
2	0	1	$ACD \leftarrow -(\lvert ACD\rvert - \lvert ACS\rvert)$	Subtract	$SD \leftarrow SD$	$SD \leftarrow SS$
3	1	0	$ACD \leftarrow +(\lvert ACD\rvert - \lvert ACS\rvert)$	Subtract	$SD \leftarrow SD$	$SD \leftarrow SS$
4	1	1	$ACD \leftarrow -(\lvert ACD\rvert + \lvert ACS\rvert)$	Add	$SD \leftarrow SD$	–
			Subtract Instruction			
5	0	0	$ACD \leftarrow +(\lvert ACD\rvert - \lvert ACS\rvert)$	Subtract	$SD \leftarrow SD$	$SD \leftarrow \sim SS$
6	0	1	$ACD \leftarrow -(\lvert ACD\rvert + \lvert ACS\rvert)$	Add	$SD \leftarrow SD$	–
7	1	0	$ACD \leftarrow +(\lvert ACD\rvert + \lvert ACS\rvert)$	Add	$SD \leftarrow SD$	–
8	1	1	$ACD \leftarrow -(\lvert ACD\rvert - \lvert ACS\rvert)$	Subtract	$SD \leftarrow SD$	$SD \leftarrow \sim SS$

NOTE

The microprogram is implemented such that the source can be subtracted from the destination but the destination cannot be subtracted from the source.

5.2.2 Description of Exponent Processing

During exponent alignment, the relative magnitude of the operands is detected by subtracting the smaller exponent from the larger exponent – the difference being the number of right shifts the smaller number is to be shifted. If this number is very small compared to the other number, it can be completely shifted out of the register. To avoid needless shifting in these cases, the relative magnitude of the numbers is detected and falls into one of the following five classes (see FP11 flow diagram), and Figure 5-1:

1. **EQ – (exponents equal)**. In this case, the exponents of the operands are equal and no exponent alignment is necessary. The mantissas can simply be added in the FALU.

2. **GT – (greater than).** The operand in the AR is greater than the operand in the BR. The operand in the BR is the same operand that is stored in the QR, and since the BR cannot be shifted, the QR is right shifted until the exponents associated with the mantissas in the QR and AR are equal. Then the contents of the QR is transferred to the BR. Figure 5-1 shows the various ranges of magnitudes. If single-precision floating-point format is specified, the difference between the two exponents must not be greater than 25_{10} to be in the GT class. If double-precision floating-point format is specified, the difference between the exponents can be no greater than 57_{10}. The reason that 25_{10} shifts must be exceeded when the single-precision word, for example, is only 24 bits (23 bits plus hidden bit) is that the number must be completely shifted out of the register including the rounding bit slot, before the GT class can be exceeded.

3. **LT – (less than).** The operand in the AR is less than the operand in the BR, and in this case, the AR is right shifted to fall in this class. The AR EXP - BR EXP difference should result in a number more positive than minus 25_{10} for single-precision or minus 57_{10} for double-precision floating-point.

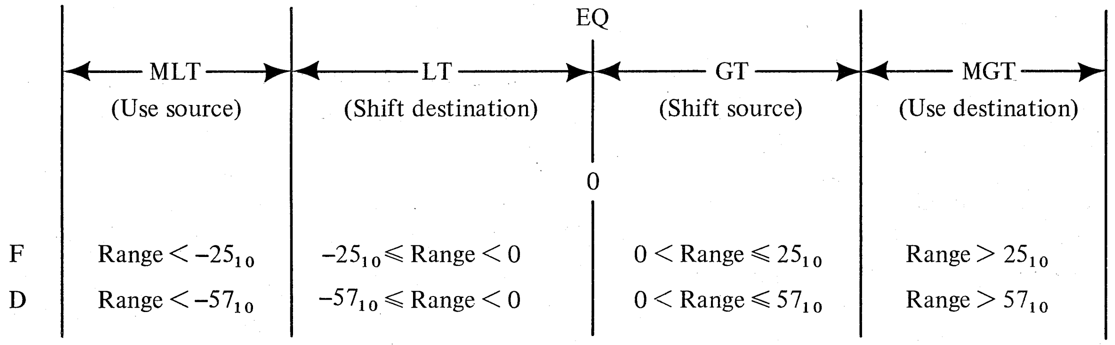
4. **MGT – (much greater than).** In this case, the operand in the AR is much greater than the operand in the BR and when the QR is right shifted to align exponents, the number contained therein would be completely shifted out of the QR. This fact is detected by the FP11 hardware; thus, unnecessary shifting is prevented. Effectively, the operand in the AR is the result in this case.

5. **MLT – (much less than).** The operand in the AR is much less than the operand in the BR. In this case, right shifting the AR to align the exponents would zero out the quantity in the AR. This fact is detected by the FP11 hardware, thus avoiding the necessity of performing unnecessary shifting operations. The quantity in the BR is effectively the result. The exponent in the BA and the mantissa in the BR are loaded into the destination AC.

Consequently, in the last two cases (MGT, MLT) where one operand is much larger or smaller than the other operand, the addition is never performed, and the result is the result of the larger quantity. In the first three cases (EQ, GT, LT), the two operands are added or subtracted by the hardware after they are aligned.

$$\text{RANGE} = \text{EXP}_D - \text{EXP}_S$$

If positive, shift source
If positive and $>25_{10}$ (single precision) or 57_{10} (double precision), use destination as result
If negative, shift destination
If negative and $<25_{10}$ (single precision) or 57_{10} (double precision), use source as result



	MLT (Use source)	LT (Shift destination)	GT (Shift source)	MGT (Use destination)
F	Range $< -25_{10}$	$-25_{10} \leqslant$ Range < 0	$0 <$ Range $\leqslant 25_{10}$	Range $> 25_{10}$
D	Range $< -57_{10}$	$-57_{10} \leqslant$ Range < 0	$0 <$ Range $\leqslant 57_{10}$	Range $> 57_{10}$

Figure 5-1 Exponent Magnitudes

5.2.3 Testing For Normalization

After the required addition or subtraction operation has been performed, the result in the AR is tested to ensure that it can be normalized. If the number in the AR is negative, it indicates that the number cannot be 0. If the AR is positive, a possibility exists that it can be 0. Consequently, 1 is subtracted from the AR and if the result is negative (change of signs) the number in the AR is known to be 0 and cannot be normalized. If there is no sign change in the subtraction the AR contains a positive nonzero number, which can be normalized.

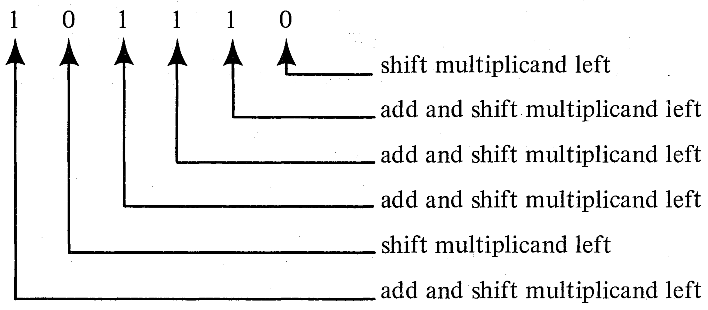
After normalization, the result is rounded or truncated depending on the setting of the FT bit in the program status register. The floating condition codes are also set.

5.3 FLOATING-POINT MULTIPLICATION

The FP11 Floating-Point Processor employs a rather complex method of shifting over 1s and 0s to perform multiplication. In order to familiarize the reader with this method, several concepts of this technique are first described followed by a description of the hardware employed in the FP11.

5.3.1 Fundamental Concepts

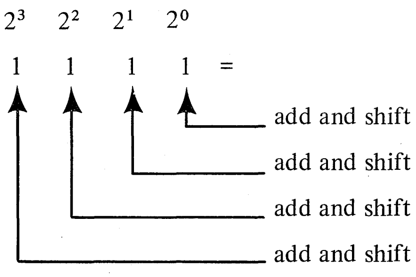
One simple method used in multiplication is to examine the multiplier on a bit-by-bit basis. If the bit is a 0, the multiplicand is shifted left one place. If the bit is a 1, the multiplicand is added to the partial product and is then shifted left one place.


1 0 1 1 1 0
shift multiplicand left
add and shift multiplicand left
add and shift multiplicand left
add and shift multiplicand left
shift multiplicand left
add and shift multiplicand left

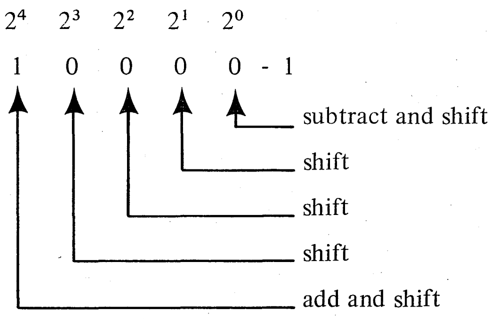
The same result can be obtained by shifting the partial product and the multiplier right one place as opposed to shifting the multiplicand left one place.

The method just described becomes rather time consuming because each 1 in the multiplier requires an addition. A method is desired where addition can be replaced with shifts inasmuch as shifting consumes less time. An improvement over this method is a process of shifting over 1s and 0s.

In order to implement shifting over 1s and 0s, the binary configuration of a number is represented in a different manner. For example, the binary number 1111 can be represented as 10000-1. Both expressions are equivalent and are equal to 15_{10}. Note that the second representation of the number contains only two 1s, requiring only two arithmetic operations whereas the first representation of the number contains four 1s for a total of four addition operations. The operations for each representation are performed as shown on the following page.



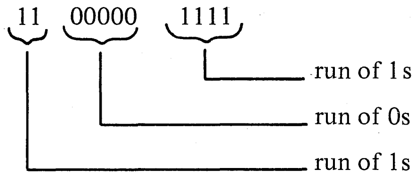
Old Method



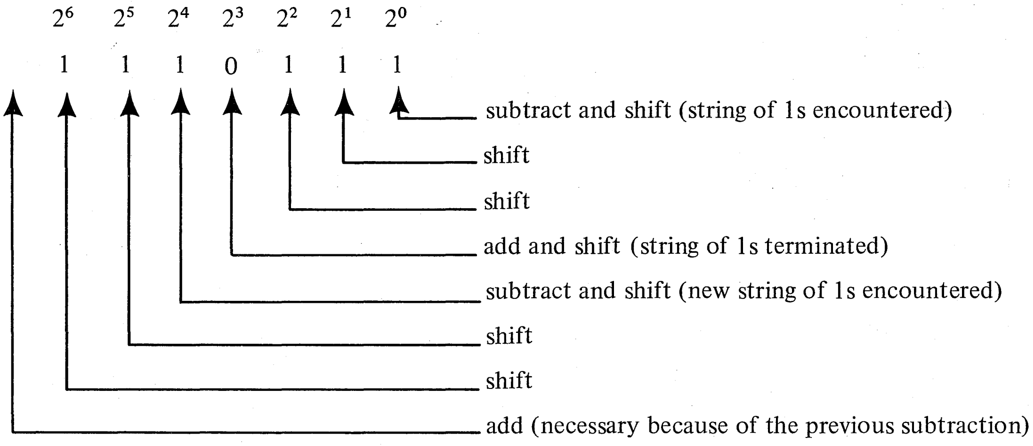
Shifting Over 1s And 0s

Note that a subtraction occurs in the bit position corresponding to the least significant 1 in the string, and an addition occurs 1-bit position beyond the most significant bit position in the string. This method proves most advantageous where long strings of 1s occur. Worst case occurs for alternating 1s and 0s.

An additional improvement over this method is developed where an isolated 1 occurs in a string of 0s or an isolated 0 occurs in a string of 1s. In this method, the multiplier is examined two bits at a time to look for runs of 1s or 0s. A run is defined as a string of two or more consecutive identical bits as shown below.

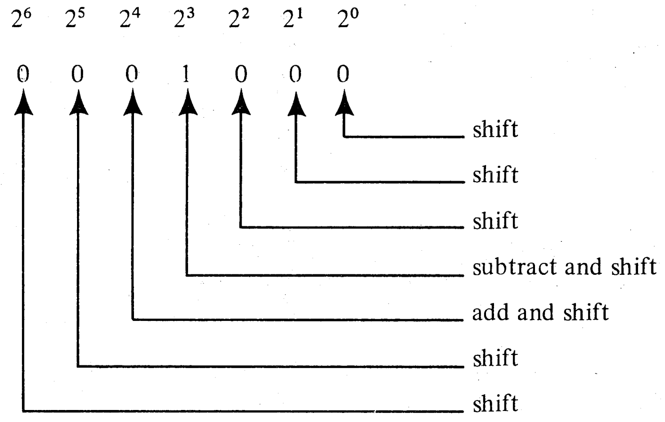


To see how this improved technique is implemented, consider the example of an isolated 0 in a string of 1s as shown in the following example:


2^6 2^5 2^4 2^3 2^2 2^1 2^0
1 1 1 0 1 1 1
subtract and shift (string of 1s encountered)
shift
shift
add and shift (string of 1s terminated)
subtract and shift (new string of 1s encountered)
shift
shift
add (necessary because of the previous subtraction)

Note in this example that in the 0 bit position an add is performed followed by a subtraction in the next bit position. This situation can be reduced to one arithmetic operation by performing the subtraction where the isolated 0 is located. Consequently, adding the 2^3 bit position (8_{10}) and subtracting the 2^4 bit position (16_{10}) is the same as merely subtracting the 2^3 bit position (8_{10}) both methods yielding –8. Another important point is that the last bits encountered in the multiplier are a run of 1s. Since a subtraction is first performed, when the run is encountered, it is necessary to conclude the operation with an addition occurring one bit beyond the most significant bit position.

Now, consider the case where an isolated 1 occurs in a string of 2s.



At first glance, this seems more cumbersome than the simple method first described. However, this can be reduced to one arithmetic operation (an addition) occurring where the 1 bit is encountered. Thus, instead of subtracting the 2^3 bit (-8_{10}) and adding the 2^4 bit ($+16_{10}$), the same result is obtained by adding the 2^3 bit ($+8_{10}$).

5.3.2 Multiply Hardware

With these principles in mind, the following paragraphs describe the implementation of the shifting over 1s and 0s method. The multiplicand is loaded in the BR register via the QR register, and the multiplier is then loaded in the QR register. The AR register is initially cleared and retains the partial products as they are accumulated. The hardware contains a step counter that keeps track of the number of shifts. This counter is preset with the 1's complement of the number of bits in the multiplier and is incremented after each shift or after each arithmetic operation followed by a shift. The counter is checked during each step and the multiplication is complete when the step counter goes to all 1s. Bits QR59 through QR3 in the QR are loaded. The extension bits, QR2 through QR0, are cleared. These bits are an extension of the QR register and are used for rounding operations. The testing of the bit pattern of the multiplier is done in a high-speed 2-bit register (MR1 and MR0), which has a copy of the appropriate bits of the QR. MR0 is always initialized to 0. MR1 is initialized with the contents of QR3 if double-precision floating point is specified or is initialized to the contents of QR35 if single-precision floating point is specified. During the multiply operation, MR1 is shifted into MR0 and QR4 (double-precision) or QR36 (single-precision) is shifted into MR1. Note that the initialization requires an extra shift at the start of the multiply operation. The floating-point hardware also contains a STRG1 (string of 1s) flip-flop, which is set by two consecutive 1s and reset by two consecutive 0s. The flip-flop is initially reset. Figure 5-2 shows a flow diagram with three variables: MR1, MR0, and STRG1. If MR1 and MR0 are both 0s and the STRG1 flip-flop set, the multiplicand is added to the partial product. If MR1 is a 0, MR0 is a 1, and STRG1 is a 0, the multiplicand is also added to the partial product. Note that the QR (containing the multiplier) and the AR (containing the partial product) are right shifted and the BR (containing the multiplicand) is not shifted.

Figure 5-3 shows a state diagram based on the state of MR1, MR0, and the STRG1 flip-flop. For example, if all three are in the 0 state, the next shift could cause all three to remain in the 0 state or a 1 could be shifted into MR1. These are the only possible states that can be entered when all three variables are initially 0. Listed below

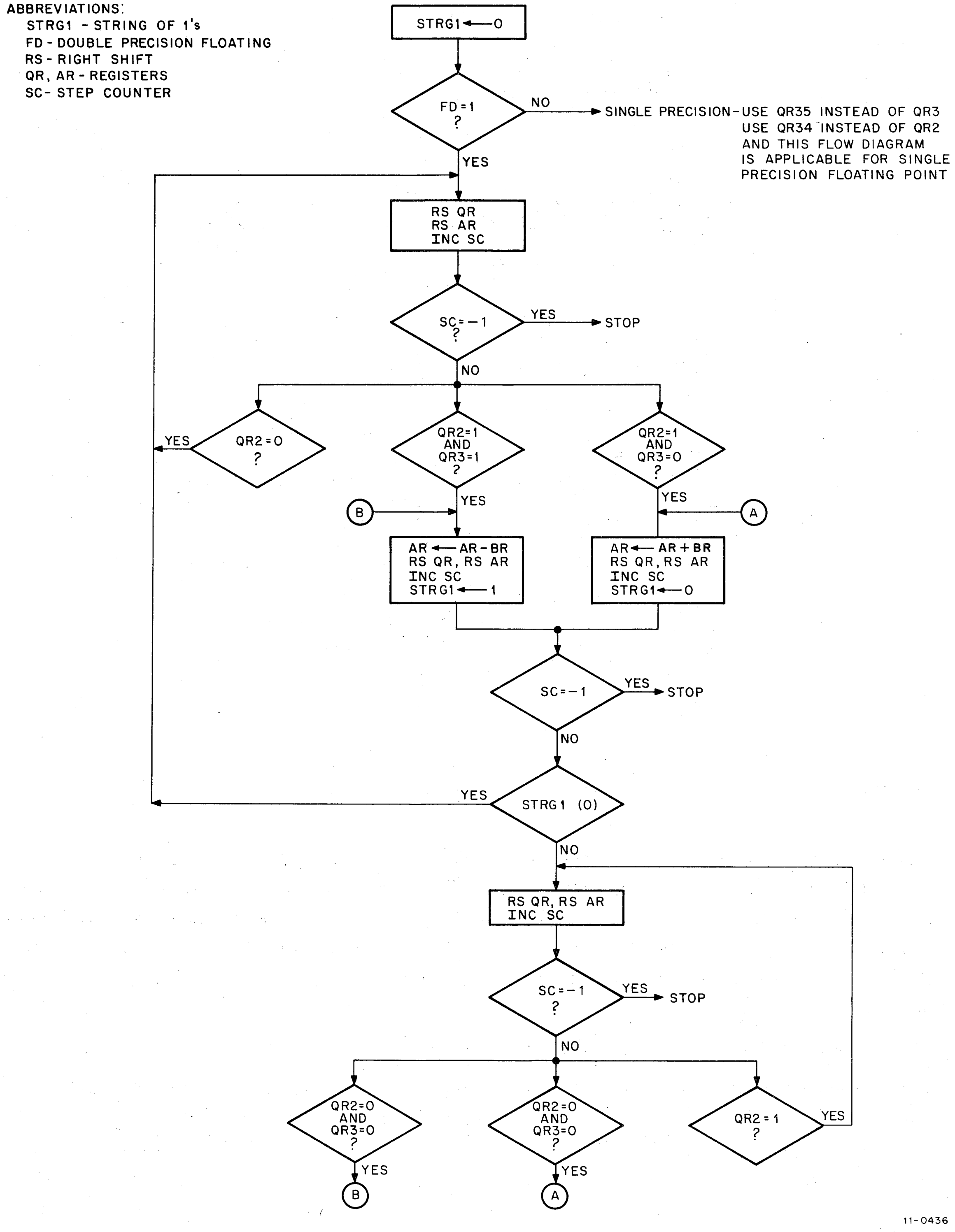
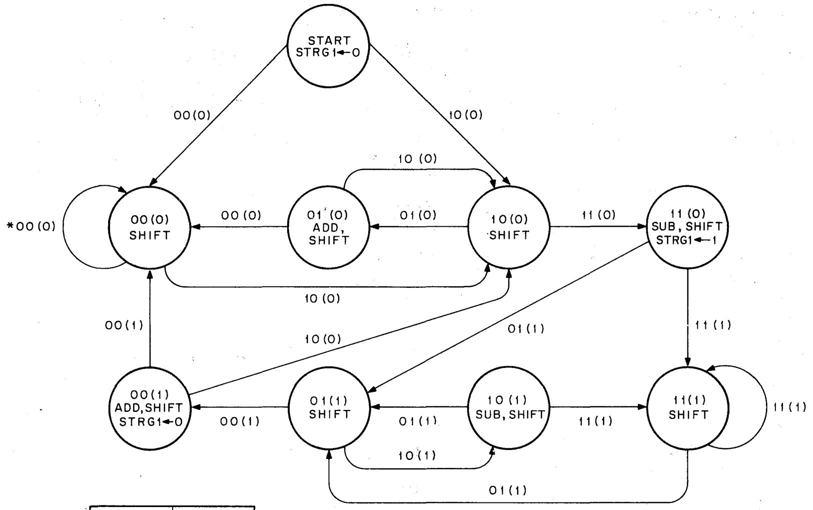


Figure 5-2 Multiply Flow Diagram



MR1	MRØ		
QR3(DBL)	QR2 (DBL)	STRG1	FUNCTION
QR35(SNG)	QR34 (SNG)		
0	0	0	RIGHT SHIFT QR, AR, INCREMENT SC**
0	1	0	AR←BR+AR, RIGHT SHIFT QR, AR, INCREMENT SC
1	0	0	RIGHT SHIFT QR, AR, INCREMENT SC
1	1	0	AR←AR-BR, RIGHT SHIFT QR, AR, SET STRG1, INCREMENT SC
0	0	1	AR←AR+BR, RIGHT SHIFT QR, AR, RESET STRG1, INCREMENT SC
0	1	1	RIGHT SHIFT QR, AR, INCREMENT SC
1	0	1	AR←AR-BR, RIGHT SHIFT QR, AR, INCREMENT SC
1	1	1	RIGHT SHIFT QR, AR INCREMENT SC

*For double precision format 00(0) = QR3, QR2, (STNG1)
For single precision format 00(0) = QR35, QR34, (STNG1)

**The step counter is set to the two's complement of the number of bits in the multiplier and is checked for zero after each incrementation.

11-0437

Figure 5-3 Multiply State Diagram

the state diagram is a table describing the functions performed as a result of the various bit configurations. For example, if MR1 is a 1, MR0 is a 1, and STRG1 is reset, the multiplicand is subtracted from the partial product, the step counter is incremented, the AR and QR registers are right shifted one place, and the STRG1 flip-flop is set. This table is very helpful in working through a typical multiplication example in order to determine the next sequence of events. Figure 5-4 provides some typical examples using 6-bit numbers for simplicity. The following points should be carefully observed in studying the examples.

1. Subtraction is performed using 2's complement arithmetic.
2. If the previous arithmetic operation was a subtraction, a 1 is shifted into the most significant bit of the AR when the AR is right shifted. Conversely, if the previous arithmetic operation was an addition, a 0 is shifted into the most significant bit of the AR when the AR is right shifted.

Example: $0.75_{10} \times 0.5_{10} = 0.375_{10}$

Step	AR						QR					QR 3	QR 2	STNG 1	BR						Functions Performed
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	QR ← MULTIPLIER, BR ← MULTIPLICAND, SC ← -7, STRG1 ← 0
1							0	0	1	0	0	0	0	0							RS QR, RS AR, INC. SC TO -6
2							0	0	0	1	0	0	0	0							RS QR, RS AR, INC. SC TO -5
3							0	0	0	0	1	0	0	0							RS QR, RS AR, INC. SC TO -4
4							0	0	0	0	0	1	0	0							RS QR, RS AR, INC. SC TO -3
5							0	0	0	0	0	0	1	0							RS QR, RS AR, INC. SC TO -2
6	0	0	1	1	0	0															AR ← AR + BR, RS QR, RS AR, INC. SC TO - 1 END MULTIPLY

Answer = $0.01100 = 0.25_{10} + 0.125_{10} = 0.375_{10}$

Example: $0.75_{10} \times 0.7187_{10} = 0.53906_{10}$

Step	AR						QR					QR 3	QR 2	STNG 1	BR						Functions Performed
0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	1	0	0	0	QR ← MULTIPLIER, BR ← MULTIPLICAND, SC = -7, STRG1 ← 0
1							0	0	1	0	1	1	1	0							RS QR, RS AR, INC SC TO -6
2	1	1	0	1	0	0	0	0	0	1	0	1	1	1							AR ← AR – BR, RS QR, RS AR, SET STRG1, INC SC TO -5
3	1	1	1	0	1	0	0	0	0	0	1	0	1	1							RS QR, RS AR, INC SC TO -4
4	1	1	1	1	0	1	0	0	0	0	0	1	0	1							RS QR, RS AR, INC SC TO -3
5	1	1	0	0	1	0	0	0	0	0	0	0	1	1							AR ← AR – BR, RS QR, RS AR, INC SC TO -2
6	1	1	1	0	0	1	0	0	0	0	0	0	0	1							RS QR, RS AR, INC SC TO -1
	0	1	0	0	0	1															AR ← AR + BR, NO FINAL SHIFT, END MULTIPLY

Answer = $0.10001_2 = .5_{10} + .03125_{10} = 0.53125$

Note: With six bits, of significance, the answer 0.53125 is the closest possible answer to the true result of 0.53906.

NOTE

1. **By investigating state of QR3, QR2, and STRG1, the next function performed can be determined.**
2. **When the AR is right shifted, the MSB retains the same bit polarity it had before the shift occurred.**

QR3	QR2	STNG1	Function
0	0	0	RS QR, RS AR, INC SC
0	1	0	AR ← BR + AR, RS QR, RS AR, INC SC
1	0	0	RS QR, RS AR, INC SC
1	1	0	AR ← AR – BR, RS QR, RS AR, INC SC, SET STRG1
0	0	1	AR ← AR + BR, RS QR, RS AR, INC SC, RESET STRG1
0	1	1	RS QR, RS AR, INC SC
1	0	1	AR ← AR – BR, RS QR, RS AR, INC SC
1	1	1	RS QR, RS AR, INC SC

Figure 5-4 Examples of Floating-Point Multiplication

3. Initially, MR0 and STRG1 are 0s and, thus, an extra shift will occur at the beginning of a multiply operation regardless of the state of MR1 (see table in Figure 5-2).

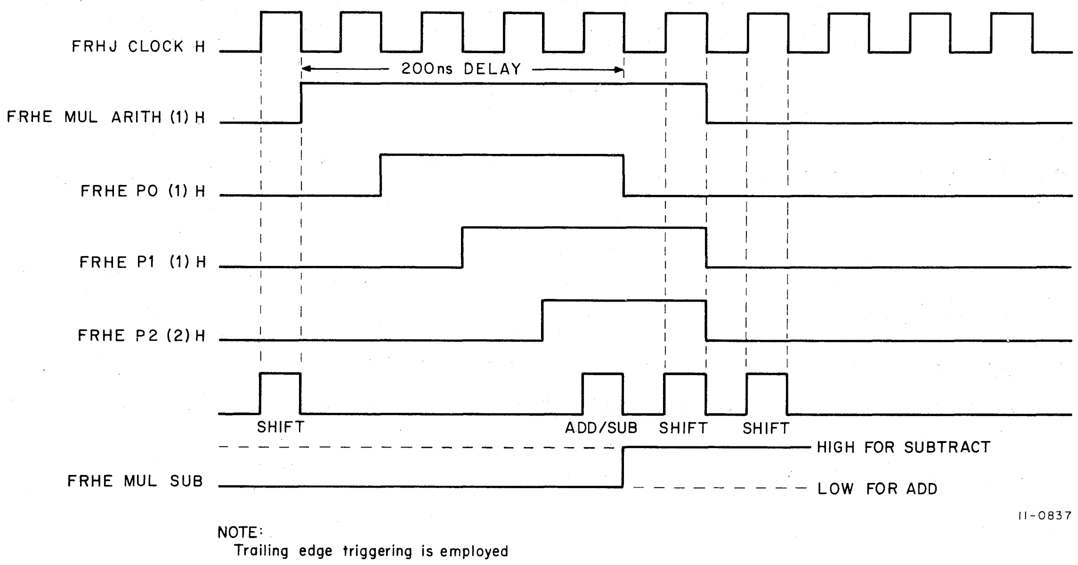
4. Arithmetic operations are performed only after state 10 (0) or 01 (1), where the leftmost bit represents MR1, the middle bit represents MR0, and the bit in parenthesis represents STRG 1.

5. A string of 1s occurring immediately to the right of the binary point requires a final addition one place beyond the binary point. This addition is not followed by a shift.

6. Although not shown in the example, the sign of the operand stored in the accumulator is stored in SD (sign of destination) and the sign of the other operand is stored in SS (sign of source). Upon conclusion of the multiplication, the signs are exclusively ORed – if they are the same, the sign of the product is positive – if they are different, the sign of the product is negative. The resultant sign is left in SD.

5.3.3 Multiply Timing

The timing for the multiplication operation is shown in Figure 5-5. The basic clock rate is 50 ns, which is the rate at which shifting occurs. Note that events occur at the trailing edge of the clock pulses. When the actual arithmetic operation (addition or subtraction) takes place, a 200 ns delay (4 clock pulses) is incurred. This is accomplished by setting the MUL ARITH flip-flop. This flip-flop is set for add or subtract operations during multiplication and, when set, inhibits shifting until the product is loaded in the AR. During normal shifting operations, the MUL ARITH flip-flop is in the reset state.

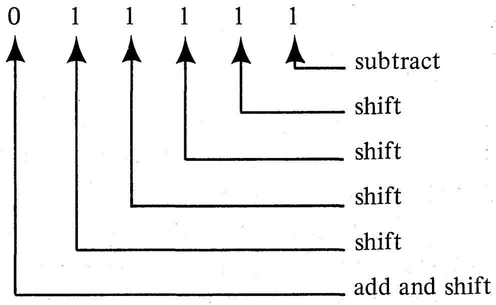


NOTE:
Trailing edge triggering is employed

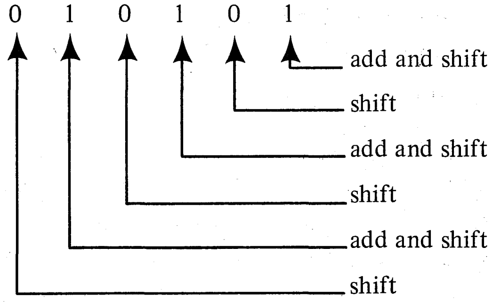
Figure 5-5 Multiply and Divide Timing Diagram

Refer again to the state diagram for multiplication shown in Figure 5-3; the two states that precede a state involving an arithmetic operation are 10 [0] and 01 [1]. Detection of either of these states causes the MUL ARITH flip-flop to set with the next clock pulse. The MUL ARITH flip-flop, in turn, enables the pause logic consisting of flip-flops P0, P1, and P2. The three flip-flops (P0, P1, P2) produce a 200 ns delay to allow time for completion of the arithmetic operation. P0 is set on the next clock pulse occurring after MUL ARITH is set, P1 is set on the next clock pulse occurring after P0 is set, and P2 is set on the next clock pulse occurring after P1 is set. The setting of P2 enables P0 to be reset on the next clock pulse, similar to a ring-tail counter. The resetting of P0, in turn, causes P1 and P2 to get reset. P0, when set, switches the AR control lines from shift to load and causes the AR to be loaded rather than shifted; P2, when high, is used to enable the AR clock pulses and then goes low to disable the pause logic and consequently enable the shift pulses.

Normally, in the multiply algorithm, the last step encountered in a string of 1s is an add and shift or merely a shift if a string of 1s has not been encountered (see the following examples).

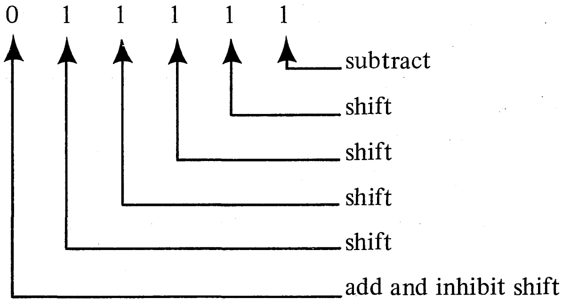


String of 1s

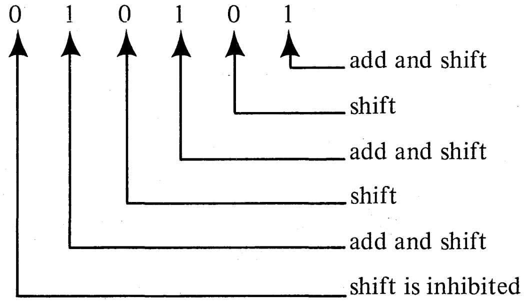


No String of 1s

If the string of 1s should occur in the most significant bit positions, it is necessary to inhibit the shift following the add operation (see examples below). The shift will occur if the string of 1s is not present.



String of 1s



No String of 1s

The hardware implements this by setting the MUL SUB flip-flop when a subtraction in a string of 1s occurs. The flip-flop is reset by an add operation and, therefore, this flip-flop remains set until the add and shift operation, which terminates a string of 1s.

The step counter is preset to the 1's complement of the number of shifts that are required. For each shift that occurs, the step counter is incremented. Multiplication is terminated when the step counter sequences to all 1s (77_8). With the MUL SUB flip-flop set (indicating that the last arithmetic operation was subtract and that a string of 1s was encountered), shifting occurs and the step counter is incremented for each shift. If the step counter sequences to 77_8 before the add operation occurs, the shift following the add is inhibited. If the add operation occurs before the step counter sequences to 77_8, the shift following the add is allowed to occur.

5.4 DIVISION

Digital computers have various methods available for performing division. Several that are briefly described in the following paragraphs are: restoring division, non-restoring division, and non-restoring division utilizing the normalizing principle. This latter method is the most efficient and is the one employed in the FP11 Floating-Point Unit.

5.4.1 Restoring Division

When dealing with positive numbers in restoring division, the divisor is first subtracted from the dividend, yielding a remainder. If the subtraction is successful (indicating that the dividend is larger than the divisor), a 1 is entered into the quotient. If the subtraction is unsuccessful, a 0 is entered in the quotient and the remainder is restored back to its original value; this is done by adding the divisor to the remainder. The disadvantage is that two arithmetic operations (a subtraction and an addition) are required when the subtraction is unsuccessful. In the next cycle, the remainder is left shifted one place (which is equivalent to multiplying by 2), the divisor is subtracted from the remainder, and the result is examined. If the subtraction is successful, a 1 is entered in the quotient; if not, a 0 is entered and the remainder is restored. This process continues until an appropriate number of quotient bits have been determined. The sign of the dividend and divisor can be handled separately. If they are both of the same sign, a positive quotient results; if different, a negative quotient results.

5.4.2 Non-Restoring Division

The chief advantage of non-restoring division over restoring division is that the remainder need not be restored in the same cycle if the subtraction result is unsuccessful. The steps in restoring divide for an unsuccessful subtraction are:

R = remainder
D = divisor

1. R-D /subtract
2. R-D+D /restore remainder
3. (R-D+D) x 2 /left shift new remainder
4. (R-D+D) x 2-D /subtract divisor
(R-D+D) x 2-D = 2R-D

The steps performed in non-restoring divide for an unsuccessful subtraction are:

1. R-D /subtract
2. (R-D) x 2 /left shift new remainder
3. (R-D) x 2 + D /add divisor
(R-D) x 2 + D = 2R - 2D + D = 2R - D

Note that the results in either case are the same (2R-D), but that the restoring divide required an additional arithmetic operation. An example of non-restoring division is shown in Figure 5-6.

Example: $0.11000_2 \div 0.10001_2 = 1.01101_2$

$1.01101_2 = 0.10110 \times 2^1 = 0.6875_{10} \times 2 = 1.375_{10}$

$0.75_{10} \div 0.53125_{10} = 1.4$

Step	Function	Register 3 (R3)	Register 2 (R2)	Register 1 (R1)
1	R2 - R1 POS.	0 0 0 0 0 0	0 1 1 0 0 0	0 1 0 0 0 1
	R3 ← 1	0 0 0 0 0 1	0 0 0 1 1 1	
	LS R2	0 0 0 0 0 1	0 0 1 1 1 0	
2	R2 - R1 NEG.		1 1 1 1 0 1	
	R3 ← 0	0 0 0 0 1 0		
	LS R2		1 1 1 0 1 0	
3	R2 + R1 POS.		0 0 1 0 1 1	
	R3 ← 1	0 0 0 1 0 1		
	LS R2		0 1 0 1 1 0	
4	R2 - R1 POS.		0 0 0 1 0 1	
	R3 ← 1	0 0 1 0 1 1		
	LS R2		0 0 1 0 1 0	
5	R2 - R1 NEG.		1 1 1 0 0 1	
	R3 ← 0	0 1 0 1 1 0		
	LS R2		1 1 0 0 1 0	
6	R2 + R1 POS.		0 0 0 0 1 1	
	R3 ← 1	1 0 1 1 0 1		
	LS R2		0 0 0 1 1 0	

NOTE

Because the dividend is larger than the divisor, the quotient must be greater than 1. The quotient, in this case, is not in true normalized form; thus, it must be right shifted one place, and the associated exponent must be incremented.

Figure 5-6 Example of Non-Restoring Division

5.4.3 Non-Restoring Divide Using Normalizing

Non-restoring divide using the normalizing principle provides a further improvement over non-restoring divide. When a trial subtraction is performed in this case, the result is examined to determine if it is normalized. If not, the remainder and quotient are left shifted until the number is normalized. For each left shift, an arithmetic operation is eliminated. When the number becomes normalized, the divisor is subtracted from or added to the new remainder and the result is again examined. If unnormalized, the remainder and quotient are left shifted until the remainder is normalized. If normalized, a new subtraction or addition is performed. When dealing with positive numbers, the divisor (in normalized form) is subtracted from the remainder, which, if unnormalized, is by definition smaller than the divisor, as shown in the following example:

Remainder	0.01111
Divisor	0.10001 (normalized)

As a result, leading 0s can be shifted over when dealing with positive numbers. Conversely, with negative numbers, leading 1s can be shifted over.

Initially, the dividend and divisor are assumed positive. The normalized dividend is loaded in the AR and the normalized divisor is loaded in the BR. The QR is initially cleared and is used to accumulate the partial bits of the quotient as they are calculated.

In order to understand the divide algorithm, it is necessary to refer to the flow chart shown in Figure 5-7. Because the AR is initially positive, the BR is subtracted from the AR, the difference being placed in the AR. Both the AR and QR are left shifted and the complement of the most significant bit of the AR is shifted into the least significant bit of the QR. The number in the AR is now examined to determine if it is normalized. If it is not, the number is normalized by a routine, which will subsequently be described. If the number is normalized, it must be determined if it is a positive or negative number. If it is positive, the BR is subtracted from the AR, the AR and QR are left shifted with QR_{LSB} receiving the complement of AR_{MSB}. If negative, the BR is added to the AR with QR_{LSB} receiving the complement of AR_{MSB}. The AR is again examined to determine if it is normalized. If it is normalized, another addition (if the number is negative) or subtraction (if the number is positive) is performed, the QR and AR are left shifted, and the complement of AR_{MSB} is shifted into QR_{LSB}. If not, the number is normalized as described below.

In order for a number to be normalized, it must be in the form of 0.1 xxxx (positive number) or 1.0 xxxx (negative number) with x designating a *don't care.* The number 0.00011, for example, can be normalized by three left shifts (shifting over 0s), yielding 0.11000. The three 0s to the right of the binary point have positional significance but have no numerical value. The quotient is left shifted three places and 0s are shifted into the least significant bit positions. The number 1.11100 can be normalized by three left shifts (shifting over 1s), yielding 1.00000. This is a negative number and in 2's complement form; thus, the 1s being shifted over are in reality 0s and have no numerical value. In this case, the quotient is left shifted three places and 1s are shifted into the least significant bit positions. A step counter is preset to the 1's complement of the number of bits in the multiplier and is incremented for each shift. When the counter is incremented to all 1s, the division is terminated.

Figure 5-8 shows the state diagram for floating-point division. This is interpreted in the same manner as the diagram for floating-point multiplication. Note that the number in the AR is assumed to be normalized and of the form 0.10 or 0.11 (the initial states). The number in the BR at this time is also assumed positive and normalized.

Several examples of the normalizing principle are shown in Figure 5-9. The first example has a dividend larger than the divisor, and the second example has a divisor larger than the dividend.

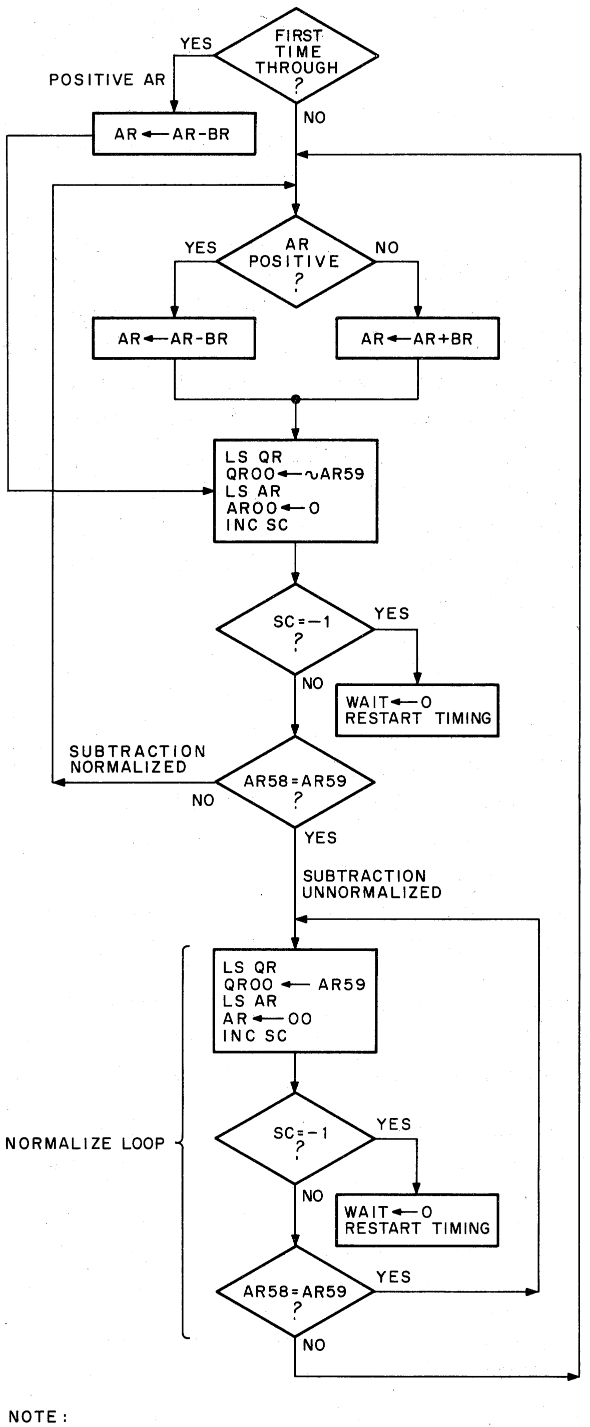
It should be noted in all cases where the dividend is larger than the divisor, the quotient will be of the form 1.xxxx, having a significance greater than 1. This number is not in true normalized form; consequently it must be shifted to the right one place yielding 0.1xxx. This reduces the number by a power of 2, and in order to maintain the same equivalence, the exponent associated with the number must be incremented, which increases the number by a power of 2. The floating-point divide algorithm can best be described by stating the rules associated with the algorithm. These are summarized below.

RULES FOR FLOATING-POINT DIVIDE

1. For the first time, the AR, containing the dividend, is positive. Consequently, subtract the BR from the AR and place the result in the AR. Left shift the QR and the AR with the complement of AR_{MSB} being shifted into QR_{LSB}. Examine the AR; if it is normalized proceed to Step 2; if not, proceed to Step 3.

2. If the AR is positive, subtract the BR from the AR; if it is negative, add the BR to the AR. In either case, left shift the QR and AR shifting the complement of AR_{MSB} into QR_{LSB}. If the number is normalized, repeat Step 2; if not, go to Step 3.

(continued on page 5-17)

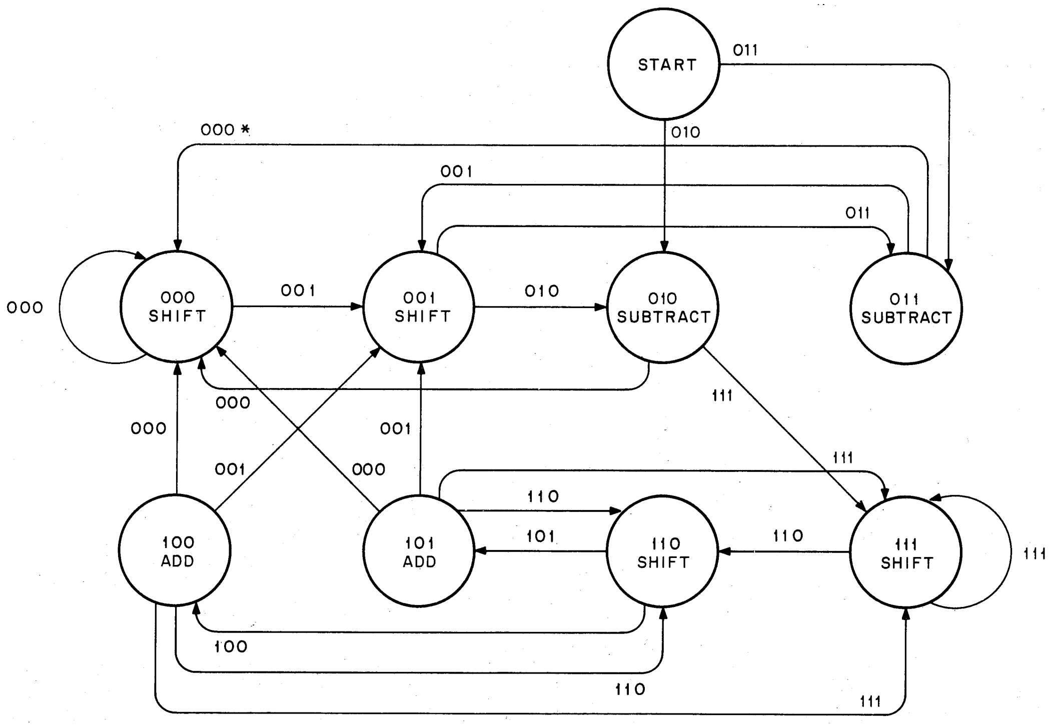


NOTE:

DIVIDEND IS MADE POSITIVE AND LOADED IN AR
DIVISOR IS MADE POSITIVE AND LOADED IN BR
BOTH NUMBERS ARE NORMALIZED PRIOR TO DIVIDE

11-0444

Figure 5-7 Divide Flow Diagram



*Three digits shown throughout state diagram refer to bits AR59, 58, and 57.
For example,

0 1 0
AR57 = RR0
AR58 = RR1
AR59 = RR2

NOTE
BR is always positive and normalized.

11-0443

Figure 5-8 State Diagram for Divide

3; Left shift the QR and AR shifting AR_{MSB} into QR_{LSB}. Note that during normalize, QR_{LSB} receives AR_{MSB} and not the complement of AR_{MSB} as is done immediately after an arithmetic operation.

Again examine the AR. If normalized, return to Step 2; if not, repeat Step 3 until the AR is normalized, then return to Step 2.

NOTE

Since the divisor shown in the example is six bits, a total of six shifts will occur before the divide is terminated.

In floating-point division, the sign of the dividend is stored in the sign of the destination (SD), and the sign of the divisor is stored in the sign of the source (SS). The sign of the quotient is determined by an exclusive OR of the two. In other words, if the signs are the same, the exclusive OR is 0, yielding a positive sign; if the signs are different, the exclusive OR is 1, yielding a negative sign.

5.4.4 Divide Timing

The timing for floating-point divide is the same as that employed for floating-point multiplication with the three pause flip-flops providing a 200 ns delay during arithmetic operations. During these operations, the AR NORM flip-flop is set, indicating a normalized result in the AR. If AR NORM is reset, the AR is shifted left until the contents become normalized.

Example: $0.11000_2 \div 0.10100_2 = 0.10011_2 \times 2_2$ (Dividend Larger Than Divisor)

$0.75_{10} \div 0.625_{10} = 1.2_{10}$ **NOTE: In binary, quotient is 1.1874 which is as close to 1.2 as possible using six bits.**

Step	**QR**						**AR**						**BR**						**Functions Performed**
	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	0	Set Step counter to – 7 AR is positive first time through
1	0	0	0	0	0	1	0	0	1	0	0	0							AR ← AR - BR, LS AR, LS QR, QR_{LSB} ← ~ AR_{MSB}, INC SC TO – 6 AR is now unnormalized
2	0	0	0	0	1	0	0	1	0	0	0	0							LS AR, SL QR, QR_{LSB} ← AR_{MSB}, INC SC TO -5 AR is now normalized and positive
3	0	0	0	1	0	0	1	1	1	0	0	0							AR ← AR - BR, LS AR, LS QR, QR_{LSB} ← ~ AR_{MSB}, INC SC TO -4 AR now unnormalized
4	0	0	1	0	0	1	1	1	0	0	0	0							LS QR, LS AR, QR_{LSB} ← AR_{MSB}, INC SC TO -3 AR still unnormalized
5	0	1	0	0	1	1	1	0	0	0	0	0							LS QR, LS AR, QR_{LSB}← AR_{MSB}, INC SC TO -2 AR now normalized and negative
6	1	0	0	1	1	0	1	1	0	1	0	0							AR ← AR + BR, LS QR, LS AR, QR_{LSB} ← ~ AR_{MSB},INC SC TO -1 Sign of QR is negative, RS QR and increment exponent
	0.	1	0	0	1	1													Divide complete – Quotient in QR

Example: $0.10000_2 \div 0.10100_2 = 0.11001_2$ (Divisor Larger Than Dividend)

$0.5_{10} \div 0.625_{10} = 0.8_{10}$ **NOTE: In binary, quotient is 0.78125 which is as close to 0.8 as possible using six bits.**

Step	**QR**						**AR**						**BR**						**Functions Performed**
	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	Set Step counter to –7 AR is positive first time through
1							1	1	1	0	0	0							AR ← AR - BR, LS AR, LS QR, QR_{LSB} ← ~ AR_{MSB}, INC SC TO -6
2	0	0	0	0	0	1	1	1	0	0	0	0							LS AR, LS QR, QR_{LSB} ← AR_{MSB}, INC SC TO -5 AR still unnormalized
3	0	0	0	0	1	1	1	0	0	0	0	0							LS AR, LS QR, QR_{LSB} ← AR_{MSB}, INC SC TO -4 AR is now normalized and negative
4	0	0	0	1	1	0	1	0	1	0	0	0							AR ← AR + BR, LS AR, LS QR, QR_{LSB} ← ~ AR_{MSB}, INC SC TO -3 AR is unnormalized and negative
5	0	0	1	1	0	0	1	1	1	0	0	0							AR ← AR + BR, LS AR, LS QR, QR_{LSB} ← ~ AR_{MSB}, INC SC TO -2 AR is now unnormalized and negative
6	0	1	1	0	0	1	1	1	0	0	0	0							LS AR, LS QR, QR_{LSB} ← ~ AR_{MSB}, INC SC TO -1 Divide complete – quotient in QR

Figure 5-9 Examples of Floating Point Division

CHAPTER 6
FP11-B LOGIC DIAGRAM DESCRIPTIONS

6.1 INTRODUCTION

This chapter describes the logic diagrams associated with the FP11-B Floating-Point Processor. This chapter, in conjunction with the signal glossary in Appendix B, provides an adequate description of the FP11-B logic diagrams.

6.2 DETAILED LOGIC DIAGRAM DESCRIPTIONS

The FP11-B logic diagrams are divided into four groups of prints – each group corresponding to one of the four FP11-B hex modules. The prints are designated by a four-letter code and are classified in one of the following four groups, whereby the first three letters of the code are defined as follows:

FRL	Fraction Data Path Low Order	M8115-0-01
FRH	Fraction Data Path High Order	M8114-0-01
FRM	FP ROM and ROM Control	M8112-0-01
FXP	Floating-Point Exponent Data Path	M8113-0-01

NOTE
The fourth letter in each group designates the sheet number of the print within the group specified, i.e., FRHA, FXPB where A and B refer to the sheet numbers.

The FRL group of prints contains the following logic:

a. lower half of FALU
b. lower half of AR
c. lower half of BR
d. lower half of QR
e. floating-point status
f. ACMX
g. scratch pad (AC7–0)
h. BMX

The FRH group of prints contains the following logic:

a. upper half of FALU
b. upper half of AR
c. upper half of BR

(continued on next page)

d. upper half of QR

e. clock logic, times states, time pulses

f. sign of source (SS) and sign of destination (SD) logic

g. fractional control logic

The FRM group of prints contains the following logic:

a. Control ROM

b. Control ROM address register

c. Scratchpad addressing logic

d. ROM multiplexers

e. ROM data buffer

f. Interface logic

The FXP group of prints contains the following logic:

a. EALU

b. EMX

c. Step counter

d. FIR

e. BA register

f. BD register

g. U Break register

h. DIMX

6.2.1 FRHA

This sheet shows the upper half of the QR and BR. Bits 58 through 35 of the QR are shown and bits 59 through 36 of the BR are also shown (refer to descriptions of FRLL and FRLM).

6.2.2 FRHB, FRHC, FRHD

These three sheets show the upper half of the AR and the FALU. Bits 59 through 36 of the AR and FALU are shown (refer to descriptions of FRLE, FRLF, FRLH, FRLJ and FRLK).

6.2.3 FRHE

This print contains the following circuitry (which is described in subsequent paragraphs):

a. MR1 and MR0 register

b. MUL ARITH flip-flop

c. Pause logic

d. STRG 1 flip-flop

e. AR Control

f. QR Control

g. MUL SUB flip-flop

h. AR clock logic

(continued on next page)

i. QR clock logic

j. Sign bit

6.2.3.1 **Multiply** – When a multiply or divide operation is designated, the FP11-B enters a Pause state where auxiliary hardware controls the fractional logic data paths. The AR, BR, and QR are initialized prior to this, and the ALU is set up to look at the AR (A input to ALU). MR0 and STRG 1 are cleared and MR1 contains the same value that was loaded into QR03 (double precision) or QR35 (single precision). Note that the microprogram must issue a LDQ1 before a LDQ0 to initialize MR1 correctly.

As described in the multiply algorithm, the first operation in the multiply subroutine is a shift. QR04 (double precision) or QR36 (single precision) is shifted into QR03 or QR35, respectively, and is also shifted into MR1; the content of MR1 is shifted into MR0. If the bit pattern of MR1, MR0, and STRG 1 is such that an arithmetic operation is required, MUL ARITH will set on the next clock (see multiply timing) and enable the pause logic which, in turn, will inhibit the AR and QR clock. The pause logic allows time for the data on the ALU input lines to settle before the add or subtract operation is performed. The ALU control is now selected by the multiply/divide hardware to do an A plus B or A minus B, as a result of the ALU select signals. The result of the operation is then set up to be loaded in the AR on the next CLK AR signal. A load occurs as a result of P0 going to a 1, which overrides all other inputs to the AR select lines and causes ARS1 and ARS0 to go high, thus specifying a load of the AR. In order to actually load the AR, the trailing edge of the AR clock must occur. However, when MUL ARITH was set, the CLK AR and CLK QR pulses were disabled. This disable is removed by P2 going to a 1, which allows two clocks to occur. The first allows the result of the add or subtract to be clocked in the AR, and the second allows both the AR and QR to be shifted following this operation. At the end of the first clock, P0 goes low forcing AR1S1 low which, in conjunction with AR1S0 high, sets up the AR for a right shift that occurs on the trailing edge of the second clock. AR1S0 remains high because CSB O (0) L is true (see Multiply Timing).

6.2.3.2 **MR1 and MR0 Register** – MR1 and MR0 are 74S74 D-Type flip-flops used in the FP11-B to speed up multiply operations. They are used in conjunction with the STRG 1 flip-flop to determine strings of 1s and strings of 0s. Before multiplication, the multiplier is loaded in the QR. QR59 through 35 are loaded and then QR34 through 3 are loaded from the scratchpad accumulator. QR02, QR01, and QR00 are loaded and with 0s. These bits are used for rounding operations. Note that when SCR OUT 00 is loaded into QR35 it is also copied into MR1 but when QR 03 is loaded from SCR OUT 00, either MR1 is latched if FD is on a 0 or is loaded from SCR OUT 00 if FD is on a 1. When the QR is right shifted, the content of QR04 (for double precision) or QR36 (for single precision) is shifted into MR1, and the content of MR1 (which contained the contents of QR03 or QR35) is shifted into MR0. Consequently, the content of MR1 and MR0 contains two successive bits of the QR. These flip-flops are monitored along with the STRG 1 flip-flop to determine the bit pattern of the multiplier.

Note that during a multiply CSB bits 2, 1, and 0 are high (ROM bits 29 through 27), which disables the direct clear input to MR0. During other operations such as division, this register is held cleared.

6.2.3.3 **MUL ARITH** – The MUL ARITH flip-flop is used during multiply to indicate that an actual arithmetic operation is to take place. The operation will be an add or subtract, depending on the bit patterns in MR1, MR0, and STRG 1. The MUL ARITH flip-flop actually anticipates an arithmetic operation with the two patterns designated *before shift* in Table 6-1. The next clock pulse shifts a 1 or 0 into MR1. If it is a 1, a subtract operation is performed; if 0 is shifted into MR1, an add operation is performed.

Table 6-1
Arithmetic Anticipation

	MR1	MR0	STRG 1	MUL ARITH	
Before Shift	0	1	[1]	0	
After Shift if QR04 = 1*	1	0	[1]	1	Subtract
After Shift if QR04 = 0*	0	0	[1]	1	Add
Before Shift	1	0	[0]	0	
After Shift if QR04 = 1*	1	1	[0]	1	Subtract
After Shift if QR04 = 0*	0	1	[0]	1	Add

***QR36 (if single-precision format)**

For example, in the second entry in the table there is a string of 1s with an isolated 0. Rather than do an add to terminate the string of 1s followed by a subtract in the next higher bit position, a subtract is performed. Note that the direct clear input to MUL ARITH is disabled during multiply because CSB bits 2, 1, and 0 are held high. During other operations, such as division, the flip-flop is held reset.

6.2.3.4 **Pause Logic** – The pause logic is used in multiplication and division and provides a 200 ns delay to perform addition or subtraction operations within the multiply or divide subroutines. The logic utilizes three 74S74 D-Type flip-flops. For multiply or divide operations, CSB2 and CSB1 (ROM bits 29 through 27) are both 0s, which disables the direct clear input to the pause flip-flops. During all other operations, the direct clear is enabled and the pause flip-flops are held cleared.

For multiply operations, the pause logic is enabled due to MUL ARITH going to a 1; during divide operations, AR NORM (1) enables the pause logic.

6.2.3.5 **STRG 1 Flip-Flop** – The STRG 1 flip-flop is a 74S112 J-K edge-triggered flip-flop used to indicate whether a strings of 1s or strings of 0s are present. The flip-flop will toggle only under the following two sets of conditions:

a. If MR1 and MR0 are both 1s and the STRG 1 flip-flop is a 0, the next clock pulse will force STRG 1 to a 1, indicating a string of 1s has been found.

b. If MR1 and MR0 are both 0s and STRG 1 is a 1, the next clock pulse will force STRG 1 to a 0, indicating the start of a string of 0s.

6.2.3.6 **AR Control** – The AR is controlled from the AR control bits (ROM bits 26 and 25), from the call subroutine bits (ROM bits 29, 28, and 27), or from the multiply/divide logic (P0). The AR can do a right shift or left shift one place, as a result of the ARC bits as shown below:

ARC1	ARC0	
0	0	Load AR
0	1	Shift AR left
1	0	Shift AR right
1	1	NOP

When the FP11 enters a multiply or divide subroutine, the ARC bits must select NOP and the CSB signals take precedence and direct the AR as follows:

CSB2	CSB1	CSB0	Function
0	0	0	Multiply BR with QR result in AR.
0	0	1	Divide AR by BR – result in QR.
0	1	0	Shift AR right by number in SC. Shift in 0s.
0	1	1	Shift AR left until normalized and count number of shifts in SC. Shift in 0s.
1	0	0	Shift QR right by number in SC. Shift in 0s.
1	0	1	Shift QR left by number in SC. Shift in 0s.
1	1	0	Shift QR right by number in SC. Shift 1s (sign bit remains 0)
1	1	1	NOP

When an addition or subtraction is to be performed within the multiply or divide subroutine, the pause logic is enabled and P0 (1) overrides the CSB bits, causing both ARS1 and ARS0 to go high which specifies a load operation. ARS1 and ARS0 are the signals that direct the AR to perform one of the following functions:

ARS1	ARS0	Function
0	0	NOP
0	1	Shift right
1	0	Shift left
1	1	Load

6.2.3.7 QR Control – The QR is controlled from the QRC (bits 22 and 21 of the ROM) bits and the ACC (bits 37 through 35 of the ROM) bits. The QRC bits direct the QR to perform one of the following functions:

QRC1	QRC0	Function
0	0	Load QR01 if ACC2 (0). Otherwise, load QR00.
0	1	Shift QR left
1	0	Shift QR right
1	1	NOP

The ACC bits specify the appropriate 16-bit word of the 64-bit AC to be used. ACC2 (0) specifies quadrant [3:2] of the scratchpad; ACC2 (1) specifies quadrant [1:0].

Because the scratchpad is 32 bits wide, the QR is loaded in two halves – the upper half is controlled by QR1S1 and QR1S0, and the lower half is controlled by QR0S1 and QR0S0. Note that when the upper half of the QR is loaded, both QR1S1 and QR1S0 are enabled; when the lower half is loaded, both QR0S1 and QR0S0 are enabled. During shifting, the QR signals are controlled together. The QR1 and QR0 signals direct the QR to perform one of the following functions:

QR (Upper Half) Control

QR1S1	QR1S0	Function
0	0	NOP
0	1	Shift right
1	0	Shift left
1	1	Load

QR (Lower Half) Control

QR0S1	QR0S0	Function
0	0	NOP
0	1	Shift right
1	0	Shift left
1	1	Load

6.2.3.8 MUL SUB Flip-Flop – The MUL SUB flip-flop is set when a subtract operation is performed during a multiply subroutine and is reset when an add operation is to be performed. The MUL SUB flip-flop performs two functions:

a. It allows the multiply operation to be terminated with an addition and inhibits final shift if the multiply is terminated in a string of 1s.

b. It also determines what is shifted into AR59 during right shift operations.

In order to set MUL SUB, MUL ARITH must be set because this enables the pause logic. Also, MR1 must be a 1, which indicates subtract. If MR1 is a 0, MUL SUB is reset, which indicates an add (see Paragraph 6.2.3.3).

6.2.3.9 AR Clock – The AR is clocked at TS4 when the AR control bits (bits 26 and 25) of the ROM specify a load. The AR is also clocked when bit 29 (CSB2) from the ROM is a 0 and the FP11 is in the Wait state. This occurs for a multiply, divide, right or left shift of the AR. The AR clock logic can be disabled as a result of one of the following three conditions:

a. In a multiply, the pause logic is enabled due to MUL ARITH (1) when the required addition or subtraction is to be performed. In order to allow the data on the lines to the ALU time to settle, clock pulses are disabled due to P2 (0). When P2 goes to a 1, the clocking of the result of the add or subtract operation can be performed (see Figure 5-5).

b. During normalizing and dividing, the AR clock is inhibited by AR NORM (1) H. In the case of normalizing, P2 is held on a 0 and the setting of the AR NORM flip-flop signifies the end of the operation. In the case of dividing, AR NORM (1) H indicates another arithmetic operation is to be performed. Consequently, the pause logic must be enabled as in *a.* above.

c. If the step counter increments all 1s and MUL SUB is reset, the AR clock is inhibited. If MUL SUB is set, one AR clock is allowed to load the result of a final addition. This load also resets MUL SUB disabling further AR clocks.

6.2.3.10 QR Clock – The QR is clocked at TS3 if QRC bits from the ROM are both 0s, which specifies a load operation. The QR can also be clocked if CSB1 is on a (0) and if the FP11 is in the Wait state and if P0 is a 0. CSB1 (0) corresponds to CSB fields 0, 1, 4, and 5 in the ROM (bits 29 through 27), which specify the following:

Field	Description
0	Multiply QR with BR – result in AR.
1	Divide AR by BR – result in QR.
4	Shift QR right by number in SC. Shift in 0s.
5	Shift QR left by number in SC. Shift in 0s.

P0 is set in the multiply/divide routine when an add/subtract operation is being performed. This ensures that the QR is clocked only during shift operations. Finally, the QR can be clocked by the CSB bits corresponding to a field of 6 [CSB2 (1), CSB1 (1), and CSB0 (0)], which specifies a right shift of the QR by the number in the SC and specifies that 1s are to be shifted in (the sign bit remaining 0).

The same logic that causes the AR clock to be disabled also causes the QR clock to be disabled (see AR clock).

6.2.4 FRHF

This sheet contains the sign bit logic, RSQR in logic, RR bits (RR2, RR1, and RR0) associated with the AR register, the AR NORM flip-flop, the LSQR IN logic, and the SS (sign of source) and SD (sign of destination) logic.

6.2.4.1 Sign Bit – QR59 is the sign bit of the QR and is clocked whenever the QR is clocked. QR59 is loaded with either a 0 or with the contents of QR58. A 0 is loaded into QR59 whenever:

a. CSB0 from the ROM is 0. The corresponding CSB field combinations are 0, 2, 4, and 6. Field 2 can be disregarded because it deals with the AR. For all other combinations (multiply, shift QR right with 0s in, and shift QR right with 1s in), QR59 is loaded with a 0.

b. QR59 is loaded with 0 when ENAB QRS0 is true. This is true when the ROM control is being used to shift the QR right (QRC1 on a 1 and QRC0 on a 0).

c. QR59 is loaded with 0 when LOAD QR is true. This signal is true when the ROM control is used to load the QR (QRC1 on a 0 and QRC0 on a 0). In all other cases QR58 is shifted into QR59.

6.2.4.2 RSQR IN – RSQR IN is an input to QR58 and is a function of QR59 or CSB1 (1) and CSB0 (0). ROM CSB fields 2 and 6 are specified for these bit patterns. However, field 2 has no effect because it is a right shift of the AR. Field 6 is a right shift of the QR. Consequently, when CSB1 and CSB0 are 1 and 0, respectively, the gate is enabled and RSQR IN goes to a 1. If the gate is disabled, RSQR IN follows the value of QR59 and is transferred to QR58.

6.2.4.3 RR2, RR1, RR0 – The bits designated RR2, RR1, and RR0 are used for division in order to speed up normalizing operations. RR2 corresponds to AR59 (sign bit), RR1 corresponds to AR58 (MSB of fraction), and RR0 corresponds to AR57. The RR bits are generated when the flip-flop associated with them is set. This can occur as a result of two conditions: 1) when a load is specified (ARS1 and ARS0 both high) and the corresponding FALU bit is present or 2) during a left shift, which occurs when ARS0 goes low.

As an example, when ARS1 and ARS0 are high, and FALU bit 59 is a 1, RR2 is set when the AR is clocked. The 1 being loaded in AR59 is also loaded in RR2. When ARS0 goes low, the second NAND gate at the input to RR2 is enabled (~ Load AR) for a left shift and RR1 (AR58) is shifted into RR2. A similar situation occurs with RR0 (AR57) shifting into RR1 and AR56 shifting into RR0.

Speed of division operations is increased by anticipating the normalization of a number. The bit patterns used to anticipate normalization are:

RR2	RR1	RR0	
0	0	1	Positive number
1	1	0	Negative number

The only bit patterns that normalize an unnormalized number in one shift are of the above shown configuration. If either of the above patterns is present, AR NORM will set on the next CLK AR pulse; consequently, as soon as the number is normalized, AR NORM is set to indicate this. Direct clear input to AR NORM holds AR NORM reset for anything other than divide or normalize operations. Direct set input to AR NORM ensures that the flip-flop is set when entering the divide subroutine, because the dividend in the AR is guaranteed to be normalized.

6.2.4.4 **LSQR IN** – To understand how LSQR IN (1) is generated, refer to the divide flow algorithm; this algorithm can be divided into two categories: one occurring during the normal shifting, and one occurring immediately after the add or subtract operation is performed. If the normal shifting prior to add or subtract is taking place, the value of AR bit 59 (represented by RR2) is shifted into QR bit 31 (if single precision) or QR bit 00 (if double precision). This condition is enabled with P2 on a 0 (see Figure 5-4). However, P2 is on a 1 for the shift performed immediately after the add/subtract operation. In this case, it is necessary to shift the complement of AR59 [RR2 (0)] into QR31 or QR00, depending on the designated format. Note that FD (0) allows AR59 to be shifted into QR bit 31 and FD (1) allows AR59 to be shifted into QR bit 00.

6.2.4.5 **SS Logic** – The SS logic consists of a 74H74 D-type flip-flop and associated gating. If the ACF field in the ROM (bits 34 through 32) is equal to zero (ACS) or one (AC V 1) and the upper half of the QR is to be loaded, the sign of the source is set from SCR OUT 31 H (sign bit) of the appropriate scratchpad accumulator. This means that when the microprogram loads the most significant 25 bits of the QR from the source AC (ACS) the most significant bit of scratch (bit 31) is also loaded in SS. The sign of the source can also be forced to a 1 by the SGN control bits in the ROM (bits 42 and 41). This is accomplished with SIGNC1 (1) and SIGNC0 (0). The SS flip-flop is clocked on the trailing edge of TS4 by the same conditions which enabled the D input to the flip-flop.

6.2.4.6 **SD Logic** – The SD logic consists of a 74H74 D-type flip-flop and associated gating. If the ACF field in the ROM is equal to two (ACD) or three (ACD V 1) and the upper half of the QR is to be loaded, the sign of the destination is set from SCR OUT 31 H (sign bit) of the scratch pad accumulator. This means that when the QR is loaded from a destination AC the most significant bit (SCR OUT 31) is also loaded into SD. This sign of the destination can also be set from the SGN control bits in the ROM according to the following conditions:

SIGNC1	**SIGNC0**	**Function**	

0	0	SD ← ~ SS if subtract,
		else SD ← SS
			[SUB H ∧SS (0) H V SUB L ∧SS (1) H]

0	1	SD ← SS ⊻ SD	[Exclusive OR of SS and SD]

For example, if the SGN bits in the ROM are selected for a field of 0, the complement of SS, which is SS (0) H, is transferred to SD for a subtract (SUB H). If the instruction being performed is not a subtract instruction (SUB L), SS, designated by SS (1) (H), is transferred to SD. The SS and SD flip-flops are cleared whenever the FIR is loaded (FRMJ READY CLR L).

6.2.4.7 **Step Counter** – The step counter is clocked in the Wait state of a hardware subroutine (refer to CSB bits 29 through 27 of ROM) if FRHE MUL DIV DISABLE is not on and the P0 flip-flop is cleared. These two signals combine to inhibit the step counter from being clocked while the pause logic (see Paragraph 6.2.3.4) is operating.

6.2.5 FRHH

This sheet contains the time state generator, pause, and maintenance pause (MPAUSE) logic and associated time state driver circuits.

6.2.5.1 Time State Generator – The time state generator consists of four 74S74 D flip-flops and associated gating. When the INIT switch on the console is depressed, the INIT signal clears the time state generator and the MPAUSE flip-flop. When time states 4, 3, and 1 go to a 0, time state 3 is initiated with the next clock pulse (see Figure 6-1). The generator then sequences to time state 4, time state 1, time state 2, and then to the Wait state or directly from time state 2 to time state 3. The Wait state is between state 2 and state 3. Each time state consumes 50 ns. If the FP11 flow diagram does not indicate a Wait state, the complete ROM cycle consumes 200 ns. If the Wait state is required, the total time is 200 ns plus the Wait period. During state 4, the ROM buffer is loaded from the ROM and during the next state 2, the next address of the ROM is clocked. The Wait state is initiated on the trailing edge of time state 2 when the PAUSE flip-flop is a 1.

6.2.5.2 PAUSE Flip-Flop – The following three conditions cause the PAUSE flip-flop to set:

a. **FP ACKN WAIT** – When the FP11 enters a new state in which a trap might occur, the FP11 anticipates an FP ACKN signal from the 11/45 in response to an FP trap signal from the FP11. This signal occurs after the interrupt and it is, therefore, necessary to initiate the Wait state.

b. **SUB CALL** – If any of the three CSB bits are 0, indicating a hardware subroutine operation, the PAUSE flip-flop is turned on. All 1s in the CSB bits indicate a NOP.

c. **FP ATTN WAIT** – The FP11 enters the Wait state while waiting for FP ATTN from the 11/45.

In each of the above three instances, the PAUSE flip-flop is clocked at the trailing edge of time state 1. The PAUSE flip-flop is cleared:

a. When the CSB bits specify a normalize and the normalize is completed, which is represented by AR NORM (1).

b. When the 11/45 sends an FP ATTN to the FP11, indicating that the 11/45 is now ready to receive or send data to the FP11.

c. When the step counter has fully incremented to all 1s to indicate completion of the operation.

d. When the 11/45 responds to an FP TRAP by issuing FP ACKN.

e. When ICLR is set an initialize condition is established which clears all major registers.

The PAUSE flip-flop, when reset, allows the WAIT flip-flop to be cleared on the trailing edge of the next clock pulse and allows state 3 to set.

6.2.5.3 MPAUSE Flip-Flop – The MPAUSE flip-flop is used in conjunction with the W131 Maintenance Module. A switch on this card removes the direct clear input from MPAUSE and allows this flip-flop to be armed by ROM + UBS. This signal results from a micromatch occurring between the control ROM address register and the microbreak register or by setting the appropriate switches on the maintenance card. Note that MPAUSE operates in parallel with PAUSE and also prevents the FP11 from sequencing to state 3 from state 2.

The remaining logic on this sheet shows the time state driver circuits (A and B outputs), which are necessary because of loading requirements.

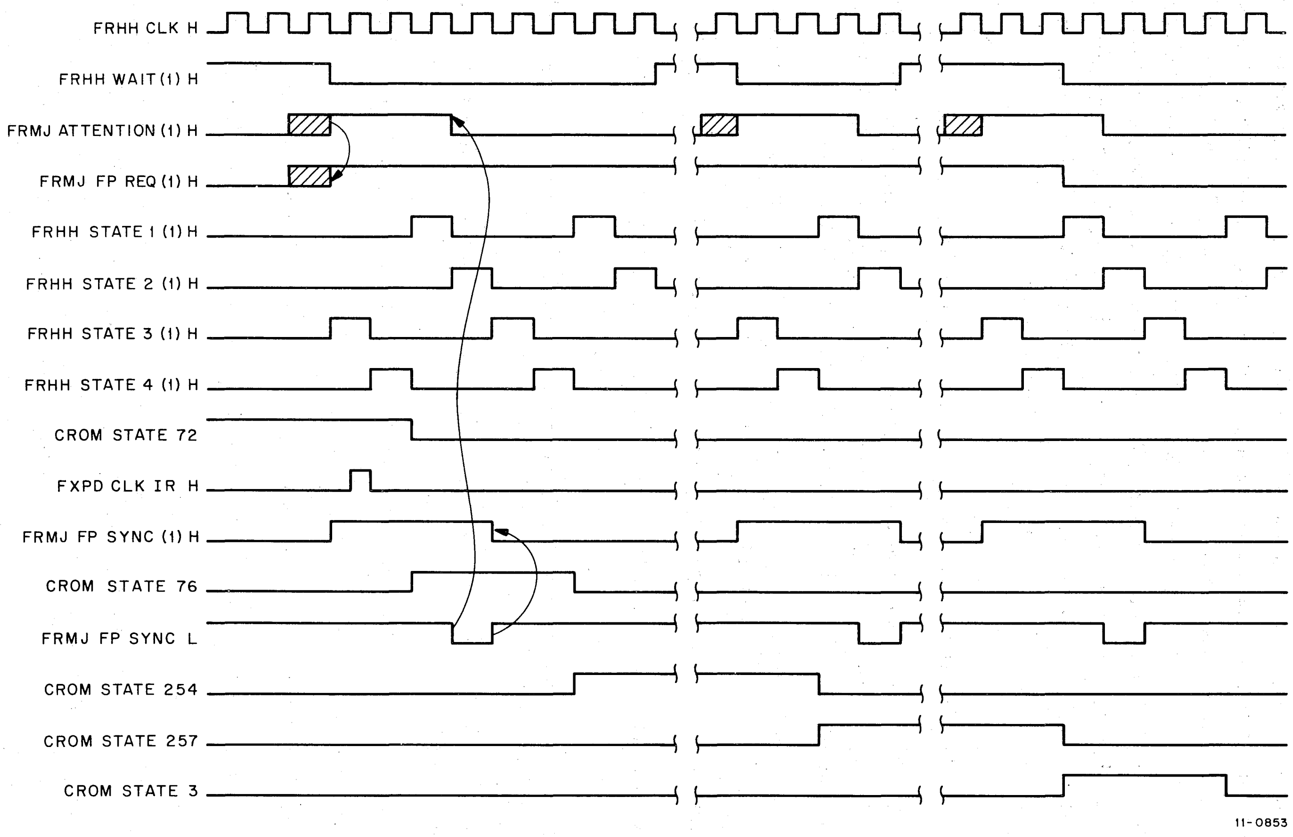


Figure 6-1 Time State Generator Timing Relationships

6.2.6 FRHJ

This sheet shows the 20 MHz crystal clock, the variable RC clock, and a source synchronizer providing switching between the two clock sources. If switching should be attempted during a crystal clock pulse, the crystal clock pulse is completed before the RC clock is switched in and vice versa. The RC clock is used for maintenance and is a variable clock source whose frequency can be adjusted by the variable resistor shown.

The source synchronizer operates in conjunction with the switches on the FMAA Maintenance Card (see KM11 Maintenance Set Manual – cards W130 and W131). S3 selects the crystal clock when off or the RC clock when on. S4 is a MAINT STPR switch that allows the function specified by S2 and S1 to be stepped. If S2 and S1 are off, normal operation occurs. If S2 is on and S1 is off, a single ROM cycle occurs each time the MAINT STPR is depressed. If S2 is off and S1 is on a micromatch between the CRAR (control ROM address register) and the microbreak register will stop the clock; and if both S2 and S1 are on, a single clock pulse will occur each time the stepper is depressed.

The J-K flip-flop that is clocked by the single time stepper is complemented each time the stepper is depressed. The MPAUSE flip-flop on FRHH is set during TP1 when a single ROM cycle is selected or a micromatch occurs. When the single time stepper is depressed, the MSWITCH CNTU flip-flop (see FRHH) goes to a 0 and resets the MPAUSE flip-flop. At time state 4 MSWITCH CNTU goes to a 1 inhibiting the direct clear to MPAUSE.

6.2.7 FRLA, FRLB, FRLC, FRLD

These logic prints show the ACMX logic and the scratch pad accumulators. The ACMX consists of 16 dual-section 74153 multiplexers. The EALU, FALU, Floating-Point Status word, and the B register condition codes (BN and BZ) provide inputs to ACMX. Common select lines at S1 and S0 provide selection of one of four inputs from each half of the chip.

There are a total of eight 3101 scratchpad accumulator chips. The dual outputs from two multiplexer chips are applied to a scratchpad accumulator chip. An SCR WRITE signal, if low, causes data to be written into scratch and, if high, causes data to be read out of scratch. Note SCR WRITE 1 and SCR WRITE 0 versions of this signal are necessary because of loading problems.

Other inputs to the scratchpad are used to determine the AC specified and the quadrant specified. SCR ADDRS 2, SCR ADDRS 1, and SCR ADDR 0 are a modified version of ACF bits (bits 34, 33, and 32) of the CROM word and selects source AC, destination AC, AC6, or AC7 (refer to CROM word format). Bits ACC2, ACC1, and ACC0 are applied to inputs A3 and CS in the accumulator and are decoded to yield the quadrant specified. Each quadrant is 16 bits and is specified by a number from 3 through 0. Quadrant 3 is bits 63 through 48, 2 is bits 47 through 31, 1 is bits 31 through 16, and 0 is bits 15 through 0 (see Figure 6-2). For example, if bit ACC2 is a 1 and ACC1 is a 0, the 3101 chips containing bits 31 through 16 are specified. This is quadrant 1. If bit ACC0 is a 0, quadrant 0 is also enabled which, according to the ROM word format, gives a field of 4. This can be verified by referring to the CROM word format. To be more specific, ACC2 is connected to the most significant address select line of all the 3101 chips. ACC1 is connected to the chip select of the four 3101s that contain quadrants 1 and 3 of the AC. ACC0 is connected to the chip select of the four 3101s that contain quadrants 0 and 2 of the AC.

6.2.8 FRLE, FRLF, FRLH, FRLJ, and FRLK

These logic prints show the lower half of the AR register (bits 35 through 0) and the lower half of the FALU (bits 35 through 0). The AR register consists of nine 74194 shift register chips, each chip having six inputs.

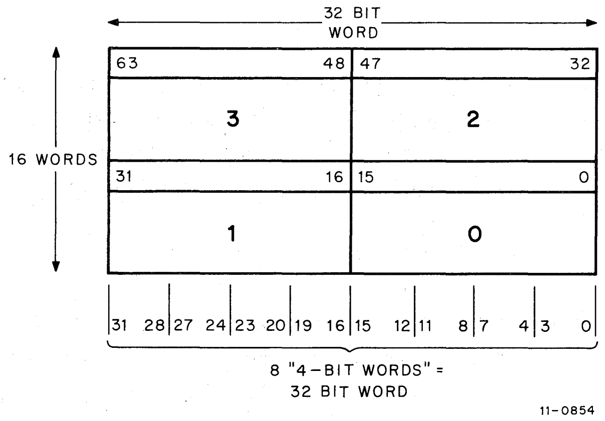


Figure 6-2 Scratchpad Configuration

One bit to the AR register chip is from the next higher order bit of the preceding chip and provides the right shift capability of the AR. Bit 12 feeds bit 11, bit 8 feeds bit 7, and bit 4 feeds bit 3. A second input to the AR is from the next lower order bit of the succeeding chip. This provides the left shift capability of the AR, where bit 3 feeds bit 4, bit 7 feeds bit 8, and bit 11 feeds bit 12. The other four inputs to each of the nine AR chips allow it to be parallel loaded with data from the FALU. The AR is simultaneously clocked by CLK AR, which is applied to all AR chips.

6.2.8.1 FALU Control – The 4-bit output of each AR chip is applied to the A side of FALU along with four corresponding bits from the BR register. See FRHE logic diagram description for discussion of AR select lines.

The FALU in the FP11 may perform one of 18 functions. In order to ascertain the desired function, five control lines are supplied to the FALU. These are designated ALUM and ALUS3 through ALUS0, and derived from ALU bits 19 through 16 of the control ROM word.

6.2.8.2 Carry-Look-Ahead – Associated with the FALU are 74182 carry look-ahead generators. Each carry look-ahead anticipates the carry for a total of four FALU chips. Eight of the FALU chips are associated with the two 74182 carry look-ahead circuit generators on the FRL prints. The ninth FALU chip is handled by a carry look-ahead generator located on the FRH prints. The carry look-ahead circuitry is used to speed up arithmetic operations.

A second level of carry look ahead is provided between each group of four FALU chips (see sheet FRLK). This circuit anticipates a carry between groups of four FALU chips, by looking at the three lowest order groups of FALU and providing a carry, if required, to the three highest order groups of FALUs.

6.2.8.3 Rounding – The rounding logic for double-precision floating-point format is shown on sheet FRLE. The data path that handles the fraction has three extra bits (bits 2, 1, and 0) that are carried for rounding purposes. The logic is implemented such that only the most significant bit, AR2, is examined. If this bit is a 1, 1 is added to bit 03 in the FALU. If this bit is a 0, nothing is added to the FALU.

For single-precision floating-point format, the word is located in bits 63 through 32 of the AR. The logic is implemented such that only the most significant rounding bit (AR34) is examined. If this bit is a 1, 1 is added to bit 35 of the FALU. If this bit is a 0, the FALU is unaffected.

6.2.8.4 Increment – For certain integer operations in long integer mode, the word is to be incremented - for example, when converting a 1's complement number to a 2's complement number. For long-integer format, the 32 integer bits are stored in bits 50 through 19 of the scratchpad (bits 50 through 35 for single-integer format). Therefore, when the long-integer word is to be incremented, 1 is added to bit 19 of the FALU (see sheet FRLH), if the FMX select signal designated FMXC1 is on a 0. Similarly, when the short-integer word is to be incremented, 1 is added to bit 35 of the FALU if FMXC1 is on a 0.

6.2.9 FRLL, FRLM

These prints show the low order 36 bits of the QR and BR registers, the EXP NEQ 0 (exponent not equal to 0) logic and the LSQR31 in H logic.

6.2.9.1 QR – The QR consists of nine 74194 left/right shift chips. Four of the inputs are the normal scratchpad outputs. The other two are inputs from the adjacent QR chips to provide the right shift/left shift capability, just as described in the AR register. Note that bits QR02, QR01, and QR00 are output from the QR, but the corresponding bits are never input from scratch and are grounded. The loading of the QR is described in the description of the FRH group of prints.

6.2.9.2 BR – The BR consists of six 74174 flip-flop chips each with six inputs and six corresponding outputs. The BR is loaded by CLK BR, which occurs on the trailing edge of TS4 if BR control (bit 24 of the CROM word) is a 0. The BR is cleared by CLR BR, which occurs during TS2 if BR control is a 1.

6.2.9.3 EXP NEQ 0 – The EXP NEQ 0 logic generates the input to QR58 (hidden bit) so that QR bit 58 will be loaded with a 1 if the exponent is not 0. If the exponent is 0, the fraction is assumed to be 0.

6.2.9.4 LSQR31 IN H – LSQR31 IN H is used as the input to QR31 when left shifting the QR. During normal left shifts, QR30 is applied to QR31; during single-precision divide, the partial quotient bits are shifted into QR bit 31.

6.2.10 FRLN

This sheet shows the BMX, consisting of eight 74153 dual four-to-one line multiplexers. Each section has four input and one output. Two control lines (BMX C1 and BMX C0) select one of four inputs from each section.

The A inputs are inputs from the EALU, the B inputs are SCR31 through SCR16 outputs, the C inputs are SCR15 through SCR0 outputs, and the D inputs are SCR30 through SCR23 outputs. Note that the D input selects the exponent portion of the AC right justified (bits 7 through 0) while bits 15 through 8 are 0s.

6.2.11 FRLP

This sheet shows the floating status register, floating condition code loading, and the FER flip-flop.

6.2.11.1 Floating Status Register – The floating status register consists of 74175 D-type flip-flop chips, a 74H74 flip-flop, and associated gating. The status register can be loaded by the LD FPS instruction (170100) or by control ROM at the appropriate time to generate the floating condition codes. If the register is to be loaded with bit 4 on a 1, the CPU must be in KERNEL mode. The FP11 enters maintenance mode and the maintenance mode flip-flop will set. The programmer, by use of the status register, can set up enables for various interrupt conditions. For example, by setting EALU bit 09 to a 1 and loading the status register, the floating interrupt on overflow (FIV) is enabled. If an overflow occurs, an interrupt will be raised.

6.2.11.2 **Floating Condition Codes** – The floating condition codes can be loaded from two different sources:

a. For the LD FPS instruction, the output of the EALU is enabled to the D inputs of the four condition code bits. The FRMF LD FPSC signal generated from the ROM enables the clocking of the condition code flip-flops.

b. If either of the FC control bits are on a 0, the condition code flip-flops will be clocked. In this case, the FP11 is not doing a LD FPS instruction; FN will be set if the result is negative; FZ will be set if the exponent is 0; FV will be set from the conditions of the EALU, which contains the exponent of the result at this time; and FC is set or cleared from the ROM control.

6.2.11.3 **FER Flip-Flop** – The FER (floating error) flip-flop is set if a floating-point exception occurs or by bit 15 of the LOAD FPS instruction. It is cleared by a zero in bit 15 of the LOAD FPS instruction.

6.2.12 FRMA, FRMB

These two sheets show the 8-bit control ROM address register (CRAR) and the control ROM multiplexers. The address register uses 74S74 D-type edge triggered flip-flops, and the control ROM multiplexers use 74151 8-to-1 line multiplexers. The two most significant bits of the address are not modified so there are only a total of 6 multiplexers.

6.2.12.1 **Control ROM Address Register** – The circuitry is designed such that the ROM can sequence to the next address, which may or may not have been modified by the branching conditions, or can sequence to the Ready state, or can trap to a service routine. If the next address is not modified, bits D07 through D00 provide the inputs to each flip-flop in the address register provided a trap condition is not present. The five conditions that can cause a trap are:

a. **INIT AND 11/45 ABORT** – sets INIT F and forces the next address to 0.

b. **Microbreak** – sets UBRK F flip-flop which forces the next address to location 4.

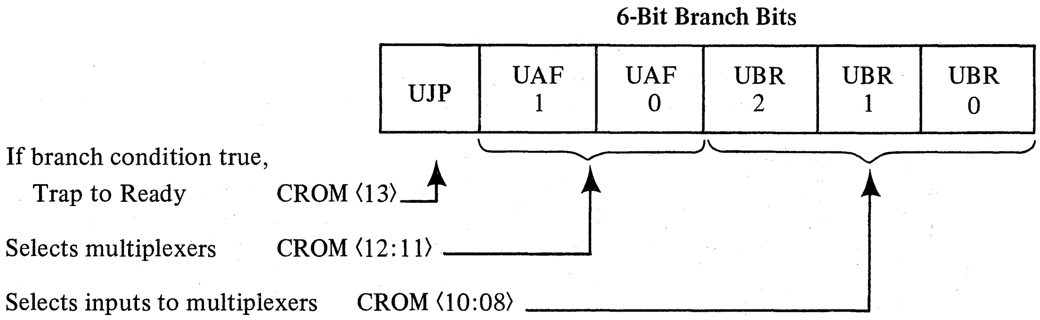
c. **1120 ABORT** – sets ABORT F and forces the next address to location 10.

d. **Floating Minus 0 (FM0)** – sets FM0 F flip-flop and forces the next address to 20.

e. UJP enabled and bits 0 and 1 of the next address on 1s generate GO TO READY L which forces the next address to 3.

If the next address is to be modified, the address bits that are to be modified are switched from 0s to 1s. This is accomplished by forcing the associated multiplexer output to a 1. If the address bit is a 1, it cannot be modified, because the normal ROM output (D7 through D0) sets the associated flip-flop regardless of the multiplexer output.

6.2.12.2 **Address Modification** – The conditions to be used to modify an address are selected by the six control ROM bits, three of these being the UBR (microbranch) bits, two being the UAF (microaddress field) bits, and one being the UJP (microjump) bit.



The three UBR bits are applied to each of the six multiplexers and uniquely specify one of the inputs to the multiplexer. If UBR bits 2, 1, and 0 are all 1s, the multiplexer output goes to 0, which indicates no modification takes place. For all other combinations, the multiplexer output goes to a 1 if the selected branch condition is true. The UAF bits specify the multiplexer(s) as follows:

UAF1	UAF0	Multiplexers Selected
0	0	0 through 5 if UBR is even (UBR0 on a 0) 2 through 5 if UBR is odd (UBR0 on a 1)
0	1	0 Multiplexer selected
1	0	1 Multiplexer selected
1	1	Both 0 and 1 Multiplexers selected

Note that all six multiplexers are selected if UAF1 and UAF0 are 0s and the UBR field is 0, 2, 4, or 6; and multiplexers 2 through 5 are selected, if UBR is 1, 3, 5, or 7. If a multiplexer is not selected, its output is low and the associated address bit will not be modified.

Table 6-2 shows how the multiplexer (specified by the UAF bits) and the inputs to the multiplexer (specified by the UBR bits) combine to create certain branching conditions. For example, if the UBR bits are all 0s, the UAF bits are also 0s, multiplexers 0 through 5 are specified. As a result, signals that are true at the A inputs to each multiplexer will cause the multiplexers' output to go high and cause that address bit to be modified (see 74S74 IC description in Appendix A).

When the UAF bits are both 0s and there are no trap conditions present, SELECT UBRMXB is generated, which is applied to the STB0 inputs of the multiplexers and selects multiplexers 2 through 5. Multiplexers 0 and 1 are specified by utilizing other combinations of the UAF bits as shown on FRMB.

6.2.12.3 **Traps** – Sheet FRMA shows the logic associated with the trap conditions. For the UBRK trap to occur, the FP11 must be in maintenance mode, and out of the Ready state, and a match must have occurred between the UBR register and the CRAR. The floating minus zero trap occurs when the sign bit is a 1 and the exponent is 0. This is detected on the output of the ACMX where the data is in complement form. Note that when a trap condition is present, UTRAP A is generated (see sheet FRMB). This signal is applied to bits 0 and 1 of the CRAR and inhibits the ROM and multiplexer inputs to these two stages. This prevents the FP11 from going to the Ready state. UTRAP A and GO TO READY are *ORed* to generate UTRAP B. This signal is applied to bits 7 through 2 of the CRAR and inhibits the ROM bit from setting the register but allows the trap condition to set the register.

Table 6-2
Multiplexer Branching Conditions

		5	4	3	2	1	0
Multiplexer Inputs	A	SUB FRAC	FIRD4	FIRD3	FIRD2	FIRD1	FIRD0
	B	FIR07 (1)	FIR06 (1)	FIR11 (1)	FIR10 (1)	AR50 (0)	SD (1)
	C	RNG2	RNG1	RNG0	0	BB1Z (1)	BN (0)
	D	0	0	0	FIU (1)	IL (0)	Immediate
	E	0	0	0	FT (1)	~ (FC ∧ FIC)	FD (0)
	F	FIRD6	FIRD5	0	~ CONV SP	~ (FV ∧ FIV)	M0
	G	0	0	FIR08 (0)	AR58 (0)	AR59 (0)	BZ (1)
	H	0	0	0	0	0	0

6.2.13 FRMC, FRMD

The ROM is contained on these prints and consists of sixteen 74187 Read Only Memory chips, providing a matrix of 256 64-bit words. Each ROM chip contains 256 4-bit words; 8 bits of address are required to select one of the 256 words. The 8 address lines are applied to all chips in parallel, and the output of each ROM is 4 bits wide yielding a 64-bit ROM word.

6.2.14 FRME, FRMF

These logic prints show the ROM buffer, which consists of 74175 D-type flip-flop chips. Each chip receives four ROM outputs and provides a pair of outputs for each input. The pair is simply the 0 and 1 output of a flip-flop toggled by the associated input.

Only 14 buffer chips outputting 56 bits are required, because the 8 bits of next address are not applied to the ROM buffer but instead are applied to the control ROM address registers through some branch condition gating logic. In order to provide additional outputs, the signals designated CONTROL SEL 2, CONTROL SEL 1, and CONTROL SEL 0 (FRMF), in turn, are octally decoded to produce eight unique outputs. Only five of the eight outputs are presently utilized.

The ROM buffer is loaded on the trailing edge of TS4 by CLK RB C L as are the eight additional outputs.

6.2.15 FRMH

This sheet shows the decoding of the ALU select lines, the scratch address lines, SCR WRITE, clocking of the BR, and BACMX selection.

6.2.15.1 **ALU Select** – Normally, the ALUS3 through ALUS0, ALUM, and ALUCIN signals are driven from the ROM ALU control signals (ALUC3 through ALUC0). Note that there are four ROM output signals from the ALU control (ALUC3 through ALUC0), which are decoded to produce six ALU signals (ALUS3 through ALUS0, ALUM, and ALUCIN). ALUM is a mode bit that is low when an arithmetic function is performed and is high for a logical function. ALUCIN is a carry input that is required only when ALUM is a 0 (i.e., when an arithmetic operation is being performed). The ROM signals produce the ALU select signals if the FP11 is not in the arithmetic subroutine (MUL DIV is high).

When the FP11 enters the multiply or divide subroutine, MUL DIV goes low and permits the subroutine signals (MR1 and RR2) to drive the ALU select lines to the correct configuration (see Table 6-3). FRAC MUL is set by the decoding of the CSB bits in the ROM, indicating the FP11 is in the multiply subroutine. DIV OR NORM is set by the decoding of the CSB bits in the ROM, indicating the FP11 is in a divide subroutine.

As an example of how the ROM controls the ALU select lines, consider the subtract functions as selected by fields 2 and 6. The subtract function can thus be shown as specified.

ALUC3	ALUC2	ALUC1	ALUC0		
0	X	1	0	=	Field of 2 or 6

The ALUC3, ALUC1, and ALUC0 signals are decoded to yield the FORCE SUB signal, which drives ALUS2 to a 1 for fields 2 and 6. Note that most of the entries (except for fields 2, 10, and 15) in the table are on a 1:1 correspondence with the numerical value of the control field. Field 2 creates a FORCE SUB signal that causes ALUS2 to go to a 1; field 10 creates a FORCE ADD signal that causes ALUS0 to go to a 1; and field 15 creates a signal that causes ALUS1 to go to a 1. In these instances, either the ALUCIN or ALUM bits is varied to differentiate between the ALU functions.

Table 6-3
ALU Control Selection

ALU Control Field (ALUC3–ALUC0)	Function	ALU Select Lines				Mode ALUM	Carry in ALUC1	
		ALUS3	ALUS2	ALUS1	ALUS0			
0	~A	0	0	0	0	1	X	
1	~(A V B)	0	0	0	1	1	X	
2	A minus B	0	1	1	0	0	0	Drive ALUS2 low
3	0	0	0	1	1	1	X	
4	~(A ∧B)	0	1	0	0	1	X	
5	~B	0	1	0	1	1	X	
6	A minus B minus 1	0	1	1	0	0	1	
7	A∧~B	0	1	1	1	1	X	
10	A plus B plus 1	1	0	0	1	0	0	Drive ALUS0 low
11	A plus B	1	0	0	1	0	1	
12	B	1	0	1	0	1	X	
13	A∧B	1	0	1	1	1	X	
14	1	1	1	0	0	1	X	
15	A minus 1	1	1	1	1	0	1	Drive ALUS1 low
16	A V B	1	1	1	0	1	X	
17	A	1	1	1	1	1	X	

X = don't care
0 = low
1 = high

The ALUM (mode bit) signal is driven low for arithmetic operations and is high for logical functions. When not in a multiply or divide subroutine, ALUM is driven low for a ROM ALU field of 15 designating A minus 1. FORCE ADD (created by fields 10 or 11) or FORCE SUB (created by fields 2 or 6) also cause ALUM to go low. Normally ALUCIN is high, however, it is driven low in fields 0, 2, 10, and 12. The signal has no meaning for fields 0 and 12 because the ALUM signal is a 1 in that field. ALUCIN can also be driven low by MUL SUB or DIV SUB when in a subroutine.

6.2.15.2 **SCR Address** – SCR ADRS 2 through SCR ADRS 0 bits are decoded as a result of the ACF bits as shown in Table 6-4.

Table 6-4
Scratch Address Selection

Field	ACF2	ACF1	ACF0	Function	SCR ADRS 2	SCR ADRS 1	SCR ADRS 0
0	0	0	0	ACS	Looks at FIR02, 01, and 00 if address mode 0 is specified. These bits can address ACs 0 through 5. If not mode 0, ACC is specified.		
1	0	0	1	ACS V1			
2	0	1	0	ACD	Looks at FIR06 and 07,of instruction. These bits can address ACs 0 through 3.		
3	0	1	1	ACD V1			
4	1	0	0	AC6	1	1	0
5	1	0	1	AC7	1	1	1
6	1	1	0	AC6	1	1	0
7	1	1	1	AC7	1	1	1

The remaining logic shows: *a.* the gating for clocking and clearing of the BR, *b.* the SCR WRITE 0 and SCR WRITE 1 signals which occur during TS 4 when ACRE (Accumulator Read) is on a 0, and *c.* BACMX C1 (1) and BACMX C0 (1) which are the buffered ACMX control lines used to select one of four inputs to the ACMX.

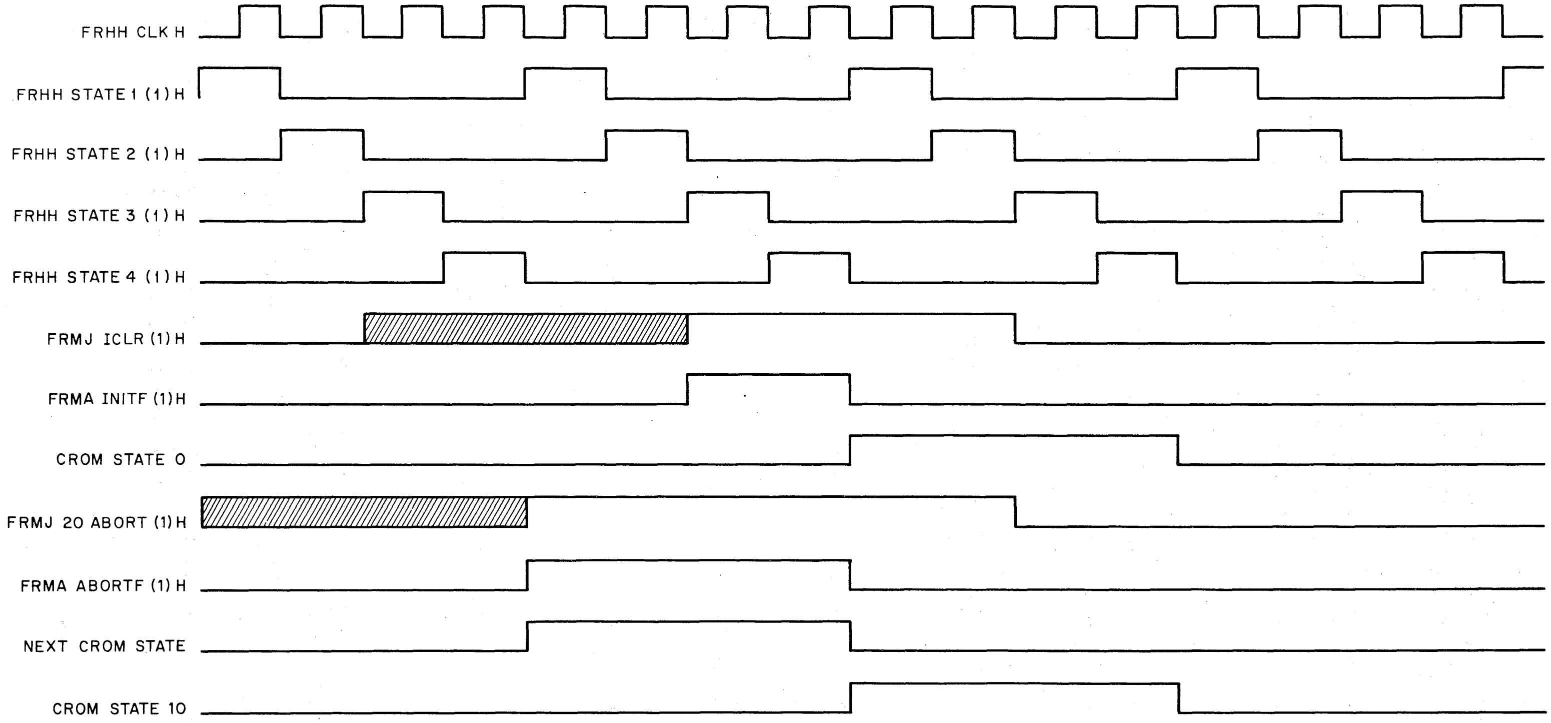
6.2.16 FRMJ

This sheet shows some of the decoding of the interface signals (FPC1, ADR INC, etc.) and contains the logic associated with INIT, ICLR, FP REQ, FP SYNC, and FP ATTN.

6.2.16.1 **ICLR and 20 ABORT** – The ICLR and 20 ABORT flip-flops are set from asynchronous external sources. ICLR is set by INIT or INTR CLR and FP REQ (1). INTR CLR is generated by the 11/45 if an abort condition is found. Note that the FP11 is trapped back to the Ready state. Both ICLR and 20 ABORT are cleared under ROM control. Figure 6-3 shows the timing associated with ICLR, INIT, and 20 ABORT.

6.2.16.2 **Set ATTENTION** – The FP ATTN signal from the CPU sets the ATTENTION flip-flop. This signal indicates that the CPU is requesting the transfer of data to or from the FP11. If the FP11 is in the Ready state, the setting of the ATTENTION flip-flop clears the PAUSE flip-flop (see sheet FRHH), allowing the time state generator to advance. This sequences the FP11 out of the Wait state.

6.2.16.3 **Set FP REQ** – FP REQ is set by ATTN (1) and FIRC (0), which is true only in the Ready state.



NOTE:
DISBLO IN STATE 10 CAN NOT CLEAR ICLR SO THAT IF POWER ON CAUSES START AT CROM ADDRESS 10 INIT WILL STILL BE SEEN.

11-0808

Figure 6-3 ICLR, INIT, and 20 Abort Timing Relationships

6.2.16.4 Set FP SYNC – The FP SYNC flip-flop is normally set under ROM control by SYNC (0) H in time state 3.

6.2.16.5 Clear ATTN – The ATTN flip-flop is cleared by SET SYNCF L, which occurs when the FP SYNC flip-flop is set. This allows the CPU to raise another FP ATTN signal for additional transfers.

6.2.16.6 FP SYNC L – FP SYNC L is a synchronizing signal sent to the CPU in response to FP ATTN and is generated during TS2 of the next ROM state following the setting of the FP SYNC flip-flop. FP SYNC L, being delayed until TS2, allows time for FP REQ to be cleared if no more data transfers are required.

6.2.16.7 Clear FP SYNC – The issuing of FP SYNC L clears the FP SYNC flip-flop so that only one FP SYNC is issued. The FP SYNC flip-flop can also be cleared at TP4 by CLR SYNC if the instruction contained in the IR is a CONV SP class and the DISBL SYNC signal from ROM control is present. The reason for clearing FP SYNC at this time is to delay FP SYNC L to allow conversion of the data before storing. This delay allows the 11/45 CPU to monitor BR requests during the data conversion.

6.2.16.8 Clear FP REQ – The FP REQ flip-flop is cleared by DISBL 1 (0) from the ROM control.

6.2.17 FXPA and FXPB

The EMX and EALU and constant field decoding are shown on logic prints FXPA and FXPB.

6.2.17.1 EMX – The 16-bit EMX consists of eight 74153 ICs, each IC capable of processing two bits of data. It selects one of four inputs via two select lines – EMX C1 (1) H from the control ROM and EMX C0. The truth table for the EMX is as follows:

S1	S0	
EMX C1 (1)	EMX C0	Input Selected
L	L	BA
L	H	MPX DATA
H	L	CNST
H	H	SC

6.2.17.2 EALU – The 16-bit EALU consists of four 74181 ICs, each IC capable of processing four bits. These four bits represent the outputs of two EMX chips. Five select lines (S3, S2, S1, S0, and M) provide the capability of selecting a wide variety of arithmetic operations (see 74181 IC description in Appendix A).

6.2.17.3 Constant Field Decoding – The FXPA print shows additional decoding logic, which decodes the constant fields to produce the desired constant. Constant fields which are not mapped 1 to 1 are:

Constant Field	Constant
0	200
11	100000
15	100004
20	220
32	70
33	71
36	74
37	75

The constant field is specified by bits 57 through 53 of the ROM word. For a constant field of 0, bit 7 of the EALU is enabled, yielding 200_8. For a constant field of 20_8, bit 57 (field bit 04) of the ROM word is also a 1, enabling bit 4 of the EALU as well as bit 7 to yield 220_8. For constant fields 11 and 15, CNST bit 15 is generated. CNST bit 00 is decoded for all odd fields except 11 and 15. CNST bit 03 is generated for constant fields of 10, 12, 13, 14, 16, and 17 because bit 03 is common to all these fields.

The EN CNST 7X signal is generated for constant fields of 32, 33, 36, and 37 as shown below:

Bit 57	Bit 56	Bit 55	Bit 54	Bit 53	Constant	
4	3	2	1	0	Field	Constant
1	1	0	1	0	32	70
1	1	0	1	1	33	71
1	1	1	1	0	36	74
1	1	1	1	1	37	75

EN CNST 7X is applied to the EMX and becomes the B input to EALU bit 05. CNST 4 and CNST 3 are enabled in order to generate EN CNST 7X. These signals are applied to EALU bits 04 and 03, respectively, and as a result EALU bits 5, 4, and 3 are enabled to generate an octal 7.

6.2.18 FXPC

This logic print contains the 16-bit BA and 16-bit BD registers, which are used as storage registers. Each register consists of two 74174 D-type flip-flop chips and one 74175 D-type flip-flop chip. The 74174 ICs are 6-bit chips and the 74175 is a 4-bit chip.

6.2.18.1 BD Register – The BD is loaded by CLK BD from the control ROM. This occurs during the trailing edge of the clock pulse in TS4, if bit 49 of the control ROM word (BDC 0) is on a 0.

6.2.18.2 BA Register – The BA is loaded by CLK BA from the control ROM if bit 48 of the ROM word (BAC 0) is on a 0. CLK BA is generated on the trailing edge of the clock pulse in TS4, if bit 48 (BAC 0) of the control ROM word is on a 0. A second clock input is available from the interface unit with the 11/20 Central Processor. This signal is designated FICC CLK BA.

NOTE
Both the BA and BD registers are loaded with data from the BMX (see description of sheet FRLN).

6.2.19 FXPD

This sheet shows the 12-bit instruction register, which receives the instruction from the buffered BR inputs. The register consists of three 74175 D-type flip-flop chips and is loaded on the leading edge of TS3 when bit 59 of the ROM word (FIR control) is a 0.

The remaining logic on the sheet shows the two instruction ROMs. The five inputs to each ROM select one of 32 8-bit words depending on the specified instruction (see Table on FXPM-M8113-0-0). The instruction ROMs are 8598 ICs, and the outputs of each ROM are applied to the branch logic on FXPE.

6.2.20 FXPE

This sheet shows the decoding associated with the branching conditions. The A and B outputs from the instruction ROMs are collector ORed to set up the various branches. AD1 and AD2 are the constants used to update the CPU general register, depending on the format specified. FCLD LD EN is the enable level used to cause the floating condition codes to be copied in the CPU. The remaining signals on this sheet represent the decoding of the floating instruction register to set up various addressing modes and/or branching conditions.

6.2.21 FXPF

This sheet shows the B condition code logic, the range ROM used to determine the magnitude of the exponent difference between two numbers, the logic used to develop the SUB FRAC signal, and the illegal op code detector.

6.2.21.1 B Condition Code Logic – The condition code logic contains a 74175 flip-flop chip to generate the B condition codes. If BB1Z is a 1, the upper byte (bits 15 through 8) of the last word loaded in the BA or BD register is 0. If BN is a 1, the last word loaded in the BA or BD register is negative (bit 15 on a 1). If BZ is a 1, the last word loaded in the BA or BD register is 0 (bits 15 through 00). The 74175 flip-flops are clocked on the trailing edge of TS4 when the BA control (bit 48 of CROM) or BD control (bit 49 of CROM) is a 0. When this occurs, the BA or BD register is loaded so that the B condition codes reflect the condition of the data loaded in the BA or BD register.

6.2.21.2 Range ROM – The range ROM is a 256-word x 4-bit ROM used to determine the difference between two exponents. The ROM provides a 3-bit output, which is applied to the branching logic. EALU bits 9 through 0 are provided at the input. The reason for this circuitry is described in Paragraph 5.2.2.

6.2.21.3 SUB FRAC – The SUB FRAC signal is developed when the hardware is to perform a subtraction. This occurs when an add instruction with unlike signs or a subtract instruction with like signs is specified. If SUB FRAC is not present, the FP11 sequences through the add branch where the hardware performs an add operation (see sheet FRMA).

6.2.21.4 Illegal Op Code Detector – The illegal op code logic examines bits 15 through 12 of the word for all 1s yielding a 17_8 op code. If any of these bits go to 0, an illegal op code sign is generated. This is used to force FIRD5 and FIRD6 low, which causes the microprogram to branch to the illegal op code routine (see sheet FXPE).

6.2.22 FXPH

This print shows the 74H74 floating-point (FD) and integer (IL) flip-flops. Also shown are sixteen 74S05 open-collector drivers, which provide the BMX outputs to the B condition code logic on FXPF.

The combinaticn of bits FIR06 and FIR03 are set up to specify one of the following five instructions.

FIR03	FIR06	Instruction	
0	1	LD FPS	(Load FP Status)
0	0	SET F	(Floating-point)
1	0	SET D	(Double Floating-point)
0	0	SET I	(Integer)
1	0	SET L	(Long Integer)

If FIR06 (1) H is true and EALU bit 06 is present, the D input to the IL flop is enabled. When the flip-flop is clocked, the IL bit in the status register is set. If FIR06 (1) H is true, and EALU bit 07 is present, the D input to the FD flip-flop is enabled. When the flip-flop is clocked, the FD bit in the status register is set. Both flip-flops are clocked at the trailing edge of TS4. The LD FPSC (0) H signal is true for the five instructions listed above and FIR06 (1) L is true in the case of the LD FPS instruction.

If FIR06 (0) H is true, one of the other four instructions (SET D, SET F, SET I, SET L) is specified. This is dependent on FIR03, FIR00, and FIR01. If FIR02 (1) H is true, either the IL or FD flip-flop is set, depending on which gets clocked. The flip-flop that gets clocked is determined by FIR01 or FIR00. Note that FIR06 (1) L is now disabled. If FIR01 (0) L is true, FD is clocked; if FIR00 (0) L is true, IL is clocked.

The remaining logic on the sheet shows the open-collector inverters, which are collector ANDed to supply the inputs to the B condition code logic on FXPF.

6.2.23 FXPJ

The logic on this print is used for maintenance purposes and consists of eight D-type flip-flops, and two 7485 4-bit decoders. The eight D-type flip-flops are housed in two 74175 ICs – four per IC. These flip-flops comprise the 8-bit U Break register, which is loaded from EALU bits 7 through 0.

Because the FP11 does not have the capability of determining what state the CROM is in, the U Break register and decoding logic are designed for this purpose. This allows the programmer or maintenance personnel to load an 8-bit address into the U Break register. When the CROM sequences to this address, it is detected and a μ match signal is generated. Detection occurs because the contents of the microbreak register matches the controls of the CRAR (Control ROM Address Register).

The CROM address is loaded in TS2 but the CROM buffer is actually loaded in TS4. Note that two versions of the μ match signal are available – one occurring as soon as the μ match signal is generated. This signal is sent to the FRM module to produce an interrupt when a match is detected. The second version of this signal enables a D-type flip-flop, which is clocked at the same time as the CROM data buffer (TS4). The output of this flip-flop provides a synchronizing signal for oscilloscope use. This flip-flop sets at the beginning of the required ROM state and remains true for the entire ROM state.

6.2.24 FXPK

The FXPK print shows the DIMX and the drivers, that the BD register feeds for communication with the 11/45.

6.2.24.1 DIMX – The DIMX consists of four 74158 quadruple 2 line-to-1 line multiplexers. Inputs are from the CPU BAMX (designated BAMX15 to BAMX00) or from the DATA IN lines (designated BR15 to BR00).

When the FP11 is in the Ready state, the BAMX data input to MPX DATA15 through MPX DATA00 (DIMX) is enabled, which allows the address of the instruction to be loaded into the scratchpad accumulator via the EMX, EALU, and ACMX. The instruction is fed to the FIR directly from the buffered BR lines. In this way, both the PC and the instruction are transferred to the FP11 at the same time. When the FP11 is not in the Ready state, the buffered BR lines are enabled through the DIMX.

6.2.24.2 **Drivers** – The sixteen 74H01 2-input positive NAND gates are used as open-collector drivers to drive the outputs from the BD register to the Central Processor Unit, via the BUS INTD lines. The BUS INTD lines are a fast internal bus also used by the Central Processor and solid state memory.

6.2.25 FXPL

This print shows the six-stage step counter, which consists of a four-stage 74191 binary counter and two 74S112 J-K edge-triggered flip-flops. The S-type flip-flops are used for bits 0 and 1 to ensure sufficiently fast set-up time on the input gates to the SCZ flip-flop. The 1's complement of the number of shifts to be performed is loaded into the step counter from EALU bits 5 through 0. This occurs during TP4. The step counter is used as an up-counter to count the number of shifts used in normalizing, arithmetic operations, or in the aligning of exponents.

The step counter is preset to the 1's complement of the number of shifts required. Termination of the operation is detected when the step counter sequences to all 1s. When this occurs, SCZ goes to a 1. Note that all inputs to the SCZ flip-flop are 1s, except for SC00 which is a 0. The next increment of the step counter causes the count to go to all 1s and SCZ to be set. Also note that SC EQ XX1111 is true when all four bits (bits 2, 3, 4, and 5) are 1s.

If in maintenance mode, the direct clear to the SC LOADED flip-flop goes high, allowing this flip-flop to be set by a LD SC maintenance instruction. When the flip-flop is set, subsequent loads of the step counter by the microprogram will be inhibited. When the step counter increments to all 1s or when maintenance mode is disabled, SC LOADED is reset allowing load pulses to occur.

CHAPTER 7
MAINTENANCE

7.1 INTRODUCTION

This chapter describes some of the maintenance techniques and tools available for maintenance of the FP11. A description of the FMAA Maintenance Module, display features, maintenance instructions, and diagnostic programming is also provided.

7.2 MAINTENANCE MODULE

The maintenance module consists of an indicator switch board (W131) and a driver board (W130 or W133) mounted piggy back in slot E1 of the KB11-A mainframe. The following floating-point signals may be displayed on the indicator board:

a. TPH
b. T1
c. T2
d. T3
e. T4
f. FP WAIT
g. FP ATTN
h. FP REQ
i. FP SYNC
j. Four floating-point condition codes (FZ, FC, FV, FN)
k. Two lights on the indicator board are unused but their pins are available on the back plate in order to allow the maintenance engineer to look at signals he may be interested in. The two available pins are E01 F2 and E01 H2. A high signal (+3V) is needed to turn on the light.

The following CPU signals are also displayed:

a. BUST
b. MEM
c. REF REQ1
d. REF REQ2
e. CPFC1/FPEC1
f. BBSY
g. MSYN
h. SSYN

(continued on next page)

i. CNTL OK

j. AERF

k. PAR ERR

l. SERF

m. T5

The switches on the maintenance module are:

S4 – MAINT STPR switch
S3 – crystal clock/RC clock

S2	S1	
0	0	Normal operation
1	0	Single ROM cycle
0	1	Microbreak stop
1	1	Single time pulse

S3 is placed into the RC clock position where the clock period can be varied for maintenance purposes. It is usually placed in the crystal position for normal operation.

NOTE
During maintenance, when any of the floating-point modules are inserted in extender boards, the clock must be in the RC position and set to more than 50 ns per clock period. It is also recommended that multilayer extender boards be used.

S4 is a MAINT STPR switch that allows the function selected by the combination of switches S1 and S2 to be performed. For example, if S2 is on and S1 is off, a single ROM cycle will occur each time the MAINT STPR stepper (S4) is depressed. The cycle will stop between TS2 and TS3. This feature can be used where maintenance personnel suspect a specified instruction is not sequencing through the proper branches. Maintenance personnel can operate in single ROM cycle mode and compare the ROM address on the console to the ROM address on the flow diagram to ensure that the proper branches are being taken. If S2 is off and S1 is one and the MAINT STPR is depressed, the FP11 will stop between TS2 and TS3 when a match occurs between the CRAR (control ROM address register) and the microbreak register. If the MAINT STPR switch is depressed again, the machine recycles until a second micromatch occurs at the same ROM address. This microbreak register is loaded by the by the LDUB instruction and provides maintenance personnel with a convenient means of sequencing to a desired state without manually depressing the single time stepper for each state sequenced through.

If S2 and S1 are both on, a clock transition occurs each time the MAINT STPR stepper is depressed. This allows the FP11 to be stopped with the clock pulse high or low in order to examine gate conditions in the logic. A second feature is that if the CPU could not cycle on the instruction, the operator could single clock up to the point of failure to see if the data paths are set up properly. Note that both the crystal and RC clock can be controlled by switches S4, S2, and S1.

7.2.1 Time Margining Using Maintenance Module

The timing of the RC clock can be varied using the maintenance module with S4 in the RC position, by adjusting potentiometer R32 on the M8114 module. The limits are from 45 ns minimum to 500 ns maximum.

The time margins should be checked periodically to locate any potential problems due to increase in propagation delays or flip-flop switching times.

7.3 SPECIAL MAINTENANCE INSTRUCTIONS

A set of five maintenance instructions are available to assist maintenance personnel. These instructions are described in the following paragraphs.

7.3.1 LDUB – Load Microbreak Register (170003)

This instruction causes the lower eight bits of general register 3 in the CPU to be loaded into the microbreak register. LDUB can be used for the functions described in the following paragraphs, depending on the FMM bit (bit 4) in the program status word (FPS).

NOTE

The FMM bit in the status word is used to enable special maintenance logic. In order to set this bit, the CPU must be in KERNEL mode.

With the FMM bit set, the microprogram will be aborted through the trap routine ROM address to the Ready state after the state specified by the address (next sequential ROM state) in the microbreak register is detected. If the Interrupt Enable bit (bit 14) of the floating-point processor status word is set, the CPU will trap to location 244. An exception code of 16 will be stored in the FEC (floating exception code) register. The contents of the FEC register can be transferred to the CPU by the STST (store status) instruction. A second function, available as a result of the LDUB instruction, is that the maintenance personnel can use the address match as a scope sync independent of the FMM bit. When the ROM address matches the contents of the microbreak register, the UMATCH flip-flop is set at the leading edge of TS1. The set output of this flip-flop (pin DK1 of slot 4 in the FXP module) is used as a scope sync to allow visual observation of events that occur during a particular ROM state. UMATCH is cleared at the trailing edge of TS4, which provides maintenance personnel with a sync signal that occurs at the beginning of a specified ROM state and ends at the beginning of the next ROM state.

7.3.2 LDSC – Load Step Counter (170004)

This maintenance instruction loads the 1's complement of the least significant six bits of general register 4 into the step counter. LDSC sets the SC LOADED flip-flop, provided FMM (bit 4) of the processor status word is set (CPU must be in KERNEL mode to set FMM), which inhibits the ROM from loading the step counter. When the step counter is incremented to all 1s, the SC LOADED flip-flop is cleared. As a result of this instruction, maintenance personnel can set up the step counter to do a specified number of steps in a multiply or divide routine and can stop where desired to examine the contents of the registers.

7.3.3 STA0 – Store AR In AC0 (170005)

This instruction transfers the contents of the AR to AC0, as described below:

$$\text{AR}\langle 57:35\rangle \rightarrow \text{AC0}\langle 57:35\rangle \text{ if FD} = 0$$

$$\text{AR}\langle 57:3\rangle \rightarrow \text{AC0}\langle 57:3\rangle \text{ if FD} = 1$$

7.3.4 STQ0 – Store QR In AC0 (170007)

This instruction transfers the contents of the QR to AC0, as described below:

QR ⟨57:35⟩→ AC0 ⟨57:35⟩ if FD = 0

QR ⟨57:3⟩→ AC0 ⟨57:3⟩ if FD = 1

NOTE

The STA0 and STQ0 instructions are used to store the contents of the AR and QR (internal registers) in an AC. Since the contents of the AC can be transferred to memory, this provides maintenance personnel with a means of checking the contents of the AR and QR registers.

7.3.5 MRS – Maintenance Right Shift (170006)

The Maintenance Right Shift instruction shifts the AR or QR one bit position to the right. This instruction is used in conjunction with the STA0 instruction to allow AR59 and AR58 to be examined. Two MRS instructions are necessary to transfer AR59 to AR57 and AR58 to AR56. The MRS instruction is also used in conjunction with the STQ0 instruction to allow bits QR59 and QR58 to be examined. Two MRS instructions are necessary in order to shift QR59 to QR57 and QR58 to QR56. AR59 and AR58 as well as QR59 and QR58 represent the sign bit and hidden bit, respectively. These bits are not transferred between the CPU and the FP11 but are used in data calculations by the Floating-Point Unit. Therefore, in order to examine the state of these two bits, the use of the MRS instruction is required.

7.3.6 Maintenance Instruction Programming Example

The following program demonstrates the use of the FP11 maintenance instructions. This program is a multiplication example, whereby the contents of the AR and QR are typed out with each incrementation of the step counter from 1 through 71. Note that the MRS instruction is used in order to get AR and QR bits 59 and 58 into general register R5 for the typeout in each pass through the loop.

The fractional part of the multiplicand which is 1/2 or 0.1 is stored in the BR and the fractional part of the multiplier which consists of alternating 1s and 0s is stored in the QR. The multiplier has an exponent of 200 and the multiplicand has an exponent of 204. The sign bit is a 0 and the hidden bit is a 1. The result of each step of the multiplication is stored in the AR. The typeout of the listing after each step of the multiplication is shown following the example.

The contents of the AR and QR are typed out 57 times. On the 58th typeout, the step counter is not set and this last typeout represents the final product.

```
001000  012706  START:    MOV    #600,%6       ;SET UP STACK POINTER AT 600
001002  000600
001004  170127            LDFPS  #40220        ;DISABLE INTERRUPTS; SET DOUBLE AND MAINT. MODE
001006  040220
001010  172667            LDD    MLYR,AC2      ;LOAD MULTIPLIER IN AC2
001012  000204
001014  012703            MOV    #230,%3       ;SET REG. 3 to 230
001016  000230
001020  170003            LDUB                 ;SET MBR TO 230
001022  005004            CLR    %4            ;CLEAR COUNTER
001024  005204  NXTMUL:   INC    %4            ;INCREMENT COUNTER
001026  170004            LDSC                 ;LOAD 1'S COMPLEMENT OF R4 INTO SC
001030  012705  LSTMUL:   MOV    #QR+10,%5     ;SET UP REG. 5 TO STORAGE TABLE
001032  001166
001034  172567            LDD    MCND,AC1      ;LOAD MULTIPLICAND INTO AC1
001036  000150
001040  171102            MULD   AC2,AC1       ;DO PARTIAL MULTIPLY
001042  170007            STQ0                 ;TRANSFER QR TO AC0
001044  174045            STD    AC0,-(5)      ;STORE QR IN TABLE
001046  042715            BIC    #177600,@5    ;CLEAR SIGN AND EXPONENT
001050  177600
001052  170005            STA0                 ;STORE AR IN AC0
001054  174045            STD    AC0,-(5)      ;STORE AR IN TABLE
001056  042715            BIC    #177600,@5    ;CLEAR SIGN AND EXPONENT
001060  177600
001062  170006            MRS                  ;SHIFT AR AND QR RIGHT ONE PLACE
001064  170006            MRS                  ;SHIFT AR AND QR RIGHT ONE PLACE
001066  170007            STQ0                 ;TRANSFER QR TO AC0
001070  174067            STD    AC0,TEMP      ;MOVE AC0 TO TEMP
001072  000134
001074  016703            MOV    TEMP,%3       ;MOVE MOST SIGNIFICANT 7 BITS OF QR TO R3
001076  000130
001100  042703            BIC    #177600,%3    ;CLEAR SIGN AND EXPONENT
001102  177600
001104  006303            ASL    %3            ;SHIFT MSB OF QR ONE PLACE LEFT
001106  006303            ASL    %3            ;SHIFT MSB OF QR ONE PLACE LEFT
001110  050365            BIS    %3,10(5)      ;SET QR59 AND QR58 IN TABLE
001112  000010
001114  170005            STA0                 ;STORE AR IN AC0
001116  174067            STD    AC0,TEMP      ;MOVE AC0 TO TEMP
001120  000106
001122  016703            MOV    TEMP,%3       ;MOVE MOST SIGNIFICANT 7 BITS OF AR TO R3
001124  000102
001126  042703            BIC    #177600,%3    ;CLEAR SIGN AND EXPONENT
001130  177600
001132  006303            ASL    %3            ;SHIFT MSB OF AR ONE PLACE LEFT
001134  006303            ASL    %3            ;SHIFT MSB OF AR ONE PLACE LEFT
001136  050315            BIS    %3,@5         ;SET AR59 AND AR58 IN TABLE
001140  004567            JSR    %5,PRINT      ;PRINT AR AND QR
001142  000234
001144  000410            BR     .+22          ;BRANCH OVER ARGUMENTS
001146  000000  AR:       .FLT4  0             ;AR STORED IN THESE FOUR LOCATIONS
001150  000000
001152  000000
001154  000000
001156  000000  QR:       .FLT4  0             ;QR STORED IN THESE FOUR LOCATIONS
```

```
001160  000000
001162  000000
001164  000000
001166  020427              CMP     %4,#71      ;HAVE 71 PASSES BEEN DONE
001170  000071
001172  100714              BMI     NXTMUL      ;NO—DO NEXT PASS
001174  001402              BEQ     LSTPAS      ;YES—DO LAST PASS
001176  000167              JMP     START       ;THIS MULTIPLY COMPLETE—DO NEXT ONE
001200  177576
001202  005204  LSTPAS:     INC     %4          ;INDICATE 72ND PASS
001204  000167              JMP     LSTMUL      ;DO LAST PASS WITHOUT LOADING SC.
001206  177620
001210  040052  MCND:       .WORD   040052
001212  125252              .WORD   125252
001214  125252              .WORD   125252
001216  125252              .WORD   125252
001220  040000  MYLAR:      .WORD   040000
001222  000000              .WORD   000000
001224  000000              .WORD   000000
001226  000000              .WORD   000000
001230  000000  TEMP:       .FLT4   0
001232  000000
001234  000000
001236  000000
        000001              .END
```

TYPEOUT OF QR AND AR

Step	AR	QR
1	0000000000000000000	1252525252525252525
2	0000000000000000000	0525252525252525252
3	1000000000000000000	0252525252525252525
4	0400000000000000000	0125252525252525252
5	1200000000000000000	0052525252525252525
6	0500000000000000000	0025252525252525252
7	1240000000000000000	0012525252525252525
8	0520000000000000000	0005252525252525252
9	1250000000000000000	0002525252525252525
10	0524000000000000000	0001252525252525252
11	1252000000000000000	0000525252525252525
12	0525000000000000000	0000252525252525252
13	1252400000000000000	0000125252525252525
14	0525200000000000000	0000052525252525252
15	1252500000000000000	0000025252525252525
16	0525240000000000000	0000012525252525252
17	1252520000000000000	0000005252525252525
18	0525250000000000000	0000002525252525252
19	1252524000000000000	0000001252525252525
20	0525252000000000000	0000000525252525252
21	1252525000000000000	0000000252525252525
22	0525252400000000000	0000000125252525252
23	1252525200000000000	0000000052525252525
24	0525252500000000000	0000000025252525252
25	1252525240000000000	0000000012525252525
26	0525252520000000000	0000000005252525252
27	1252525250000000000	0000000002525252525
28	0525252524000000000	0000000001252525252
29	1252525252000000000	0000000000525252525
30	0525252525000000000	0000000000252525252
31	1252525252400000000	0000000000125252525
32	0525252525200000000	0000000000052525252
33	1252525252500000000	0000000000025252525
34	0525252525240000000	0000000000012525252
35	1252525252520000000	0000000000005252525
36	0525252525250000000	0000000000002525252
37	1252525252524000000	0000000000001252525
38	0525252525252000000	0000000000000525252
39	1252525252525000000	0000000000000252525
40	0525252525252400000	0000000000000125252
41	1252525252525200000	0000000000000052525
42	0525252525252500000	0000000000000025252
43	1252525252525240000	0000000000000012525
44	0525252525252520000	0000000000000005252
45	1252525252525250000	0000000000000002525
46	0525252525252524000	0000000000000001252
47	1252525252525252000	0000000000000000525
48	0525252525252525000	0000000000000000252
49	1252525252525252400	0000000000000000125
50	0525252525252525200	0000000000000000052
51	1252525252525252500	0000000000000000025
52	0525252525252525240	0000000000000000012
53	1252525252525252520	0000000000000000005
54	0525252525252525250	0000000000000000002
55	1252525252525252524	0000000000000000001
56	0525252525252525252	0000000000000000000
57	1252525252525252525	0000000000000000000
58	1252525252525252525	0000000000000000000
	1252525252525252525	0000000000000000000

7.4 CONSOLE DISPLAY FEATURES

The PDP-11/45 console can be used to display the floating-point ROM address and, under certain conditions, can display the contents of the EALU.

7.4.1 Display of ROM Address

The 16 DATA indicators on the console can be used to display the 8-bit FP11 ROM address and the 8-bit CPU ROM address. The FP11 ROM address is displayed on the left-most DATA indicators (bits 15–08) and the CPU ROM address is displayed on the right-most indicators (bits 07–00). The four-position data selector switch on the console must be set to the μ ADDR FPP/CPU position to display the ROM address.

NOTE

If the FMAA maintenance module is set up to do single ROM cycles or μ match, the FP11 ROM address displayed is the next ROM address; i.e., the address of the next ROM state to be cycled. The reason for this is that the ROM address changes at the end of time state 2 and a Pause or Wait state occurs between time state 2 and time state 3. If the FMAA maintenance module is set up to do single clock cycles during time states 1 and 2, the ROM address displayed is the current address, and for single clock cycles during time states 3 and 4, the ROM address displayed is the next address.

7.4.2 Display of EALU Contents

In certain ROM states of the CPU the contents of the EALU may be displayed on the lower 16 ADDRESS indicators (bits 15–00) on the PDP-11/45 console. These CPU ROM states are unique to F class instructions and are listed as follows:

ROM State	Octal Address
FOP.30	173
FOP.50	211
FOP.60	362
FOP.70	316
FOP.80	376
FOP.90	375
FOP.40	36
FSV.20	225

NOTE

The content of the EALU at any of these ROM states is dependent on the FP ROM state occurring at that time. Both the FP11 and the CPU should be set up for single-step operation using both the CPU and FP11 Maintenance Boards to see meaningful data in these ROM states.

The eight-position address selector switch on the console must be set to CONS.PHYS, or PROG.PHYS.

7.5 MAINTENANCE PROGRAMMING

This section describes some simple programs that can be performed by maintenance personnel to ascertain if certain areas of the logic are working properly.

PROGRAM 1

```
        000000          AC0=%0
        177570          SWR=177570
        003000          .=3000
003000  172400                  LDF AC0,AC0     ;FCLASS AND MODE 0
003002  000000                  HALT
```

PROGRAM 2

```
        003010          .=3010
003010  172400                  LDF AC0,AC0     ;FCLASS AND MODE 0
003012  000776                  BR .-2          ;LOOP ON INSTRUCTION
003014  000000                  HALT            ;SHOULD NEVER HALT
```

PROGRAM 3

```
        003020          .=3020
003020  016701  174544          MOV SWR,%1      ;SWITCHES TO CP REGISTER 1
003024  170101                  LDFPS %1        ;CP REGISTER 1 TO FPS
003026  170200                  STFPS %0        ;FPS TO CP REGISTER 0
003030  020100                  CMP %1,%0       ;DOES R0=R1?
003032  001400                  BEQ .+2         ;TEST
003034  000000                  HALT            ;IS NOT THE SAME
003036  000137  003020          JMP @#3020      ;RESTART PROGRAM
```

PROGRAM 4

```
        003050          .=3050
003050  170011                SETD              ;PUT FPU IN DOUBLE PRECISION
003052  172437  004000        LDD @#4000,AC0    ;LOAD AC0 FROM LOCATION 4000
003056  174037  004010        STD AC0,@#4010    ;STORE AC0 IN LOCATION 4010
003062  000000                HALT              ;EXAMINE DATA TO SEE IF SAME
        004000          .=4000
004000  011111                11111             ;FIRST WORD OF LOAD
004002  022222                22222             ;SECOND WORD
004004  033333                33333             ;THIRD WORD
004006  044444                44444             ;FOURTH WORD
        000001          .END
```

The following general assumptions can be made about each of the programs.

1. In program number 1 it can be assumed that control can be transferred between the FP11 and the CPU and also that the FP11 can cycle on the LDF instruction.

2. In program number 2 it can be assumed that the floating-point instruction can be run dynamically and that the interface signals are being properly generated. The program is useful for scoping control signals between the FP11 and the CPU. In addition, it is probable that both the FP11 and the CPU control ROMs will pause in the Ready state and the LDF instruction can be looped on.

3. In program number 3 it can be assumed that single-operand fetches work and that the data is being transferred properly. The program allows the data placed in the switch register to be transferred from the CPU to the FP11, back to the CPU, and then compared.

NOTE
Bits 12 and 13 of the floating-point status word are unused and should be set to 0 on the switch register.

4. In the program number 4 two floating-point words from memory are transferred to the FP11 and then transferred back to memory. It can generally be assumed that the QR, BR, FALU, ACMX, and scratch pad are operating correctly. For single-precision mode, the same program can be utilized if the SETD instruction is replaced with the SETF instruction.

NOTE
Refer to Chapter 4 of the *PDP-11/45 System Maintenance Manual* for information on diagnostic programming.

APPENDIX A
SIGNAL GLOSSARY

Signal Mnemonic	Logic Print	Function
ACC ⟨2:0⟩ (AC Control)	FRME	Three bits used to select a 16 or 32 byte location within the accumulator.
ACF ⟨2:0⟩ (AC Field)	FRME	Three ROM bits used to specify accumulator address.
ACMX ⟨07:00⟩ ACMX ⟨15:08⟩ ACMX ⟨23:16⟩ ACMX ⟨31:24⟩	FRLA FRLB FRLC FRLD	Outputs of ACMX, which are applied to scratch pad accumulator.
ACMXC1, ACMXC0 (ACMX Control)	FRME	ROM bits used to select inputs to ACMX.
ACRE (AC Read)	FRME	ROM bit used to specify AC Read on a 1 and AC Write on a 0.
ADDR INCR (Address Increment)	FRMF	ROM bit that causes the BA register to be incremented twice.
AD2, AD1	FXPE	A 2-bit constant field that is decoded to a value of 0, 2, 4, or 8 and indicates how much the program counter is to be incremented. This is based on the number of memory cycles required to represent the operand, as shown:

AD2	AD1	No. of Operands
0	0	8
0	1	4
1	0	2
1	1	0

Signal Mnemonic	Logic Print	Function
ALUCIN	FRMH	One of the six inputs to the ALU and produces a carry under certain conditions (see Table 6-3).
ALUC ⟨3:0⟩ (ALU Control)	FRME	Four ROM bits used to select function performed by ALU.
ALUM	FRMH	Mode bit which, when set, indicates a logical function is to be performed by the ALU and, when reset, indicates an arithmetic function is to be performed.

(continued on next page)

Signal Mnemonic	Logic Print	Function
ALUS ⟨3:0⟩	FRMH	Select lines which specify the function to be performed in the ALU. (See Table 6-3).
ARC1, ARC0 (AR Control)	FRME	ROM bits used to control the shifting or loading of the AR.
AR ⟨59:52⟩	FRHD	Outputs from AR which are directly applied to the FALU.
AR ⟨51:44⟩	FRHC	
AR ⟨43:36⟩	FRHB	
AR ⟨35:32⟩	FRLK	
AR ⟨31:24⟩	FRLJ	
AR ⟨23:16⟩	FRLH	
AR ⟨15:08⟩	FRLF	
AR ⟨07:00⟩	FRLE	
AR NORM (1) H	FRHF	If the AR is normalized, this flip-flop is set when the AR is clocked. If the AR is unnormalized, the flip-flop remains reset. The flip-flop is held cleared except for divide and left shifting of the AR.
ARS1, ARS0	FRHE	AR select signals used to specify function performed by AR. These signals are derived from AR control bits 26 and 25 from the ROM or from ROM CSB bits 29 through 27 during an add or subtract operation.
ATTENTION (1) H	FRMJ	This signal is set by FP ATTN and is used to force the FP11 out of the Wait state by clearing FRHH PAUSE.
BA ⟨15:00⟩	FXPC	Represents the 16-bit outputs of the BA register.
BAC (BA Control)	FRMF	ROM bit used to load the BA register.
BACMXC1, BACMXC0	FRMH	Buffered ACMX control lines used to specify inputs to ACMX.
BB1Z	FXPF	When this signal is a 1, it indicates that the exponent (bits 15 through 8) is 0 and that the exponent has not overflowed.
BD ⟨15:00⟩	FXPC	Represents the 16-bit outputs of the BD register.
BDC (BD Control)	FRMF	ROM bit used to load the BD register.
BMX ⟨15:00⟩	FRLN	Outputs of the BMX which represent data from one of four input sources.
BMX ⟨15:00⟩ H	FXPH	These are inputs to the condition codes and convey information regarding the state of the exponent (overflow, underflow, negative, etc.).
BMXC1, BMXC0 (BMX Control)	FRME, FRMF	ROM bits used to select inputs to BMX.

(continued on next page)

Signal Mnemonic	Logic Print	Function
BN	FXPF	This signal is a 1 when bit 15 is a 1 and indicates that the exponent has underflowed.
BRC0 (0) H (BR Control)	FRME	ROM bit used to load the BR.
BRC1 (0) H (BR Control)	FRME	ROM bit used to load the BR.
BR ⟨59:36⟩	FRHA	Outputs from BR which are applied to FALU.
BR ⟨35:24⟩	FRLM	Outputs of the BR register which are applied to the FALU. Bit 35 of the BR is applied to FALU through the FMX, which is used for rounding purposes.
BR ⟨23:00⟩	FRLL	Outputs of the BR register which are applied to the FALU. Bits 3, and 19 of the BR are applied to the FALU through the FMX, which is used for rounding purposes.
BUS INTD ⟨15:00⟩		Open collector bus lines that run between the BD register and the 11/45 to provide for flow of data to the 11/45. The FP11 can be disconnected from the bus by the 11/45 gating signal TMCF FP READ.
BZ	FXPF	This signal is a 1 when bits 15 through 0 are all 0s.
CLK AR A, CLK AR B	FRHE	CLK AR A is used to clock the AR during load and shift operations. CLK AR B is a copy of this signal necessary for additional drive requirements.
CLK BA	FXPC	A signal used to load the BA register at the end of TS4 if the BA control (bit 48) from the CROM is a 0.
CLK BD	FXPC	A signal used to load the BD register at the end of TS4 if the BD control (bit 49) from the CROM is a 0.
CLK BR A, CLK BR B	FRMH	Signals which clock the BR during the latter half of TS4. Two signals available to satisfy drive requirements.
CLR BR A, CLR BR B	FRMH	Signals used to clear the BR during TS2. Two signals available to satisfy drive requirements.
CLK IR	FXPD	A signal used to load the FIR at the beginning of TP3 if the FIR control (bit 51 of the CROM) is a 0.
CLK QR A, CLK QR B	FRHE	CLK QR A is used to clock the QR during load and shift operations. CLK QR B is a copy of this signal necessary for additional drive requirements.
CLK RB A, CLK RB B	FRME	This signal clocks the ROM output buffer on the trailing edge of TS4. Two sources available for increased drive capability.

(continued on next page)

Signal Mnemonic	Logic Print	Function
CLK SC	FRHF	A signal used to clock the step counter in multiply or divide. The signal is inhibited during the actual arithmetic operation or when the AR is normalized.
CLOCK A,B,C,D	FRHJ	Four clock lines from the clock driver used to supply clock signals to the FP11.
CLOCK A,B,C, RTN	FRHJ	Return lines for the four clock driver lines which supply signals to the FP11.
CLR QR A, CLR QR B	FRHF	Signals used to clear the QR in time state 2 with ROM control bit QRC2 on a 0.
CLR SYNC	FRMJ	ROM derived control signal occurring during the latter half of TS4 which causes the SYNC flip-flop to be cleared if a CONV SPECIAL-type instruction is issued.
CNST BIT 00, AND 15	FXPA	These are signals which, when enabled, will force a bit into bit 00, or bit 15 of the EMX, respectively.
CNST BIT 03	FXPA	A constant bit used in bit location 3 of the constant word which is fed into the B input of EALU via the EMX.
CNSTF4–CNSTF0	FRMF	Five ROM bits used for various constants employed in the FP11. When accompanied by RDFN CNSTF (1) these bits are used for control functions.
CONTROL SEL2–CONTROL SEL0	FRMF	Three ROM bits used for encoding up to seven additional functions, such as LOAD FPSC, LOAD UBC, REG WRITE, etc.
CONV SP	FXPE	When a store exponent, store floating to integer, or store floating to double instruction is issued, CONV SP is generated to initiate a conversion routine based on the instruction.
COUT10 COUT09 COUT06,07,08	FRHD FRHC FRHB	Carry output to succeeding FALU chip for carry propagation.
CRAR 1 D L	FRMB	Input to CRAR 1 which represents the next state of the address register bit.
CRAR 0 D L	FRMB	Input to CRAR 0 which represents the next state of the address register bit.
CRAR ⟨07:04⟩	FRMA	Bits 7 through 4 of the 8-bit control ROM address register. These bits are generated by the control ROM. CRAR bits 4 and 5 can be modified by some branch conditions which have been satisfied.
CRAR ⟨03:00⟩	FRMB	Bits 3 through 0 of the 8-bit control ROM address register. These bits are generated from the control ROM and can be modified by branch conditions which have been satisfied.

(continued on next page)

Signal Mnemonic	Logic Print	Function
CSB ⟨2:0⟩ (Call Subroutine bits)	FRME	Three ROM bits used to specify functions to be performed during an arithmetic subroutine.
D ⟨63:32⟩ D ⟨31:00⟩	FRMC FRMD	32 control ROM outputs which are applied to the ROM output buffer. Bits D07 through D00 are not applied to the output buffer but are directly applied to the control ROM address register.
DISABLE SC (Disable Step Counter)	FRMF	Second level decoding of ROM bits 61 through 59 to yield 4_8.
DISABLE SYNC	FRMF	Second level decoding of ROM bits 61 through 59 to yield 3_8
DIV SUB	FRMH	Indicates that a subtract function is to be performed by the ALU in a divide subroutine.
DROM ⟨A7:A0⟩	FXPD	Decoded instruction ROM outputs used with DROM B7 – DROM B0 to specify branching conditions necessary to perform each instruction.
DROM ⟨B7:B0⟩	FXPD	Decoded instruction ROM outputs used with DROM A7 – DROM A0 to specify the branching conditions necessary to perform each instruction.
DSBL1, DSBL0	FRMF	ROM bits used to clear FP REQ and/or disable BRQ monitor.
EALU ⟨15:00⟩	FXPA,FXPB	16 outputs from EALU whose content depends on inputs supplied and function specified to be performed in EALU.
EMXC0	FXPA	One of the select lines to EMX. Used in conjunction with EMXC1 (1) H to select one of four inputs.
EMXC1, EMXC0 (EMX Control)	FRMF	ROM bits used to select inputs to EMX.
EN CNST 7x	FXPA	This signal is generated in order to generate an octal digit of $7x_8$ for constants of 70, 71, 74, and 75 specified by constant fields of 32, 33, 36, and 37, respectively.
ENABLE FM0	FRMJ	This signal enables the floating minus zero interrupt if the gating on sheet FRMA detects a minus zero condition. This is detected by a negative sign and an exponent of all 0s.
ENABLE FV	FRLP	Indicates that a positive exponent has overflowed, setting bit 8 out of EALU. Occurs only when CROM enables Floating Condition Code output.
ENAB QRS0L	FRHE	ROM derived signal indicating that a right shift should be performed in the QR, with 0s being shifted into the MSB.

(continued on next page)

Signal Mnemonic	Logic Print	Function
EXP EQ 0 L	FRLM	When these signals are enabled, it indicates that the exponent is equal to 0. SCR OUT 30–23 can represent the eight bits of exponent.
EXP EQ 0 H	FRLM	
FALU59–FALU52	FRND	Outputs from FALU dependent on inputs from AR and BR and function specified in FALU.
FALU51–FALU44	FRNC	
FALU43–FALU36	FRNB	
FALU35–FALU32	FRLK	
FALU31–FALU24	FRLJ	
FALU23–FALU16	FRLH	
FALU15–FALU08	FRLF	
FALU07–FALU00	FRLE	
FC	FRLP	Bit 0 of the FP11 program status register which, when set, indicates that the integer, obtained from conversion of a floating-point number, is too large to be stored in the specified register. This is a result of the STCXJ instruction. FC also indicates that absolute value of floating-point result was larger than largest integer which can be represented by 56 bits (D) or 24 bits (F).
FCC1, FCC0 (Floating Condition Codes)	FRMF	ROM bits used to determine inputs to floating condition codes.
FC INT	FRLP	The AND of FIC (1) H and FC (1) H. If the programmer wishes to trap only on the setting of a condition code, he sets FIC (1) H. When the condition code is set (designated by FC (1)), FC INT is generated which causes an FP TRAP if the interrupt enable bit is set.
FCLD EN H	FXPE	Used to signify that the floating condition codes must be loaded into the CPU.
FD	FXPH	The floating double flip-flop, when set, indicates double-precision floating point and when reset, indicates single-precision floating-point.
FER (Floating Error)	FRLP	Bit 15 of the FP11 program status register which is set by CROM when FP11 sequences into error state.
FIC (Floating Interrupt on Conversion Error)	FRLP	Bit 8 of the FP program status register which, when set, will cause an interrupt if the FC bit (indicating a conversion error) is set.
FID (Floating Interrupt Disable)	FRLP	Bit 14 of the FP11 program status register which, when set, allows all interrupts to be disabled.
FIR11–FIR00	FXPD	Represents the 12-bits of the instruction word stored in the floating instruction register.
FIRC (FIR Control)	FRMF	ROM bit used to load the floating instruction register; also used to indicate the Ready state of the FP11, since this is the state during which the IR is loaded.

(continued on next page)

Signal Mnemonic	Logic Print	Function
FIRD6–FIRD0	FXPE	The outputs of the two instruction decoder ROMs, when collector ORed, generate FIRD6 through FIRD0. These are branch conditions which are supplied to the CROM MPX to produce the proper sequence for each instruction.
FIU (Floating Interrupt on Underflow)	FRLP	Bit 10 of the FP11 program status register which, when set, allows a floating underflow to cause an interrupt.
FIUV (Floating Interrupt on Undefined Variable)	FRLP	Bit 11 of the FP11 program status register which, when set, allows a minus zero from memory to cause an interrupt.
FIV (Floating Interrupt on Overflow)	FRLP	Bit 9 of the FP11 program status register which, when set, allows an overflow to produce an interrupt condition.
FL	FXPH	The long-integer flip-flop, when set, indicates long-integer format and, when reset, indicates short-integer format.
FMM	FRLP	Floating Maintenance Mode – bit 4 of the FP11 program status register which is used to set FP11 into maintenance configuration. This can only be done while 11/45 is in Kernel mode.
FM0F (1) H	FRMA	This flip-flop is set on a floating minus zero and causes the FP11 to trap to 20 when the 8 bits of exponent are all 0s and the sign bit is negative.
FMXC1, FMXC0	FRME	ROM bits used for rounding or incrementing operand in AR.
FN (Floating Negative)	FRLP	Bit 3 of the FP11 program status register which is the FP11 version of the N condition code.
FORCE ADD	FRMH	Combination of ALUC (ALU Control) signals which force ALUS3–ALUS0 (ALU select) signals to specify an add operation.
FORCE SUB	FRMH	Combination of ALUC (ALU Control) signals which force ALUS3–ALUS0 (ALU select) signals to specify a subtract operation.
FP ACKN WAIT	FRHH	Forces the pause flip-flop to turn on causing the FP11 to sequence to Wait state where it produces FP TRAP and waits for FP ACKN.
FP ADR INC	FRMJ	This signal causes the address of the operand to be incremented by 2.
FP ATTN WAIT	FRHH	ROM control signal which forces FP11 into Wait state to wait for FP ATTN.
~FPC1 L	FRMJ	Indicates a DATO transfer when high and a DATI transfer when low.

(continued on next page)

Signal Mnemonic	Logic Print	Function
FP REQ (1) H	FRMJ	Signal used in conjunction with FP SYNC to indicate that more data words are desired. When FP SYNC is returned to the CPU in the absence of FP REQ, the memory cycles are terminated.
FP SYNC	FRMJ	A signal sent to the CPU indicating that data has been accepted or that the FP11 is ready to send or receive data.
FP SYNC (1) H	FRMJ	This signal enables FP SYNC to be sent to the CPU in TS2 of the next ROM state.
FP TRAP	FRHH	Indicates that the FP11 is issuing a trap command.
FRAC MUL	FRHE	This signal is derived from ROM bits 29 through 27 (CSB 2 through 0). When these bits are all 0s, a multiply operation is indicated.
FT (Floating Truncate)	FRLP	Bit 5 of the FP11 program status register, which, when set, causes the result of any floating-point operation to be truncated rather than rounded.
FV (Floating Overflow)	FRLP	Bit 1 of the FP11 program status register which is the FP11 version of the V condition code.
FV INT (Floating Overflow Interrupt)	FRLP	The AND of FIV (1) H and FV (1) H. If the programmer wishes to trap only on the setting of the overflow bit, he sets FIV (1) H. When the overflow bit is set (designated by FV (1)), FV INT is generated which causes an FP TRAP if the interrupt enable bit is set.
FZ (Floating Zero)	FRLP	Bit 2 of the FP11 program status register which is the FP11 version of the Z condition code.
GATE BD B1, GATE BD B2	FXPK	Allows data to be gated from the BD to the 11/45–two sections available for adequate drive.
GEN 12, PROP 12	FRHC	Outputs from first level carry look-ahead which are used as inputs to second level carry look-ahead.
GO TO READY	FRMB	Causes FP11 to trap to ready if the μJMP CROM bit is on and next address has 2 LSBs of 1.
ICLR (1) H	FRMJ	This signal is an enable to the INITF flip-flop which allows the INITF flip-flop to be set synchronously with the FP11. ICLR is set by INIT or INTR CLR where INTR CLR is the CPU signal.
ILL OP CODE	FXPF	Indicates an illegal op code in that bits 15 through 12 of the instruction are not all 1s, yielding the 17_8 op code assigned to FPU.
IMMEDIATE	FXPE	Indicates register R7 and modes 1, 2, or 4.
INIT	FRHH	A signal asserted by the processor when the start key is depressed, when a reset instruction is executed, or when the power fail sequence is initiated.

(continued on next page)

Signal Mnemonic	Logic Print	Function
INITF B	FRMA	Output from INIT flip-flop which is at TP2 by ICLR (1), a function of INIT. INTF B is used to direct clear the CRAR.
L2 COUT10, 11, 12 L	FRLK	Carry outputs from the second level carry look-ahead circuit. These outputs represent carry between the first level carry look-ahead circuits.
LD QR	FRHF	A signal developed from ROM bits 22 and 21 to load the QR.
LDQR ⟨59:35⟩	FRHE	ROM derived signal indicating that the upper half of the QR is to be loaded.
LDQR ⟨34:00⟩	FRHE	ROM derived signal indicating that the lower half of the QR is to be loaded.
LDSC	FXPL	A signal occurring at TP4 which is used to load the six-stage step counter.
LOAD FPSC (Load floating-point status control)	FRMF	Second level decoding of ROM bits 61 through 59 to yield 0_8.
LOAD UBC (Load µbreak control)	FRMF	Second level decoding of ROM bits 61 through 59 to yield 1_8.
LSQR IN H	FRHF	This signal, when high, causes a 1 to be shifted into QR00 (double precision) or QR31 (single precision) and when low causes a 0 to be shifted in.
LSQR31 IN H	FRLM	Used in single-precision floating point and represents the bit that is shifted into bit position 31 of the QR during a left shift.
LSQR00 IN H	FRHF	This signal is generated during divide with double-precision floating-point format specified and represents the bit shifted into the LSB position of the QR.
M0 OR 1	FXPE	A mode 0 or mode 1 instruction has been decoded.
M PAUSE (1) H	FRHH	A flip-flop used during maintenance which stops the state counter between time state 2 and time state 3 and allows the Wait state to be turned on.
MPX DATA ⟨15:00⟩	FXPK	A multiplexer which selects the contents of the BR register or the BAMX which contains the value of the program counter when the FP instruction was fetched. This MPX allows both the floating instruction and floating return address to be transferred simultaneously to the FP11.
MR1 (1), MR0 (1)	FRHE	These are two flip-flops used during the multiply subroutine to store the two LSBs of the QR for increased speed.

(continued on next page)

Signal Mnemonic	Logic Print	Function
MSWITCH CNTU (1) H	FRHH	A maintenance signal which allows manually switching out of the Wait state by clearing MPAUSE flip-flop.
MUL ADD	FRMH	Indicates an add operation in the multiply subroutine.
MUL ARITH (1) H	FRHE	This flip-flop is set under following conditions: **MR1 MR0 STRG1** 0 1 [1] 1 0 [0] and indicates an arithmetic operation is to occur.
MUL DIV	FRMH	Indicates a multiply or divide subroutine has been selected by the ROM. When MUL DIV is true, ALUC signals are disabled.
MUL DIV DISABLE	FRHE	This signal disables the multiply or divide subroutine by inhibiting the clock pulses from clocking the QR or AR. This signal is generated when an arithmetic operation occurs (MUL ARITH (1)), when an operand is normalized, or when the step counter is fully incremented.
MUL DIV H	FRHE	This signal is derived from ROM bits 29 and 28 (CSB bits 2 and 1). When both bits are 0, a multiply or divide operation is indicated.
MUL DIV LSQR31 L	FRHF	This signal is generated during divide with single-precision floating-point format specified and represents the bit shifted into QR31.
MUL SUB	FRMH	Indicates that a subtract function is to be performed by the ALU in a multiply subroutine.
MUL SUB (1) H	FRHE	This flip-flop is set when a subtract operation is performed during a multiply subroutine and is initiated by a string of 1s.
PAUSE (1) H	FRHH	A flip-flop which stops the state counter between state 2 and 3 and allows the Wait state to be turned on.
P0, P1, P2 (1) H	FRHE	Pause flip-flops which provide a 200 ns delay to inhibit clocking the AR and QR during an add or subtract operation in an arithmetic subroutine.
QR0 S0, QR0 S1, QR1 S0, QR1 S1	FRHE	QR select signals used to specify function performed by QR. The signals are derived from QR control bits 22 and 21 from the ROM and are used in conjunction with ACC2 to determine what accumulator is loaded into the QR and whether it is loaded into the lower or upper half of the QR

(continued on next page)

Signal Mnemonic	Logic Print	Function
QR59 (1) H	FRHF	This is the QR sign bit flip-flop, and its value depends on whether the QR is being right shifted or left shifted.
QR ⟨58:35⟩ QR ⟨34:23⟩ QR ⟨22:00⟩	FRHA FRLM FRLL	Outputs from QR directly applied to BR.
QRC ⟨2:0⟩ (QR Control)	FRME	Three ROM bits used to specify if QR is to be shifted or loaded.
RC	FRHJ	An RC clock used to vary the frequency during maintenance mode.
RDFN CNSTF (Redefined Constant Field)	FRMF	ROM bit used to redefine constant field bits 57 through 53, so that they may be used for control purposes.
READY CLR	FRNJ	Occurs every time the IR is loaded and is used to ensure that certain flip-flops are initialized at the beginning of each instruction.
REG 6 or 7	FXPE	Indicates that register 6 or register 7 has been addressed.
REG WRITE (Register Write)	FRMF	Second level decoding of ROM bits 61 through 59 to yield 2_8.
RNG ROM 2, 1, 0	FXPF	Outputs of the range ROM used to determine the magnitude difference between the two exponents involved.
ROM + UBS	FRHJ	Signal which represents a CROM address comparison or single ROM cycle in maintenance mode. This signal sets MPAUSE at TP1 time to force the FP11 into the Wait state.
RR2, RR1, RR0 (1) H	FRHF	Three flip-flops used in division to speed up the add or subtract operations within the divide subroutine. They are high-speed duplications of AR59, AR58, and AR57 and can only be left shifted.
RS QR IN H	FRHF	Input to MSB of QR register during a right shift operation.
SC ⟨05:00⟩	FXPL	Outputs of the step counter which indicate number of shifts that have occurred during normalizing or the number of shifts that must occur during some arithmetic operation.
SCC (Step Counter Control)	FRMF	ROM bit used to load the step counter.
SC LOADED (0) H		Maintenance signal used to allow one load of the step counter and inhibits further loading until step counter overflow.

(continued on next page)

Signal Mnemonic	Logic Print	Function
SCR OUT ⟨31:24⟩	FRLD	Outputs from scratchpad accumulator which are applied to BMX or QR.
SCR OUT ⟨23:16⟩	FRLC	
SCR OUT ⟨15:08⟩	FRLB	
SCR OUT ⟨07:00⟩	FRLA	
SCR ADDRS ⟨2:0⟩	FRMH	Address lines which select one of eight accumulator scratchpads (AC0–AC7).
SCR WRITE 1, SCR WRITE 0	FRMH	A control signal used to command a write into the scratch pad accumulator. Two signals available to satisfy drive requirements.
SC EQ XX1111	FXPL	Indicates four outputs (SC5 through SC2) are all 1s.
SCZ (1) H	FXPL	A signal used to indicate that the step counter is fully incremented to all 1s, which causes termination of the operation being performed.
SD (1) H	FRHF	This flip-flop represents the sign of the destination operand and can be set by SCR OUT 31 when the upper half of the QR is to be loaded, by the exclusive OR for SS and SD with a SGNC field of 1 (see Paragraph 6.2.4.6), or by SS or its complement with a SGNC field of 0.
SELECT UBRMXB	FRMA	Indicates that the UAF bits are selecting the appropriate branch combinations and the FM0, ABORT, μBRK conditions are not enabled.
SET FER	FRMJ	A signal generated by the ROM control to set the error flag. The FER signal does not cause an interrupt; however, the same ROM control will cause an interrupt on sheet FRHH if the FIE bit is not enabled.
SET SYNCF	FRMJ	A direct set to the SYNC flip-flop which is produced at TS2 time when commanded by the control ROM or when a FM0 trap is going to occur.
SIGNC1, SIGNC0 (Sign)	FRMF	ROM bits used to determine sign of source and sign of destination.
SS (1) H	FRHF	This flip-flop represents the sign of the source operand and can be set by the sign bit (SCR OUT 31) of the scratchpad accumulator during loading of the upper half of the OR, or by the SGN bits in the control ROM.
SS XOR SD	FRHF	A signal which represents exclusive OR of SS and SD.
START EN	FRHH	Enables the state counter to restart from the Wait state on the next clock pulse.
STATE 4 (1) H	FRHH	Indicates that the FP11 is in time state 4.
STATE 3 (1) H	FRHH	Indicates that the FP11 is in time state 3.

(continued on next page)

Signal Mnemonic	Logic Print	Function
STATE 2 (1) H	FRHH	Indicates that the FP11 is in time state 2.
STATE 1 (1) H	FRHH	Indicates that the FP11 is in time state 1.
STRG 1 (1)	FRHE	This flip-flop when set, indicates a string of 1s is present.
STR ZERO (Store Zero)	FRLP	Indicates that all eight bits of the exponent which are stored in the scratch pad are 0s.
SUB	FXPE	Indicates hardware is to subtract two numbers.
SUB CALL	FRHH	A signal which is enabled for any of the seven subroutine calls designated by the CSB bits in the ROM.
SUB FRAC	FXPF	Denotes that the hardware is doing a subtraction with like signs or an addition with unlike signs.
SYNC	FRMF	ROM bit used to enable FP SYNC.
TP4	FRMA	A pulse occurring during the latter half of TS4.
TP2	FRMA	A pulse occurring during the latter half of TS2.
TS3, TS4, TS1, TS2	FRHH	Output signals which represent the four time states of the FP11 and which are applied to maintenance indicator lights on the FMAA card.
TS4A, TS4B	FRHH	TS4A and TS4B represent the TS4 signal after being applied through a driver.
TS3A, TS3B	FRHH	TS3A and TS3B represent the TS3 signal after being applied through a driver.
TS2A, TS2B	FRHH	TS2A and TS2B represent the TS2 signal after being applied through a driver.
UAF1, UAF0 (Microaddress field)	FRME	ROM bits used in conjunction with UBR and UJP bits for microbranching.
UBR2–UBR0 (Microbranch)	FRME	Three ROM bits used in conjunction with UAF and UJP bits for microbranching.
μBRKF (1) H	FRMA	This signal indicates that a match has occurred between the control ROM address register and the μBreak register and causes the FP11 to trap state 4 and interrupt if the interrupt enable is on. This signal occurs during maintenance mode and the FP11 must be in some state other than the Ready state.
μJMP (Microjump)	FRME	ROM bit used in conjunction with UBR and UAF bits for microbranching.
μMATCH H	FXPJ	A signal generated when the contents of the control ROM address register match the contents of the microbreak register. The FP11 can be programmed to stop when a match occurs.

(continued on next page.

Signal Mnemonic	Logic Print	Function
μMATCH (1) H	FXPJ	A delayed version of μMATCH H which represents one complete ROM state. This signal is used for synchronization in scope loops.
μTRAP B L	FRMA	This signal, when low, disables the ROM from the CRAR and allows the trap bit to be enabled.
WAIT (1) H	FRHH	Indicates the FP11 is in the Wait state.
WAITS	FRHH	An output signal which represents the Wait state of the FP11. This signal is applied to a maintenance indicator light on the FMAA board.
XTAL	FRHJ	The basic 20 MHz clock for the FP11.

READER'S COMMENTS

FP11 FLOATING POINT PROCESSOR
DEC-11-HFPAA-C-D

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 ______________

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

DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS 01754