EK-FP11C-MM-001

FP11-C floating-point processor maintenance manual

digital equipment corporation • maynard, massachusetts

1st Edition, May 1976

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

Digital Equipment Corporation assumes no responsibility for any errors which may appear in this manual.

Printed in U.S.A.

This document was set on DIGITAL's DECset-8000 computerized typesetting system.

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

DEC
DECCOMM
DECsystem-10
DECSYSTEM-20
DECtape
DECUS
DIGITAL
MASSBUS
PDP
RSTS
TYPESET-8
TYPESET-11
UNIBUS

CONTENTS

Page

CHAPTER 1 INTRODUCTION

CHAPTER 2 INTERFACE

CHAPTER 3 DATA AND DATA FORMATS

CONTENTS (Cont)

Page

CHAPTER 4 CONTROL ROM

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

ILLUSTRATIONS

TABLES

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The FP11-C Floating-Point Processor is a hardware option used with the PDP-11/70 Central Processor. The FP11-C enables the PDP-11 Central Processor to perform arithmetic and logic operations using floating-point arithmetic. The prime advantage is increased speed without the need of writing complex floating-point software routines. The FP11-C has single- and double-precision floating-point capability. Before describing the FP11-C Floating-Point Unit, several fundamentals of floating-point arithmetic are presented.

1.2 FLOATING-POINT NUMBERS

Data processed by digital computers may be represented by integers (whole numbers). Integers are sometimes called fixed-point numbers because the radix point never varies.

NOTE

The radix point represents the base of the number system employed. For example, the decimal system is a system with a base of 10, which means that the radix point is the decimal point. In the binary system, which is a base 2 system, the radix point is the binary point.

Since the radix point marks the least significant place of the integer, it is generally not included in the representation of the numbers. Data can be accurately represented by these numbers over a range specified by the length of the integers. Very small or very large quantities can be represented simply by adjusting the units used in the system. For example, the integer 001 can represent 1 microsecond or 1 second. Assume for a moment that 001 is a binary number and the three digits necessary to express it have a range from 000 to 111_2 (or from 0_{10} to 7_{10}); therefore, only microseconds or seconds could be expressed in the range of these three digits. This example illustrates one of the disadvantages imposed by representing data only by whole numbers, i.e., the range and accuracy limitations.

When both large and small numbers are to be expressed in a system without increasing the number of places in the data, a system which incorporates fractional values is employed. The position of the radix point in these numbers is not predetermined, and, hence, these numbers are called floating-point numbers. An example of a type of floating point representation of numbers is scientific notation where a number is represented by some value multiplied by the radix raised to some power.

Example:

$1{,}000{,}000 = 1 \times 10^6$

In this representation, the significant figures are written and the magnitude is adjusted to the correct value by the power of 10.

There are many ways to represent a number in scientific notation as shown in the example below.

$$
\begin{aligned}
512 &= 51200. \times 10^{-2} \\
&= 5120. \times 10^{-1} \\
&= 512. \times 10^{0} \\
&= 51.2 \times 10^{1} \\
&= 5.12 \times 10^{2}, \text{ etc.}
\end{aligned}
$$

Several examples in other base systems are shown below.

Decimal No.	Base 10	Base 8	Base 2
64	0.64×10^{2}	0.1×8^{3}	0.1×2^{7}
33	0.33×10^{2}	0.41×8^{2}	0.100001×2^{6}
1/2	0.5×10^{0}	0.4×8^{0}	0.1×2^{0}
1/16	0.625×10^{-1}	0.4×8^{-1}	0.1×2^{-3}

Note that in each of the examples above, only significant digits are retained in the final result and the radix point is always to the left of the most significant digit. Establishing the radix point in a whole number is accomplished by shifting the number to the right until the most significant digit is to the right of the radix point. Each right shift causes the exponent to be incremented by one. Establishing the radix point on a fractional number is done by shifting the number left until all leading zeros are eliminated. Each left shift causes the exponent to be decremented by one.

To summarize, the value of the number remains constant if the exponent is incremented for each right shift of the number and decremented for each left shift. Normally, the representation of a number in which the nonsignificant leading zeros have been removed is the most convenient representation. Consider the following examples:

Problem A – Represent the number 75_{10} as a binary normalized floating-point number. A binary normalized floating-point number (Paragraph 1.3) is one in which the whole number has been converted to a fraction with leading zeros eliminated and the exponent adjusted accordingly.

1. Integer conversion
   $75_{10} = 1001011_{2}$ 111 101

2. Convert to fraction and exponent
   $1001011.0 \times 2^{0} = 0.1001011 \times 2^{7}$
   Fraction = 0.1001011
   Exponent = 111

Problem B – Represent the number 0.25_{10} as a binary floating-point number.

1. Integer conversion
   $0.25_{10} = 0.01_{2}$

2. Convert to fraction and exponent
   $0.01 \times 2^{0} = 0.1 \times 2^{-1}$
   Fraction = 0.1
   Exponent = -1

1.3 NORMALIZATION

In digital computers, the number of places in the fraction is limited. Retention of nonsignificant leading zeros can decrease accuracy by taking places which could be filled by significant digits. For this reason, a process called normalization is used in the FP11-C. The normalization process consists of testing the fraction and shifting it until it is in the form 0.1.... The exponent is increased or decreased by the number of shifts of the fraction to retain equivalence to the original number. Since digits to the right of the binary point are weighted with negative powers of two, the smallest normalized fraction will be 1/2 (0.1000...). The largest normalized fraction is 0.1111....

Restricting the values which the fraction part of the number can take does not restrict the possible values of the number because the exponent part will determine magnitude.

Figure 1-1 shows an unnormalized fraction which must be left-shifted six places to become normalized. The exponent is decreased by six to maintain equivalence to the original number.

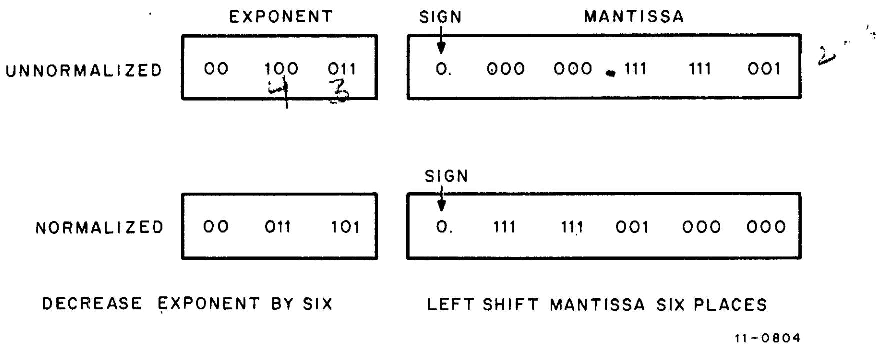


Figure 1-1 Floating-Point Representation

1.4 EXCESS 200 NOTATION

The magnitude of numbers that can be handled by a floating-point processor is dependent on the size of the exponent register. The FP11-C utilizes an 8-bit exponent register. Eight bits of exponent provide a range of 400_8 exponents from 0 to 377_8. However, this range does not allow a means to express both positive and negative exponents. In digital computers, a popular method of expressing positive and negative numbers utilizes 2's complement notation. A disadvantage of 2's complement notation is that an overflow has to occur to go from the least negative number to 0, as shown below.

	2's Complement		Excess 200
Positive Exponents	177 Most positive exponent ↕ 0 Least positive exponent	Positive Exponents	377 Most positive exponent ↕ 200 Least positive exponent
Negative Exponents	377 Least negative exponent ↕ 200 Most negative exponent	Negative Exponents	177 Least negative exponent ↕ 0 Most negative exponent

To avoid this, the FP11-C utilizes excess 200 notation, where the exponents are "biased" by 200 as shown in the chart. As a result of this "bias," 200 must be subtracted from the exponent calculation in multiplication since exponents are added and, in division, 200 must be added to the exponent calculation since exponents are subtracted.

To understand why 200 must be subtracted from the exponent calculation during multiplication, consider the following:

$$\begin{array}{l} \text{Exponent A} + 200 \\ \underline{\text{Exponent B} + 200} \\ \hline \text{Exponent A} + \text{Exponent B} + 400 \end{array}$$

Both exponent A and exponent B are biased by 200, yielding a bias of 400. However, only a bias of 200 is desired in excess 200 notation. It is, therefore, necessary to subtract 200 from the exponent calculation.

To understand why 200 must be added to the exponent calculation during division, consider the following:

$$\begin{array}{l} (\text{Exponent A} + 200) \\ -(\text{Exponent B} + 200) \\ \hline (\text{Exponent A} - \text{Exponent B} + 200 - 200 = \text{Exponent A} - \text{Exponent B} + 0 \end{array}$$

However, since the result is to be in excess 200 notation, 200 must be added to the exponent, yielding Exponent A - Exponent B + 200.

Several simplified examples are shown below to illustrate this concept.

Example 1 - Multiplication

$2 \times 3 =$

	Fraction		**Exponent**
2 =	0.100	X	202
3 =	0.110	X	202

$$\begin{array}{rcr} \textbf{Fraction Calculation} & \quad & \textbf{Exponent Calculation} \\ 0.100 & & 202 \\ \underline{0.110} & & \underline{+202} \\ 1000 & & 404 \\ \underline{100} & & \underline{-200} \\ 0.011000 & & 204 \end{array}$$

Normalize the fraction by left shifting one place and decreasing the exponent by 1.

Fraction	**Exponent**
0.11000	203

Example 2 – Division

16 ÷ 4 =

	Fraction		Exponent
16 =	.10000	X	205
4 =	.10000	X	203

Fraction Calculation	Exponent Calculation
$$\begin{array}{r} 1.000 \\ 0.10000\,\overline{)\,0.10000.000} \end{array}$$	$$\begin{array}{r} 205 \\ -203 \\ \hline 2 \\ +200 \\ \hline 202 \end{array}$$

Normalize the fraction by right shifting one place and incrementing the exponent.

1.000 = 0.100 × 203

This number is equivalent to 4.

1.5 FLOATING-POINT ADDITION AND SUBTRACTION

For floating-point addition or subtraction, the exponents must be aligned or equal. If they are not aligned, the fraction with the smaller exponent is shifted right until they are. Each shift to the right is accompanied by an incrementation of the associated exponent. When the exponents are aligned or equal, the fraction can be added or subtracted. 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 fractions are added; the exponent value dictates the final location of the binary point.

$$\begin{array}{rl} 0.\ 101\ 000\ 000\ 000\ 000 \times 2^6 = 50_8 = & 40_{10} \\ +0.\ 111\ 000\ 000\ 000\ 000 \times 2^3 = \ \ 7_8 = & 7_{10} \end{array}$$

1. To align exponents, shift the fraction with the smaller exponent three places to the right and increment the exponent by 3, and then add the two fractions.

$$\begin{array}{rl} 0.\ 101\ 000\ 000\ 000\ 000 \times 2^6 = 50_8 = & 40_{10} \\ +0.\ 000\ 111\ 000\ 000\ 000 \times 2^6 = \ \ 7_8 = & 7_{10} \\ \hline 0.\ 101\ 111\ 000\ 000\ 000 \times 2^6 = 57_8 = & 47_{10} \end{array}$$

2. To find the true value of the answer, move the binary point six places to the right.

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

1.6 FLOATING-POINT MULTIPLICATION AND DIVISION

In floating-point multiplication, the fractions are multiplied and the exponents are added. For floating-point division, the fractions are divided and the exponents are subtracted.

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

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

$$\begin{array}{r l}
0.1110000 & \times 2^3 = 7_8 = 7_{10} \\
\times 0.1010000 & \times 2^3 = 5_8 = 5_{10} \\
\hline
00000000 & \\
1110000 & \\
0 & \\
1110000 & \\
\hline
.10001100000000 &
\end{array}$$

Move the binary point six places to the right.

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

1.7 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-C Floating-Point Processor are listed below.

- Performs arithmetic operations on 32- or 64-bit floating-point numbers. The 32-bit number contains 23 bits of fraction, 8 bits of exponent, and 1 bit of sign. The 64-bit number consists of 55 bits of fraction, 8 bits of exponent, and 1 bit of sign.
- Includes special instructions to optimize input/output routines and mathematical subroutines.
- Utilizes microprogramming techniques for greater flexibility and reduced cost.
- Compatible with existing PDP-11 address modes.
- Utilizes overlap processing, i.e., CPU and FP11-C can run simultaneously.
- Allows 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.

- Is capable of converting 32- or 64-bit floating-point numbers to 16- or 32-bit integers during the Store class of instructions.

- Is capable of converting single-precision floating-point to double-precision floating point and vice versa during the Load or Store classes of instructions.

- Contains floating-point condition codes that can be copied into the CPU condition codes 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.

- High floating-point throughput.

1.8 SIMPLIFIED BLOCK DIAGRAM DESCRIPTION

Figure 1-2 shows a simplified block diagram of the floating-point processor. The major elements of the FP11-C are the exponent calculation logic, sign processor, the accumulators, and the fraction calculation logic.

31 30 29 28 27 26 25 24 23

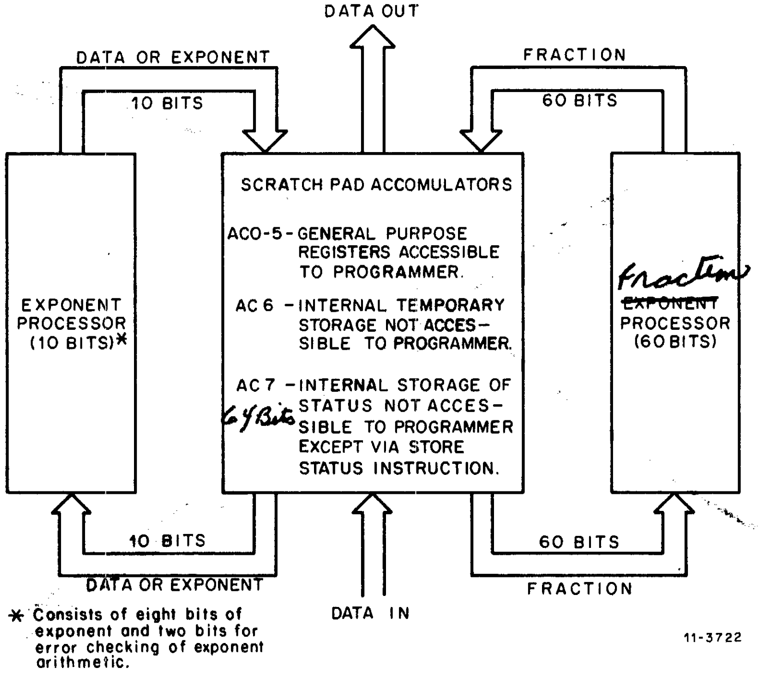

DATA OUT
DATA OR EXPONENT
10 BITS
FRACTION
60 BITS
SCRATCH PAD ACCOMULATORS
AC0-5 - GENERAL PURPOSE REGISTERS ACCESSIBLE TO PROGRAMMER.
AC 6 - INTERNAL TEMPORARY STORAGE NOT ACCES-SIBLE TO PROGRAMMER.
AC 7 - INTERNAL STORAGE OF STATUS NOT ACCES-SIBLE TO PROGRAMMER EXCEPT VIA STORE STATUS INSTRUCTION.
64 Bits
EXPONENT PROCESSOR (10 BITS)*
Fraction
~~EXPONENT~~
PROCESSOR (60 BITS)
10 BITS
DATA OR EXPONENT
DATA IN
60 BITS
FRACTION
* Consists of eight bits of exponent and two bits for error checking of exponent arithmetic.
11-3722

Figure 1-2 FP11-C Simplified Block Diagram

The exponent calculation logic connects to a 10-bit wide data path that processes exponent information as follows: 8 bits of exponent, 1 bit of sign indicating arithmetic result of the exponent calculation, and 1 bit for overflow.

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 60-bit scratchpad accumulator.

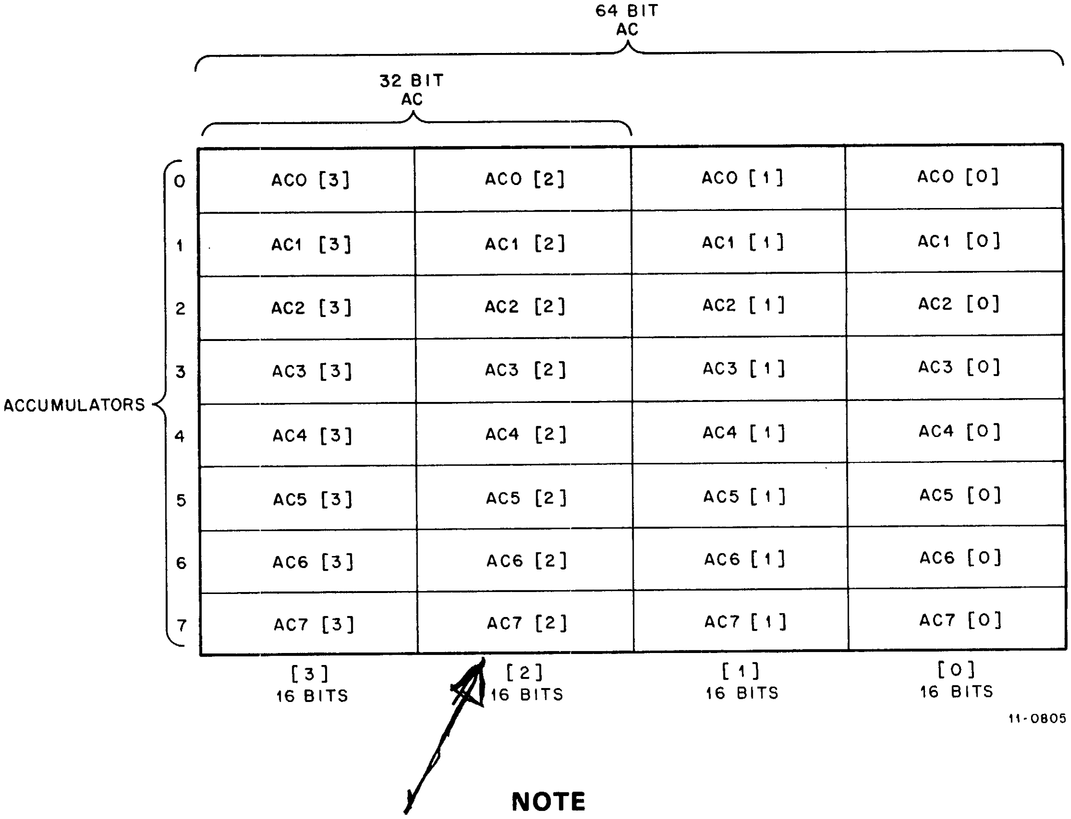
The accumulators (ACs) are general-purpose read/write scratchpad memories with nondestructive readout. Accumulators 5 through 0 are used for storage of general-purpose data and for register-to-register operations. Accumulator 6 is used for temporary internal storage and is not accessible by the programmer.

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

1. Floating Exception Code (FEC) - A number that identifies the last cause of an interrupt by the FP11-C.
2. Floating Exception Address (FEA) - The address of the last instruction that caused the interrupt.

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

The ACs are interpreted as being 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 leftmost 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 64-bit words or four 16-bit words (quadrants). Examples of the designated AC and the length of the word contained therein are: AC5[3:2], AC3[3:0].



	[3] 16 BITS	[2] 16 BITS	[1] 16 BITS	[0] 16 BITS
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]

NOTE
AC7[2] contains address of instruction.

Figure 1-3 Accumulator Configuration

The number following the AC designates one of eight accumulators, and each number in the bracket denotes a 16-bit quadrant. In the first example, AC5 contains two 16-bit quadrants [3:2]; in the second case, AC3 contains four 16-bit quadrants [3:0]. 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.

1.9 MEMORY/FP11-C 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-C. Floating-point numbers are already in sign and magnitude format and are transferred directly to the FP11-C without being converted. When the FP11-C finishes processing the numbers, they can be transferred back to memory as 2's complement integers or sign and magnitude floating-point numbers. Floating-point numbers are always normalized.

The example below shows the numbers +2 and -2 represented in 2's complement notation and in sign and magnitude notation. Note that positive numbers appear the same whether expressed in 2's complement notation or sign and magnitude notation.

2's Complement Notation		Sign and Magnitude Notation
+2	000010	000010 Sign ↑ → Magnitude
-2	111110	100010 Sign ↑ → Magnitude

1.9.1 FP11-C Hidden Bit

All numbers (fractions) transferred to the fraction calculation logic are transferred as positive fractions of the form 0.1xxxx. Since the number is normalized, 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. The hidden bit provides another bit of significance in the FP11-C. Words transferred from memory to the FP11-C consist of a sign, 8 bits of exponent, and 23 bits of fraction (single-precision) or 55 bits of fraction (double-precision). When the word is transferred to the FP11-C, the hidden bit is inserted, resulting in 24 bits of fraction (single-precision) or 56 bits of fraction (double-precision).

Figure 1-4 shows the format of a word as it appears in memory and as it appears when stored internally in the FP11-C.

Regardless of whether the floating-point number is positive or negative, the fraction is treated as a positive normalized fraction (bit 59 = 0, bit 58 = 1). Bits 59 and 58 are dropped when the floating-point word is reassembled and stored back in memory.

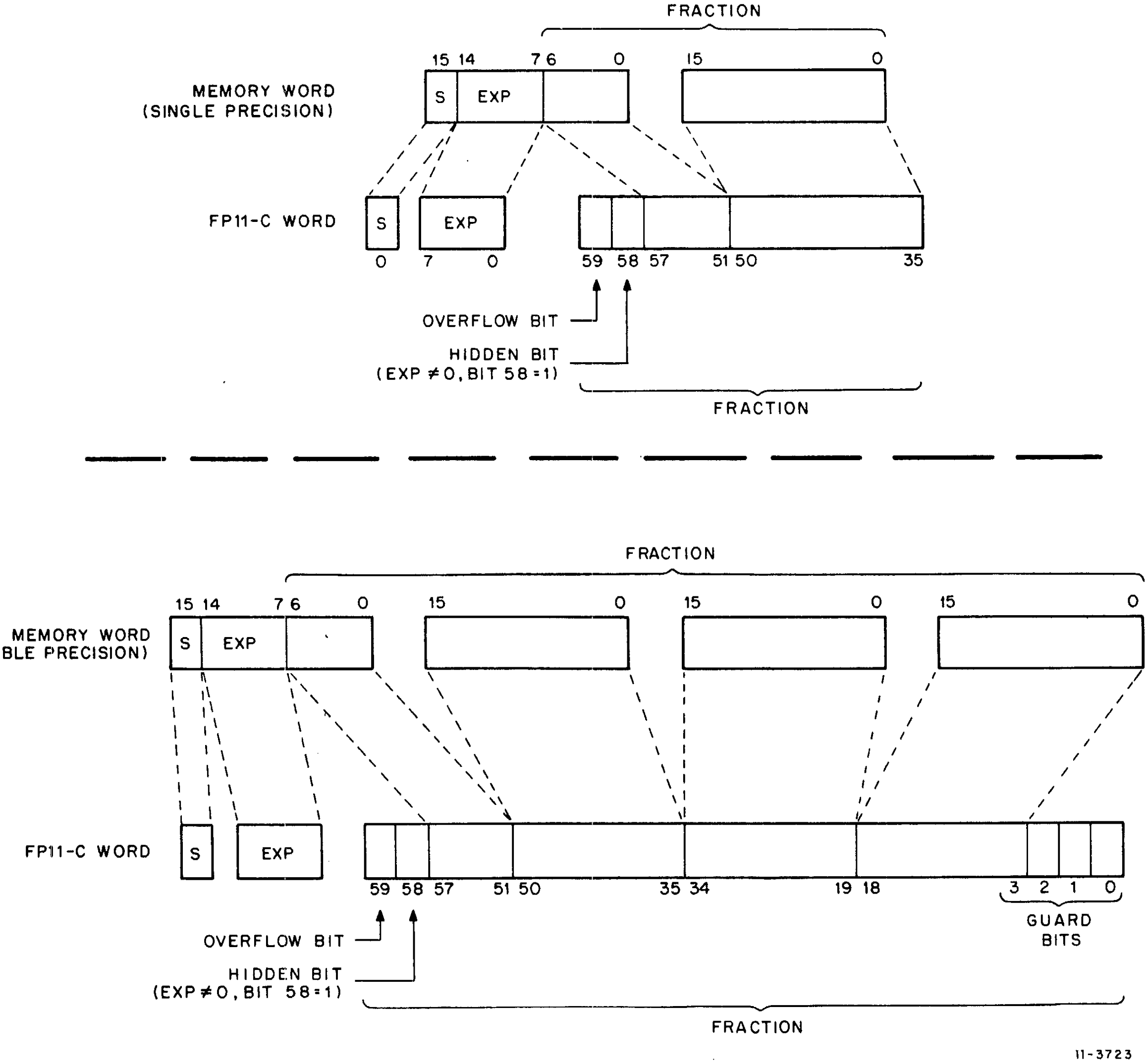


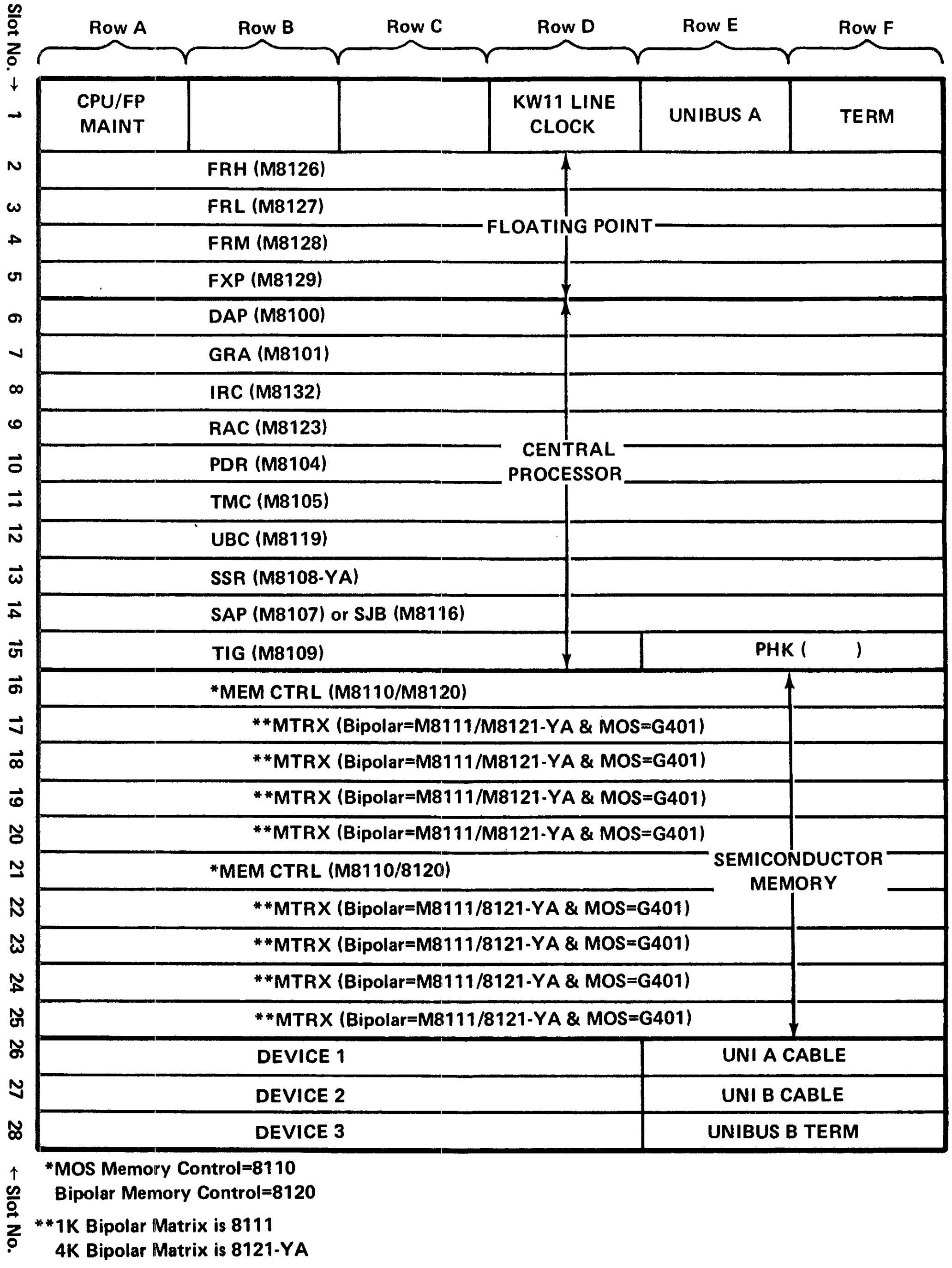
Figure 1-4 Memory/FP11-C Bit Relationships

1.10 FP11-C PHYSICAL DESCRIPTION

The FP11-C Floating-Point Processor is used with the KB11-C (PDP-11/70 CPU) or the KB11-D (PDP-11/45,50,55 CPU). The FP11-C consists of four multilayer hex modules that are plugged into the prewired KB11-D Mainframe or KB11-C Mainframe. The four modules plug into slots 2, 3, 4, and 5 and take-up rows A through F. (See Figure 1-5 for the KB11-D Mainframe and Figure 1-6 for the KB11-C Mainframe.) The chart below shows the slots associated with each module.

M8128	-FRM	4	A through F
M8129	-FXP	5	A through F
M8126	-FRH	2	A through F
M8127	-FRL	3	A through F

A +5 V regulator card is included and is plugged into the upper power supply in slot A. The -15 V needed for the time state generator on the FRH module in the FP11-C is supplied by regulator E, which is included as part of the Central Processor Regulator Set. The -15 V needed in the PDP-11/70 comes from the lower H7420 Power Supply.



Slot No.	Row A	Row B	Row C	Row D	Row E	Row F
1	CPU/FP MAINT			KW11 LINE CLOCK	UNIBUS A	TERM
2	FRH (M8126)					
3	FRL (M8127)					
4	FRM (M8128)			FLOATING POINT		
5	FXP (M8129)					
6	DAP (M8100)					
7	GRA (M8101)					
8	IRC (M8132)					
9	RAC (M8123)					
10	PDR (M8104)			CENTRAL PROCESSOR		
11	TMC (M8105)					
12	UBC (M8119)					
13	SSR (M8108-YA)					
14	SAP (M8107) or SJB (M8116)					
15	TIG (M8109)				PHK ( )	
16	*MEM CTRL (M8110/M8120)					
17	**MTRX (Bipolar=M8111/M8121-YA & MOS=G401)					
18	**MTRX (Bipolar=M8111/M8121-YA & MOS=G401)					
19	**MTRX (Bipolar=M8111/M8121-YA & MOS=G401)					
20	**MTRX (Bipolar=M8111/M8121-YA & MOS=G401)					
21	*MEM CTRL (M8110/8120)				SEMICONDUCTOR MEMORY	
22	**MTRX (Bipolar=M8111/8121-YA & MOS=G401)					
23	**MTRX (Bipolar=M8111/8121-YA & MOS=G401)					
24	**MTRX (Bipolar=M8111/8121-YA & MOS=G401)					
25	**MTRX (Bipolar=M8111/8121-YA & MOS=G401)					
26	DEVICE 1				UNI A CABLE	
27	DEVICE 2				UNI B CABLE	
28	DEVICE 3				UNIBUS B TERM	

*MOS Memory Control=8110
Bipolar Memory Control=8120

**1K Bipolar Matrix is 8111
4K Bipolar Matrix is 8121-YA

11-3739

Figure 1-5 FP11-C Module Layout (PDP-11/45)

	44	43	42	41	40	39	38	37	36	35	34
A	M930					M5904	M5904	M5904	M8150	M5904	M5904
B						MBSC	MBSB	MBSA		MBSC	MBSB
C						M8153	M8152	M8151	MDP	M8153	M8152
D											
E	SPC	SPC	SPC	SPC	SPC	BCT	AWR	CST		BCT	AWR
F											

	33	32	31	30	29	28	27	26	25	24	23
A	M5904	M8150	M5904	M5904	M5904	M8150	M5904	M5904	M5904	M8150	
B	MBSA		MBSC	MBSB	MBSA		MBSC	MBSB	MBSA		
C	M8151	MDP	M8153	M8152	M8151	MDP	M8153	M8152	M8151	MDP	
D											
E	CST		BCT	AWR	CST		BCT	AWR	CST		
F											

	22	21	20	19	18	17	16	15	14	13	12
A	M8141	M8145	M8144	▨	M8143	M8142	M8140	M8138	M8137	VMS	M8136
B				▨				YA		▨	
C	MAP	CDP	DTM	▨	ADM	CCB	SCC	SSR	SAP	M8139	UBC
D				▨							
E				▨						TIG	
F				▨							

	11	10	9	8	7	6	5	4	3	2	1
A	M8135	M8134	M8123	M8132	M8131	M8130	M8129	M8128	M8127	M8126	
B											CPMAINT
C	TMC	PDR	RAC	IRC	GRA	DAP	FXP	FRM	FRL	FRH	
D											KW11
E											M930
F											

11/70 MODULE UTILIZATION CHART

(Viewed from module side of backplane)

11-3738

CHAPTER 2
INTERFACE

2.1 INTRODUCTION

The floating-point processor connects directly to the PDP-11/45 or 11/70 Central Processor (Figure 2-1) and not to the Unibus. This is to allow addressing of floating-point memory references to utilize the memory management option.

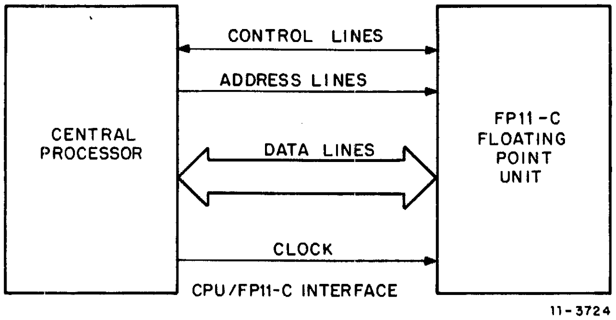


Figure 2-1 PDP-11/45, 11/70 Simplified Interface Diagram

The FP11-C depends on the CPU to fetch instructions and data. Control of the program resides in the CPU and the CPU, therefore, must initiate floating-point operation and supply addresses and data, as required.

The CPU fetches instructions from memory and decodes each instruction. If the instruction is not a floating-point instruction, it is ignored by the FP11-C. If the instruction is a floating-point instruction, it contains an op code of $17XXXX_8$ and the CPU branches to the CPU ROM (read-only memory) states (Chapter 4) associated with floating-point instructions. The CPU/FP11-C interaction is initiated when the CPU issues FP START to alert the FP11-C of a pending instruction. The FP11-C asserts FP REQ and then issues FP SYNC when it is ready to accept or transmit the operands, as in a data transfer type instruction. At this point, the CPU is calculating the address of the instruction. Upon completion of this, the CPU monitors the FP11-C for FP SYNC. When it is asserted, the CPU proceeds to read or write operands. The FP11-C controls the direction of transfer. (For Load class instructions, operands are transferred from memory to the FP11-C. For Store class instructions, the result or operand is transferred from the FP11-C to memory.)

When completion of the data transfer occurs, the FP11-C negates FP REQ and the CPU is free to execute other instructions.

2.2 INTERFACE SIGNALS

The signals that interface the CPU to the FP11-C (Figure 2-2) are described below.

Signal	Description
BAMX (15:00) H	Sixteen lines from the CPU that contain the address of the instruction.
BRA (15:00) L	Sixteen data lines that provide transfer of data from the CPU to FP11-C.
BUS INTD (15:00) L	Sixteen lines used to send data from the FP11-C to the CPU. These lines are also used by the memory management unit.
FP ACKN L	A signal from the CPU indicating that an FP TRAP was received from the FP11-C.
INTR CLR L	A signal from the CPU that indicates that the CPU is in its interrupt service routine.
FP READ L	A signal asserted by the CPU that indicates that the BUS INTD lines will be driven by data or status from the FP11-C.
FP ATTN	A signal issued by the CPU to indicate a memory cycle has been completed.
INIT L	An initialize pulse used to reset major registers in the FP11-C.
FP EXC TRAP L	This signal, when low, causes the CPU to trap to vector address 244_8(Trap vector).
AD1, AD2 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 [illegible]
0	1	Constant of 4
1	0	Constant of 2
1	1	Constant of 0

Signal	Description
FCLD EN L	This signal causes the FP11-C floating-point condition codes to be written into the CPU condition codes.
FP REG WR H	When high, this signal causes the CPU to write 16 bits of data from the FP11-C to one of the CPU general registers.
FP SYNC L	A signal from the FP11-C in response to FP START, indicating that the instruction has been started or that the FP11-C is ready to send or receive data.
FP REQ L	A signal used in conjunction with FP SYNC to indicate that 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-C is present.

ACOMX — Sixteen lines to the console that allow the outputs of ACOMX to be displayed.

RARB (07:00) — Eight lines to the console that allow the FP11-C ROM address to be displayed.

FP START — Tells the FP11-C to start executing the instruction.

FPC — Floating-point control field. Controls selection of the data out multiplexer (DOMX) and clocking of the Floating Instruction Register A (FIRA), Floating-Point Address (FPA) register, and Floating Data Register (FDR) in the FP11-C.

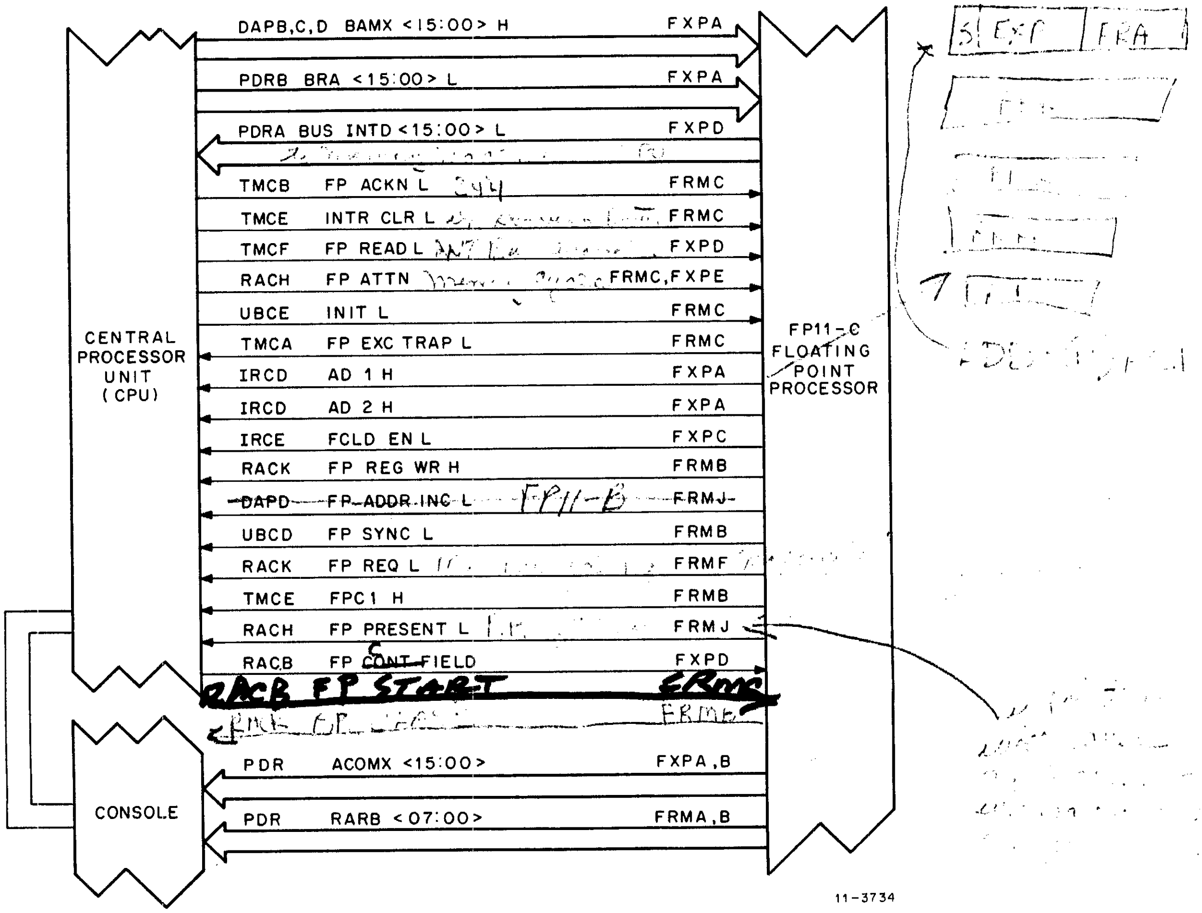


Figure 2-2 CPU/FP11-C Interface Diagram

F.P. [illegible] – floating point [illegible] – [illegible] [illegible] + [illegible]
Causes CPU [illegible]

2.3 CPU/FP11-C INTERFACE DIAGRAM DESCRIPTION

Figure 2-3 shows the interface between the CPU and the FP11-C. When the CPU fetches a floating-point instruction, the address of that instruction is sent to the FPA register in the FP11-C via the BAMX in the CPU. This provides temporary storage of the address and allows the CPU to read the address back via the data out multiplexer (DOMX), if an interrupt occurs. When the FP11-C starts to execute an instruction, the address is clocked from the FPA into the FEA register. If the instruction being executed causes a floating-point trap, the contents of the FEA are transferred to AC7[1], which stores the floating exception address (FEA) used in the store status (STST) instruction (Chapter 3).

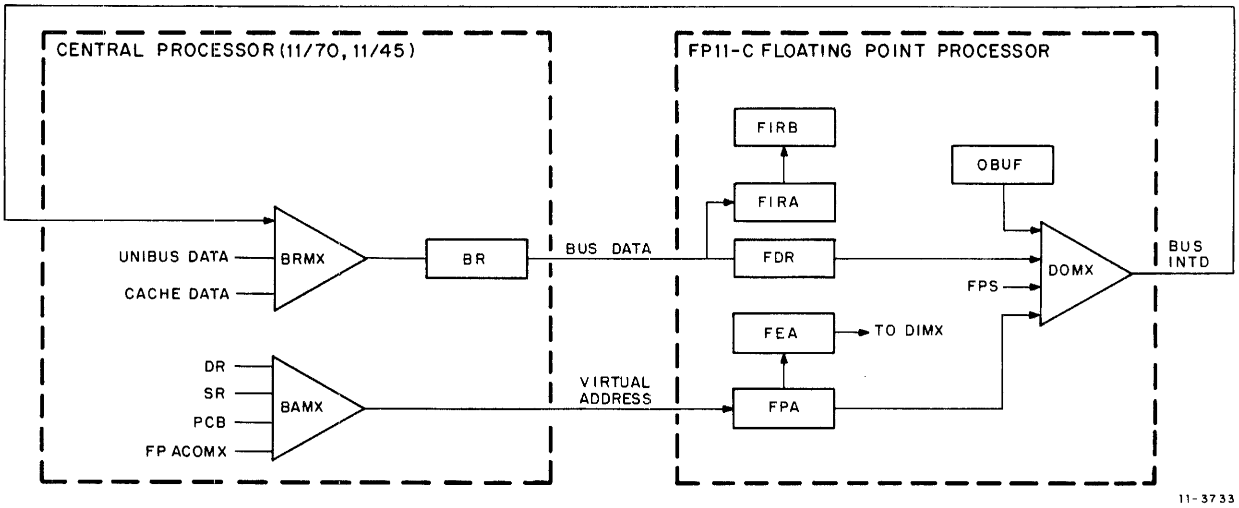


Figure 2-3 CPU/FP11-C Interface Block Diagram

The instruction is applied to Floating Instruction Register A (FIRA) in the FP11-C via the BRMX and BR in the CPU. The loading of FIRA is controlled by the CPU microcode. At the beginning of the floating-point instruction execution, the contents of FIRA are loaded into Floating Instruction Register B (FIRB). The loading of FIRB is controlled by the FP11-C microcode. FIRB contains the instruction that the FP11-C is currently executing, while FIRA contains the instruction fetched by the CPU.

The contents of the general register, which may be modified by address calculation, are transferred to the FDR in the FP11-C via the CPU general register, the BRMX, and the BR in the CPU before the address calculation is done. Thus, it can be restored if an interrupt occurs.

Data supplied to the CPU from the FP11-C is routed to memory via the DOMX, OBUF, and the scratchpad in the FP11-C.

2.4 CPU/FP11-C INTERFACE FLOW DIAGRAM DESCRIPTION

This section describes the sequence of events that occur during Load and Store instructions and also covers the special situations listed below. Load instructions cause operands to be loaded into the FP11-C. Store instructions cause the result of a floating-point operation to be stored in memory.

- Load Instruction Class – This paragraph describes the flow diagram for a Load floating-point instruction. It is assumed that the FP11-C is not busy and no interrupt is raised.

- Store Instruction Class – This paragraph describes the flow diagram for a Store floating-point instruction. It is assumed that the FP11-C is not busy and no interrupt is raised.
- FP11-C Busy – This paragraph describes what occurs when the CPU issues an instruction to the FP11-C while the FP11-C is busy.
- Interrupt Operation – This paragraph describes what occurs when the CPU issues a floating-point instruction and an interrupt occurs.
- Floating-Pause Operation – This paragraph describes what occurs during floating-pause operation, which is only used for the NEG and ABS instructions (not mode 0) (Chapter 3).
- Destination Mode 0 – This paragraph describes the sequence of events for a floating-point instruction when destination mode 0 is specified. No memory reference takes place during this mode.
- Destination Mode 0 with Interrupt Sequence – This paragraph describes the sequence of events for a floating-point instruction when destination mode 0 is specified and an interrupt has been raised.

For simplicity, the instructions are described as single-precision instructions. Double-precision words simply require the transfer of two additional data words.

It is assumed that the reader is familiar with the symbolic notation on the CPU flow diagrams. The symbolic notation associated with the FP11-C flow diagrams is described in Table 4-1 of this manual. Additional information can be found by referring to the *KB11-C Central Processor Manual* (EK-KB11C-TM-001).

2.4.1 Load Instruction Class

The sequence is initiated by the CPU fetching the instruction from memory; this is accomplished in the first two ROM states shown in Figure 2-4.

> **NOTE**
>
> **The ROM is a read-only memory which replaces groups of combinational logic. (See Chapter 4 for a detailed description.) Each block in Figure 2-4 represents a particular ROM state, i.e., there is a ROM associated with the CPU and one associated with the FP11-C.**

The CPU decodes the instruction after it has been fetched from memory and sends it to the FP11-C. If bits 15 through 12 are decoded as 17_8, the instruction is a floating-point instruction.

After the instruction has settled, the CPU transfers the instruction to FIRA in the FP11-C, then issues an FP START signal. FP START flags the FP11-C to start execution of the instruction.

After it has been ascertained that the instruction is a floating-point instruction, the CPU transfers the contents of the destination register to the FDR in the FP11-C. This is accomplished in ROM states 101 and 314 (designated in the upper-right corner of the 4th and 5th ROM states from the top of Figure 2-4). These ROM states are used to preserve the contents of the program counter and general register during interrupt service routines. Since it is assumed there will be no interrupts for this example, these states are not described.

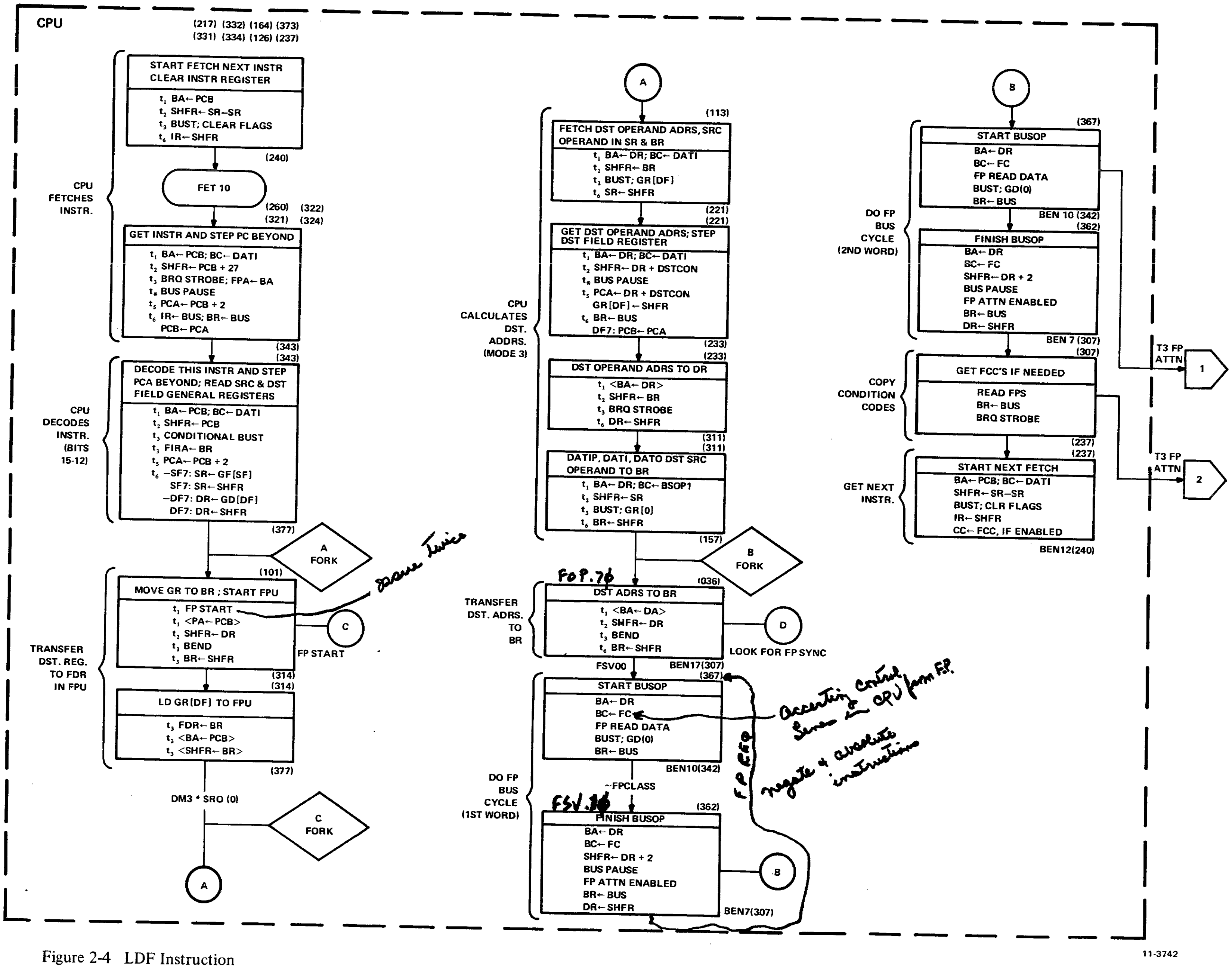


Figure 2-4 LDF Instruction
Flow Diagram - Write FP11-C
(Sheet 1 of 2)

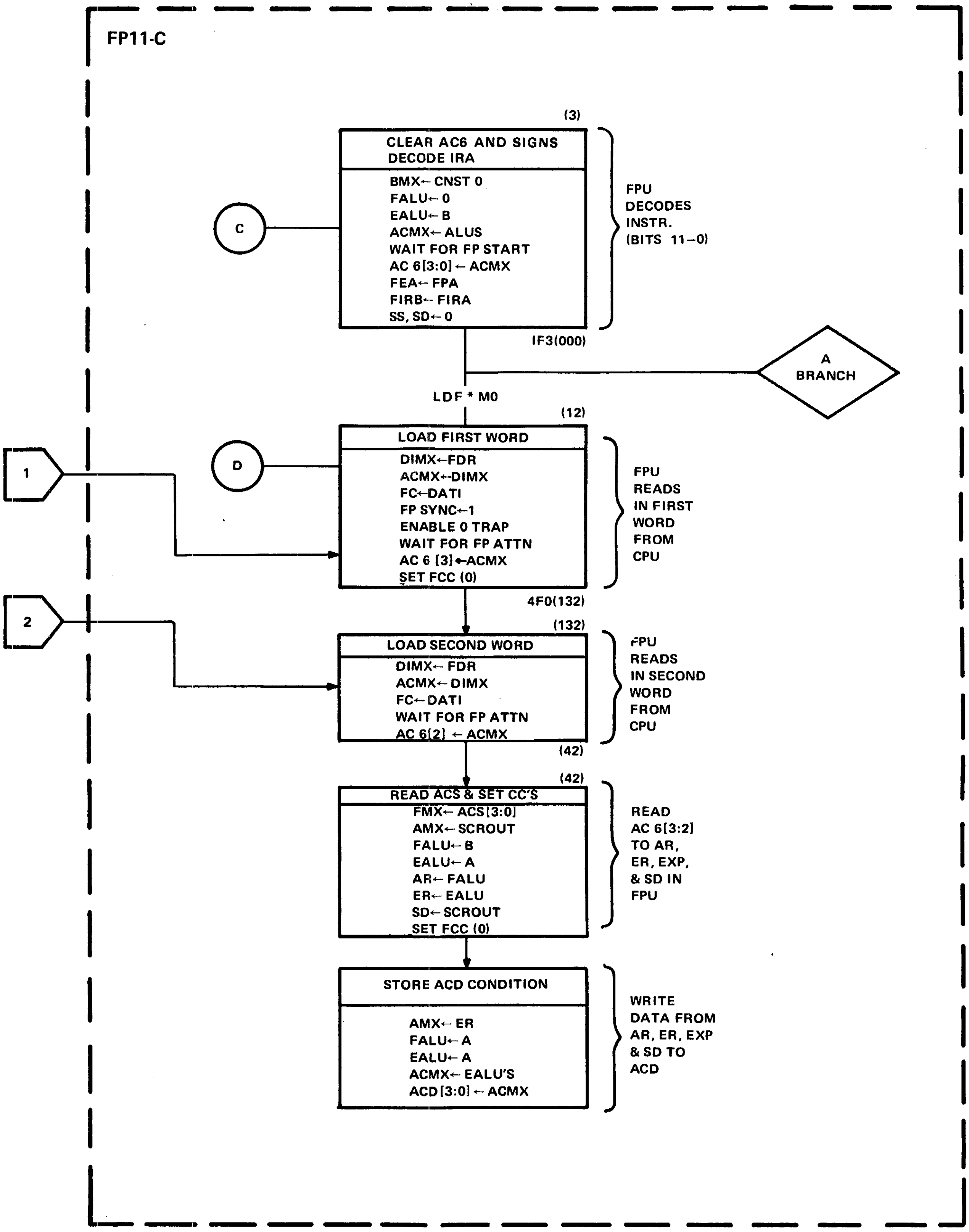


Figure 2-4 LDF Instruction Flow Diagram - Write FP11-C (Sheet 2 of 2)

At this point, the CPU calculates the destination address, which is mode 3 in this example. Mode 3 indicates that the address of the destination operand is in the location following the instruction.

When the FP11-C receives FP START, it has already decoded bits 11 through 00 of the instruction and entered the next ROM state (state 012). While the CPU is performing the address calculation, the FP11-C is waiting for the first data word (operand) from the CPU and indicates this by issuing FP SYNC.

Flow 7.13

When the CPU advances to state 036, it looks for the FP SYNC signal. Upon receipt of this signal, the CPU performs the first bus cycle, which is accomplished in states 367 and 362. The data is not transferred to the FP11-C in either of these states, however. FP ATTN is enabled in state 362 and the first data word is not transferred to the FP11-C until T3 of the state following the enabling of FP ATTN. At this time, FP ATTN is asserted on the interface. This deskew time provides settling time for the data in the BR in the CPU to stabilize before it is transferred to the FDR in the FP11-C. When the FP11-C receives the first data word, it leaves the wait state and writes the word in the FDR into AC6[3]. AC6[3] represents the most significant 16 bits of AC6.

The FP11-C then steps to state 132 to wait for the second data word. The CPU, at this time, has determined that it must perform another bus cycle by branching on FP REQ. If FP REQ is asserted, the CPU performs the second bus cycle. (See states 367 and 362, which are the same states implemented in first bus cycle.) In state 362, FP ATTN is enabled. At T3 of the next state (307) (to allow the data in the BR to stabilize), FP ATTN is issued to the FP11-C along with the second data word (operand). The FP11-C leaves the wait state and writes the operand in AC6[2].

In the next state, the CPU checks the condition codes (Chapter 3). In this example, the condition codes are not used and the CPU sequences to the next instruction fetch (state 237). While this is occurring, the FP11-C is transferring the contents of the source accumulator (AC6) to the destination accumulator (ACD).

2.4.2 Store Instruction Class

This sequence is initiated by the CPU fetching the instruction from memory, which is accomplished in the first two ROM states shown in Figure 2-5.

The CPU decodes the instruction after it has been fetched from memory. If bits $15–12_8$ are decoded as 17_8, the instruction is a floating-point instruction.

After the instruction has settled, the CPU transfers the instruction to FIRA in the FP11-C, then issues an FP START signal. FP START flags the FP11-C to start execution of the instruction.

After it has been ascertained that the instruction is a floating-point instruction, the CPU transfers the contents of the destination register to the FDR in the FP11-C. This is accomplished in ROM states 101 and 314. These states are used to preserve the contents of the program counter and general register during interrupt service routines. Since it is assumed there will be no interrupts, these states will not be described here.

At this point, the CPU calculates the destination address, which is mode 3 in this example. Mode 3 indicates that the address of the destination operand is in the location following the instruction.

When the FP11-C receives FP START, it has already decoded bits 11 through 00 of the instruction and entered the next ROM state (state 13). In T1 of this state, the FP11-C issues FP SYNC, indicating that it is ready to send the first word, and clocks the word into the OBUF. However, the CPU is still doing the address calculation.

When the CPU finishes the address calculation, it looks for FP SYNC. Since the FP11-C has issued FP SYNC, the CPU starts a bus cycle (state 367), where it reads the floating-point data. The data in OBUF, which is ACD[3], is transferred to the BR in the CPU via the FP11-C DOMX and the CPU BRMX. In CPU state 362, the word in the BR is transferred to memory. The CPU now looks for an FP REQ signal from the FP11-C. If it is asserted, the CPU knows that it must prepare to perform another bus cycle.

Meanwhile, the FP11-C has sequenced from state 013 to state 203. In this state, the contents of ACD[2] are present at the input to OBUF, and the FP11-C waits for the first FP ATTN from the CPU. When it is received, the contents of ACD[2] are clocked into OBUF and are transferred to the CPU. Since the FP REQ signal is asserted, the CPU performs another bus cycle, causing the second data word to be transferred to the BR in state 367 and causing the word to be transferred from the BR to memory in state 362. Note that this is the second 367 state and the second 362 state. The CPU now looks for an FP REQ signal. Since the example describes a single-precision word, the FP11-C negates the FP REQ line, indicating to the CPU that it is finished with the transfer.

The FP11-C sequences to state 202 where it waits for FP ATTN from the CPU. When the second FP ATTN is received, the contents of ACD[1] are clocked into OBUF. This word, however, will never be transferred to the CPU since the CPU will not perform any more bus cycles due to the negation of FP REQ.

After the CPU completes state 362, it sequences to state 307 where it can copy the floating condition codes in the FP11-C, if desired. From this state, the CPU sequences to state 237 to start the next instruction fetch.

To summarize, the first word (ACD[3]) was transferred to the CPU by the microcode in the FP11-C. ACD[3] was clocked into OBUF and then transferred to the BR in the CPU and subsequently transferred to memory. The second word (ACD[2]) was clocked into OBUF and transferred to the FP11-C by the first FP ATTN signal from the CPU. The third word (ACD[1]) was clocked into OBUF by the second FP ATTN signal. However, the word will not be transferred to the CPU since FP REQ will not be asserted by the FP11-C.

2.4.3 FP11-C Busy

Paragraph 2.4.1 discussed the interaction between the CPU and the FP11-C during a Load instruction. It was assumed that FP SYNC was issued by the FP11-C at or prior to the time that the CPU was looking for it. This meant that the FP11-C was ready to accept data. Now assume that the FP11-C does not issue FP SYNC. This indicates that it is busy or wants to raise an interrupt.

In the normal flow in Figure 2-4, the CPU steps from state 036 to 367 if FP SYNC is asserted. This is indicated by the dotted blocks in Figure 2-6.

If FP SYNC is not asserted, the CPU goes into a waiting loop (state 327 in Figure 2-6) and continues to sample the FP SYNC line. When FP SYNC finally becomes asserted (indicating that the FP11-C is ready to accept data), the CPU sequences from state 327 to state 367, which is the normal flow pattern.

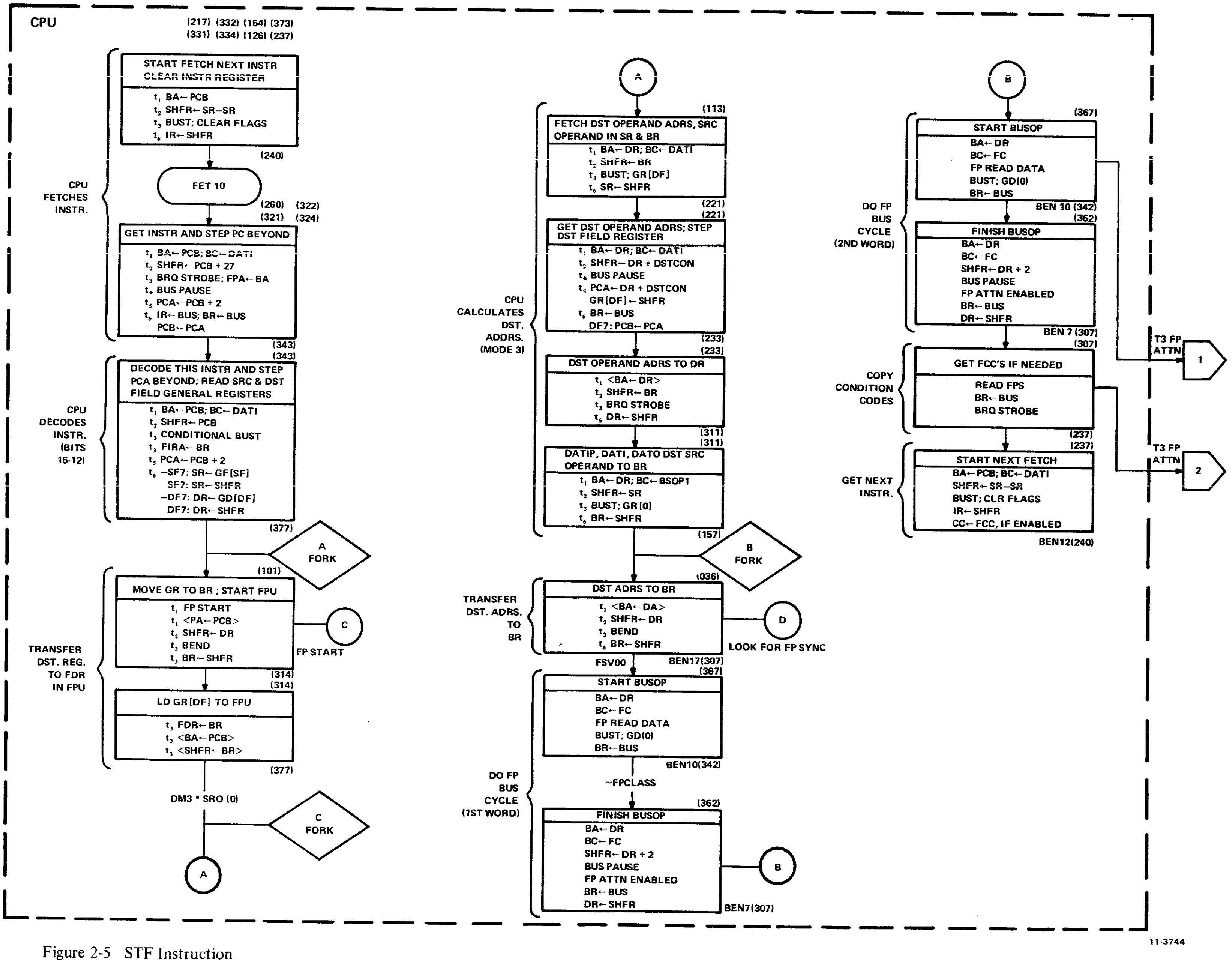


Figure 2-5 STF Instruction
Flow Diagram - Read from FP11-C
(Sheet 1 of 2)

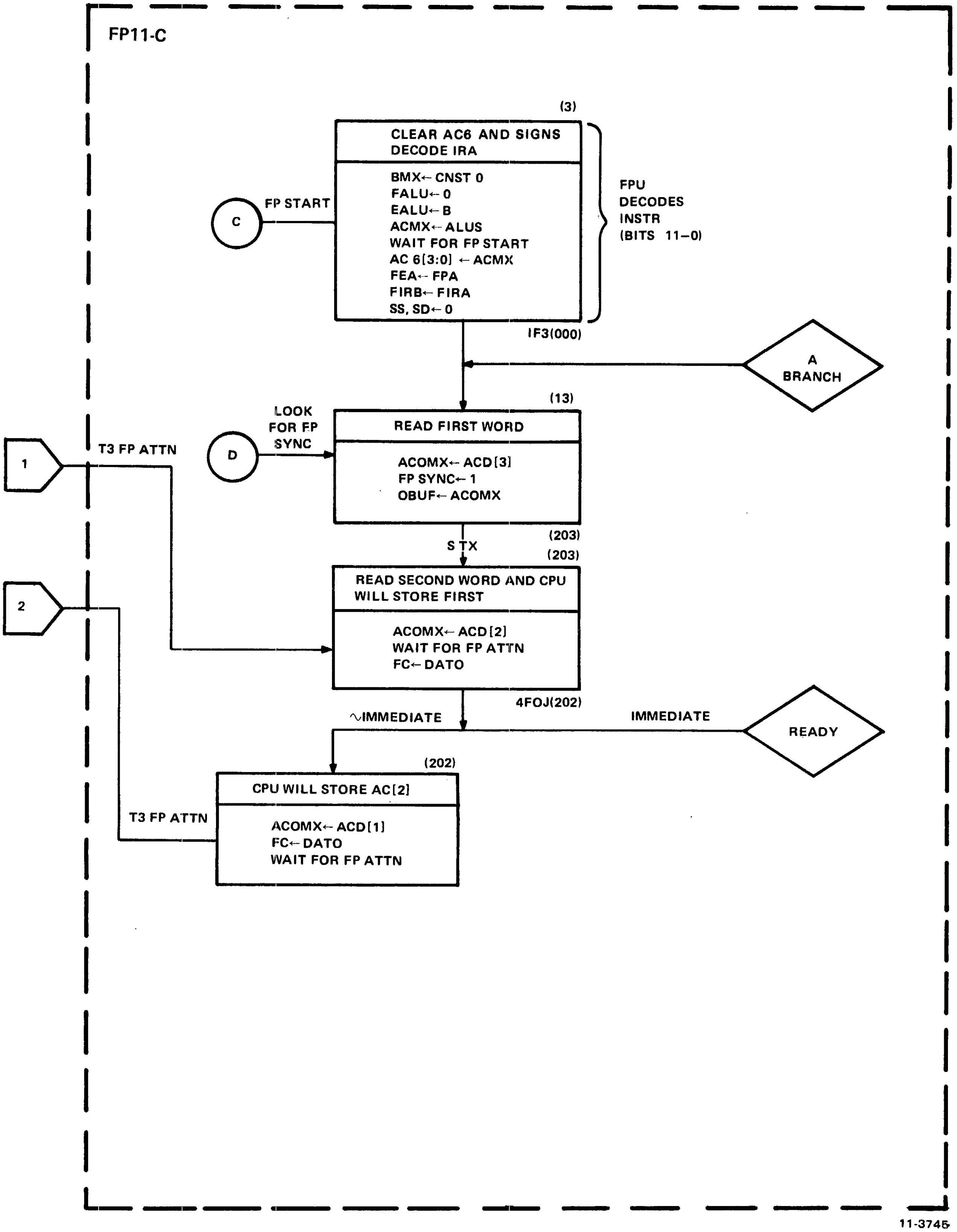
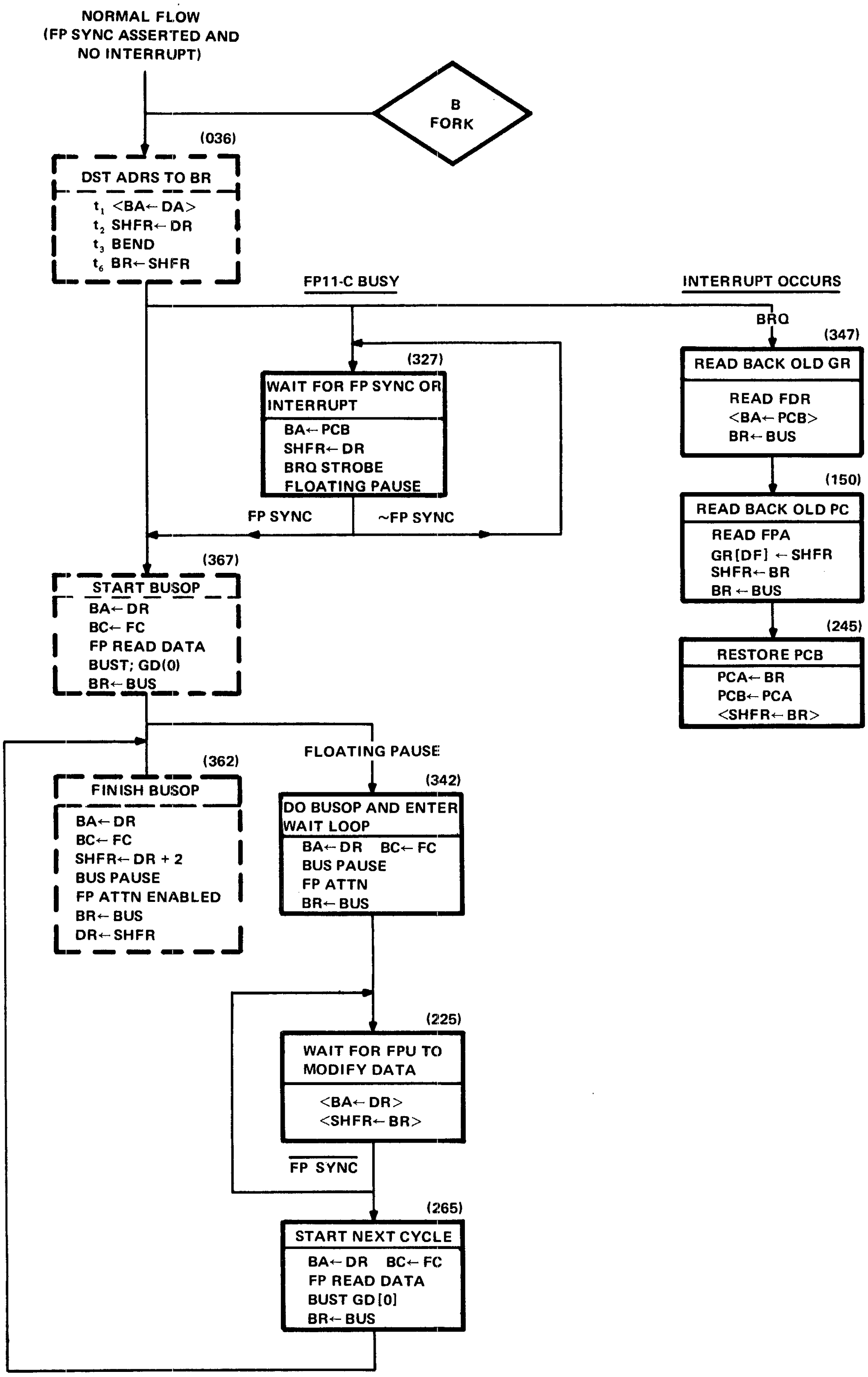


Figure 2-5 STF Instruction
Flow Diagram - Read from FP11-C
(Sheet 2 of 2)



11-3741

Figure 2-6 FP11-C Busy, Interrupt, and Floating Pause Flow Diagram

2.4.4 Interrupt Operation

The BRQ STROBE signal in state 327 (Figures 2-4 and 2-6) is a signal which clocks interrupt requests from peripherals, floating-point requests, and trap requests, and arbitrates them.

When an interrupt occurs (BRQ asserted), the CPU sequences to state 347. At this point, the CPU has already fetched the instruction, calculated the address and is ready to transfer the data. It is necessary to restore the PC and the general register to the values they contained prior to the address calculation. The values of the PC and the general register were temporarily stored in the FP11-C for the specific purpose of saving their contents in case of an interrupt. The contents of the PC are transferred to the FP11-C in state FET 10. The contents of the general register are transferred to the FP11-C in states 101 and 314.

When the interrupt is raised, the contents of the general register before the address calculation are read back to the CPU in state 347, and the contents of the PC prior to the address calculation are read back in state 150. State 245 restores the PCB in the CPU by transferring the contents of the BR (containing the FPA) to the PCB via the PCA.

This procedure of temporarily storing the contents of the PC and the general register in the floating-point FPA register and the FDR, respectively, allows the CPU to fetch instructions and do the entire address calculation before interacting with the FP11-C. However, the overhead necessary to accomplish this overlap involves the additional microcode necessary to reroute the PC and general register between the CPU and the FP11-C.

After the contents of the PC and general register have been transferred back to the CPU due to the pending interrupt, the PC and PS are pushed on the stack. The CPU enters an interrupt service routine to service the interrupt, and, upon completion of the service routine, executes an RTI which pops the PC and SP off the stack which yields the address present prior to the interrupt. The CPU now refetches the floating-point instruction and waits for FP SYNC from the FP11-C, indicating that the FP11-C is ready to send or receive data.

NOTE

If FP SYNC and the interrupt occur simultaneously, the interrupt will have higher priority and the CPU will proceed to service the interrupt and abort the floating-point instruction that was just fetched.

2.4.5 Floating-Pause Operation

The floating-pause branch (states 367, 342, 225, and 265 in Figure 2-6) is used exclusively with the NEG and ABS instructions where mode 0 is not specified. In the NEG instruction, the sign bit, which is the most significant bit in the first word of the instruction, is complemented. In the ABS instruction, the sign bit is made equal to 0. (See chapter 3 for a description of instructions.)

Both instructions require two memory references – one memory reference to read the first word from memory into the FP11-C, a pause state to modify the sign bit, and a second memory reference to transfer the word with modified sign bit back to memory. States 367 and 342 read the first word from memory. State 225 is the pause state where the sign bit is modified by the FP11-C and written back into memory. State 265 is the start of the next bus cycle to transfer the next word.

absolute

NOTE

If the exponent is 0 for the NEG or ABS instruction, the fraction is cleared, resulting in all 0s.

2.4.6 Destination Mode 0 Operation

Mode 0 instructions are instructions which do not require a memory reference. Examples are accumulator-to-accumulator transfers, and copy condition codes.

Assume an Add instruction is specified in which it is desired to add the contents of AC2 to AC3. AC2 and AC3 are scratchpad accumulators used for internal temporary storage in the FP11-C.

The CPU fetches the Add instruction, decodes it, transfers the contents of the PC, the instruction, and the general register to the FP11-C (states 343, 101, and 314 in Figure 2-4), and issues FP START. After sequencing out of state 314, the CPU comes to a large branching network called C-FORK. If the instruction is a mode 0 floating-point instruction, the CPU senses that no more data is transferred to the FP11-C and sequences to state 211 as shown in Figure 2-7. The CPU monitors the FP11-C for an FP SYNC signal, indicating that the FP11-C is ready to send or receive data.

When the CPU receives the FP SYNC signal from the FP11-C, it issues a second FP START signal which directs the FP11-C to execute the Add instruction (state 173). The reason for the second FP START signal is to inform the FP11-C that the CPU has not gone off to service the interrupt and to order the FP11-C to execute the Add instruction. Without this signal, the FP11-C would not know whether the CPU was servicing an interrupt and would repeat execution of the Add instruction, resulting in erroneous information.

If the instruction was a store operation that transferred data to a CPU general register (Chapter 3) and if FP REG WRITE is asserted, the data is written into 1 of 16 general registers in the CPU. If FP REG WRITE is negated, the CPU proceeds to the fetch state to fetch the next instruction.

2.4.7 Destination Mode 0 with Interrupt Sequence

If an interrupt occurs prior to execution of a mode 0 instruction, the CPU will restore the contents of the PC from the FPA register in the FP11-C. The CPU originally stored the contents of the PC in the FPA register during state 260, which occurs prior to the address calculation. The PC is restored in states 153 and 245. Note that the contents of the general register need not be restored since mode 0 does not modify the general register contents, and consequently, only the PC must be restored during an interrupt routine.

NOTE

With destination mode 0, the contents of the general register are sometimes used as data rather than interrupt information. For example, the Load Convert Integer, Load Floating-Point Status, and Load Microbreak Register instructions cause 16 bits of data to be transferred from the general register to the FP11-C.

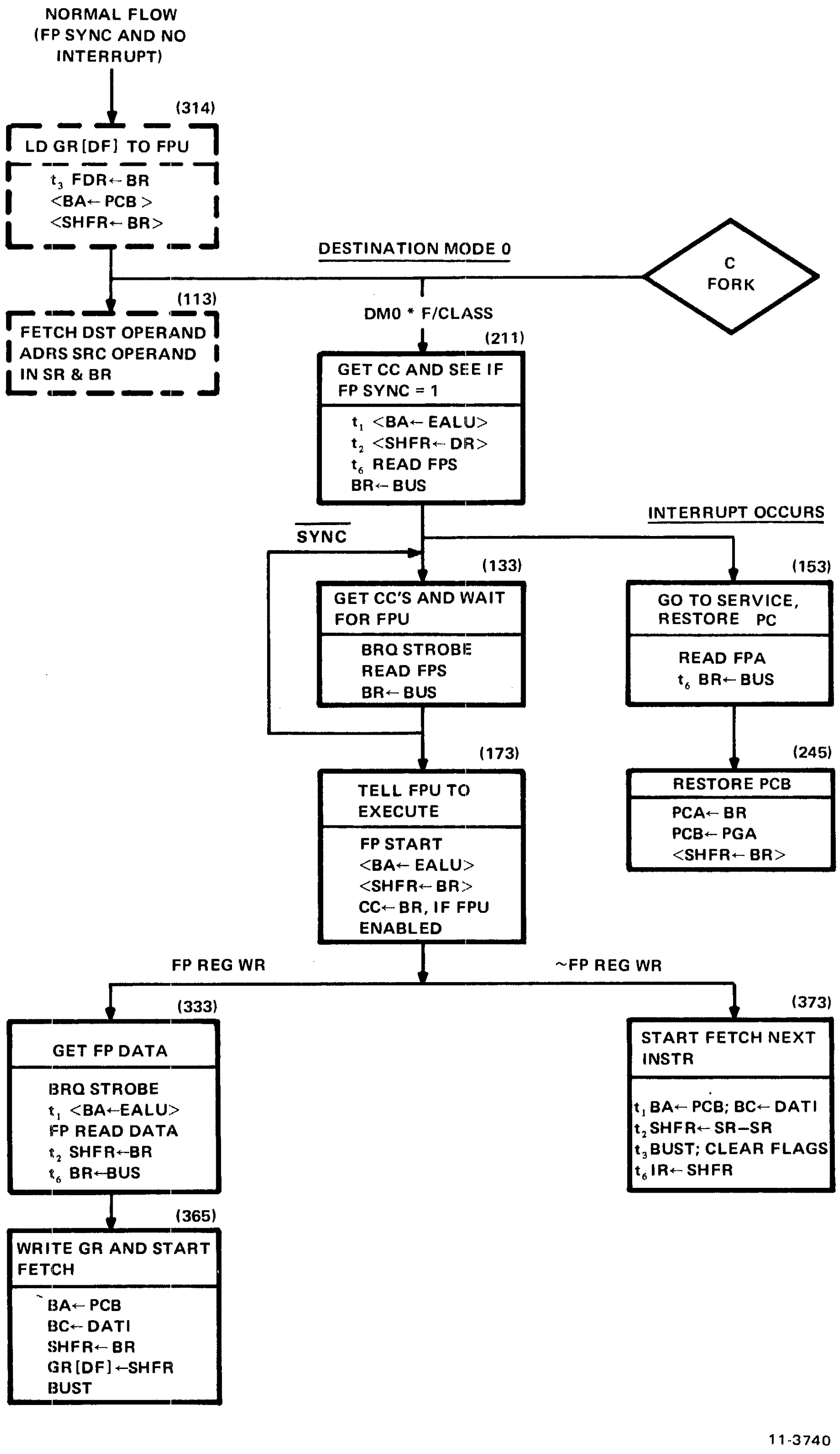


Figure 2-7 Destination Mode 0 and/or Interrupt Flow Diagram

CHAPTER 3
DATA AND DATA FORMATS

3.1 FP11-C DATA FORMATS

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

3.1.1 FP11-C Integer Format

Integer format is represented in 2's complement notation in the FP11-C. 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 −5 in both formats.

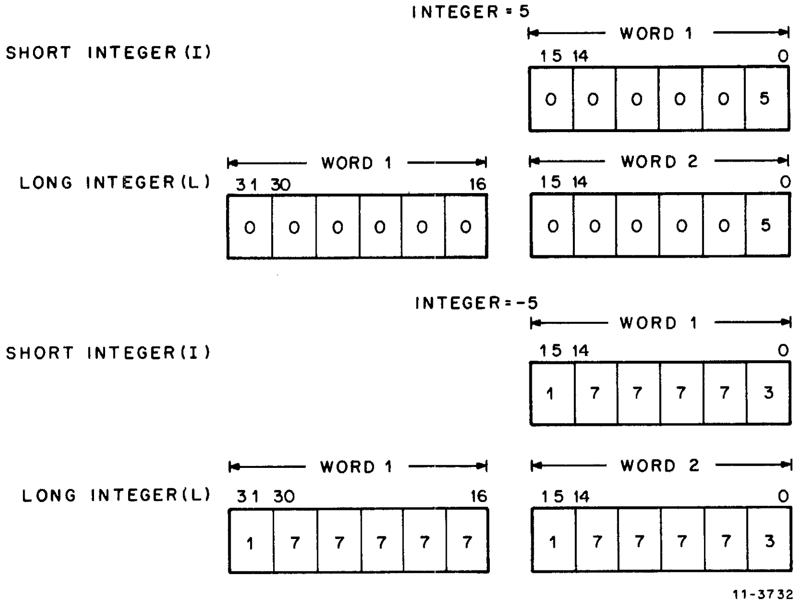
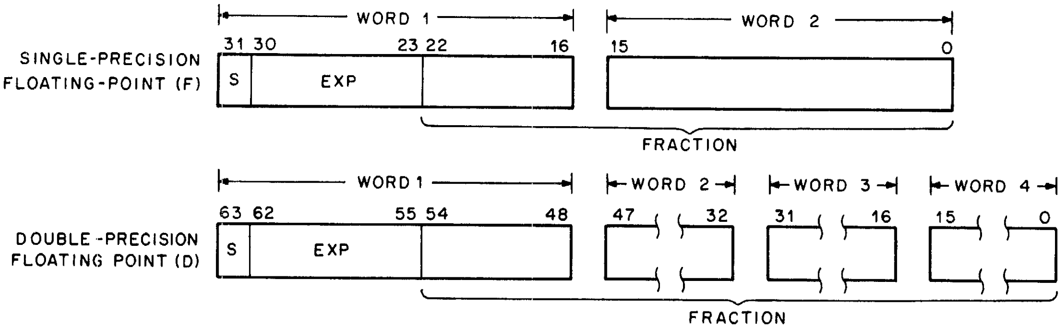


Figure 3-1 Integer Formats

3.1.2 FP11-C Floating-Point Formats

The single-precision floating-point format is 32 bits long and is designated by F; the double-precision (extended) format is 64 bits long and is designated by D. All floating-point numbers are assumed to be normalized. The 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.



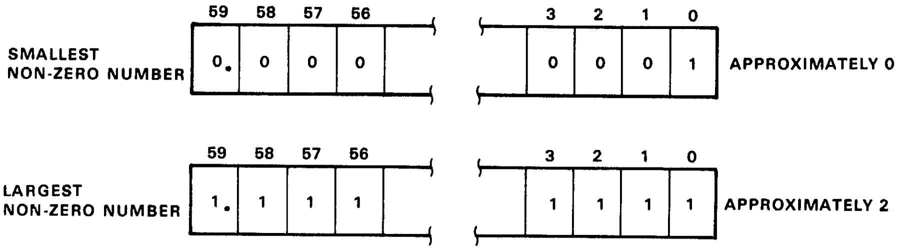
S = Sign
EXP = Exponent in excess 200 (8) notation (refer to paragraph 3.1.4)
Fraction = 23 or 55 bit fraction in sign and magnitude format. Binary point between bits 22 and 23 for F format or between bits 54 and 55 for D format.

11-3731

Figure 3-2 Floating-Point Data Formats

3.1.3 Floating-Point Fraction

Floating-point fractions have a range from approximately 0 through 2 as shown below.



11-3967

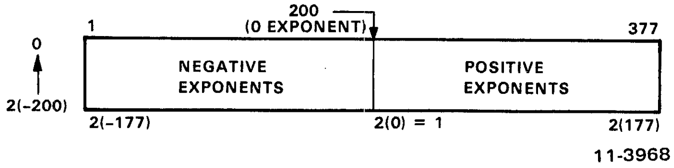
The implementation of the FP11-C only works with normalized numbers. Consequently, the FP11-C normalizes all unnormalized numbers. Normalized numbers are of the form 0.1000... to 0.111... with the zero to the left of the binary point representing an overflow bit. This bit can be set to a 1 during certain addition and subtraction operations. During addition, certain sums will produce an overflow such as 0.1000... + 0.1000... which yields 1.000.... In order to normalize the result in this case, the number must be right shifted one place and the exponent must be increased by one. During subtraction, when a larger number is subtracted from a smaller number, the overflow bit will be forced to a 1. In this case, the overflow bit indicates that the value of the result is negative. The result must then be 2's complemented and a negative sign must be affixed to the number to indicate a negative result.

3.1.4 Transfer of Operands

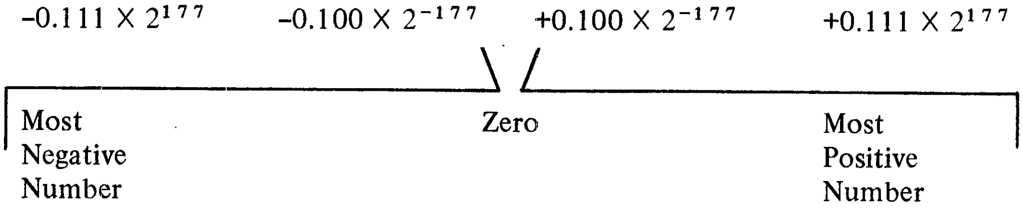
All operands transferred between the CPU and FP11-C are normalized and in sign and magnitude format, except during the execution of Load Convert Integer, Store Convert Integer, Load Exponent or Store Exponent instructions where integers are transferred to or from memory. Because in sign and magnitude 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 significant bit 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.

3.1.5 Floating-Point Exponent

The exponent in the FP11-C 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. A number with an exponent of −200 is treated by the FP11-C as 0. If the fraction is non-zero, it is forced to 0 by the FP11-C hardware.



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-C. Only three bits are shown for simplicity, but they can be extended to any number.



A number with an exponent less than 0 indicates an underflow condition, which means that the number is too small to be represented. A number with an exponent of more than 377 indicates an overflow condition, which means that the number is too large to be represented.

3.1.6 Interpretation of a Floating-Point Number

Operands or arguments stored in memory are assumed normalized and in sign and magnitude format.

Figure 3-3 shows the decimal number 32 represented in memory in sign and magnitude format. The FP11-C interprets the number as a floating-point number with sign, exponent, and fraction. Only one memory word is shown, which contains the sign, exponent, and upper bits of the fraction. An additional word from memory would be transferred to bits 50 through 35 for single-precision mode. For double-precision mode, two additional words from memory would be transferred to bits 34 through 19 and bits 18 through 03.

The lower-half of Figure 3-3 represents the decimal number 7/16 in memory and how it is interpreted by the FP11-C.

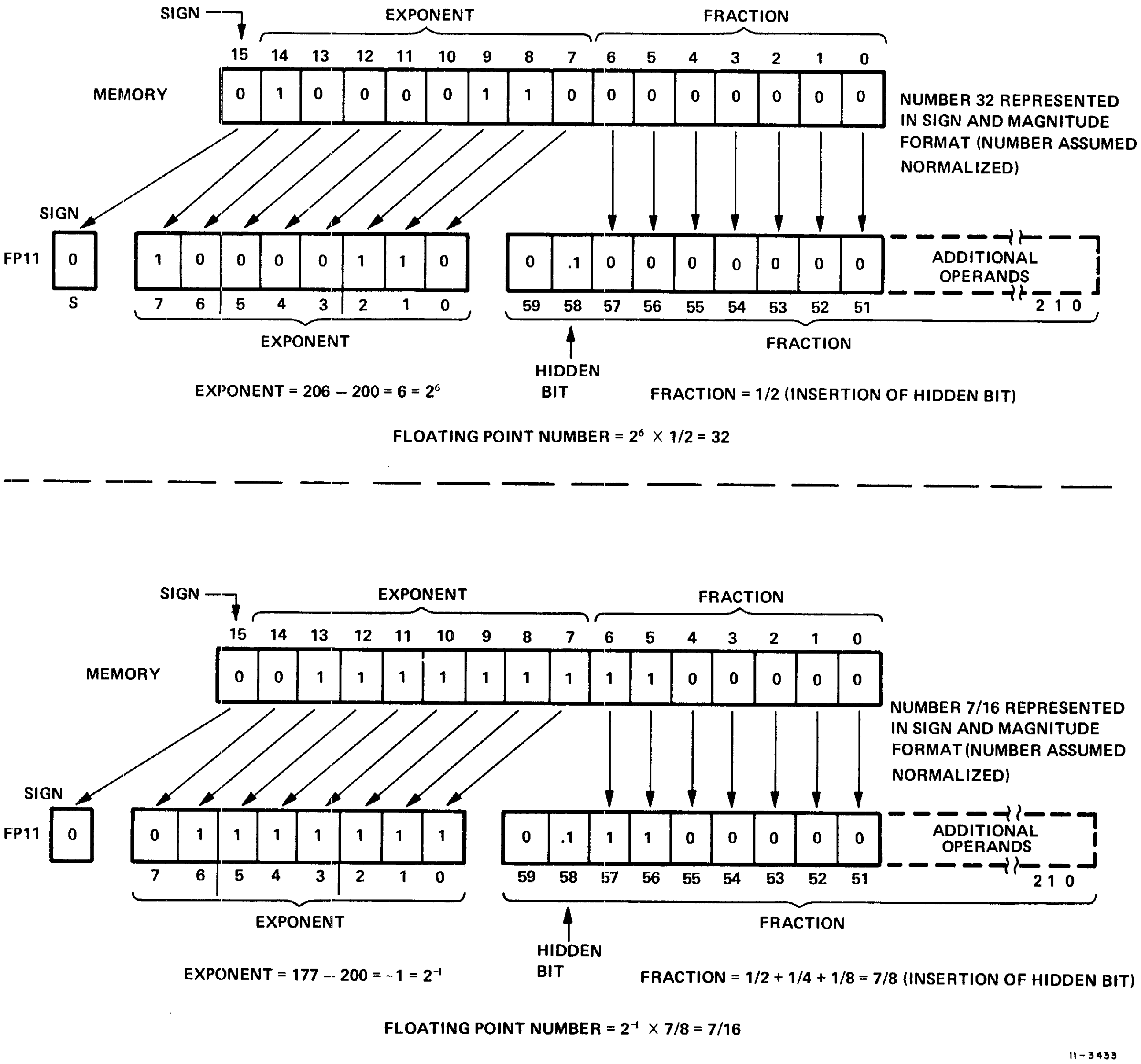


Figure 3-3 Interpretation of Floating-Point Numbers

The following instructions are an exception to the above description, in that an integer may be written into or read from memory.

Load Exponent – This instruction takes an integer from memory and creates an exponent in excess 200 notation.

Store Exponent – This instruction converts an exponent in excess 200 notation into a 2's complement integer.

Load Convert Integer to Floating – This instruction converts a 2's complement integer from memory to a floating-point number.

Store Convert Floating to Integer - This instruction converts a floating-point number to an integer and stores it in memory.

3.2 FP11-C PROGRAM STATUS REGISTER

The FP11-C contains a Program Status register; this register contains FP11-C 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 to enable various interrupt conditions. Figure 3-4 shows the layout of the Program Status register. Each bit shown in the figure is described below.

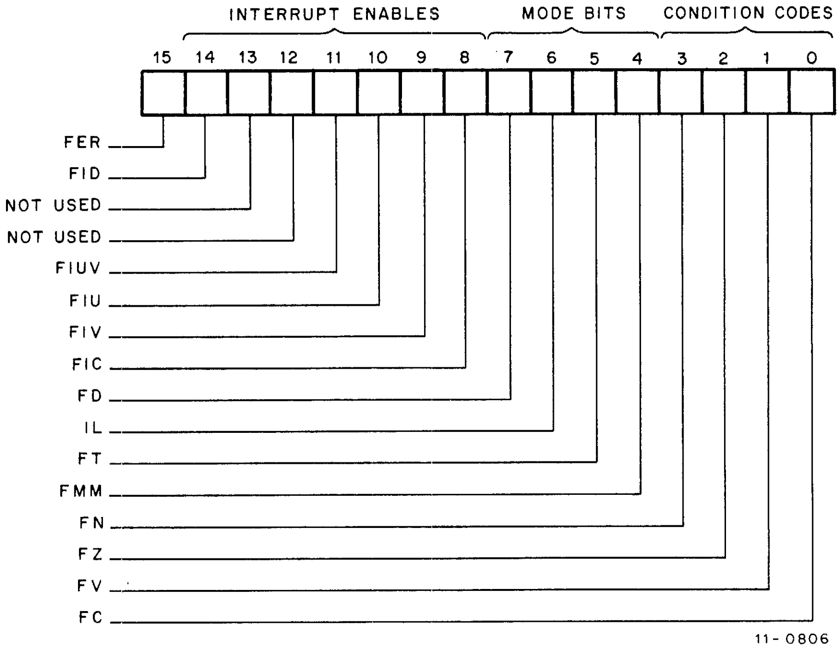


Figure 3-4 Status Register Format

FER - This bit indicates an error condition of the FP11-C.

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

FIUV (Floating Interrupt on Undefined Variable) - When this bit is set and a -0 is obtained from memory, an interrupt occurs. If the bit is not set, -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; if reset, single-precision 2's complement integer format 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-C Exception Code (FEC) Register. 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, with a brief description.

FP11-C Exception Code	**Definition**
2	Floating Op Code Error – The FP11-C causes an interrupt for an erroneous op code.
4	Floating Divide by Zero – Division by zero causes an interrupt. 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 FP11-C Program Status register. All traps are disabled if FID is set.

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

3.4 FP11-C INSTRUCTION FORMATS

The FP11-C instruction set is divided into the five formats shown in Figure 3-5.

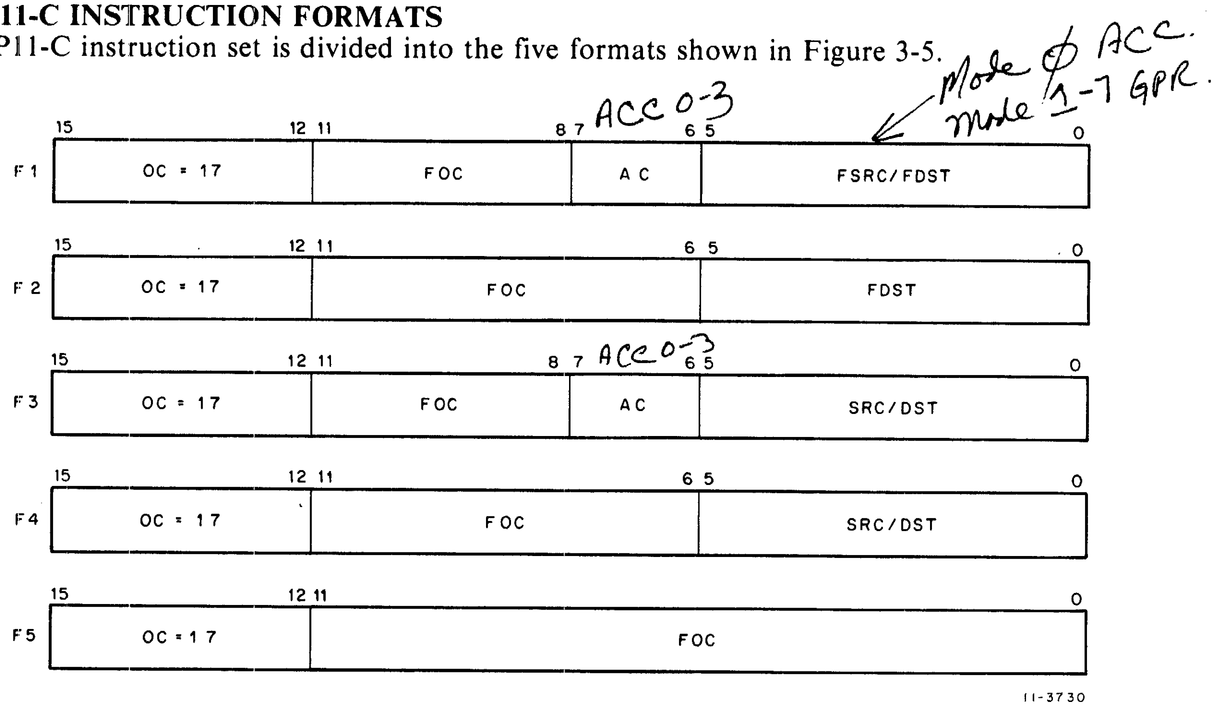


Figure 3-5 Instruction Formats

The 2-bit AC field (bits 06 and 07) allows selection of scratchpad accumulators 0 through 3 only. If address mode 0 is specified with formats F1 or F2, bits 02 through 00 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-C traps if the interrupt is enabled.

The fields of the various instruction formats (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; the code 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.

Mnemonic   Description

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-C Instructions

Instruction Format	Instruction	Mnemonic
F1	ADD	ADDF FSRC, AC ADDD FSRC, AC
F1	LOAD	LDF FSRC, AC LDD FSRC, AC
F1	SUBTRACT	SUBF FSRC, AC SUBD FSRC, AC
F1	COMPARE	CMPF AC, FDST CMPD AC, FDST
F1	MULTIPLY	MULF FSRC, AC MULD FSRC, AC
F1	MODULO	MODF FSRC, AC MODD FSRC, AC
F1	STORE	STF AC, FDST STD AC, FDST
F1	DIVIDE	DIVF FSRC, AC DIVD FSRC, AC
F1	LOAD CONVERT	LDCFD FSRC, AC LDCDF FSRC, AC
F1	STORE CONVERT	STCFD AC, FDST STCDF AC, FDST
F2	CLEAR	CLRF FDST CLRD FDST
F2	TEST	TSTF FDST TSTD FDST
F2	ABSOLUTE	ABSF FDST ABSD FDST
F2	NEGATE	NEGF FDST NEGD FDST
F3	LOAD EXPONENT	LD EXP SRC, AC
F3	LOAD CONVERT INTEGER TO FLOATING	LDCIF SRC, AC LDCID SRC, AC LDCLF SRC, AC LDCLD SRC, AC
F3	STORE EXPONENT	STEXP AC, DST

Table 3-1 Format of FP11-C Instructions (Cont)

Instruction Format	Instruction	Mnemonic
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
F4	STORE FP11's PROGRAM STATUS	STFPS DST
F4	STORE FP11's STATUS	STST DST
F5	COPY FLOATING CONDITION CODES	CFCC
F5	SET FLOATING MODE	SET F
F5	SET DOUBLE MODE	SET D
F5	SET INTEGER MODE	SET I
F5	SET LONG INTEGER MODE	SET L
F5	LOAD UBREAK REGISTER	LDUB
F5	STORE AR REGISTER IN AC0	STA0
F5	MAINTENANCE SHIFT BY N	MSN
F5	STORE QR REGISTER IN AC0	STQ0

3.5 INSTRUCTION SET

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

1. 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 designated by D. Examples are NEG F, NEG D, and SUB D.

2. 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. Examples are SET I and SET L.

3. Several convert type instructions use the symbology defined below.

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

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

   $C_{F,D}$ or $C_{D,F}$ – Convert single-floating to double-floating or double-floating to single-floating

4. Numbers in parentheses indicate bit positions; an example is AR (57:00), which indicates AR bits 57 through 00.

5. 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 the 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.

Table 3-2 Instructions Set

Mnemonic	Instruction Description	Octal Code

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

SETF	Set Floating Mode FPS Single Floating FD ← 0 32Bits	170001 F5 Format
SETI	Set Integer Mode Integer 16 Bits 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-C is in maintenance mode (FMM = 1), an interrupt is also generated and the FPU traps to the Ready state. (See Chapter 7.) traps will trap to ROM State 2 (Micromatch state) flows 2	170003 F5 Format
STA0	Store AR in AC0 ~~AC0 (54:32) ← AR (57:35) if FD = 0~~ } if FP-11B AC0 (54:0) ← AR (57:3) ~~if FD = 1~~	170005 F5 Format

~~MRS~~ MSN	Maintenance Shift by N check the string of hidden Bit
	AR ← AR shifted by 0-7 left or 0-8 right. and operation
	QR ← QR shifted by 0-7 left or 0-8 right. of shifters.
		170004
		
		F5 Format

STQ0	Store QR in AC0 ~~BR ← QR; AC (54:32) ← BR (57:35) if FD = 0~~ } if FP-11B AC0 (54:0) ← QR (57:3) ~~if FD = 1~~ during in rounding process.	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-C's Program Status Word FPS ← (SRC)	170100+SRC F4 Format

170137
Mode 3
GPR Reg 7

Table 3-2 Instructions Set (Cont)

Mnemonic	Instruction Description	Octal Code
STFPS DST	Store FP11-C's Program Status Word DST ← (FPS)	170200+DST F4 Format
STST DST	Store FP11-C's Status DST ← (FEC) DST + 2 ← (FEA) if not mode 0 or not immediate mode	170300+DST F4 Format 170425

CLRF FDST
CLRD FDST
	Clear
	FDST ← 0
	FC ← 0
	FV ← 0
	FZ ← 1
	FN ← 0
		170400+FDST
		F2 Format
		MODE 2
		GPRReg 5
		mode Φ
		mode Φ
		AC 2
		1704Φ2

TSTF FDST
TSTD FDST
	Test
	FDST ← (FDST)
	FC ← 0
	FV ← 0
	FZ ← 1 if (FDST) = 0; otherwise FZ ← 0
	FN ← 1 if (FDST) ⩽ 0; otherwise FN ← 0
		170500+FDST
		F2 Format

ABSF FDST
ABSD FDST
	Absolute
	FDST ← minus (FDST) if FDST ⩽ 0; otherwise FDST ← (FDST)
	FC ← 0
	FV ← 0
	FZ ← 1 if (FDST - 0; otherwise FZ ← 0
	FN ← 0
		170600+FDST
		F2 Format

NEGF FDST
NEGD FDST
	Negate
	FDST ← minus (FDST)
	FC ← 0
	FV ← 0
	FZ ← 1 if (FDST) = 0; otherwise FZ ← 0
	FN ← 1 if (FDST) ⩽ 0; otherwise FN ← 0
		170700+FDST
		F2 Format
		Bit 8 all ways on

LDEXP SRC, AC	Load Exponent
	AC SIGN ← (AC SIGN)
	AC EXP ← (SRC) + 200
	AC FRACTION ← (AC FRACTION)
	FC ← 0 upper limit Exponent 377 all 1's in fraction
	FV ← 1 if |AC| ⩽ UPLIM; otherwise FV ← 0
	FZ ← 1 if (AC) = 0; otherwise FZ = 0 or FZ ← 0
	FN ← 1 if (AC) ⩽ 0; otherwise FN = 0 or FN ← 0
		176400+AC*100+SRC
		F3 Format
		AC 2
		1764 200
		200
		600

176626

AC = 2

FSRC MODE = 2
GPR = 6

Table 3-2 Instructions Set (Cont)

Mnemonic	Instruction Description	Octal Code Bit 8 [illegible] 1

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; otherwise FZ ← 0
	FN ← 1 if (AC) ⩽ 0; otherwise 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.
		17700+AC~~*100~~+SRC
		F3 Format

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

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; otherwise FZ ← 0
	FN ← 1 if (DST) ⩽ 0; otherwise 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

Table 3-2 Instructions Set (Cont)

Mnemonic	Instruction Description	Octal Code

STF AC, FDST
STD AC, FDST
	Floating Store
	FDST ← (AC)
	FC ← FC
	FV ← FV
	FZ ← FZ
	FN ← FN
	move from 0 to 5 To pick up. Accumulator 5.
		174000+AC*100+FDST
		F1 Format

DIVF FSRC, AC
DIVD FSRC, AC
	Floating Divide
	AC ← (AC) / (FSRC) if | (AC) / (FSRC) | ⩾ LOLIM; otherwise
	AC ← 0
	FC ← 0
	FV ← 1 if | AC | ⩾ UPLIM
	FZ ← 1 if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) ⩽ 0; otherwise FN ← 0
	001 Exponent are 0's fraction except for hidden bit
		174400+AC*100+FSRC
		F1 Format

LDCDF FSRC, AC
LDCFD FSRC, AC
	Load Convert Double to Floating or Floating to Double
	AC ← C (F, DvD, F) (FSRC)
	FC ← 0
	FV ← 1 if |AC| ⩾ UPLIM; otherwise FV ← 0
	FZ ← 1 if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) ⩽ 0; otherwise 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 |AC| + (FSRC) ⩽ LOLIM; otherwise
	AC ← 0
	FC ← 0
	FV ← 1 if |AC| ⩾ UPLIM; otherwise FV ← 0
	FZ ← 1 if (AC)= 0; otherwise FZ ← 0
	FN ← 1 if (AC) ⩽ 0; otherwise FN ← 0
		17200+AC*100+FSRC
		F1 Format

LDF FSRC, AC or
LDD FSRC, AC
	Floating Load
	AC ← (FSRC)
	FC ← 0
	FV ← 0
	FZ if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) ⩽ 0; otherwise FN ← 0
		172400+AC*100+FSRC
		F1 Format
		accumulator 0-3 only

Table 3-2 Instructions Set (Cont)

Mnemonic	Instruction Description	Octal Code

SUBF FSRC, AC or
SUBD FSRC, AC
	Floating Subtract
	AC ← (AC) - (FSRC) if $\lvert(AC) - (FSRC)\rvert \geqslant$ LOLIM
	otherwise AC ← 0
	FC ← 0
	FV ← 1 if $\lvert AC\rvert \geqslant$ UPLIM; otherwise FV ← 0
	FZ ← 1 if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) ⩽ 0; otherwise 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; otherwise FZ ← 0
	FN ←1 if (FSRC) - (AC) ⩽ 0; otherwise FN ← 0
		173400+AC*100+FSRC
		F1 Format

MULF FSRC, AC
MULD FSRC, AC
	Floating Multiply
	AC ← (AC) * (FSRC) if $\lvert(AC) * (FSRC)\rvert \geqslant$ LOLIM;
	otherwise AC ← 0
	FC ← 0
	FV ← 1 if $\lvert AC\rvert \geqslant$ UPLIM; otherwise FV ← 0
	FZ ← 1 if (AC) = 0; otherwise FZ ←0
	FN ← 1 if (AC) ⩽ 0; otherwise 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
	$\lvert(AC)*(FSRC)\rvert \geqslant$ LOLIM or FIU = 1; otherwise AC ←0
	FC ← 0
	FV ← 1 if $\lvert AC\rvert \geqslant$ UPLIM; otherwise FV ← 0
	FZ ← 1 if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) ⩽ 0; otherwise FN ←0
	
	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.
		171400+AC*100+FSRC
		F1 Format

STCFD AC, FDST
STCDF AC, FDST
	Store Convert from Floating to Double or Double to Floating
	FDST ← C (F, DVD,F) (AC)
	FC ← 0
	FV ← 1 if $\lvert AC\rvert \geqslant$ UPLIM; otherwise FV ← 0
	FZ ← 1 if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) ⩽ 0; otherwise FN ← 0
		176000+AC*100+FDST
		F1 Format
		F, D-single-precision to double-precision floating
		D, F-double-precision to single-precision floating

6. 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 1: LD FPS Instruction

Mode 3, register 7 specified (F instruction format).

170100+SRC
SRC field is equal to 37
Basic op code is 170100
SRC and basic op code are added to yield 170137.

Example 2: LDF Instruction

AC2, mode 2, and register 6 specified (F1 instruction format).

172400+C*100+FSRC
AC = 2
2*100 = 200

172400 + 200 = 172600
FSRC is equal to 26

172600 + 26 = 172626

The information in Table 3-2 is provided in symbolic notation to provide the reader with a quick reference to the function of each instruction. The following paragraphs supplement the information in Table 3-2 by providing a description as to how the arguments are routed in an instruction, where the result is stored, etc. The reader should be familiar with the scratchpad accumulators described in Chapter 4.

To summarize, the source accumulator (ACS) can be accumulator 0, 1, 2, 3, 4, or 5 for mode 0 (non-memory references) instructions while the destination accumulator (ACD) can be accumulator 0, 1, 2, or 3. For instructions other than mode 0 instructions, memory data is loaded into AC6. ACS is then forced to be AC6, while the destination accumulator will be accumulator 0, 1, 2, or 3.

3.5.1 Arithmetic Instructions

For the Arithmetic instructions (add, subtract, multiply, divide) and the Compare instruction, one argument is in the source accumulator and one argument is in the destination accumulator. The instruction is executed and the result (except for the CMP instruction) is stored in the destination accumulator.

For the CMP instruction, the two arguments remain in the respective accumulators and there is no transfer of the result to the destination accumulator.

3.5.2 Floating Modulo Instruction

The Floating Modulo (MOD) instruction multiples two arguments and treats the product as a sign and magnitude floating-point number. The number is separated into a whole number and a fraction, with the whole number portion going into an odd accumulator and the fraction going into an even accumulator.

The whole number portion of the number contains an exponent greater than 201 in excess 200 notation, which means that the whole number has a decimal value of some number equal to or greater than one and less than N, where N is the greatest possible number that can be represented by the FP11-C.

Since the fractional portion of the number is normalized, the fraction must be equal to or greater than 0.5.

For example, assume the arguments $36.542_{10} \times 10_{10}$, which yield a product of 365.42. The 365 represents the whole number and the 42 represents the fraction. Now move the decimal point three places to the left, keeping track of the fact that this is division by 1000. Execution of the MOD instruction would strip off the first digit (3, in this case) and move the decimal point one place to the right (which is multiplying by 10). Continue this process for another two digits at which point the decimal would be where it was upon completion of the multiplication process.

3.5.3 Load Instruction

The Load instruction takes an argument from memory or from a source accumulator and copies it into a destination accumulator.

If a memory reference instruction (not mode 0) is specified, the CPU transfers the word from memory to the BR and issues a data strobe. This causes 16 bits of data to be loaded in the FDR in the FP11-C and forces the FP11-C out of the Wait state. The FP11-C executes one microstate, which stores the first word in AC6[3], since AC6 is the source accumulator for memory reference instructions. The FP11-C then goes into the Wait state. Meantime, the CPU has fetched the next word, sent it to the BR, and issued data strobe, which transfers the next word to the FDR and again forces the FP11-C out of the Wait state. The FP11-C executes another microstate, which stores the second word in AC6[2]. This process continues until the FP11-C drops the REQ line, which indicates that the required number of words have been transferred. For single-precision mode, two words will be transferred and for double-precision mode, four words will be transferred.

When the required words have been transferred, the FP11-C reads the source accumulator in the next microstate, which transfers the sign to the SS flip-flop, the exponent to the ER, and the fraction to the AR. In the following microstate, the FP11-C stores the contents of SS, ER, and AR into the destination accumulator.

For mode 0 instructions, operation is similar except that the data is already contained in ACS and is not transferred in from memory.

NOTE

If immediate mode is employed, only one 16-bit data word is read into the FP11-C. This word is read into AC6[3]. In the next microstate, the FP11-C reads the AC, which transfers the sign to the SD flip-flop, the exponent to the ER, and the upper bits of the fraction to the AR. In the next microstate, the FP11-C stores the word into the destination accumulator (AC2 in this case).

3.5.4 Store Instruction

The Store instruction transfers data from a selected scratchpad to memory. The ACOMX in the FP11-C is selected for quadrant 3 from OBUF. The first word to be transferred to memory is transferred from the scratchpad to OBUF and the second word is transferred to the input of OBUF. The CPU reads in the first word from OBUF and stores it in memory. As soon as it has accomplished this, it clocks OBUF, which transfers the second word into OBUF. The FP11-C transfers the third word to the input of OBUF. The CPU accepts the second word and stores it in memory. If single-precision mode is specified, the FP11-C negates the Request line and the third word will never be transferred. If double-precision mode is specified, the Request line will be asserted for an additional two words and a process similar to that described is repeated until all four words are transferred.

3.5.5 Load Convert (Double-to-Floating, Floating-to-Double) Instructions

The Load Convert Double-to-Floating (LDCDF) instruction assumes the source is a double-precision floating-point number and converts that number to single-precision. If the floating truncate bit is set, the number is truncated. If the bit is not set, the number is rounded by adding a 1 to the single-precision segment, provided that the MSB of the double-precision segment of the word is a 1, as shown in Figure 3-6. If the MSB is a 0, the single-precision word remains unchanged.

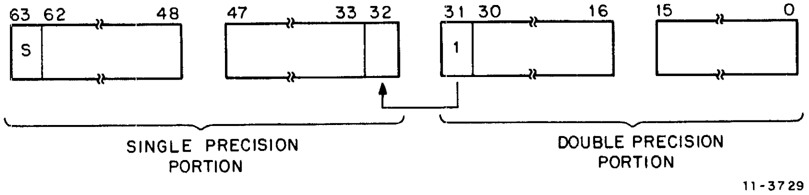


Figure 3-6 Double-to-Single Precision

This instruction requires three bus cycles for instructions that are not mode 0 instructions: two for the single-precision segment of the word and the third to examine the MSB of the double-precision segment (bit 31) to determine if rounding is to occur.

The Load Convert Floating-to-Double (LDCFD) instruction assumes the source is a single-precision number and converts that number to double-precision by appending 32 zeros to the single-precision word.

For the Load Convert instructions, the number to be converted is originally in the source accumulator and is transferred to the destination accumulator after the conversion.

3.5.6 Store Convert (Double-to-Floating, Floating-to-Double) Instructions

The Store Convert Double-to-Floating (STCDF) instruction converts a double-precision number located in the destination accumulator to a single-precision number and transfers it to the source accumulator. If the floating truncate bit is set, the floating-point number is truncated. If the bit is not set, the number is rounded. If the MSB (bit 31) of the double-precision segment of the word is a 1, 1 is added to the single-precision segment of the word (Figure 3-6); otherwise, the single-precision segment remains unchanged.

The Store Convert Floating to Double (STCFD) instruction converts a single-precision number located in the destination accumulator to a double-precision number and transfers it to the source accumulator. The single-to-double precision is accomplished by appending zeros equivalent to the double-precision segment of the word as shown in Figure 3-7.

The Store Convert instructions store the number to be converted originally in the destination accumulator and transfer the result to the source accumulator after the conversion.

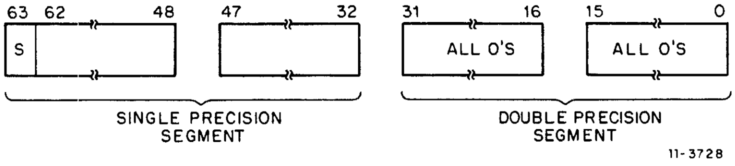


Figure 3-7 Single-to-Double Precision

3.5.7 Clear Instruction

The Clear instruction clears a floating-point number which may be stored in memory or in an accumulator. If mode 0 is specified, the FP11-C microcode clears the exponent field by selecting a constant of 0 and writing the source accumulator. The FP11-C microcode clears the sign by selecting SD ← 0 to zero the sign. The FP11-C microcode clears the fraction by selecting FALU ← 0 which presents 0s at the output of the FALU and writes 0s into the fraction scratchpad.

If mode 0 is not specified, AC6 is the source accumulator and is cleared every time the FP11-C sequences through the Ready state. In order to clear memory, then, it is only necessary to write AC6 to memory.

3.5.8 Test Instruction

The purpose of the Test instruction is to test the sign and exponent of a floating-point number. If mode 0 is specified, the exponent and sign are read from the source accumulator and transferred through ACMX via the exponent scratchpad. At the output of ACMX, the FCC (Floating Condition Code) is set to 0, which disregards the C and V bits and sets the N and/or Z bits in accordance with the sign and exponent of the source accumulator. For example, if the sign were positive and the exponent zero, the N bit would be unasserted and the Z bit would be asserted.

If mode 0 is not specified, a 16-bit word is read from memory and is applied to ACMX via the DIMX. The sign and exponent are monitored as the word is being transferred from ACMX to the exponent scratchpads. For this mode, a minus zero trap is enabled which will cause an interrupt if the word from memory is an undefined variable (a negative sign with an exponent of 0).

3.5.9 Absolute Instruction

The purpose of the Absolute instruction is to take the absolute value of a floating-point number by making the sign bit equal to 0.

If mode 0 is specified, the sign of the number in the source accumulator is made equal to 0. The exponent of the number is tested in the ER. If the exponent is 0, a 0 is written into the source accumulator. If the exponent is non-zero, the fraction and exponent are restored to the source accumulator.

If mode 0 is not specified, the sign bit in memory is zeroed. A word is transferred from memory to AC6[3]. The exponent of this word is tested in the ER. If the exponent is 0, the entire operand is made equal to 0. This will be a 2-word operand for single precision or a 4-word operand for double-precision. If the exponent is non-zero, the original fraction and exponent are restored to memory.

Negate and Absolute instructions are the only instructions which can read and write a memory location.

3.5.10 Negate Instruction

The purpose of the Negate instruction is to complement the sign of the operand.

If mode 0 is specified, the sign of the number in the source accumulator is complemented. The exponent of the number is tested in the ER. If the exponent is 0, a 0 is written into the source accumulator. If the exponent is non-zero, the fraction and exponent are restored to the source accumulator.

If mode 0 is not specified, the sign bit in memory is complemented. A word is transferred from memory to AC6[3]. The exponent of this word is tested in the ER. If the exponent is 0, the entire operand is made equal to 0. This will be a 2-word operand for single- precision or a 4-word operand for double-precision. If the exponent is non-zero, the fraction and exponent are restored to memory.

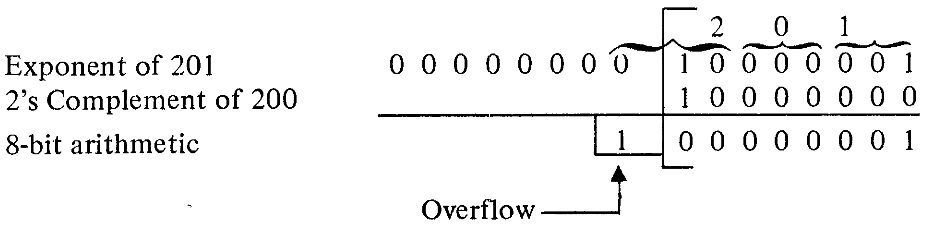
3.5.11 Load Exponent Instruction

The Load Exponent instruction (LD EXP) is used to change the value of an exponent of a number loaded into the FP11-C. This is accomplished by taking the 16-bit, 2's complement number from memory, adding the constant of 200 (since the exponent field in the FP11-C must be expressed in excess 200 notation), and storing the result in the 8-bit exponent field of the destination accumulator (ACD). The exponent processor in the FP11-C is only 10 bits wide (8 bits of exponent, 1 bit of sign, and 1 overflow bit). The question is now raised how the 16-bit number from memory is processed by a 10-bit exponent processor and stored in an 8-bit exponent field in the FP11-C. In order to accomplish this, the FP11-C hardware performs some functions which are not immediately obvious. The following paragraphs attempt to describe the operation.

First, the possible legal range of numbers in memory range from 000000 to 177777_8. The possible legal range of exponents in the FP11-C falls into two classes: 0 through 177 (when 200 is added to any of these numbers, the sum stays within the legal 8-bit exponent field - from 200 to 377) and 177601 through 177777 (when 200 is added to any of these numbers, the sum stays within the legal 8-bit exponent field - from 1 to 177).

Any other number from memory is illegal and will result in either an overflow or an underflow condition. Several examples follow to clarify this concept.

Example 1: LD EXP 000201



This number (201), when added to the constant of 200 in the FP11-C, yields a result which is larger than the 8-bit capacity of the exponent field and is designated an overflow condition.

Example 2: LD EXP 100200

```
                                                 2     0     0
Exponent = 100200          1 0 0 0 0 0 0 0   |  1 0 0 0 0 0 0 0
2's Complement of 200                    +   |  1 0 0 0 0 0 0 0
                           ---------------------------------------
                                       | 1 | |  0 0 0 0 0 0 0 0
                                         ^
                           Underflow ----┘
```

This number (100200), when added to the constant of 200 in the FP11-C, yields a result which is more negative than can be expressed by the 8-bit exponent field and is designated an underflow condition.

If a programmer inadvertently loads a number which cannot be represented in the 8-bit exponent field, the FP11-C hardware will detect the condition and will flag the system with either an overflow or underflow trap. The following paragraphs describe how the FP11-C hardware determines whether the number is legal, too large, or too small to be represented by the FP11-C.

When the 2's complement, right-justified number in memory is transferred to the FP11-C, the FP11-C expects a floating-point number and strips off bit 15 as a sign bit, bits 14 through 07 as the exponent field, and bits 06 through 00 as part of the fraction field. Bit 15 is routed to the sign scratchpad, bits 14 through 07 are routed to AC6[3], and bits 06 through 00 are stored in the SC register via the BMX and the EALU. Bits 14 through 07 are then read into the ER and the appropriate condition codes in the EALU are set.

If bits 14 through 07 are all 1s and bit 15 (sign bit) is a 1, the number is a legal negative number. If bits 14 through 07 are all 0s and bit 15 is a 0, the number is a legal positive number. Any other combination is either underflow or overflow. The conditions described below can be found on the FP11-C diagrams.

Legal Positive Number – This case occurs when bit 15 is a 0 (designated by BN) and bits 14 through 07 are all 0s (designated by BZ being asserted). This means that the constant of 200 can be added to the number and no error condition will occur.

Legal Negative Number – This case occurs when bit 15 is a 1 (designated by BN) and bits 14 through 07 are non-zero (designated by BZ). If bits 14 through 07 are all 1s, the number is a legal negative number except for a special case where bits 06 through 00 are all 0s. If any of bits 14 through 07 is a 0, then the number is an illegal negative number. The legality of the number is tested by adding a 1 to bits 14 through 07. If all the bits become 0 (meaning a carry was propagated all the way through bits 14 through 07), it means that the bits were all 1s to begin with. Several examples are shown below.

Example 1: Bits 14 through 07 all 1s

```
       14 13 12 11 10 9 8 7
        1  1  1  1  1 1 1 1
                         +1
     ┌----------------------
   1 |  0  0  0  0  0 0 0 0
```

All 0s indicates bits 14 through 07 were all 1s, which represents a legal negative number.

Example 2: Bits 14 through 09 all 1s, bit 08 and 07 = 0

```
14 13 12 11 10 9 8 7
 1  1  1  1  1 1 0 0
                  +1
--------------------
 1  1  1  1  1 1 0 1
```

Since the result is not all 0s, it indicates that bits 14 through 07 were not all 1s to begin with and represented an illegal negative number. This is an underflow condition since bit 15 = 1.

Example 3: Special Case
A special case occurs when bit 15 is a 1, bits 14 through 07 are 1s, and bits 06 through 00 are 0s. When the constant of 200 is added to this number, the result is all 0s. This is registered as an illegal negative number, as shown in the example below.

```
                      15  14  13  12  11  10  9 8 7 6 5 4 3 2 1 0
177600                 1   1   1   1   1   1  1 1 1 0 0 0 0 0 0 0
Add constant of 200                              +1 0 0 0 0 0 0 0
                      -------------------------------------------
                       0   0   0   0   0   0  0 0 0 0 0 0 0 0 0 0
```

This is the only time that bits 15–07 are all 1s and still represent an illegal number. In all other cases with bits 15–7 = 1, the number is a legal negative number.

Example 4: Overflow
When bit 15 is a 0 (designated by BN) and any of bits 14 through 07 are non-zero, a number greater than the FP11-C's capacity is indicated.

```
                     15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
                      0  0  0  1  0  0 0 0 0 x x x x x x x
Add constant of 200                        + 1 0 0 0 0 0 0 0
                     ---------------------------------------
                      0  0  0  1  0  0 0 0 1 x x x x x x x
```

X = don't care

This result indicates a number greater than 377, which is the largest number the 8-bit exponent field in the FP11-C is capable of representing.

Example 5: Underflow
This case occurs when bit 15 is a 1 (designated by BN being asserted) and bits 14 through 07 are all 0s (designated by BZ being asserted). It indicates a negative number too large to be handled by the 8-bit exponent field in the FP11-C.

```
                     15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
                      1  0  0  0  0  0 0 0 0 x x x x x x x
Add constant of 200                        + 0 1 0 0 0 0 0 0
                     ---------------------------------------
                      1  0  0  0  0  0 0 0 0 x x x x x x x
```

X = don't care

The result is a very large negative number which exceeds the capacity of the FP11-C.

3.5.12 Load Convert Integer to Floating Instruction

The Load Convert Integer instruction takes a 2's complement integer from memory and converts it to a floating-point number in sign and magnitude format. If short integer mode is specified, the number from memory is 16 bits and is converted to a 24-bit fraction (single-precision) or a 56-bit fraction (double-precision), depending on whether floating or double is specified. If long integer mode is specified, the number from memory is 32 bits and is converted to a single-precision or double-precision number depending on whether floating or double mode is specified. The integer is loaded into bits 50 through 35 if short integer is specified or into bits 50 through 19 if long integer is specified. It is then left-shifted nine places so that bit 50 is transferred to bit 59 as shown below.

59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1

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

11-3912

The integer is then assigned an exponent of 217_8 short integer. This is the result of adding 200_8 (since the exponent is expressed in excess 200 notation) to 17_8, which represents 15_{10} shifts. This number of shifts is the maximum number required to normalize a number. If long integer mode is specified, the integer is assigned an exponent of 237_8, which represents 31_{10} shifts.

The 2's complement integer is tested by examination of bit 59 to see if it is a positive or negative number. If bit 59 is 0 (positive number), it could represent 0 or a positive integer. This is tested by subtracting 1 from the integer, which is stored in the upper 16 bits of the AR. If the integer is 0, the subtraction will produce a ripple borrow, causing bit 59 to go to a 1. If this occurs, the integer is 0 and 0 is stored in the destination accumulator.

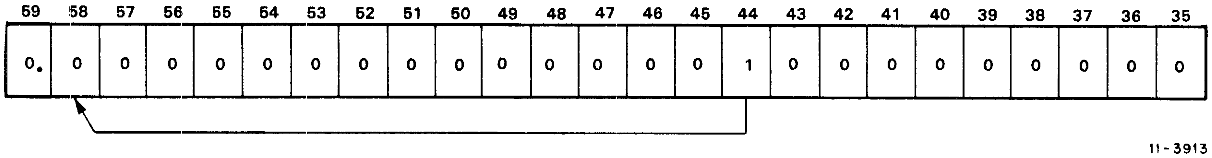
If bit 59 remains a 0 when a 1 is subtracted from the AR, it indicates a positive non-zero integer and 1 must be added back to the AR to restore the integer to its original value. The number is then normalized by left-shifting until bit 58 becomes a 1.

If bit 59 is 1 (negative number), the integer is negative and is 2's complemented, prefixed by a negative sign, and then normalized.

To normalize a number, bit 59 (MSB) of the fraction must be equal to 0 and bit 58 must be made equal to one. To do this, the integer is shifted the required number of places to the left and the exponent value is decreased by the number of places shifted.

Several examples that illustrate the procedure follow.

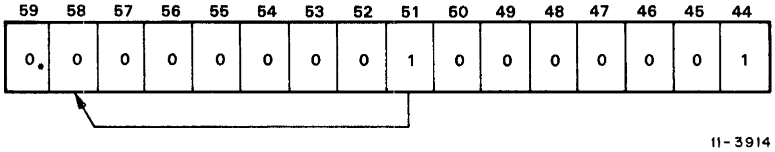
Example 1: Integer of 1



59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35
0.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

EXP = 217_8	Shift integer 15 places to the left to normalize.
$- 17_8$	Bit 59 = 0, bit 58 = 1
200_8	Decrease exponent by 15_{10} which is 17_8.

Example 2: Integer of 129



59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44
0.	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

EXP = 217_8	Shift integer seven places to the left to normalize.
-7_8	Bit 59 = 0, bit 58 = 1
210_8	Decrease exponent by 7_{10}, which is 7_8.

3.5.13 Store Exponent Instruction

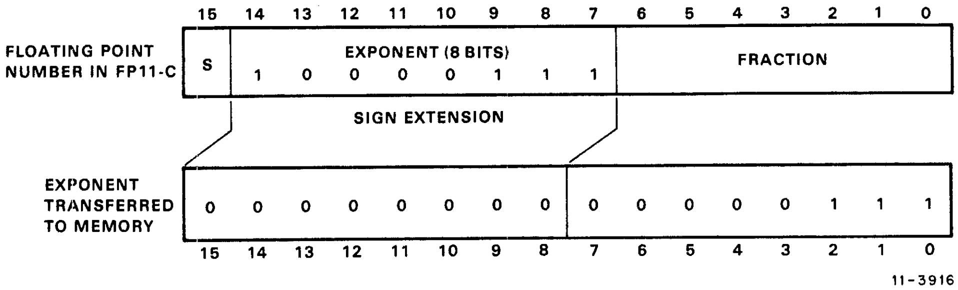
The Store Exponent (ST EXP) instruction accesses a sign and magnitude floating-point number in the FP11-C, extracts the 8-bit exponent field from this number, subtracts a constant of 200 (since the exponent field is expressed in excess 200 notation), and stores the resultant exponent in a 16-bit memory location. The number is stored in memory as a 2's complement, right-justified number with the sign of the exponent extended through the unused bit locations.

The legal range of exponents is from 0 to 377, expressed in excess 200 notation. This means that the number stored in memory ranges from –200 to 177 after the constant of 200 has been subtracted. The subtraction of 200 is accomplished by taking the 2's complement of 200 and adding it to the exponent field. Although the exponent field is only eight bits, the FP11-C subtracts the constant of 200 using 8-bit arithmetic and the additional eight bits are merely sign extension bits.

Two examples that illustrate the process follow: one using an exponent greater than 200 and the next using an exponent less than 200.

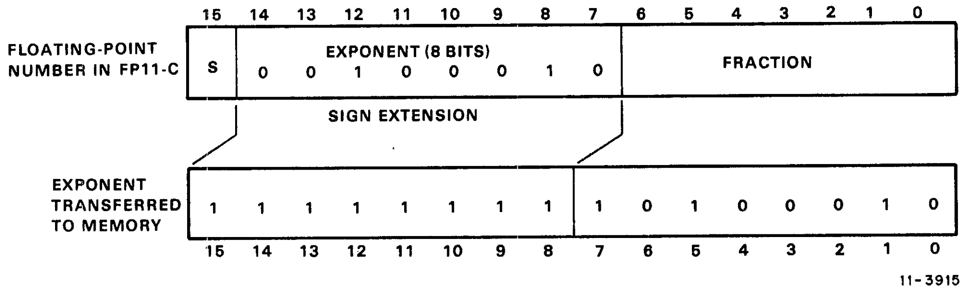
Example 1: Exponent = 207

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Constant of 200	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1's Complement of 200	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
																+1
2's Complement of 200	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Exponent of 207									1	0	0	0	0	1	1	1
2's Complement of 200	+1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Result = 000007	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1



Example 2: Exponent = 42

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Constant of 200	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1's Complement of 200	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
																+1
2's Complement of 200	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Exponent of 42									0	0	1	0	0	0	1	0
2's Complement of 200	+1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Result = 177642	1	1	1	1	1	1	1	1	1	0	1	0	0	0	1	0



The exponent processor in the FP11-C is a 10-bit processor (8 bits of exponent, 1 bit for overflow, and 1 bit for sign). To implement the transfer of the 16-bit exponent to memory, the FP11-C does the following. The 8-bit exponent field is supplied to one input of the EALU and the constant of 200 is applied to a second input to the EALU (Figure 3-8). The EALU performs the subtraction and applies the result in a right-justified format to the DIMX. The sign bit is sampled; if it is a 1, 1s are extended through the upper eight-bit positions in the memory location; if it is a 0, 0s are extended through the upper eight-bit positions in memory. Thus, the 8-bit exponent has been converted to a 16-bit, 2's complement, sign extended number.

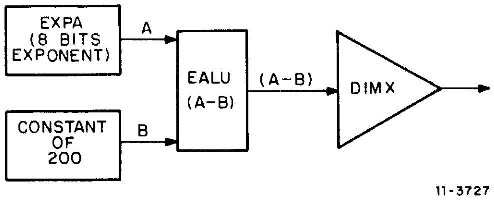


Figure 3-8 Exponent Path for STEXP

3.5.14 Store Convert Floating-to-Integer Instruction

The Store Convert Floating-to-Integer instruction takes a floating-point number which is expressed in sign and magnitude format and converts it to an integer number for transfer to memory.

The four classes of this instruction are:

1. *STCFI* – Convert single-precision, 24-bit fraction to a 16-bit integer (short integer mode)
2. *STCFL* – Convert single-precision, 24-bit fraction to a 32-bit integer (long integer mode)
3. *STCDI* – Convert double-precision, 56-bit fraction to a 16-bit integer (short integer mode)
4. *STCDL* – Convert double-precision, 56-bit fraction to a 32-bit integer (long integer mode)

The floating-point number to be converted is first transferred to the AR and ER. The FP11-C then subtracts 201_8 from the exponent to determine if there is an integer (number greater than or equal to 1). If the result of the subtraction is negative, it indicates that the exponent is less than 201 and the integer value of that number is 0. In this case, the FZ bit is set and 0s are sent to memory. If the result of the subtraction is positive, it indicates that the exponent is greater than 1 so the number is an integer.

NOTE
The reason 201 is subtracted rather than 200 is that the smallest legal exponent is 201. An exponent of 200 results in a number less than 1.

A second test is made by the FP11-C to determine if the integer is within the range of numbers which can be represented by a 16-bit integer (short integer mode) or 32-bit integer (long integer mode). This means that the exponent must be less than 2^{14} for short integer mode or less than 2^{30} for long integer mode. To test this, the FP11-C subtracts constants of 17_8 (short integer mode) or 37_8 (long integer mode) from the unbiased exponent. If the result of the subtraction is positive, it indicates that the floating-point number is too large to be represented as an integer. In this case, the FCC bit is set to 0 and 0s are sent to memory. If the result of the subtraction is negative, the integer is within the range of numbers that can be stored in memory.

The most significant bits of the fraction (bits 58 through 51) are presently stored in the AR. The resultant integer is to be stored in bits 50 through 35 (AC6[2]) if the number is short integer or in bits 50 through 19 (AC6[2:1]) if the number is long integer. In effect, this provides a contiguous location of 16 bits or 32 bits to store the resultant integer.

The FP11-C subtracts a constant of 10 from the exponent, which at this point is the floating-point exponent -201_8 and -17_8 if short integer, or -37_8 if long integer. The result of the subtraction indicates the number of places the fraction is to be right-shifted to be in the proper position. If the floating point number is positive (SD = 0), then the integer conversion is complete and the number is transferred to memory via AC6. If, however, the floating-point number is negative (SD = 1), the integer must be 2's complemented. The number is transferred to the FALU where it is 1's complemented and is then stored in the AREG. The FALU is then set up to perform A + B where it adds the integer increment bit to the contents of the AREG. The integer increment bit is inserted in bit 35 in short integer mode or bit 19 in long integer mode. By using this bit, only that portion of the number being converted to an integer is incremented. The FALU now performs an A + B operation to get the 2's complement of the number by adding 1 to the 1's complement. The 2's complement integer is now stored in AC6[2] if short integer or in AC6[2:1] if long integer. Bit 50 (MSB) of AC6 is reserved for the sign bit. If the number being converted is positive, bit 50 remains a 0; if the number is negative, bit 50 becomes part of the integer as a result of the 2's complementing.

Once the floating-point number has been converted to an integer (and 2's complemented if negative), it is transferred from AC6 to ACOMX, OBUF, DOMX and subsequently to memory. To better understand the procedure, Example 1 shows the number 4 in sign and magnitude format being converted to integer. Single-precision format and short integer mode are specified.

Example 1: Store Convert Floating to Integer (STCFI)

Exponent = 203_8
Fraction = 0.10000...

$203 = 2^3$, $0.10000 = 1/2$

Integer to be stored = $2^3 \times 1/2 = 4$

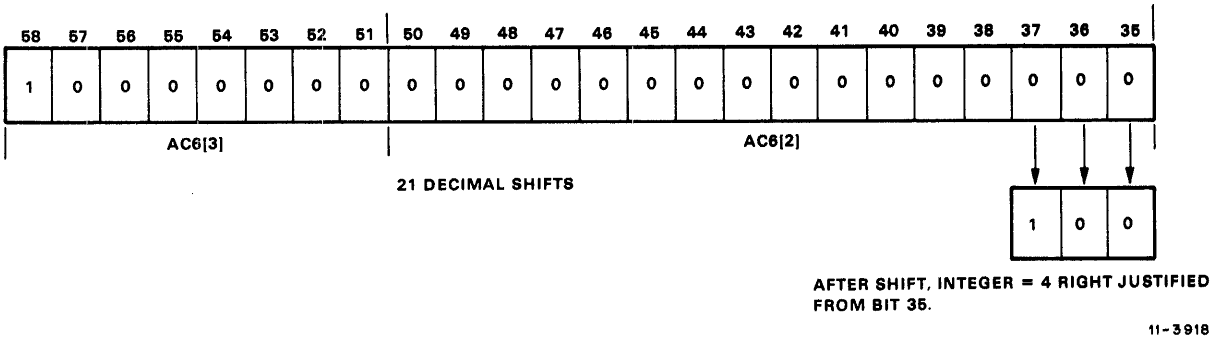
$$\begin{array}{rr l} \text{Exponent =} & 203 & \\ \text{Subtract} & \underline{201} & \\ \text{Unbiased Exponent =} & 2 & \text{Indicates there is an integer} \end{array}$$

Determine if integer is less than 2^{14}

$$\begin{array}{r l} 2_8 & \\ \underline{-17_8} & \\ -15_8 & \text{Indicates that the integer can be represented in 16 bits. If the result is positive, the integer is greater than can be represented by 16 bits.} \end{array}$$

Generate shift count by subtracting constant of 10_8 (which right-shifts numbers from bit 58 to bit 49 in AC6[2].

$$\begin{array}{r l} -15_8 & \\ \underline{-(+)10_8} & \\ -25_8 & = 21_{10} \text{ right shifts} \end{array}$$



This example assumed a positive number, so conversion is complete after 21 right shifts. If the number was negative, the integer must be 2's complemented.

Example 2: Convert Floating-Point to Integer Special Case

There is one specific number that requires a special conversion from a floating-point number to an integer. The number $+2^{15}$ cannot be represented in the CPU as a 16-bit integer, since this number is 0 in sign and magnitude format. The number -2^{15} can be represented and would normally produce a conversion error. However, the FP11-C recognizes this number as a special case with a negative sign, an exponent of 220, and a fraction of 1/2. The exponent is tested as follows.

When 201 is subtracted from the biased exponent followed by the subtraction of 17 from the unbiased exponent, the resultant exponent is 0.

$$\begin{array}{r} 220_8 \\ \underline{-201_8} \\ 17_8 \\ \underline{-17_8} \\ 0 \end{array}$$

In this instance, the FP11-C knows the exponent to be 220 and the sign to be negative; otherwise, a conversion error results. The fraction of the special number is tested as follows: a 17-bit mask in the QR is formed from bit 59 through bit 44.

	AC6[2]									AC6[1]							
	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
QR MASK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

11-3919

The special number is ANDed with the mask to yield all 0s except for bit 58 = 1. At this point, the FP11-C left-shifts one bit position to place bit 58 into bit 59 and decrements bit 00. A test is then performed on bit 59. If the fraction is the special number described, the borrow will be propagated all the way through the register with bit 59 being forced to 0. In this instance, the FP11-C increments the number to bring back the number to its original state. Since the number will be stored in part of AC6[2] and AC6[1], it is necessary to right shift the number nine places so that bit 58 is shifted to bit 49 and all other bits are shifted accordingly, with the final numbers now stored in AC6[1]. If bit 59 was equal to 1 after the least significant bit was decremented, it indicates that there was a 1 somewhere between bits 59 and 43 and, consequently, this number is not the special case. In this situation, the FP11-C exits with a conversion error.

3.5.15 Load FP11's Program Status

This instruction causes 16 bits to be transferred from the CPU to the FPS (Floating-Point Status) register. The 16 bits would contain status information for use by the FP11-C in order to specify the mode of operation and interrupt enables (-0 trap, overflow, underflow - Paragraph 3.2).

3.5.16 Store FP11's Program Status

This instruction transfers the 16 bits of the FPS (Floating Point Status) to a specified destination (memory or general register).

3.5.17 Store FP11's Status

The Store FP11's Status (STST) instruction reads the FEA (Floating Exception Address) from AC7[2] and the FEC (Floating Exception Code) from AC7[1], and is used during a floating-point error condition. The FEA is a 16-bit address while the FEC uses only the lower four bits of a memory location. If destination mode 0 is specified, then the FEC is stored in the CPU general register (the FEA is not stored). If destination mode 0 is not specified, the FEC is stored in the CPU followed by the FEA. Normal operation is to have the interrupt trap enabled. When an error occurs, the CPU traps to interrupt vector 244 and issues the STST instruction to determine the type of error. If the interrupt trap is disabled, FEC and FEA are still loaded into AC7. When the error occurs, the error bit sets but it necessitates testing the error bit after each instruction.

NOTE

The STST instruction should be used only after an error has occurred, since in all other cases the instruction contains irrelevant data or contains the conditions that occurred after the last error.

3.5.18 Copy Floating Condition Codes

The Copy Floating Condition Codes (CFCC) instruction copies the four floating condition codes (FC, FZ, FV, FN) into the CPU condition codes (C, Z, V, N).

3.5.19 Set Floating Mode

The Set Floating Mode (SETF) instruction clears the FD bit (bit 07 of FPS register) to a 0 and denotes single-precision operation.

3.5.20 Set Double Mode

The Set Double Mode (SETD) instruction sets the FD bit (bit 07 of FPS register) to a 1 and denotes double-precision operation.

3.5.21 Set Integer Mode

The Set Integer Mode (SETI) instruction clears the IL bit (bit 6 of FPS) to a 0 and indicates that short integer mode (16 bits) is specified.

3.5.22 Set Long Integer Mode

The Set Long Integer Mode (SETL) instruction sets the IL bit (bit 6 of FPS) to a 1 and indicates that long integer mode (32 bits) is specified.

NOTE

The following instructions are maintenance instructions and are primarily used to locate system failures.

3.5.23 Maintenance Shift Instruction

The Maintenance Shift (MSN) instruction is used to check the storing of the hidden bit and the operation of the shifters (QSHFR and ASHFR).

The contents of the AR and the QR are shifted right or left the number of times specified by bits 06 through 00 of CPU general register 4. The shift count ranges from $-10_{\underset{8}{\cancel{10}}}$ to $+7_{10}$. Negative is a right-shift and positive is a left-shift.

3.5.24 Store AR in AC0

The Store AR in AC0 (STA0) instruction is used for diagnostic purposes to check the contents of the AR register at certain microstates during the execution of an instruction.

The contents of the AR are transferred to AC0 except for bit 59 (overflow bit), bit 58 (hidden bit), and three guard bits (bits 02 through 00). Also, eight bits of exponent are transferred to AC0 from the ER. The sign bit is not affected.

3.5.25 Load Microbreak (Load Ubreak) Register

The Load Microbreak Register instruction copies the low byte (bits 07 through 00) of processor general register 3 into the microbreak register in the FP11-C. The microbreak register is an 8-bit register which is used for microbreak traps and for generating sync pulses for scope loops. To do a microbreak trap, load the microbreak register with the microstate desired to trap to and set the FMM (maintenance mode) bit in the FPS register. Everytime the FP11-C sequences through the same microstate which is loaded in the microbreak register, it will trap to ROM state 2 (micromatch state).

3.5.26 Store QR in AC0

The Store QR in AC0 (STQ0) instruction is used for diagnostic purposes to check the contents of the QR register at certain microstates during the execution of an instruction.

The contents of the QR are transferred to AC0 except for bit 59 (overflow bit), bit 58 (hidden bit), and three guard bits (bits 02 through 00). Also, eight bits of exponent are transferred to AC0 from the ER. The sign bit is not affected.

3.6 FP11-C PROGRAMMING EXAMPLES

This paragraph shows two programming examples using the FP11-C instruction set. In 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 3-pass loop, whereby each loop does the calculation indicated below.

Example 1 [(A+B)*(C–D)]*X

```
000000    172467    000122    LDF     A,AC0      ;LOAD AC0 FROM A
000004    172067    000122    ADDF    B,AC0      ;AC0 HAS (A+B)
000010    172567    000122    LDF     C,AC1      ;LOAD AC1 FROM C
000014    173167    000122    SUBF    D,AC1      ;AC1 HAS (C–D)
000020    171001              MULF    AC1,AC0    ;AC0 HAS (A+D)*(C–D)
000022    174467    000752    DIVF    X,AC0      ;AC0 HAS (A+D)*(C–D)/X
000026    174067    000752    STF     AC0,Y      ;STORE (A+D)*(C–D)/X IN Y
```

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

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

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

Example 2

```
000100  012700  000003          MOV   #3,%0      ;SET UP LOOP COUNTER
000104  012701  000146          MOV   #D+4,%1    ;SET UP POINTER TO
                                                  COEFFICIENTS
000110  172526                  LDF   (6)+,AC1   ;POP X FROM STACK
000112  170400                  CLRF  AC0        ;CLEAR OUT AC0
000114  172044          LOOP;   ADDF  - (4),AC0  ;ADD NEXT COEFFICIENT
                                                 ;TO PARTIAL RESULT
000116  171001                  MULF  AC1,AC0    ;MULTIPLY PARTIAL
                                                  RESULT BY X
000120  077003                  SOB   %0,LOOP    ;DO LOOP 3 TIMES
000122  172044                  ADDF  -(4),AC0   ;ADD X TO GET RESULT
000124  174046                  STF   AC0,-(6)   ;PUSH RESULT ON STACK
```

CHAPTER 4
CONTROL ROM

4.1 INTRODUCTION

Figure 4-1 is a block diagram of the FP11-C data path which consists of the CPU/FP11-C interface, sign and exponent processor logic, and fraction processor. The registers and multiplexers shown in Figure 4-1 are briefly described in the following paragraphs.

4.1.1 Floating Instruction Register A (FIRA)

The FIRA is a 12-bit temporary holding register for the floating-point instruction. This instruction is transferred from the BR register in the CPU. The FIRA register is only 12 bits long because the four upper bits of the instruction contain the 17_8 op code (which specifies floating-point instructions) and are not sent to the FP11-C. This register is clocked by the CPU.

4.1.2 Floating Instruction Register B (FIRB)

The FIRB is a 12-bit register which accepts the instruction from the FIRA as it starts operation. The FIRB stores the instruction currently being executed. Upon completion of the execution of this instruction, and if there is another floating-point instruction, the instruction in the FIRA is sent to the FIRB and the FP11-C will then proceed to execute this instruction.

4.1.3 Floating Data Register (FDR)

The FDR is a 16-bit register which accepts data from the CPU via the BR and allows the data to be read back to the CPU via the DOMX or allows the data to be routed internally to the FP11-C via the DIMX.

4.1.4 Floating Point Address (FPA) Register

The FPA register is loaded from the CPU and stores the address of the instruction currently being executed by the CPU.

4.1.5 Floating Exception Address (FEA) Register

The FEA register is loaded by the FP11-C sequencing through start or the ready state. It stores the exception address which represents the address of the instruction being executed. If an error occurs, the exception address is transferred to AC7[2] and can be buffered back to memory via execution of a Store Status (STST) instruction.

4.1.6 Data Out Multiplexer (DOMX)

The DOMX is a 16-bit, 4-input multiplexer which accepts data from one of four input sources to be written back to the CPU. The four inputs are:

1. OBUF – Register that holds data from the accumulators which is to be written into memory.

2. FPS – The floating-point status which is stored in the FPS register. This register routes the status of the FP11-C to the CPU.

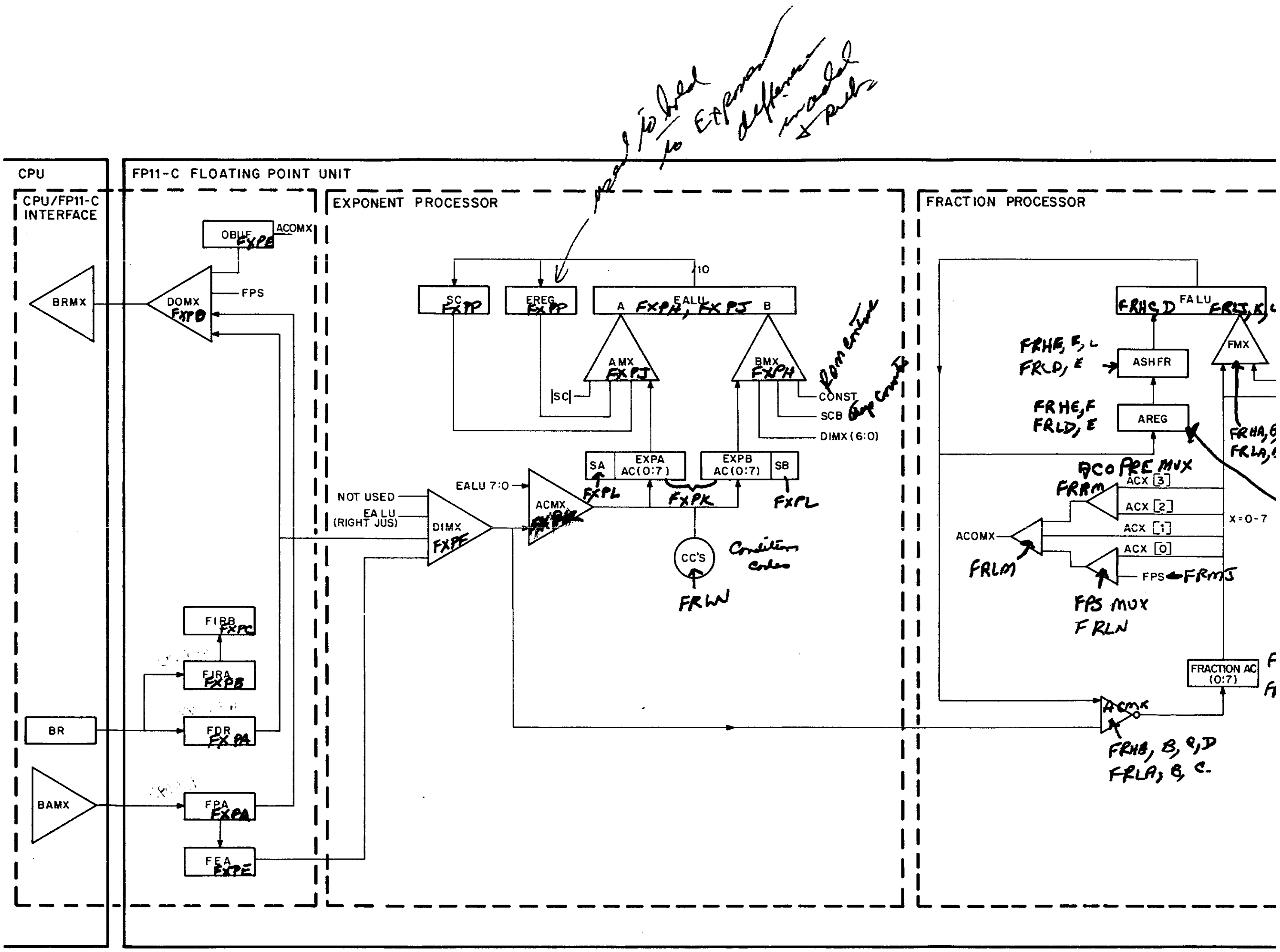


Figure 4-1 FP11-C Data Paths

3. FPA – Holds the floating-point address of the FP11-C instruction currently being executed.

4. FDR – Holds the contents of the general register during address calculation. The FDR is also used as a temporary storage register to hold data from memory.

4.1.7 Data In Multiplexer (DIMX)

The DIMX is a 16-bit wide, 4-input multiplexer that accepts data from one of 3 input sources. The inputs are:

1. FEA – The FEA (floating exception address) stores the address of the FP instruction currently being executed.

2. EALU – Unit that manipulates the exponents.

3. FDR – Holds the contents of the general register during address calculation. The FDR is also used as a temporary storage register for data from memory.

4.1.8 Accumulator Multiplexer (ACMX)

The ACMX is a 16-bit wide, 2-input multiplexer which routes data to the exponent scratchpads (EXPA and EXPB) and to the fraction scratchpads (FRACTION AC0:7).

One of the inputs to the multiplexer is from the EALU, FALU, and sign bit. The other input to the ACMX is the 16-bit operand from the DIMX. The EALU input representing the exponent is transferred to both exponent scratchpads. The DIMX input is applied to the fraction scratchpads in one or three 16-bit segments, depending on the precision.

For example, in a double-precision instruction, the first word is stripped, with the eight bits of exponent and one bit of sign going to EXPA and EXPB and the seven bits of fraction being routed to the fraction scratchpads. The 16 bits of the second operand are routed to the fraction scratchpads, as are the third and fourth operands.

4.1.9 Exponent A (EXPA) and Exponent B (EXPB) Scratchpads

There are two sets of exponent scratchpads which are nine bits wide (eight bits of exponent and one bit of sign). Each set of scratchpads (accumulators) has an exponent field associated with each fraction scratchpad. For example, if AC4 is specified, the exponent scratchpad is addressed as accumulator address 4.

Two exponent scratchpads are implemented because they allow the source and destination exponents to be checked at the same time. They also allow the FP11-C to take the difference of two exponents in one ROM state. This feature is utilized during execution of the Add or Subtract instructions. Whenever data is written into an accumulator, EXPA and EXPB are addressed the same. If the EXPB scratchpads are addressed, the fraction scratchpads follow the same address.

4.1.10 Condition Codes

The circle at the output of ACMX, which is labeled CC's, represents the condition codes, which may be floating or branch condition codes. The condition codes are:

FC (Floating Conversion Error) – The FC bit is used for integer conversion errors when floating point numbers are converted to integers.

FV (Floating Overflow) – The FV bit is set via bit 08 of the EALU and indicates that the exponent has overflowed.

FZ (Floating Zero) -- The FZ bit denotes a zero exponent and is set if bits 07 through 00 (exponent field) are all 0s or if DIMX bits 14 through 07 are all 0s. Bits 14 through 07 represent the exponent field from memory, assuming the number is a floating-point number.

FN (Floating Negative) - The F bit denotes a negative number (sign bit = 1) and can be set by the sign control field or when the ACMX is enabling the DIMX and DIMX bit 15 is a 1. The DIMX path is used to set the N bit during a Load instruction.

BN (Branch Negative) - The BN bit is set from EALU bit 9 (which is the sign bit of the exponent processor) or if bit 15 of the DIMX is 1 when the DIMX is enabled by ACMX.

BZ (Branch Zero) - The BZ bit denotes a zero exponent and is set if bits 07 through 00 (exponent field) are all 0s or if DIMX bits 14 through 07 are all 0s. Bits 14 through 07 represent the exponent field from memory, assuming the number is a floating-point number.

4.1.11 A Multiplexer (AMX)

The AMX is a 10-bit wide, 4-input multiplexer which accepts inputs from one of the following four sources:

SC - 10-bit working register in the exponent path.

EXPA - One of the two sets of exponent scratchpads.

|SC| - The absolute value of the step counter which is used to calculate exponent alignment during an add or subtract instruction.

EREG - 10-bit working register in the exponent path.

4.1.12 B Multiplexer (BMX)

The BMX is a 10-bit wide, 4-input multiplexer which accepts inputs from one of the following four sources.

1. CONST - A number from 0 to 377_8 which is used by the microprocessor.
2. SCB - The shift control bus which determines how many shifts have occurred in the fraction processor.
3. DIMX (6:0) - Low-order bits of DIMX used during a LD EXP instruction.
4. EXPB - One set of scratchpads in the exponent path.

4.1.13 Exponent Arithmetic Logic Unit (EALU)

The EALU Unit is a 10-bit wide unit that is capable of performing both arithmetic and logical functions between the A and B inputs.

4.1.14 Step Counter (SC)

The SC is a 10-bit register used in arithmetic operations to store the shift count.

4.1.15 E Register (EREG)

The EREG is a 10-bit working register which holds the exponent of the result.

4.1.16 Fraction Accumulator (AC7:0)

The fraction accumulators are referred to as the fraction scratchpads or scratchpad accumulators and are utilized to store the fraction of a floating-point number. The scratchpads are actually eight 56-bit wide accumulators designated AC7 through AC0. Accumulators 0 through 5 are working accumulators; accumulator 6 is a special accumulator used as a temporary buffer between the FP11-C and memory; and accumulator 7 is an accumulator which stores the FEA (floating exception address) and FEC (floating exception code).

The address of the fraction scratchpads follows the address of the EXPB scratchpads. For example, if EXPB is addressed as a source, the fraction scratchpads are addressed as a source.

4.1.17 Accumulator Out Multiplexer (ACOMX)

The ACOMX is a 16-bit wide, 4-input multiplexer which accepts the four quadrants of the accumulators and the floating-point status (FPS). The FPS is multiplexed by an additional multiplexer labeled FPSMX.

4.1.18 Fraction Multiplexer (FMX)

The FMX is a 60-bit wide, 2-input multiplexer which accepts the fraction from the scratchpad accumulators or accepts a shifted fraction from the QSHFR, which is utilized during arithmetic operations. The FMX is also used during rounding.

4.1.19 Fraction Arithmetic Logic Unit (FALU)

The FALU is a 60-bit wide unit that is capable of performing arithmetic and logical operations between the A and B inputs. Two levels of carry look-ahead are provided.

4.1.20 Accumulator Register (AREG)

The AREG is a 60-bit wide register used to manipulate the fractions during floating-point operations.

4.1.21 Accumulator Shifter (ASHFR)

The ASHFR is a 60-bit shift network used to shift the AREG during arithmetic operations. It will shift left from 0 to 7 shifts and will shift right from 0 to 8 shifts.

4.1.22 Q Multiplexer (QMX)

The QMX is a 60-bit wide, 2-input multiplexer which accepts inputs from the scratchpad accumulators or from the QSHFR.

4.1.23 Q Register (QREG)

The QREG is a 60-bit wide register used during arithmetic operations. The register is loaded from scratchpad outputs or from QREG to QREG via the QMX.

4.1.24 Q Shifter (QSHFR)

The QSHFR shifts the output of the Q register. The QMX, QREG, and QSHFT are described with more detail with the arithmetic alqorithms in Chapter 5.

4.2 DATA PATH ROUTING FOR LOAD INSTRUCTION

To better understand the FP11-C data path (Figure 4-1), assume that the CPU has fetched a floating-point instruction and that the instruction is a double-precision Load instruction of the form LDD TEM, AC2. This instruction causes a double-precision word from memory to be loaded into AC2.

The CPU sends the following to the FP11-C:

1. Address of the LDD instruction is sent to the FPA register.

2. Bits 11 through 00 of the floating-point instruction are sent to the 12-bit FIRA. Bits 15 through 12 of the instruction are decoded as a 17_8, which is a floating-point op code. These bits are not sent to the FIRA.

3. Contents of the general destination register are sent to the FDR.

The CPU then performs the address calculation. When the FP11-C is ready to accept the first data word from the CPU, it issues FP SYNC.

The first data word consists of one bit of sign, eight bits of exponent, and seven bits of fraction. The sign and exponent are routed to the exponent scratchpads (EXPA and EXPB) via the DIMX and ACMX. The seven bits of fraction are routed to the fraction accumulator via the DIMX and the ACMX. This occurs in microstate 21. (Refer to applicable flow diagrams.) Since this is a memory reference (not mode 0), the FP11-C forces the source accumulator to be accumulator 6. This means that the seven bits of fraction are routed to the fraction scratchpads, and the eight bits of exponent and one bit of sign are routed to the EXPA and EXPB scratchpads. These 16 bits of data are defined as quadrant [3] of accumulator 6, in this case. In the next microstate (142), the FP11-C accepts the second data word and transfers it to AC6[2] via the FDR, DIMX, and ACMX. The third data word is transferred in microstate 200 to AC6[1] via the same data path. In microstate 201, the fourth data word is transferred via FDR, DIMX, and ACMX to AC6[0]. The second, third, and fourth data words are all fractions. The total fraction then consists of an overflow bit (bit 59), a hidden bit (bit 58), and 55 bits of fraction, as shown in Figure 1-4.

In the next microstate (42), the source exponent that was loaded in scratchpads EXPA and EXPB is applied to the EALU. The ACF field is set for 4 by the microcode. This indicates EXPA is ACS and EXPB is ACS.

NOTE

When the scratchpads (EXPA and EXPB) are written, they are written with the same data simultaneously.

The EALU microcode is set to 4 which is A. Consequently, the A input to EALU is enabled and the sign and exponent from EXPA are routed through the EALU to the EREG.

During this same microstate, the fraction is transferred to the AREG via FMX and the FALU. The FALU is set to 2 by the microcode. This enables the B input which passes the fraction through FALU to AREG.

At this point, the sign and exponent are stored in EREG and the fraction is stored in AREG. In the next microstate (state 16), the ACF is set to 3 by the microcode, which is ACD/ACD. This causes the addresses of EXPA, EXPB, and the fraction scratchpads to be the destination accumulator. Consequently, the contents of EREG are routed through AMX, EALU (A input is selected), and ACMX to exponent scratchpads EXPA and EXPB associated with the destination accumulator (accumulator 2, in this case).

At the same time that the source exponent in the ER is transferred to the destination exponent scratchpad, the fraction, which is stored in AREG, is routed through ASHFR, FALU (A input is selected), and ACMX and is written into the destination accumulator (AC2).

To summarize, the data from memory was routed to the FP11-C via the BR in the CPU and the FDR in the FP11-C. The sign and exponent (upper bits of the first data word) were transferred to the exponent processor. Since the hardware forced the source accumulator to be AC6, the exponent was transferred to the EXPA and EXPB scratchpads associated with AC6. The fraction (consisting of a portion of the first data word and all of the second, third, and fourth data words) was routed to AC6 in the fraction processor. The instruction specified the destination accumulator as AC2. Consequently, the microcode was set up to route the exponent and fraction from AC6 to AC2, causing the double-precision word from memory to be written into AC2.

4.3 CONTROL ROM

The FP11-C 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-C is composed of 256 76-bit words. Eight bits of each word represent 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 sequencing to the next address, branches to the modified address. This action is shown in Figure 4-2. Note that the RAR (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.

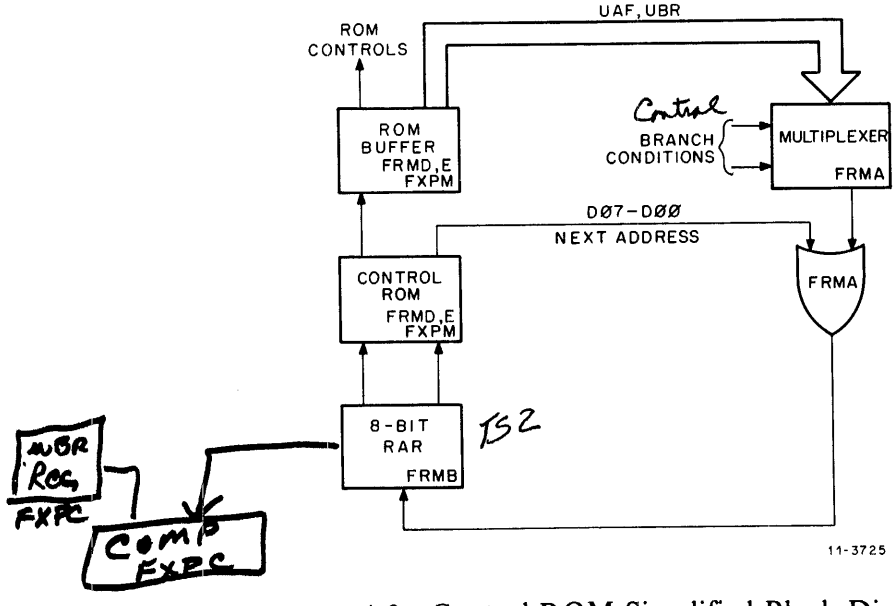


Figure 4-2 Control ROM Simplified Block Diagram

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 CPU 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 –0 is loaded from memory.

4.3.1 ROM Field Descriptions

Each block on the flow diagrams in the print set and in this manual represents a specific ROM word. The number of ROM words necessary to execute a floating-point instruction depends on the instruction. Table 4-1 shows how each ROM word is subdivided into fields and briefly defines the purpose of each field.

Table 4-2 shows bits (67:65), which represent the microbranch field. These bits are used in conjunction with the UAF field for branch modification.

1. Only bits 03 through 00 of the next address (see NAD bits in Table 4-2) can be modified; bits 06 and 07 cannot be changed. There are four exceptions to this:

   Initialize - FP11-C traps to state 0
   Floating Minus Zero - FP11-C traps to state 1
   UTRAP (Microtrap) - FP11-C traps to state 2
   UJP (Microjump) - FP11-C jumps to state 3

   The associated bits are modified; for example, on the UJP, bits 07 through 02 are modified to 0; bits 01 and 00 are 1s.

2. Only NAD (Next Address) bits on a 0 can be modfied, i.e., NAD bits on a 0 can be modified to 1s, but NAD bits on a 1 cannot be modified.

3. The branch condition(s) being used must be true. See Table 4-2 for the branch conditions of the UBR field. 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 08) is decoded to determine which conditions will be used to modify the next address, i.e., if UBR = 6, we can use FIR bit 8 (0) to modify NAD bit 03, AR58 (1) to modify NAD bit 2, etc. These modifications are, of course, contingent on the prior listed conditions and also on the decoding of the UAF field (Table 4-2).

The UAF field (ROM bits 60 and 61) is decoded to determine which bits of the NAD can be modified. See Table 4-2 for this octal decoding. There are four multiplexers which, if enabled, allow corresponding NAD bits to be modified. For example, if UAF = 2, bits 01 and 00 of the ROM address can be modified. In this case, bits 03 and 02 of the next address are unaltered.

The NAD field (ROM bits 75–68) gives the next ROM address to be sequenced, subject to modification if selected.

As an example of microbranching, refer to block 105 labeled SET BZ FOR 6:0, SAVE ACD[3] shown on sheet 1 of the flow diagrams. From ROM state 105, one of four different ROM addresses can be selected subject to the conditions of BZ and BN. The NAD field contains 360 as indicated by the 360 in parentheses shown in the lower right-hand corner of the block. The term 3F2 preceding the 360 refers to the UBR and UAF fields of the ROM word in location 105. The 3 represents the octal decode of the UBR bits and the F2 is the decode of the UAF bits. A UBR of 3 selects the D3 input to each of the branching logic multiplexers shown on logic diagram FRMA. A UAF of 2 specifies a branch enable of 1:0. (See the FP11-C data path diagram in the print set.) This indicates that multiplexers 0 and 1 are enabled for branching conditions while multiplexers 2 and 3 provide the normal outputs with the branching logic inhibited. There are four possible addresses that the ROM will branch to: addresses 360, 361, 362, or 363. The branch address is dependent on the condition of BN and BZ.

If BN and BZ are both 1s, the D3 inputs to multiplexers 0 and 1 are negated, providing a high output from each multiplexer. The high outputs inhibit NOR-AND gates E59 and E69, yielding FRMA RAD 00 L and FRMA RAD 01 L, respectively. In this case, the next address is 360 and no branching occurs. If BN and BZ are both 0s, the D3 input to both multiplexers is enabled, causing NOR-AND gates E59 and E69 to be enabled and yielding RAD 00 H and RAD 01 H. This creates a next address of 363. If BN is a 0 and BZ is a 1, only the D3 input to the 0 multiplexer is enabled and causes RAD 00 H to be asserted and the next address is forced to 361. If BN is a 1 and BZ is a 0, the D3 input to multiplexer 1 is enabled, asserting RAD 01 H, which creates the next address of 362.

These possible branch conditions are summarized in the truth table below.

BN	BZ	FRMA RAD 00 H	FRMA RAD 01 H	NEXT ADDRESS
0	0	1	1	363
0	1	1	0	362
1	0	0	1	361
1	1	0	0	360

4.3.2 Masking Out Branch Conditions

In certain situations, the UAF field allows certain multiplexers on logic diagram FRMA to be modified for branching conditions. If it is desired to inhibit the branching on one of these 8 multiplexers, the NAD field is used to mask out this condition. Consider the following example: ROM state 326 on sheet 12 of the flow diagrams with a NAD field of 365 and a UAF field of 3F2. The 3 in 3F2 designates the D3 input to the multiplexers shown on logic diagram FRMA and the F2 specifies a UAF of 2, which enables multiplexers 0 and 1. Thus, the D3 input to both multiplexers is enabled. However, the microcode in this state is designed to only branch on the BN condition code, which is the D3 input to multiplexer 1. Consequently, it is necessary to mask out the D3 input to multiplexer 0, so the FP11-C will not branch on BZ. Since only NAD bits on a 0 can be modified, NAD bit 0 can be masked by making it equal to 1, independent of any branching condition. Examining ROM address 326 again, it can be seen that the next address is 367 if ~BN is asserted and is 365 if BN is asserted, regardless of the state of BZ.

	ROM Address Bit			
Branching Condition	76	543	210	Next Address
~BN	11	110	111	367 if BN is positive
BN	11	110	101	365 if BN is negative

Note that only NAD bit 01 can be modified, since NAD bit 00 is forced to a 1 to inhibit branching. Another example occurs in ROM state 332 on sheet 12 of the flow diagrams. This block has a next address of 332 with a branching field of 3F1. The 3 specifies the D3 input to multiplexers 3 and 2. These multiplexers are designated by the UAF field of 1. Since the flow diagram shows the only branch condition to be SWR (shift within range) present at the D3 input to multiplexer 2, it is necessary to mask out multiplexer 3 since the UAF specifies multiplexers 3 and 2 and it is only desired to branch on multiplexer 2. The multiplexer is masked out by forcing NAD bit 03 to a 1 so it cannot be modified as shown below.

	ROM Address Bit			
Branching Condition	76	543	210	Next Address
~SWR	11	011	010	332 if ~SWR is asserted
SWR	11	011	110	336 if SWR is asserted

Table 4-1 ROM Fields

Bits	Field	Field Setting	Definitions
(75:68)	NAD 7:0 (Next Address) (See Table 4-2)	Refer to flow diagrams	An 8-bit field to address 256 words in the control ROM and indicates the next address to the ROM.
(67:65)	UBR (Microbranch)	0 No branch 1 Branch enable 2 Branch enable 3 Branch enable 4 Branch enable 5 Branch enable 6 Branch enable 7 Branch enable Refer to flow diagrams	A 3-bit field used to select one of eight branch conditions in an 8-bit multiplexer.
(64)	UJMP Clear micro address Bits (Microjump)	0 NOP 1 Trap to ROM state 3 if RAR (1:0) = 3	A 1-bit field for deciding when the operation is done. If next address bit is 3, this bit enables a microtrap which forces the ROM back to the Ready state.
(63)	ACKN WAIT (Acknowledge Wait)	0 NOP 1 Wait for FP ACKN	A 1-bit field which informs the FP11-C to wait for FP ACKN if this bit is asserted.
(62)	ATTN WAIT (Attention Wait)	0 NOP 1 Wait for FP ATTN	A 1-bit field which informs the FP11-C to wait for FP ATTN if this bit is asserted.
(61:60)	UAF (Microaddress Fork) 4-14	0 Branch enable 0 1 Branch enable 3:2 2 Branch enable 1:0 3 Branch enable 3:0	A 2-bit field which selects certain combinations of the four 8-input branching multiplexers shown on logic diagram FRMA. For example, if UAF = 1, multiplexers 3 and 2 are enabled. If UAF = 3, all four multiplexers are enabled.
(59:56)	FALUC (Fraction Arithmetic Logic Unit Control)	0 A and B 1 Not A 2 B 3 A and not B 4 A plus B 5 A plus B plus 1 6 A minus 1 7 A minus B 10 Zero 11 not used 12 Not used 13 Not used 14 A plus B Conditional 15 Not used 16 Mpy/Div Cond 17 A	A 4-bit control field which selects one of the functions specified in the field setting column.

Table 4-1 ROM Field (Cont)

Bits	Field	Field Setting	Definitions
(55:53)	EALUC (Exponent Arithmetic Logic Unit Control)	0 A plus B 1 A plus B plus 1 2 A minus B minus 1 3 A minus B 4 A 5 B 6 Not used 7 Not used	A 3-bit control field which selects one of the functions specified in the field setting column.
(52)	DISABLE INTR (Disable interrupt)	0 Allow interrupt clear 1 Disable interrupt clear	A 1-bit field which is used to determine whether to execute or abort an instruction. If the bit is asserted, the instruction is executed and aborts are disabled.
(51:48)	SHIFT CONTROL	0 EALU 1 DIV 2 MUL 3 NORM 4 ALIGN 5 Not used 6 Not used 7 0 SHIFT 1X-CLR Control	A 3-bit field which selects one of the functions specified in the field setting column.
(47:45)	ACC (Accumulator control)	0 NOP 1 [0] 2 [1] 3 [3] 4 [2] 5 [3:0] 6 [3:2] 7 [3:0] COND. (FD)	A 3-bit field which controls the writing of the scratchpads. The scratchpads can be written a quadrant at a time, two quadrants at a time, or all four quadrants.
(44)	LONG CYCLE	0 180 ns state 1 240 ns state	Normal microstate consists of 180 ns and is subdivided into T1, T2, and T3 with 60 ns between each pulse. Provision is made for extending the microstate to 240 ns by adding an additional 60 ns between T1 and T2. This feature is used in divide. Bit 44, when asserted, yields the 240 ns microstate.
(43)	QR CLOCK	0 NOP 1 CLK QR	A 1-bit field which controls clocking of the QR.
(42, 29, 28)	SIGN CONTROL	0 SD ← SD, SS ← SS 1 SD ← SC09, SS ← SS 2 SD ← SS, SS ← SS 3 SD ← NOT SS; SS ← SS 4 SD ← SD XOR SS; SS ← SS 5 SD ← SD XOR SUBTRACT; SS ← SS 6 SD ← 0, SS ← 0 7 SD ← SCROUT, SS ← SCROUT	A 3-bit control field used to determine the sign of source and sign of destination.

Table 4-1 ROM Field (Cont)

Bits	Field	Field Setting	Definitions
(41:40)	AR CONTROL	0 NOP 1 CLK AR 2 CLR AR (34:00) 3 CLR AR (59:00)	A 2-bit field which controls clocking of the AR.
(39:38)	ACOMX (Accumulator Output Multiplexer)	0 ACX [0] 1 ACX [1] 2 ACX [2] 3 ACX [3]	A 2-bit field which selects one of four quadrants to be read out of scratchpads.
(37:36)	FILL CONTROL	0 RS • 0 • QR 1 RS • 1 • QR 2 RS • 0 • AR 3 RS • AR59 • AR	A 2-bit field which controls whether 0s or 1s are right-shifted into the AR and QR. Bit 36 is dedicated to the QR and bit 37 is dedicated to the AR.
(35:34)	FMX (Fraction Multiplexer)	0 SCROUT 1 Q SHIFTER 2 COND • AC SCROUT (FD) 3 ROUND, INT, INC.	A 2-bit field which controls the selection of the FMX and the strobe inputs to FMX. For single-precision, the lower inputs are disabled and for double-precision, all inputs are enabled
(33:32)	QMX (Q Multiplexer)	0 SCROUT 1 QSHIFTER 2 COND SCROUT (FD) 3 QUOTIENT	A 2-bit field which controls the selection of QMX.
(31:30)	FCC (Floating Condition Codes)	0 FN, FZ ← CONDITIONS, FC, FV ← 0 1 FN, FZ, FV ← CONDITIONS, FC ← 0 2 FC ← 1 3 NOP	A 2-bit field which controls the floating condition codes.
(27:25)	MISC CONTROL (Miscellaneous Control)	0 NOP	
		1 FPS CLOCK	Used during LDFPS instruction to load FPS register.
		2 UBRK CLOCK	Used during load microbreak instruction to clock the microbreak register.
		3 DOMX MOD	Used during STFPS instruction to force DOMX output to select FPS instead of data (OBUF)
		4 OBUF CLK	Used during store operations to load the first 16 bits of data to the CPU into the OBUF register.
		5 CLR QR	Sets Q register to zero.
		6 LD FP REQ	Load a counter from the IR decode for the number of memory cycles minus 1 that the CPU will perform.

Table 4-1 ROM Field (Cont)

Bits	Field	Field Setting	Definitions
(27:25) (Cont)		7 FP CLASS	Informs the CPU to do one memory cycle and wait for the FP11-C to modify data before continuing on.
(24)	FIR CLK (Floating Instruction Register Clock)	0 NOP 1 LOAD FIRB	A 1-bit field which only occurs only in the Ready state. If the bit is asserted, the IR is enabled to be loaded.
(23)	FP SYNC (Floating-Point Sync)	0 NOP 1 FP SYNC	A 1-bit field which causes FP SYNC to be issued to the CPU if this bit is asserted.
(22)	EN FM0 (Enable Floating minus 0)	0 Disable FM0 Trap 1 Enable FM0 Trap	A 1-bit field used to enable floating minus 0 trap. When enabled, floating minus 0 occurs with negative numbers having an exponent of 0.
(21)	FPC1 (Floating-Point C1 Line)	0 DATI 1 DATO	A 1-bit field used to specify a DATI when FPC1 is negated and a DATO when FPC1 is asserted.
(20)	FP REG WR (Floating-Point Register Write)	0 NOP 1 WRITE	A 1-bit field used by the FP11-C during mode 0 to write into the general register if FP REG WR is asserted.
(19:18)	AMX (A Multiplexer)	0 EXPA 1 ABS SC 2 SC 3 ER	A 2-bit field that controls the selection of AMX in the exponent processor.
(17:16)	BMX (B Multiplexer)	0 EXPB 1 Constants (0-377_8) 2 DIMX [6:0] 3 Shift Control	A 2-bit field that controls the selection of BMX in the fraction processor
(15:14)	DIMX (Data In Multiplexer)	0 FDR 1 FEA 2 EALU 3 Not Used	A 2-bit field that controls the input to DIMX.
(13)	SC CLK (Step Counter Clock)	0 NOP 1 CLK SC	A 1-bit field that causes the step counter to be loaded when the bit is a 1.
(12)	ER CLK (Exponent Register Clock)	0 NOP 1 CLK ER	A 1-bit field which causes the ER to be loaded when the bit is a 1.

Table 4-1 ROM Field (Cont)

Bits	Field	Field Setting	Definitions
(11:9)	ACF (Accumulator Control Field)	0 Not used 1 ACDV1/ACDV1 2 ACS/ACD 3 ACD/ACD 4 ACS/ACS 5 ACD/ACS 6 AC6/AC6 7 AC7/AC7	A 3-bit field which controls selection of the scratchpads. EXPA is selected by the field to the left of the slash and EXPB is selected by the field to the right of the slash, For example, if ACF is a 2, EXPA is selected to the source accumulator and EXPB and the fraction scratch pad is selected to the destination accumulator.
(8)	ACMXC (Accumulator Multiplexer Control)	0 ALUS 1 DIMX	A 1-bit field which selects the ALUs if the bit is a 0 or selects the DIMX if the bit is a 1.
(7:0)	CNST (Constant)	0 37.7$_8$ 8 integral Exponent (200 [illegible])	An 8-bit field used to specify a constant between 0 and 377$_8$.

Table 4-2 Microbranch Field

Bits	Field	Field Setting	Next Address Field 75:68 Definition							
			NAD7	NAD6	NAD5	NAD4	NAD3	NAD2	NAD1	NAD0
(67:65)	UBR2–UBR0 (Microbranch field used in conjunction with UAF field to specify branch modification	0	0	0	0	0	0	0	0	0
		1			FIRD5	FIRD4	FIRD3	FIRD2	FIRD1	FIRD0
		2					ADD * SC < 8	SUB * SC < 8	SC09 (1) H	SC00 (1) H
		3					DIV DONE (0)	SWR (1)	BN (0) H	BZ (0) H
		4					FIRB06 (1) H	FIRB08 (1) H	SD (1) H	IMMED H
		5					OUT OF RNG	IL = 0	AR59 (0) H	BOU + BZ
		6					EXPA = 0	EXPB = 0	FV (1) * FIV	FC (1) * FIC
		7					IL + IMMED	M0 = H	FIU (0) H	FD (0)

MAF
61:64

Normal change

which condition will be
MBR 1Ø-Ø8 used Generaly Can't Change upper Bits

MAF 60,61 which Nad. Bits Can are changed.

Also use do modifie Bits 6+7 Can not be changed.

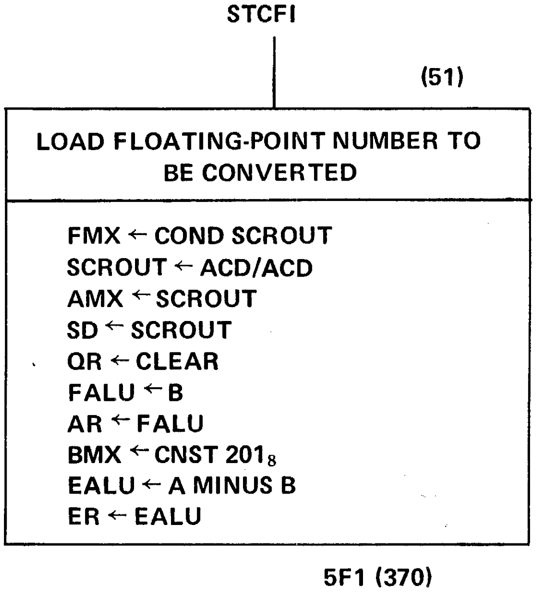
Can change NAD Bits from 0's to 1's
but not from 1's to 0's.

Initialize - FP11-C traps to state Ø
Floating minus Zero - FP11-C traps to state 1
UTRAP (MICROTRAP) - FP11-C traps to state 2
UJP (microjump) - FP11-C Jumps to state 3
Ø7-Ø2 0's Bits Ø1ØØ are 1's

4.3.3 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.

This block is ROM state 51 shown on page 7 of the flow diagrams and is part of the Store Convert Floating to Integer instruction.



The current address of this word is 51_8 shown in the upper-right corner of the block; the next address is 370, shown in the lower-right corner of the block. The 370 is preceded by 5F1 where the 5 designates the UBR field and the F1 designates the UAF (microaddress fork) bits.

Functionally, this ROM state transfers the number to be converted from the destination accumulator to the AR, initializes the fraction processor, and sets up a test to determine whether the number has an integer portion. In order to see how this is accomplished, each step in the ROM state is described. FMX ← COND SCROUT indicates that the FMX is selected for conditional scratchpad outputs on the fraction processor. If FD = 0, the FMX contains 24 bits of fraction (single-precision); if FD = 1, the FMX contains 56 bits of fraction (double-precision).

The next statement, SCROUT ← ACD/ACD, indicates that the addressing mode of the scratchpad is selecting the destination accumulator by monitoring FIR bits 07 and 06. Thus, EXPA and EXPB are selected for the exponent of the destination accumulator. AMX ← SCROUT indicates that the output of EXPA is routed to the AMX, and SD ← SCROUT causes the SS and SD flip-flops to be loaded from the scratchpad output which contains SD at this time. At this point, the destination accumulator has been transferred to the SD flip-flop, the exponent scratchpad, and the fraction scratchpad.

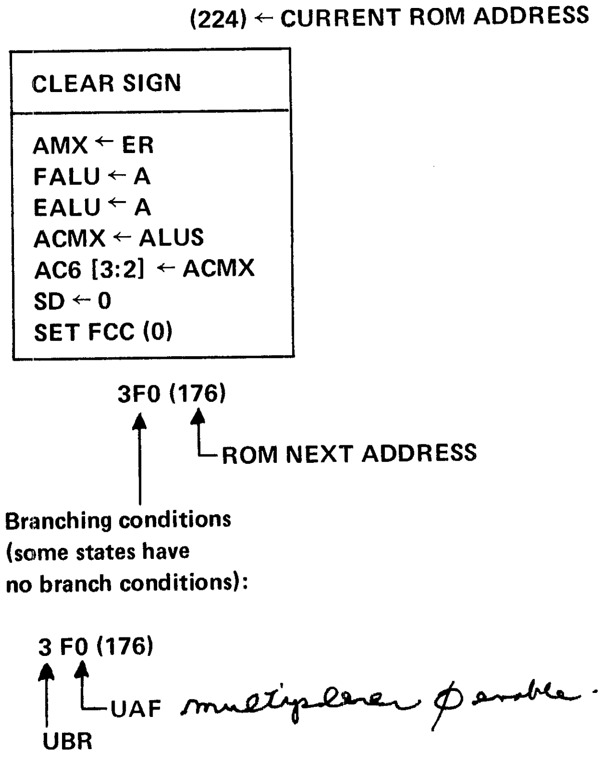
The next function to be performed is initialization of the fraction processor. First, the QR is cleared at time 2 of ROM state 51. Next, the FALU is selected for the FMX and the output of the FALU is loaded into the AR. This is accomplished by FALU ← B, and AR ← FALU. A constant of 201_8 is then loaded into BMX (BMX ← CNST 201) to test if the number has an integer portion. The next statement EALU ← A MINUS B, causes the constant of 201_8 to be subtracted from the exponent of the destination accumulator. The result of the subtraction currently in the EALU is transferred to the ER (ER ← EALU).

To summarize, this ROM state transferred the sign of the destination accumulator to SD, transferred the destination exponent to the ER after subtracting a constant of 201_8 to determine if the number was an integer, and transferred the fraction to the AR.

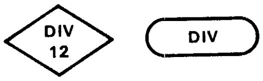
4.3.4 Control ROM Flow Diagram

This section describes the flow diagrams associated with the FP11-C. General points concerning the flow diagram symbology are described first. Table 4-3 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 5.



2. The flow diagram contains diamond shaped symbols with connector names listed inside. Below the connector name is the sheet reference. The diamond is connected to an oval-shaped symbol of the same connector name. The following symbols are reproduced from sheet 12 of the flow diagram. This indicates that the flow is connected to an oval symbol with the designation DIV. This oval symbol is on sheet 12 as referenced by the number in the bottom of the diamond.



11-3920

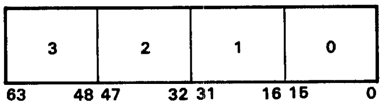
3. Certain connector names have numbers following them, which are used to differentiate between connectors of the same category. For example, on sheet 7 of the flow diagram there are diamond symbols designated LDCF.1 and LDCF.2. These symbols are connected to oval symbols.

4. Several statements of the following forms are on the flow diagrams:

   AC7[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 02 through 00, while the D following AC in the third statement specifies the destination accumulator designated by bits 07 and 06 of the FIR if address mode 0 is used; otherwise, AC6 is the source 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] specifies bits 63 through 32
[3:0] specifies bits 63 through 0

11-3921

   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 V1
0	0	0	1
0	1	1	1
1	0	2	3
1	1	3	3

In the MODF instruction, the fraction portion of the number is stored first, followed by the whole number. If an odd accumulator is specified, the fraction portion is destroyed by the storing of the whole number. If an even accumulator is specified, the fraction is stored in an even-numbered accumulator, while the whole number is stored in an odd-numbered accumulator which is the "OR 1" accumulator.

Table 4-3 Flow Diagram Statements

Statement	Description
ACD[3:0] ← ACMX	All four quadrants of the destination accumulator are written from the ACMX.
ACD[3:0]COND ← ACMX	If double-precision is specified (FD = 1), all four quadrants of the destination accumulator are written from the ACMX. If single-precision is specified (FD = 0), quadrants [3:2] of the destination accumulator are written from the ACMX.
ACD[3:2] ← ACMX	Quadrants 3 and 2 of the destination accumulator are written from the ACMX.
ACD[3] ← ACMX	Quadrant 3 of the destination accumulator is written from the ACMX. This quadrant contains the sign, exponent, and upper bits of the fraction scratchpad.
ACD OR 1[3:0]COND ← ACMX	Destination accumulator is made odd and ACMX writes all four quadrants if FD = 1, or writes quadrants 3 and 2 if FD = 0.
AC6[3:0] ← ACMX	All four quadrants of accumulator 6 are written from ACMX.
AC6[3] ← ACMX	Quadrant 3 of accumulator 6 is written from ACMX. This quadrant contains the sign, exponent, and upper bits of the fraction scratchpad.
AC6[2] ← ACMX	Quadrant 2 of accumulator 6 is written from ACMX.
AC6[1] ← ACMX	Quadrant 1 of accumulator 6 is written from ACMX.
AC6[0] ← ACMX	Quadrant 0 of accumulator 6 is written from ACMX.
AC7[2] ← ACMX	Quadrant 2 of accumulator 7 is written from ACMX. Accumulator 7[2] stores the FEA (floating exception address).
AC7[1] ← ACMX	Quadrant 1 of accumulator 7 is written from ACMX. Accumulator 7[1] stores the FEC (floating exception code).
ACMX ← ALUS	ACMX is selected to pass data from the EALU in the exponent processor to the exponent scratchpads and is selected to pass data from the FALU to the fraction scratchpads in the fraction processor.
ACMX ← DIMX	The DIMX supplies four 16-bit words to the ACMX.

Table 4-3 Flow Diagram Statements (Cont)

Statement	Description
ACOMX← ACD[3]	ACOMX is selected to pass data from quadrant 3 of the destination accumulator to OBUF. This quadrant contains the sign, exponent, and upper bits of the fraction.
ACOMX ← ACD[2]	ACOMX is selected to pass data from quadrant 2 of the destination accumulator to OBUF.
ACOMX ← ACD[1]	ACOMX is selected to pass data from quadrant 1 of the destination accumulator to OBUF.
ACOMX ← ACD[0]	ACOMX is selected to pass data from quadrant 0 of the destination accumulator to OBUF.
ACOMX ← AC6[3]	ACOMX is selected to pass data from quadrant 3 of accumulator 6 to OBUF.
ACOMX ← AC6[2]	ACOMX is selected to pass data from quadrant 2 of accumulator 6 to OBUF.
ACOMX ← AC6[1]	ACOMX is selected to pass data from quadrant 1 of accumulator 6 to OBUF.
ACOMX ← AC6[0]	ACOMX is selected to pass data from quadrant 0 of accumulator 6 to OBUF.
ACOMX ← AC7[2]	FEA (floating exception address) is passed through ACOMX from quadrant 2 of accumulator 7 for transfer to memory.
ACOMX ← AC7[1]	FEC (floating exception code) is passed through ACOMX from quadrant 1 of accumulator 7 for transfer to memory.
ACOMX ← FPS	Floating Point Status (FPS) is routed from FPSMX to ACOMX for subsequent transfer to the CPU.
ACS[3:2] ← ACMX	Source accumulator quadrants 3 and 2 are written from the ACMX.
ACS[3:0] ← ACMX	Source accumulator quadrants 3 through 0 are written from the ACMX.
ACS[3:0]← ACMX COND	AC source quadrants 3 and 2 are written from ACMX if FD = 0; AC source quadrants 3 through 0 are written from ACMX if FD = 1.
AFILL ← SIGN EXTEND	Used during multiplication operations. If the last operation was a subtraction, 1s are shifted into the most significant bit position of the AR; otherwise, 0s are shifted in.

Table 4-3 Flow Diagram Statements (Cont)

Statement	Description
AMX ← ER	AMX selects ER to be supplied to EALU.
AMX ← SCROUT	AMX selects scratchpads to be supplied to EALU.
AMX ← SC	AMX selects the step counter to be supplied to EALU.
AR ← 0 (LOW)	Bits 34 through 00 of the AR are forced to 0.
AR ← FALU	The AR is loaded from the FALU.
BMX ← CNST X	X is a constant which may range from 0 to 377_8. This constant is selected to be supplied to the EALU via BMX.
BMX ← DIMX	BMX is selected to pass DIMX (06:00) from the DIMX to the EALU.
BMX ← SCROUT	BMX is selected to pass data from the EXPB scratchpads to the EALU.
BMX ← SHIFT COUNT	The output of the shift control circuit is a shift count number which is routed to the output of BMX.
DIMX ← EALU	Low-order eight-bits of EALU are routed to the low-order eight-bits of DIMX. The EALU is right-justified and sign extended on EALU bit 08 through the remaining bits of the DIMX.
DIMX ← FDR	DIMX is selected to pass the contents of FDR to ACMX.
DOMX ← FPS	Floating-point status (FPS) is routed to DOMX via the FPSMX and the ACOMX.
EALU ← A	Output of AMX gated through EALU.
EALU ← B	Output of BMX gated through EALU.
EALU ← A MINUS B	BMX output is subtracted from the AMX output; the output of EALU yields the difference between the two.
EALU ← A PLUS B	AMX output is added to BMX output; the output of EALU yields the sum of the two.
EALU ← A PLUS B PLUS 1	AMX output is added to BMX output and the EALU yields the sum plus 1.
EALU ← A MINUS B MINUS 1	BMX output is subtracted from AMX output and the EALU yields the difference minus 1.

Table 4-3 Flow Diagram Statements (Cont)

Statement	Description
ENABLE -0 TRAP	Enables -0 detection circuit to flag -0, which is an undefined variable, if the trap is enabled.
ER ← EALU	ER is loaded from the EALU.
FALU ← 0	Output of FALU is set to all 0s.
FALU ← A	ASHFR outputs gated to output of FALU.
FALU ← B	FMX output gated to output of FALU.
FALU ← A MINUS 1	Output of FALU yields ASHFR minus 1.
FALU ← A MINUS B	Output of FALU yields ASHFR output minus FMX output.
FALU ← A•B	ASHFR output is logically ANDed with FMX output. If both ASHFR and FMX are 1s for a particular bit position, a 1 will appear in that bit position at the output of the FALU.
FALU ← ~A	ASHFR output is 1s complemented and routed to output of FALU.
FALU ← A•~B	The output of FMX is 1s complemented and is logically ANDed with the output of ASHFR. The result of this operation is at the output of FALU.
FALU ← A PLUS B	The ASHFR output is added to the FMX output; the output of the FALU yields the sum.
FALU ← A PLUS B PLUS 1	The ASHFR is added to the FMX output; the output of FALU contains the result plus 1.
FALU ← COND A PLUS B	Used during normalize operations. If the number of shifts required to normalize is within range (7 bits of shifting), then FALU outputs the sum of ASHFR plus the round bits (FALU = A plus B). If the number of shifts required to normalize is greater than 7, FALU outputs the contents of ASHFR (FALU = A).
FALU ← COND MUL/DIV	In multiplication and division operations, FALU can yield A, A plus B, or A minus B, depending on the bit patterns in the multiplication or division operation. This is described in more detail in Chapter 5.
FC ← DATI	The FC line corresponds to the C1 line in the CPU. If FC is negated, a DATI operation is specified.

Table 4-3 Flow Diagram Statements (Cont)

Statement	Description
FC ← DATO	The FC line corresponds to the C1 line in the CPU. If FC is asserted, a DATO operation is specified.
FD ← 0 IF SETF	SETF instruction forces FD (floating double) bit to a 0, indicating that the single precision operation is specified.
FD ← 1 IF SETD	SETD instruction forces FD (floating double) bit to a 1, indicating double precision operation is specified.
FMX ← COND SCROUT	FMX produces SCROUT on bits 58–35 and 0s on bits 34 through 00 if FD = 0 (single precision) or produces SCROUT on bits 58–00 if FD = 1 (double precision).
FMX ← QSHFR	QSHFR outputs are gated to output of FMX.
FMX ← RND	The FMX is forced to all 0's except that a round bit is asserted depending on the FD (floating double) bit. If FD = 0, bits 33 through 00 are 0s, bit 34 is a 1, and bits 59 through 35 are 0. If FD = 1, all bits are 0 except bit 02, which is a 1. For the Store Convert Floating to Integer instruction, this does not apply. Instead, the integer increment bit (bit 19 if IL = 1 or bit 35 if IL = 0) is forced to a 1 and is used for 2's complementing a negative number. (See description of Store Convert Floating to Integer instruction in Chapter 3.)
FMX ← SCROUT	Fraction scratchpads are selected at output of FMX (58–00).
FP REG WR ← 1	FP REG WR is a bit in the control ROM which informs the CPU to write the contents of a general register.
FP REQ ← CNST FP REQ ← IR Decode	Used during Negate and Absolute instructions. Loads the FP11-C with a 2 for single precision or a 4 for double precision. This number dictates how many cycles are necessary to complete the instruction.
FP SYNC ← 1	An FP11-C signal which is sent to the CPU to indicate that the FP11-C is ready to accept or receive data.
FP TRAP ← 1	Sets the floating error bit (bit 15) in the FPS register which informs the FP11-C to wait for a Trap Acknowledge from the CPU if the floating interrupt disable (FID) bit is not set. If this bit is set, the FP11-C will not trap to interrupt vector 244 and will return to the Ready state.
FPS ← DIMX	Floating-point status register is clocked and loaded with output of DIMX.

FP Req. ASSERTED unless immediate mode or no memory Detected

Table 4-3 Flow Diagram Statements (Cont)

Statement	Description
IL ← 0	SETI instruction forces IL (integer long) bit to a 0, indicating short integer (16-bit) mode.
IL ← 1	SETL instruction forces IL (integer long) bit to a 1, indicating long integer (32-bit) mode.
LONG CYCLE ← 1	Used in Divide and Store Convert Floating to Integer instructions and causes FP11-C to execute a 240-ns state instead of the 180-ns state.
NOP	No operation (NOP) occurs in this state.
OBUF ← ACOMX	OBUF register is being loaded with data from ACOMX for transfer to memory by the FP11-C control ROM.
QFILL ← 1	Ones are right-shifted into the high-order bits of QR. The QR can be loaded with one to eight 1s at a time and then shifted.
QMX ← COND SCROUT	
QMX ← QUOTIENT	Used only during division, and causes quotient to be formed in QR. (See divide description in Chapter 5.)
QMX ← QSHFR	QSHFR outputs are gated to output of QMX.
QR ← CLEAR	QR register is cleared.
QR ← QMX	QR register is loaded from QMX.
SC ← EALU	The step counter is loaded from the output of the EALU.
SCROUT ← ACD/ACD	Scratchpad outputs (EXPA, EXPB, and fraction) are selected for ACD (destination accumulator).
SCROUT ← ACD/ACS	Scratchpad EXPA is selected for destination accumulator and the fraction and EXPB scratchpads are selected for source accumulator.
SCROUT ← ACS/ACD	Scratchpad EXPA is selected for the source accumulator and the fraction, and EXPB scratchpads are selected for the destination accumulator. read two accumulators at once.
SCROUT ← ACS/ACS	Scratchpad ouputs (EXPA, EXPB, and fraction) are selected for source accumulator.
SCROUT ← AC6/AC6	Scratchpad outputs (EXPA, EXPB, and fraction) are selected for accumulator 6.

SEE FMX ← COND_SCROUT

Table 4-3 Flow Diagram Statements (Cont)

Statement	Description
SD ← 0	Clears both sign flip-flops (SS and SD).
SD ← SCROUT	SS and SD flip-flops are loaded from scratchpad outputs.
SD ← SC09	Used during Store Exponent instruction. SD flip-flop is loaded with step counter bit 09 (SC09) which, in this case, is the sign of the arithmetic operation performed in the exponent processor.
SD ← SS	Contents of SS flip-flop are transferred to the SD flip-flop.
SD ← ~ SS	The complement of the SS flip-flop is transferred to the SD flip-flop.
SD ← SS COND	If a subtraction operation is specified, the complement of the SS flip-flop is transferred to the SD flip-flop. If subtraction is not specified, the contents of the SS flip-flop are transferred to the SD flip-flop.
SD ← SS XOR SD	The SS flip-flop is exclusively ORed with the SD flip-flop and the result is stored in the SD flip-flop.
SET FCC (1)	With FCC on a 1, the FN and FZ bits are set according to the results of the arithmetic operation. Also enables floating FV bit to be set in accordance with EALU bit 08.
SET FCC (0)	With FCC on a 0, the FZ and FN bits are set according to the results of the arithmetic operation.
SHIFT CONT ← DIV	Used only during division (Chapter 5). Shift control is loaded with a count equal to the number of left shifts required to normalize the partial remainder during a division operation. Both the AR and the QR are left-shifted in the microstate following SHIFT CONT ← DIV. The remainder is normalized if bit 59 = 0 and bit 58 = 1, or if bit 59 = 1 and bit 58 = 0.
SHIFT CONT ← ALIGN	Lower three bits of EALU determine number of shifts necessary to align fractions for addition or subtraction operations. If SC09 is set, shifting of the AR is ~~inhibited~~. If SC09 is not set, shifting of the QR is ~~inhibited~~. enabled enabled

Table 4-3 Flow Diagram Statements (Cont)

Statement	Description
SHIFT CONT ← MULSHF	Shift control is loaded with a count from the multiplication hardware which causes right shift of both the AR and the QR, in the microstate following SHIFT CONT ← MULSHF.
SHIFT CONT ← CLR	Initializes shift control to shift by 0, which allows the shift operation to be aborted.
SHIFT CONT ← EALU	Shift control loaded from lower three bits of EALU. Bit 06 of EALU indicates whether number is positive (left-shift) or negative (right-shift). Bits 03, 04, and 05 determine if the number is within the range.
SHIFT CONT ← NORM	Shift control is loaded from a decode of the upper bits of the AR.
SS ← SCROUT	Both SS and SD get loaded with scratchpad output.
UBR ← DIMX	Used during Load Microbreak instruction. Microbreak register loaded from output of DIMX.
WAIT FOR FP ATTN	Places FP11-C in a pause state to wait for futher action from the CPU.

CHAPTER 5
ARITHMETIC ALGORITHMS

5.1 INTRODUCTION

This chapter describes the arithmetic algorithms associated with the FP11-C. Addition and subtraction are described first, 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-C. State diagrams and examples of the multiplication and division algorithms are provided.

5.2 FLOATING-POINT ADDITION AND SUBTRACTION

Floating-point addition and subtraction are 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.

Operand	**Exponent**	**Fraction**	**Sign**
Destination	Scratchpad A	AR	SD
Source	Scratchpad B	QR	SS
Result	ER	AR	SD

For example, the exponent of the result of an addition or subtraction is found in the ER, 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 and is located in memory if mode 0 is not specified. In the latter case, the operand in memory is transferred to the FP11-C and temporarily stored in AC6.

5.2.1 Description of Sign Processing

To understand how the hardware implements sign calculations for floating-point addition and subtraction, refer to Table 5-1. The following text attempts to educate the reader in the use of this table. Normally, SS (sign of source) represents the sign associated with the source operand (ACS) and SD (sign of destination) represents the sign of the destination operand (ACD). The sign of the result is stored in SD.

When addition with quantities having like signs is specified, or subtraction with unlike signs is specified, the hardware performs an add operation. The sign of the result is positive if the quantities are positive and is negative if the quantities are negative.

Example 1:

$$\begin{array}{r} +8 \\ +7 \\ \hline +15 \end{array} \qquad \begin{array}{r} -8 \\ -7 \\ \hline -15 \end{array}$$

When subtraction is specified with quantities having unlike signs, the hardware actually performs an add operation. The sign of the result is the sign of the minuend.

Example 2:

$$\begin{array}{rcr} +8 & \qquad & -8 \\ \underline{-(-7)} & & \underline{-(+7)} \\ +15 & & -15 \end{array}$$

When addition is specified with quantities having unlike signs, the quantities are subtracted and the sign of the result is the sign of the quantity with the larger magnitude.

Example 3:

$$\begin{array}{rcrcrcr} +8 & \qquad & -8 & \qquad & +7 & \qquad & -7 \\ \underline{-7} & & \underline{+7} & & \underline{-8} & & \underline{+8} \\ +1 & & -1 & & -1 & & +1 \end{array}$$

When subtraction is specified with quantities having like signs, the quantities are subtracted, which is accomplished by changing the sign of the subtrahend and adding. The sign of the result is then the sign of the quantity with the larger magnitude.

Example 4:

$$\begin{array}{rcrcrcr} +8 & \qquad & -8 & \quad & +7 & \qquad & -7 \\ \underline{-(+7)} & & \underline{-(-7)} & & \underline{-(+8)} & & \underline{-(-8)} \\ +1 & & -1 & & -1 & & +1 \end{array}$$

The above concepts form the basis for determining the sign as shown in Table 5-1. First, note that combinations 1 through 4 are for the add instruction and 5 through 8 for the subtract instruction. In combination 1, the operands have positive like signs (SS = 0, SD = 0); in combination 4, the quantities have negative like signs (SS = 1, SD = 1). Consequently, the hardware performs an addition. In combination 2, the source operand is positive (SS = 0) and the destination operand is negative (SD = 1), while in combination 3 the source operand is negative and the destination operand is positive. Consequently, the hardware performs a subtraction since the operands are of unlike signs. The sign of the result is the sign of the quantity with the larger magnitude.

Combinations 5 through 8 define the subtract instruction. Note that combinations 6 and 7 deal with operands of unlike signs, which means that the hardware performs an add operation. Combination 5 specifies positive operands (SS = 0, SD = 0) and combination 8 specifies negative operands. Thus, the hardware performs a subtraction, with the result getting the sign of the destination if that is the larger quantity, or the complement of the sign of the source if that is the larger quantity. The source and destination operands (ACS and ACD) are added or subtracted with respect to magnitude only as indicated by the absolute value signs (|ACD| + |ACS|). Several examples illustrate this.

Table 5-1 Add and Subtract Implementations

Combination	SS	SD	Instruction	Hardware Performs	Sign of Result	
					Positive Parentheses	Negative Parentheses
			Add Instruction			
1	0	0	ACD ← +(|ACD|+|ACS|)	Add	SD ← SD	–
2	0	1	ACD ← -(|ACD|-|ACS|)	Subtract	SD ← SD	SD ← SS
3	1	0	ACD ← +(|ACD|-|ACS|)	Subtract	SD ← SD	SD ← SS
4	1	1	ACD ← -(|ACD|+|ACS|)	Add	SD ← SD	–
			Subtract Instruction			
5	0	0	ACD ← +(|ACD|-|ACS|)	Subtract	SD ← SD	SD ← ~SS
6	0	1	ACD ← -(|ACD|+|ACS|)	Add	SD ← SD	–
7	1	0	ACD ← +(|ACD|+|ACS|)	Add	SD ← SD	–
8	1	1	ACD ← -(|ACD|-|ACS|)	Subtract	SD ← SD	SD ← ~SS

Complement [illegible] and But [illegible].

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

Example 1:

Assume an add instruction is specified.

ACD = +3, SD = 0
ACS = -5, SS = 1

ACD ← + (|ACD| - |ACS|) = +(3 - 5) = -2

SD ← SS because the quantity in parentheses is negative. Therefore, ACD is loaded with 2 and SD is loaded with a 1.

Example 2:

Assume a subtract instruction is specified.

ACD = -5, SD = 1
ACS = -3, SS = 1

ACD ← -(|ACD| - |ACS|) = -(5 - 3) = -2

SD ← SD if the quantity in parentheses is positive.
SD ← ~ SS if the quantity in parentheses is negative.

The quantity in parentheses is positive, so SD remains a 1.

5.2.2 Relative Magnitude

During fraction alignment (which occurs when the exponents are unequal), the relative magnitude of the operands is detected by subtracting the exponents; the difference is the number of right shifts the smaller number is to be shifted to effectively equalize the exponents. If the exponent of this number is very small compared to the other number, it can be completely shifted out of the register it is stored in and thus will have no significance in the operation. To avoid unnecessary shifting in these cases, the relative magnitude of the numbers is tested. If the number of shifts required to align the fractions is greater than 25 (single-precision) or 57 (double-precision), the FP11-C hardware will not attempt to align the operands. In these cases the unshifted operand is the answer.

5.2.3 Testing for Normalization

All floating-point numbers must be normalized. In order to normalize a number, bit 59 must be a 0 and bit 58 must be a 1. The result of any arithmetic operation must be normalized. In addition, the fraction of the result is always positive; therefore, the hardware will simply normalize the number. In subtraction, the fraction may be negative or 0, neither of which can be normalized. After a subtraction operation has been performed in which the QR was not aligned, 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 and cannot be normalized. If the number in the AR is positive, it may 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, which cannot be normalized. If there is no sign change in the subtraction, the AR contains a positive number, which can be normalized.

During 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.

5.2.4 Floating-Point Addition

For floating-point addition and subtraction, the exponents must be equal. In general, there are two methods of accomplishing this. One is to left-shift the fraction of the larger number and decrease its exponent accordingly.Eachleft shift represents multiplication by a power of 2 and consequently, the exponent must be decreased by 1. The disadvantage of this method is that the most significant bits of the fraction are shifted out of the register they are stored in and are lost.

A second method and the one used by the FP11-C is to right-shift the fraction with the smaller exponent and increase the exponent accordingly. Each right shift corresponds to division by a power of 2 and consequently, the exponent must be increased by a power of 2. When the exponents have been made equal, the addition or subtraction can be performed. The exponent of the result then becomes the larger of the two exponents. After the addition or subtraction, the fraction must be normalized. This means that bit 59 must be equal to 0 and bit 58 must be equal to 1. The implementation of addition and subtraction will first be described from the standpoint of the addition of two numbers. This is the case where there is addition of two numbers with like signs or the subtraction of two numbers with unlike signs. In both cases, the two arguments are actually added. Several examples demonstrate this point.

Addition With Like Signs

$$\begin{array}{r}+3\\+3\\\hline+6\end{array}\qquad\begin{array}{r}-3\\-4\\\hline-7\end{array}\qquad\begin{array}{r}-4\\-6\\\hline-10\end{array}$$

Subtraction With Unlike Signs

$$\begin{array}{rrr} +3 & -3 & +6 \\ -(-4) & -(+4) & -(-2) \\ \hline +7 & -7 & +8 \end{array}$$

Note that in all examples, the two quantities are actually added. Paragraph 5.4 describes floating-point subtraction which consists of the addition of two numbers with unlike signs or the subtraction of two numbers with like signs. Several examples demonstrate this point.

Addition With Unlike Signs

$$\begin{array}{rrr} +3 & +6 & -3 \\ -4 & -7 & +5 \\ \hline -1 & -1 & +2 \end{array}$$

Subtraction With Like Signs

$$\begin{array}{rrr} -3 & +6 & -7 \\ -(-4) & -(+2) & -(-5) \\ \hline +1 & +4 & -2 \end{array}$$

Note that in these cases, a subtraction operation is actually taking place. The operation of the data path for floating-point subtraction is similar to that of floating-point addition, except that the following point must be kept in mind.

> The FALU is performing A minus B for subtraction; therefore, the result must be examined for the possible cases of 0 or negative results which require special treatment by the FP11-C hardware (Paragraph 5.4).

5.2.4.1 Hardware Implementation of Addition – The difference between the two exponents is initially stored in the step counter and represents the destination exponent minus the source exponent.

The destination fraction is loaded in the AR and the source fraction is loaded in the QR. The exponent difference which is stored in the step counter is applied to a ROM, which creates a negative absolute value that is transferred to the ER. The ER serves as a holding register for the number of right shifts to be accomplished in alignment.

5.2.4.2 Out-of-Range Flag – The ROM which creates the negative absolute value of the exponent difference also contains an out-of-range flag which is used in association with floating/ double mode. If the exponent difference is greater than 25 (floating double = 0) or greater than 57 (floating double = 1), the out-of-range flag is asserted, indicating that the difference between the numbers is such that one number would be shifted out of the register.

If the numbers are within range, the out-of-range flag is negated and the negative exponent difference is stored in the ER.

5.2.4.3 Shift-Within-Range Flag - At this point, the shift control logic is clocked and the shift-within-range flag is allowed to stabilize. If this flag is asserted, it indicates that the fractions are within eight shifts of being aligned. Consequently, the FP11-C will shift the smaller fraction until the exponents are aligned and then add or subtract the fractions, depending on the instruction. At this point, the ER is loaded with the larger exponent and no longer contains the exponent difference. The SC has the exponent difference and, based on whether the difference is positive or negative, it can be determined which exponent is larger. If the exponent difference is negative, it means that the source operand is larger than the destination operand and the source operand is referenced from the scratchpad and contains the sign of the result. If the exponent difference is positive, it means that the destination operand is larger than the source operand. In this case, the destination operand is referenced and contains the sign of the result.

If the shift-within-range flag is not asserted, the FP11-C performs eight right-shifts and subtracts eight from the step counter. If the difference is still greater than eight, the loop is reiterated, the shift-within-range remains unasserted, and another eight right-shifts are performed by the FP11-C. This looping is iterative until the shift-within-range flag is asserted, which indicates that eight or less shifts are necessary to align the exponents. When this point is reached, the remaining right-shifts are performed and the fractions are added or subtracted.

5.2.4.4 Normalizing the Result - The fractions are stored in the AR and QR and are transferred to the FALU where the addition or subtraction takes place. The result is routed back to the AR. However, en route to the AR, it is examined by a normalization shift network which determines how far and in which direction the result must be shifted to be normalized. If bit 59 = 1, the result is normalized by right-shifting one place, which makes bit 59 = 0 and bit 58 = 1. On the other hand, if bits 58 and 59 = 0, the FP11-C must left-shift the result to have a normalized number. A shift-within-range flag is also associated with normalization. If the number can be normalized within seven shifts, the shift-within-range flag is asserted, and the unnormalized number is rerouted through the ASHFR and FALU again where it is normalized. Note that the normalization shift network encodes the number of shifts and direction of shift one microstate ahead of the actual shifting.

If the shift-within-range flag is not asserted, the ASHFR shifts the result seven places and transfers the result to the FALU, which reroutes it back to the AREG. As the result is transferred from the FALU to the AREG, it is reexamined by the normalization shift network which again determines the direction and number of shifts. This loop is repeated until the shift-within-range flag is asserted, which indicates that seven or less shifts are required to normalize the result.

5.2.4.5 Truncate or Rounding - If the FP11-C is in truncate mode and the shift-within-range flag is asserted, the result is shifted the required number of shifts, routed through the A side of the FALU, and stored in the AREG. If the FP11-C is in round mode, a 1 is inserted in bit 34 (single-precision) or bit 02 (double-precision). When the shift-within-range flag is asserted, the FALU takes the result from AREG, which is within seven shifts of being normalized, shifts it, and adds the round bit (B input to FALU) to the shifted fraction. The result is now a normalized, rounded fraction. However, if the fraction contained all 1s, adding the round bit to it will cause an arithmetic overflow (bit 59 = 1). This condition is detected by the normalization shift encoder which now left-shifts the result by 1, causing bit 59 to go to 0 and bit 58 to go to 1 as the fraction is stored in the destination accumulator.

5.2.4.6 Adjusting Exponent During Normalization - When the result of the addition is being normalized, it is necessary to keep track of the number of shifts required to normalize so that the exponent of the result may be properly adjusted. The ER contains the larger of the two exponents, i.e., the exponent of the answer. This exponent is updated during normalization by adding the number of right addition $\begin{array}{r} a \times 2^b \\ + c \times 2^q \\ \hline \end{array}$

1. align exponents
   - shift the fraction of the number with smaller exponent right the number of places equal to difference in exponents
2. Larger exponent is exponent of result
3. Add fractions

shifts (or conversely subtracting the number of left shifts) directly. This is accomplished by feeding the shift count to the EALU via the BMX. In the case where the fraction is normalized by right shifting (bit 59 equal to 1), the exponent must be incremented. This is accomplished by the shift control network which asserts 1s on four lines and sends them to the BMX. These 1s are sign-extended in the BMX to ten 1s which are subtracted from the exponent in the ER. The subtraction is accomplished by 2's complement addition, which increases the exponent by 1. The example below illustrates this point.

4. normalize results

Exponent in ER	1000010100	
Sign extended input from BMX	1111111111	
Exponent in ER	1000010100	
2's Complement of sign-extended input	+0000000001	
	1000010101	This number is 1 greater than previous exponent in ER.

5.2.5 Floating-Point Subtraction

In floating-point subtraction, the source operand is subtracted from the destination operand. The source operand is loaded in the QR and the destination operation is loaded in the AR, which means that the FALU will perform AR minus QR.

The step counter is loaded with the destination exponent minus the source exponent, which represents the exponent difference between the two operands.

5.2.5.1 Negative Exponent Difference – If the exponent difference is negative (indicated by SC09 = 1), it means that the source operand in the QR is greater than the destination operand in the AR. Since the AR is the smaller number, it is right-shifted to align the fractions. When the fractions are aligned and then subtracted, the difference will be a 2's complement negative number. This number is 2's complemented to make it a positive number and the sign is adjusted to be the sign of the source operand. A simple example to demonstrate this point follows.

Example:

	25_{10}	11001	
Subtract	31_{10}	11111	Take 2's complement and add
	-6_{10}		
		11001	
2's complement of 11111		+00001	
		11010	$= 26_{10}$ which is not the correct result and which represents a 2's complement negative number.

The answer must be 2's complemented to acquire the proper result.

11010	
00101	1's complement
+1	Add 1
00110	$= 6_{10}$

ADDF (4)+, AC0

R4 = 1ØØØ

$\left.\begin{array}{l} 1ØØØ/Ø417ØØ \\ 1Ø92/ØØØØØØ \end{array}\right\} 96_{10}$

AC0 = 1Ø17ØØØØØØØ} 7_{10}

many 2^{16} Bit places

AC0[3:2]

Ø1 000 001 111 000

5.2.5.2 Determining Exponent Difference – During addition of unlike signs or subtraction of like signs (ALU performs subtract), SC bit 09 is tested. If the exponent difference in the SC is positive or zero, SC09 will be a 0. To determine if the exponent difference is 0, the FP11-C logic clocks the branch condition codes and checks BZ. If BZ is asserted, this indicates an exponent difference of 0. In this case neither the QR nor AR were shifted and the subtraction of the fractions could result in a zero difference, a negative difference, or a positive difference. If bit 59 = 1, the resultant fraction is a 2's complement negative number and must be converted to a positive sign and magnitude number.

If bit 59 = 0, the result of subtracting the fractions is either zero or positive. The test for a result of 0 is done by decrementing the result. If the result is 0, decrementing it will cause bit 59 to go to a 1 due to the borrow rippling all the way through the result. The FP11-C will then store 0s in the destination accumulator. If decrementing the result causes bit 59 to remain a 0, then the result of subtracting the fractions was positive. With a positive result, the FP11-C will add 1 to the result to restore it to its original value before storing it.

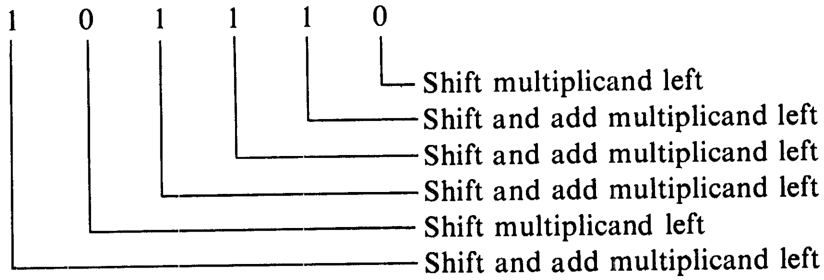
5.2.5.3 Positive Exponent Difference – If the exponent difference is positive (AR minus QR), it indicates that the destination operand is larger than the source operand. Since the QR is the source operand and is smaller than the destination operand, it means that the QR is right-shifted to align the fractions. The fractions are then subtracted. This subtraction must result in a positive number in the QR; therefore, the FP11-C does not have to test it for zero or negative, but instead normalizes it immediately.

5.3 FLOATING-POINT MULTIPLICATION

The FP11-C 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 multiplication techniques are described, followed by a description of the hardware employed in the FP11-C. Figure 5-1 is a simplified flow diagram of the multiply algorithm.

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
Shift and add multiplicand left
Shift and add multiplicand left
Shift and add multiplicand left
Shift multiplicand left
Shift and add multiplicand left

The same results is obtained in the FP11-C by shifting the partial product and the multiplier right as opposed to shifting the multiplicand left.

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, as shifting in the FP11-C effectively takes no time. An improvement over this method is a process of shifting over 1s and 0s.

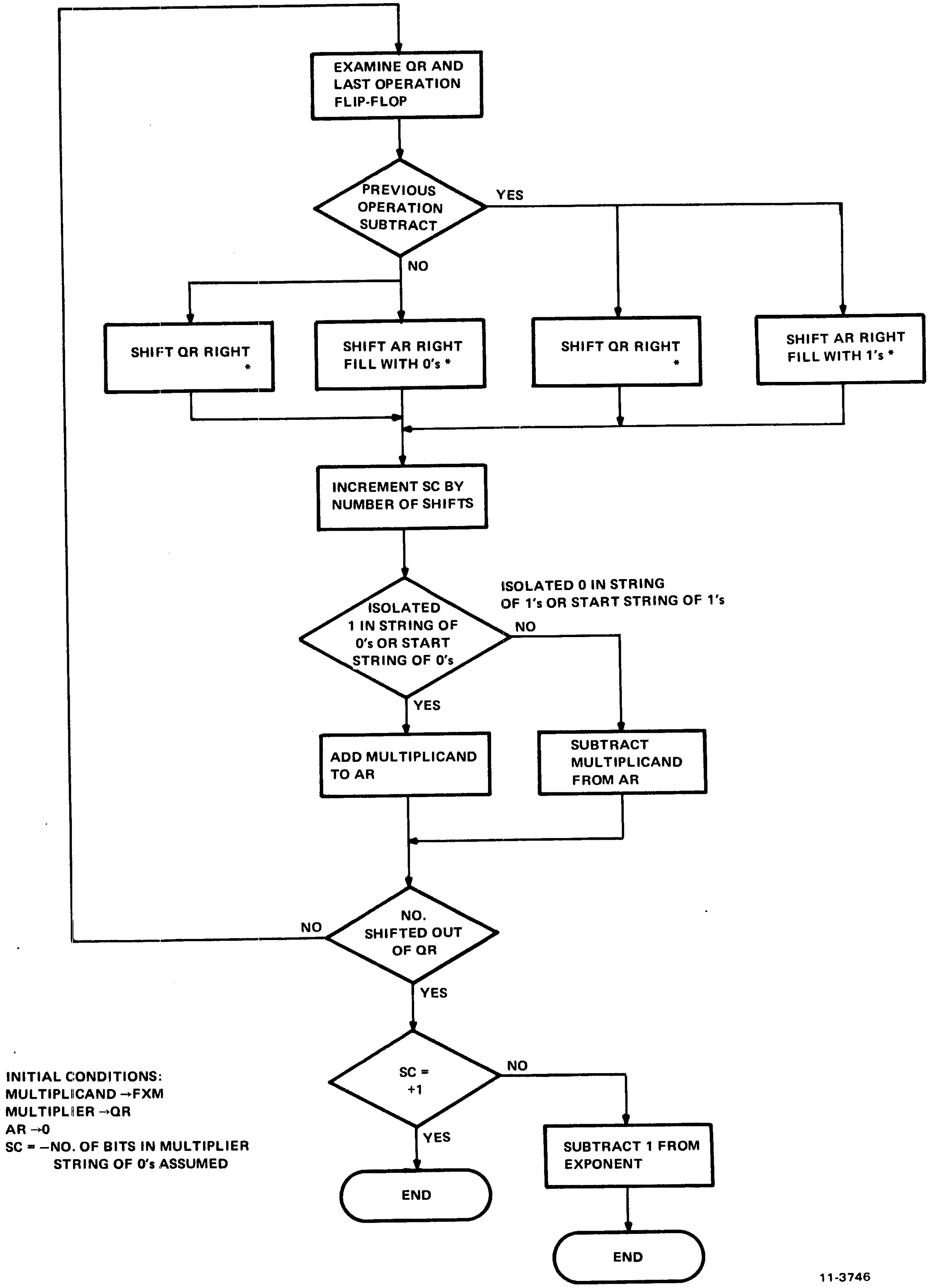
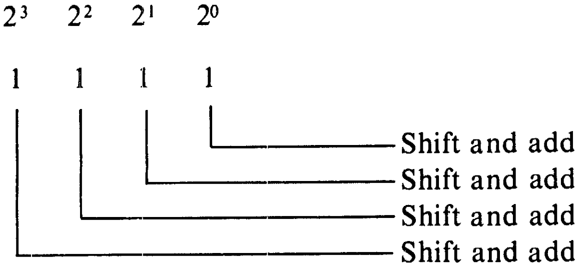

EXAMINE QR AND LAST OPERATION FLIP-FLOP
PREVIOUS OPERATION SUBTRACT
YES
NO
SHIFT QR RIGHT *
SHIFT AR RIGHT FILL WITH 0's *
SHIFT QR RIGHT *
SHIFT AR RIGHT FILL WITH 1's *
INCREMENT SC BY NUMBER OF SHIFTS
ISOLATED 1 IN STRING OF 0's OR START STRING OF 0's
ISOLATED 0 IN STRING OF 1's OR START STRING OF 1's
NO
YES
ADD MULTIPLICAND TO AR
SUBTRACT MULTIPLICAND FROM AR
NO
NO. SHIFTED OUT OF QR
YES
SC = +1
NO
YES
END
SUBTRACT 1 FROM EXPONENT
END
INITIAL CONDITIONS:
MULTIPLICAND →FXM
MULTIPLIER →QR
AR →0
SC = –NO. OF BITS IN MULTIPLIER
STRING OF 0's ASSUMED
11-3746

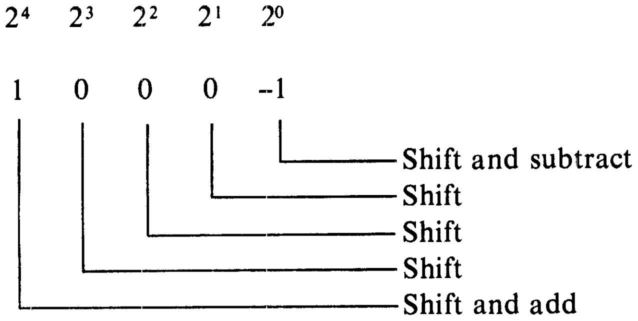
Figure 5-1 Simplified Multiplication Flow Diagram

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 01111 can be represented as 1000 – 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 below.

Old Method

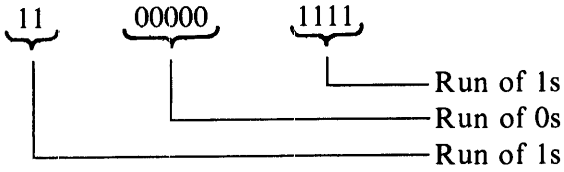


Shifting Over 1s and 0s



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

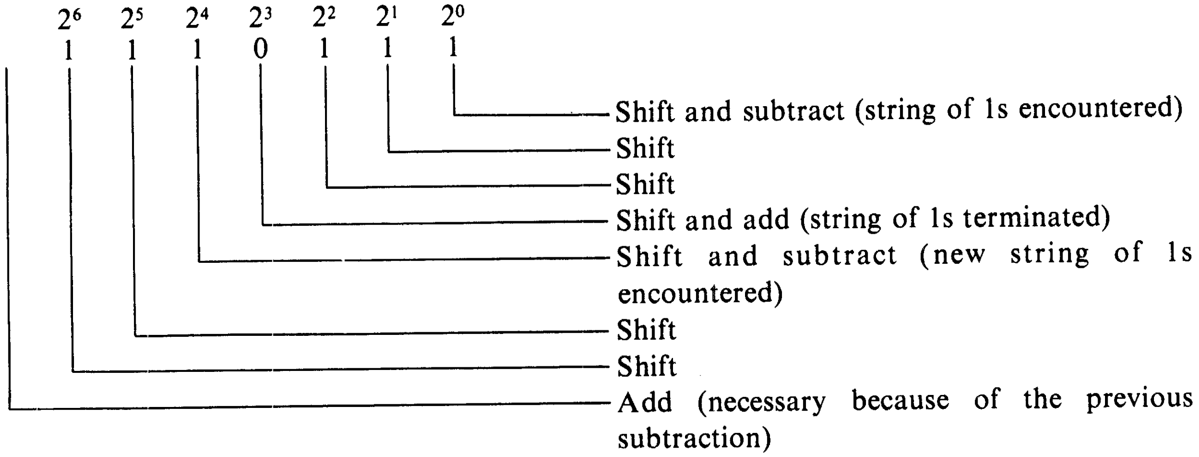
An additional improvement over this method is developed where an isolated 1 occurs in a string of 0s or an isolated 0 occurs in a string of 1s. In this method, the multiplier is examined two bits at a time to look for runs of 1s or 0s. A run is defined as a string of two or more consecutive identical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

1. multiply fractions
2. add exponents
3. normalize results

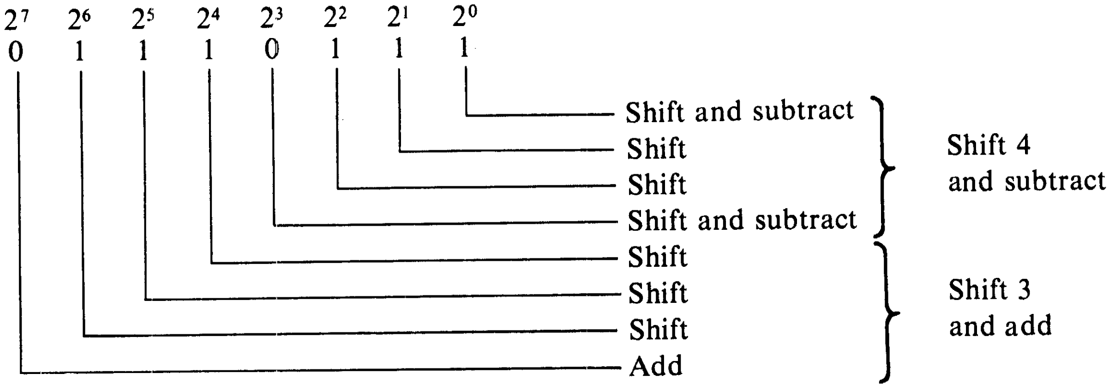
$$a \times 2^{b}$$
$$\times\, c \times 2^{d}$$
$$(A \times C) \times 2^{(b+d)}$$



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

Note in this example that in the 2^3 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.

This example can be reduced to the following:

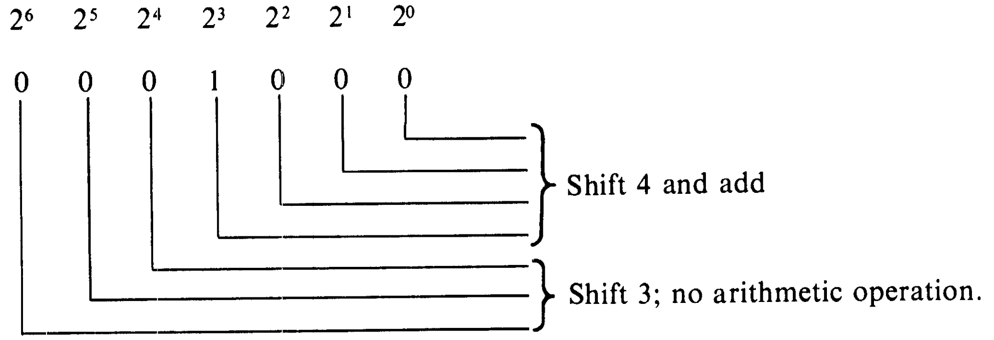


The FP11-C contains a shifting network which allows shifting to occur in parallel with and in combination with an arithmetic operation (add or subtract). The number of shifts performed during each cycle is equal to the number of shifts dictated by the multiplication algorithm until an arithmetic operation occurs, up to a maximum of six shifts. In the example above, then, the FP11-C will simultaneously do four shifts and subtract. Then, in the next microstate, the FP11-C will simultaneously do three shifts and add. This parallel shifting and simultaneous arithmetic operation provides the FP11-C with a very efficient multiplication operation.

NOTE

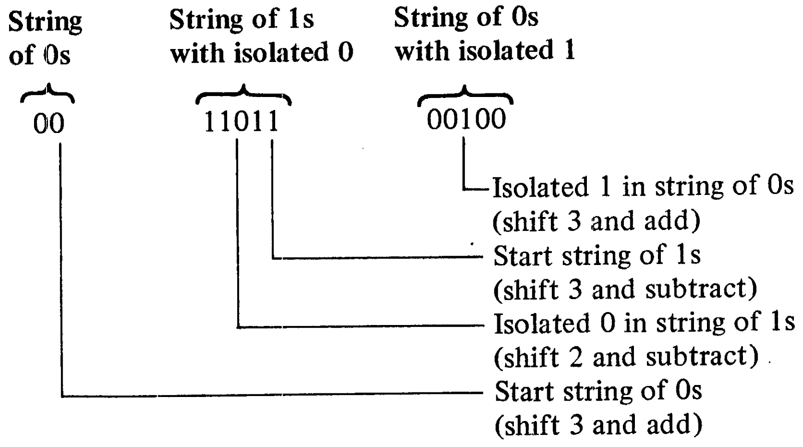
The FP11-C hardware is initialized to a string of 0s. Consequently, if a 1 is encountered as the first bit, it may be either the first 1 in a string of 1s or an isolated 1 in a string of 0s depending on what the second bit encountered is.

The FP11-C will handle an isolated 1 in a string of 0s as shown in the following example. Assume the last operation was an add and the FP11-C is in a string of 0s.

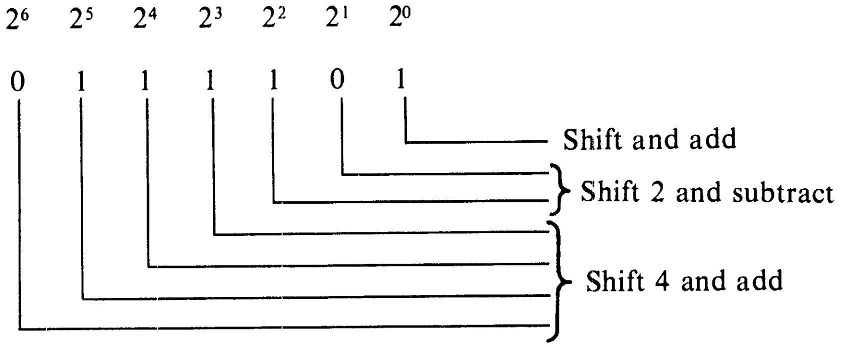


This example requires one arithmetic operation (an addition) that occurs where the 1 bit is encountered. Consequently, the FP11-C would process this example in two microstates: four shifts and an add in the first microstate and three shifts in the second microstate.

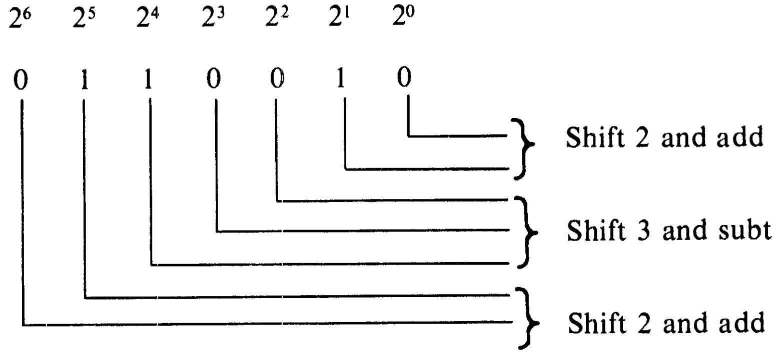
In addition to the cases of an isolated 0 in a string of 1s or an isolated 1 in a string of 0s, two other cases must be handled: termination of a string of 0s that represents the beginning of a string of 1s or conversely, termination of a string of 1s that represents the beginning of a string of 0s. The example below shows a typical multiplier and how it would appear to the FP11-C. Note that the first five digits are 00100, which corresponds to a string of 0s with an isolated 1. The next two digits denote the start of a string of 1s. Further examination reveals that there is a string of 1s with an isolated 0. This is followed by two 0s that indicate the start of a string of 0s. It is extremely important to be aware of this concept in order to understand the multiplication algorithms.


String of 0s
String of 1s with isolated 0
String of 0s with isolated 1
00
11011
00100
Isolated 1 in string of 0s (shift 3 and add)
Start string of 1s (shift 3 and subtract)
Isolated 0 in string of 1s (shift 2 and subtract)
Start string of 0s (shift 3 and add)

Several other examples are provided below.



In this example, the FP11-C would first do a shift and add, followed by a shift of two and a subtract, and concluding with a shift of four and an add. Thus, a total of three microstates would be used.



In this case, the FP11-C would do a shift by two and an add, followed by a shift of three and a subtract, and terminating with a shift of two and an add.

5.3.2 Hardware Implementation of Multiplication

To multiply in the FP11-C, the multiplicand is addressed in the fraction scratchpad which drives the input of FMX; the multiplier is loaded in the QR; and the partial product is formed in the AR which is initially cleared. Both the QR and the AR are right-shifted as the algorithm proceeds.

The FP11-C utilizes two ROM encoders to examine the seven least significant bits of the multiplier and the arithmetic operation performed in the previous microstate. One encoder is used for single-precision and examines QR bits 41 through 35 and the other is used for double-precision and examines QR bits 09 through 03. Up to six left-shifts can be performed simultaneously by the FP11-C hardware.

In some cases where six shifts can occur, the arithmetic operation must be inhibited, since the FP11-C cannot determine whether the arithmetic operation to be performed is an add or subtract. The example below shows the four cases in which the add or subtract is inhibited.

Encoder	**9**	**8**	**7**	**6**	**5**	**4**	**3**
	1	0	0	0	0	0	0
	0	0	0	0	0	0	0
	0	1	1	1	1	1	1
	1	1	1	1	1	1	1

For example, in the first number there is a string of 0s followed by a 1 in bit 09. However, it is not known whether this 1 is an isolated 1 in a string of 0s (add) or the beginning of a string of 1s (subtract). To determine this, it would be necessary to examine bit 10 which is not visible to the encoder.

In the next number (all 0s), six shifts will occur since this is the maximum number, and no arithmetic operation is indicated since bit 09 is a 0, indicating a continuation of the string of 0s.

The third number (all 1s except bit 9 = 0) is a string of 1s. However, the arithmetic operation is inhibited since it is not known whether the 0 in bit 09 is the termination of a string of 1s (add) or an isolated 0 in a string of 1s (subtract).

In the case of the fourth number (all 1s), six shifts will occur since a string of 1s is incurred and only a maximum of six shifts can be performed at a given time. Also, bit 09 is a 1 indicating a continuation of the string of 1s.

The step counter (SC) is used to keep track of the number of bits of the multiplier which have been processed as the algorithm proceeds. It is initially loaded with a negative number corresponding to the number of bits in the multiplier (24 for single-precision, 56 for double-precision) and incremented as each bit of the multiplier is shifted out.

To understand the multiplication algorithm, several examples are provided to illustrate the concepts. For simplicity, 4- and 5-bit numbers are used, although the actual numbers used by the FP11-C are much greater in length (24 bits single-precision; 56 bits double-precison).

5.3.3 Example 1 of Multiplication Algorithm

$.1000 \times 2^4 \times .1101 \times 2^4 = .01101000 \times 2^8$

```
     .1000
   X .1101
  ---------
      1000
     10000
    1000
  ---------
  .01101000
```

Initial Conditions:

Multiplicand (FMX)	.1000
Multiplier (QR)	.1101
AR	.00000000
SC	-4
String	0s
Previous Operation	Add

Cycle 1: Examine QR

Isolated 1 in string of 0s	.1101
Shift QR right 1 place	+.0110
Shift AR right 1 place	+.00000000
Add multiplicand to AR	+.1000
New AR	+.10000000
Increment SC by 1	SC = -3

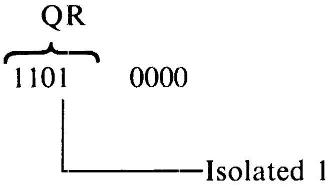
Cycle 2: Examine QR

Terminate string of 0s	.0110
Shift QR right 2 places	0.0001
Shift AR right 2 places	0.00100000
Subtract multiplicand from AR	1.1000 (2's complement add)
	1.10100000
Increment SC by 2	SC = -1

Cycle 3: Examine QR

Terminate string of 1s	.0001
Shift QR right 2 places	.0000
Shift AR right 2 places (because of previous subtract)	
Sign-extend AR with 1s	.11101000
Add multiplicand to AR	.1000
	.01101000 × 2^8
Increment SC by 2	SC = +1

Example 1 shows the number .1000 multiplied by .1101. The multiplicand is loaded at the input to FMX, the multiplier is loaded in the QR, and the AR is initialized. The step counter (SC) is set to the negative value of the number of bits in the multiplier. In this case, the SC is equal to -4. The FP11-C initially assumes the hardware is in a string of 0s and begins the operation by examining the low-order bits of the QR. This is an isolated 1 in a string of 0s.



Consequently, the QR and AR are shifted right one place. This indicates the SC should be incremented by 1. The multiplicand is then added to the AR to form the first partial product.

In the next cycle, the shifted QR is examined. Since the last operation was an add (indicating that the unit is processing a string of 0s), examination of the QR indicates termination of this string and the start of a string of 1s. This means the QR and AR will be shifted two places to the right. Consequently, the step counter is incremented by 2. Then the multiplicand is subtracted from the AR since a subtraction is performed when initiating a string of 1s. The subtraction is performed by 2's complementing the multiplicand and adding it to the AR.

In the third cycle, the QR shifted in cycle 2 is examined (.0001). In cycle 2, a string of 1s was started and examination of the QR now reveals termination of the string of 1s. Consequently, the AR and QR are shifted right two places. However, when the previous operation is a subtraction (as in cycle 2), the AR is right-shifted and sign-extended with 1s, which means that 1s rather than 0s are shifted into the upper bits of the AR. If the previous operation was an add, 0s are shifted into the upper bits of the AR. Since the QR and AR are right-shifted two places, the step counter is incremented by 2, causing it to go from -1 to +1. The multiplier in the QR now has been completely shifted out of the QR. Since the step counter is +1, (non-negative) the operation is terminated. If the step counter is 0, the exponent must be adjusted by adding 1 to it as described in the next example.

5.3.4 Example 2 of Multiplication Algorithm

.11110 × .10001 = .0111111110

```
      .11110
     X.10001
    --------
       11110
    11110000
   ---------
 .0111111110   Unnormalized
 .1111111100   Normalized
```

Initial Conditions:

Multiplicand	.11110
AR	.10001
SC	-5
String	0s
Previous Operation	Add

Cycle 1: Examine QR

Isolated 1 in string of 0s	.10001
Shift QR right 1 place	.01000
Shift AR right 1 place	.0000000000
Add multiplicand to AR	.11110
New AR	.1111000000
Increment SC by 1	SC = –4

Cycle 2: Examine QR

Isolated 1 in string of 0s	.01000
Shift QR right 4 places	.00000
Shift AR right 4 places	.0000111100
Add multiplicand to AR	.11110
New AR	.1111111100
Increment SC by 4	SC = 0

Algorithm is completed when number in QR is shifted out. If SC = 0, 1 is subtracted from exponent. This will occur if the multiplier has an isolated 1 as its most significant bit.

Example 2 shows a 5-bit multiplicand and a 5-bit multiplier, indicating that the step counter is initially set to –5. The multiplicand is transferred to the input of FMX, the multiplier is stored in the QR, and the AR is initially cleared and is used to store the partial products and, subsequently, the final result. The FP11-C initially assumes it is in a string of 0s (i.e., the last operation was an add).

1. divide fraction
2 subtract the exponent
3 normalize

In the first cycle, the QR is examined and is seen to contain an isolated 1 in a string of 0s. Thus, the AR and the QR are right-shifted one place, the step counter is incremented by 1, and the multiplicand is added to the AR.

In the next cycle, the QR is examined and is seen to contain an isolated 1 in a string of 0s. Consequently, the AR and the QR are right-shifted four places, the step counter is incremented by 4, causing it to go from -4 to 0, and the multiplicand is added to the AR. In this example, the number in the QR has been shifted completely out of the QR and the SC is 0; since the SC = 0, and not +1, it is necessary to decrease the exponent by 1. This yields the correct final result as shown.

The multiplication operation can be shown in the form of a flow diagram shown in Figure 5-1. However, much of the operation is data-dependent; in other words, the amount of shifting that occurs in each state and the arithmetic operation (add or subtract) to be performed is dependent on the data pattern (string of 1s, string of 0s, isolated 1 or isolated 0) in the QR.

5.4 FLOATING POINT DIVISION

Floating-point division is accomplished in the FP11-C hardware by a normalizing non-restoring division algorithm and is described in the following paragraphs. The dividend is loaded in the AR and the divisor is loaded in the scratchpad and is present at the input to FMX. The ER is loaded with the exponent difference plus the number of bits in the quotient plus 1. The QR is initially cleared and forms the quotient.

For single-precision operation, bits 58 through 35 form the quotient. The 24-bit quotient is initially formed in the QR extension (bits 30 through 24) up to seven bits per cycle. The QR extension is below (to the right of) the least significant bit (bit 35) of the quotient. For double-precision operation, the quotient is formed in flip-flops external to the QR, which will also be referred to as the QR extension. In actuality, the quotient is formed below QR bit 31 for single-precision and below QR bit 00 for double-precision.

5.4.1 Adding or Subtracting Divisor to Dividend

The first step in the divide algorithm is to subtract the divisor at the input to FMX from the dividend in the AR. In each successive cycle, if the AR is positive, the divisor is subtracted from it to try to drive the AR negative; if the AR is negative, the divisor is added to it to try to drive it positive. Initially, however, since the dividend and divisor contain positive normalized numbers, the first arithmetic operation will be a subtraction. After each arithmetic operation, both the QR and AR are left-shifted until AR bits 59 and 58 are different.

5.4.2 Forming Quotient Bits

The FP11-C will form up to seven bits of the quotient (QR) and remainder (AR) in each cycle. Quotient bits are formed in the QR extension and are subsequently left-shifted into the QR.

The extension bits are based on the results of the arithmetic operation and the shift-within-range (SWR) flag from the previous cycle as shown below. In the first cycle, it is assumed by the hardware that the (SWR) flag is asserted.

Result of Arithmetic Operation	Quotient Formed
AR is positive; SWR (previous cycle) asserted.	QR = 1000000
AR is positive; SWR (previous cycle) negated.	QR = 0000000
AR is negative; SWR (previous cycle) asserted.	QR = 0111111
AR is negative; SWR (previous cycle) negated.	QR = 1111111

The FP11-C will calculate the number of shifts required to normalize the AR as well as the arithmetic operation to be performed in the next cycle.

5.4.3 Shifting of AR and QR

In the next cycle, the shift control logic will shift the AR and the QR the required number of shifts and will increment the ER by this number. This operation will cause some of the bits formed in the QR extension to be shifted into the QR. The FP11-C will then add or subtract the divisor from the shifted AR. Again, if the AR was positive from the last operation, the hardware will subtract the divisor from it to try to drive it negative; if the AR was negative from the last operation, the hardware will add the divisor to it to try to drive it positive. The hardware will then calculate the number of shifts required to normalize the AR and will determine the new bits to be formed in the QR extension as a result of the arithmetic operation performed in this cycle and the state of the SWR flag in the previous cycle. This operation is repeated until the division is terminated.

5.4.4 Termination of Divide

Termination occurs when the number of bits required to normalize the QR is less than the number of bits required to normalize the AR (QR NORM ≤ AR NORM). Whenever this occurs, the hardware sets the Divide Done flag. In the next cycle, the QR and AR will be shifted by the number of shifts necessary to normalize the QR. This normalized QR is the quotient. The hardware will automatically do the arithmetic operation specified in this cycle, but it is not necessary as the quotient has already been obtained.

In the cases previously mentioned where the SWR flag in the previous cycle of a divide sequence is negated, it indicates that the AR and the QR require more than seven shifts to be normalized. When this occurs, the shift control shifts the AR and QR by seven, increments the ER by seven, and inhibits the arithmetic operation in the present cycle. This results in shifting AR and QR seven places to the left. In addition, the QR extension is filled with copies of the sign bit (AR59).

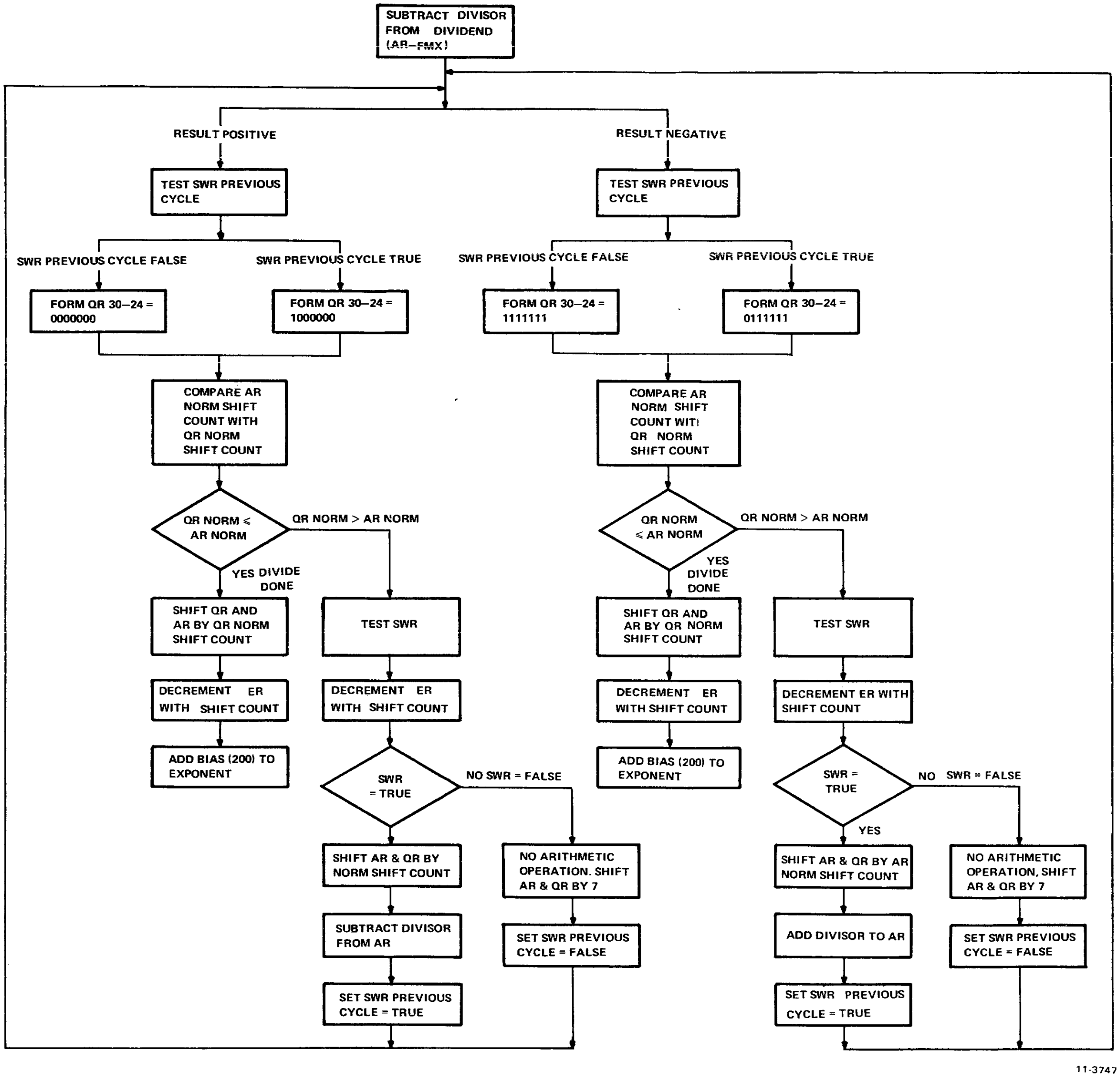
Because of the termination scheme employed in the FP11-C, the exponent of the quotient must be incremented in those cases where the quotient is greater than 1. To accomplish this, the ER is initially loaded with the exponent difference plus the number of bits to be formed in the quotient plus 1 and is decremented as the QR and AR are left-shifted.

For quotients less than 1, the QR is normalized when the number in the ER is decremented to the original exponent difference. For quotients greater than 1, the QR becomes normalized when the ER contains the original exponent difference +1. Since the normalized QR represents the quotient, the hardware will terminate one shift sooner than anticipated for quotients equal to or larger than 1 and the exponent (ER) will automatically be correct.

The reason for this is as follows: when the divisor is subtracted from the dividend in the first cycle and the result is positive (indicating quotient greater than or equal to 1), the QR extension will contain the complement of the sign bit followed by six copies of the sign. In this case, the QR extension is loaded with 1000000. When the result of the subtraction is negative (indicating quotient less than 1), the QR extension will be loaded with the complement of the sign bit plus six copies of the sign (0111111). Consequently, as the QR extension is subsequently shifted into the QR, it can be seen that a positive result (quotient greater than 1) will be normalized one shift sooner than a negative result.

5.4.5 Divide Flow Diagram Description

Figure 5-2 is a simplified flow diagram of the divide algorithm. The left-hand portion deals with a positive result in the AR and the right-hand portion deals with a negative result in the AR. The QR bits are formed based on the results of the arithmetic operation and based on the state of the SWR flag of the previous cycle. In the initial state, it is assumed that the SWR flag is asserted.


SUBTRACT DIVISOR FROM DIVIDEND (AR–FMX)
RESULT POSITIVE
TEST SWR PREVIOUS CYCLE
SWR PREVIOUS CYCLE FALSE
SWR PREVIOUS CYCLE TRUE
FORM QR 30–24 = 0000000
FORM QR 30–24 = 1000000
COMPARE AR NORM SHIFT COUNT WITH QR NORM SHIFT COUNT
QR NORM ≤ AR NORM
QR NORM > AR NORM
YES DIVIDE DONE
SHIFT QR AND AR BY QR NORM SHIFT COUNT
DECREMENT ER WITH SHIFT COUNT
ADD BIAS (200) TO EXPONENT
TEST SWR
DECREMENT ER WITH SHIFT COUNT
SWR = TRUE
NO SWR = FALSE
SHIFT AR & QR BY NORM SHIFT COUNT
SUBTRACT DIVISOR FROM AR
SET SWR PREVIOUS CYCLE = TRUE
NO ARITHMETIC OPERATION. SHIFT AR & QR BY 7
SET SWR PREVIOUS CYCLE = FALSE
RESULT NEGATIVE
TEST SWR PREVIOUS CYCLE
SWR PREVIOUS CYCLE FALSE
SWR PREVIOUS CYCLE TRUE
FORM QR 30–24 = 1111111
FORM QR 30–24 = 0111111
COMPARE AR NORM SHIFT COUNT WIT! QR NORM SHIFT COUNT
QR NORM ≤ AR NORM
QR NORM > AR NORM
YES DIVIDE DONE
SHIFT QR AND AR BY QR NORM SHIFT COUNT
DECREMENT ER WITH SHIFT COUNT
ADD BIAS (200) TO EXPONENT
TEST SWR
DECREMENT ER WITH SHIFT COUNT
SWR = TRUE
YES
NO SWR = FALSE
SHIFT AR & QR BY AR NORM SHIFT COUNT
ADD DIVISOR TO AR
SET SWR PREVIOUS CYCLE = TRUE
NO ARITHMETIC OPERATION, SHIFT AR & QR BY 7
SET SWR PREVIOUS CYCLE = FALSE
11-3747

INITIAL CONDITIONS
ER = EXP. DIFFERENCE PLUS
NO. OF BITS IN ANSWER,
AR LOADED WITH DIVIDEND,
FMX SELECTED FOR DIVISOR,
QR = 0

Figure 5-2 Simplified Division Flow Diagram

The QR and AR are examined for normalization. The FP11-C calculates the number of shifts required to normalize the AR and will determine the next arithmetic operation to be performed. Both the shifting and the arithmetic operation are performed in the next cycle, even though the number of shifts and the type of arithmetic operation is determined in the present cycle.

NOTE

If the QR can be normalized in fewer shifts than the AR, the Divide Done flag is raised, the QR and AR are left-shifted in the next cycle until the QR is normalized, the ER is decremented by the number of shifts, and the bias of 200_8 is added to the exponent to complete the operation since the correct quotient is in the QR when the QR becomes normalized.

The algorithm continues with the number of shifts and the type of arithmetic operation occuring in the cycle after they are determined. If the SWR flag from a previous cycle is negated since both the AR and the QR are out of range (more than seven shifts away from being normalized), the shift control is set for seven shifts. Both the AR and QR are left-shifted by seven and the arithmetic operation is inhibited.

As previously described, the ER is loaded with the exponent difference of the arguments plus the number of bits in the answer plus 1. The 1 adjusts for those numbers where the quotient is greater than 1. Normally, in these cases, the hardware would expect to do an extra left shift which would unnormalize the QR. However, this is inhibited by the termination scheme employed which stops shifting as soon as the QR is normalized.

5.4.6 Example 1 of Division Algorithm

$.1000011100 \times 2_{10} \div .1000001100 \times 2_{10}$

$= 540_{10} \div 524_{10} = 1.0305_{10}$

1.030
524) 54000
524
1600
1572
18

Initial Conditions:

ER = Exponent difference + number of bits in answer +1. Therefore, ER ← 11; FMX ← divisor; AR ← dividend; QR ← 0 and forms quotient; divisor and dividend are positive normalized numbers.

Exponent	**Fraction**
210	.1000011100
210	.1000001100

as we shift dividend places we get quotient places

Cycle 1: Subtract Divisor from Dividend

AR =	.1000011100
FMX =	.1000001100
AR =	.0000010000
QR =	.0000000000
QR EXT =	1000000

Positive result; SWR true; QR EXT is loaded with 1000000. AR NORM = 5, QR NORM = 00; AR NORM < QR NORM; shift control set to 5. Shift AR and QR left 5 in next cycle.

Cycle 2

Shift AR left by 5	0.1000000000
Shift QR left by 5	0.0000010000
ER = ER - 5	11 - 5 = 6
Subtract divisor from dividend	0.1000000000
	0.1000001100
AR =	1.1111110100
QR =	0000010000
QR EXT =	0111111

Negative result; SWR true; QR EXT is loded with 0111111. AR NORM = 6; QR NORM = 5; QR NORM $\leqslant$ AR NORM. Shift control set to 5. Shift AR and QR left 5 in next cycle. Set Divide Done flag.

Cycle 3

Shift AR left by 5	1.1010000000
Shift QR left by 5	0.1000001111
ER = ER - 5	6 - 5 = 1
QR normalized; set Divide Done flag	Divide Done
Add bias to exponent (200_8)	200 + 1 = 201
Fraction =	.1000001111
Answer =	$.1000001111 \times 2^1$
	= 1.000001111
	$= 1.0305_{10}$

Example 1 shows a 10-bit dividend divided by a 10-bit divisor. In the first cycle, the divisor (in FMX) is subtracted from the dividend (in the AR), resulting in a positive value. A subtraction rather than an addition is performed because the AR is initially assumed to be positive. The SWR flag is asserted, indicating that less than seven shifts are required to normalize the AR since this is the first cycle and the shift count is 0. Since five left shifts will normalize the AR, the shift control is set for a shift of five, which will shift both the AR and the QR left by five in the next cycle.

The AR and QR are left shifted by five, the ER is decremented by five, and the divisor is subtracted from the remainder (which is the AR left-shifted by five). The hardware performs a subtraction because the AR is positive and the FP11-C tries to make it go negative by subtracting the FMX from the AR. The result of the subtraction produces a negative AR, which means that the divisor will be added to the AR in the next cycle. The SWR flag from the previous cycle is asserted since the QR can be normalized in five shifts. The shift control is set for five and the QR and AR will be left-shifted by five in the next cycle. Since QR NORM is less than or equal to AR NORM (QR can be normalized in fewer shifts than the AR), the Divide Done flag is asserted, indicating completion of the operation. In the next cycle, the AR and QR are left-shifted by five, the divisor is added to the AR, the exponent is decremented by 5, and the 200_8 bias is added to the exponent in the ER. This exponent together with the normalized fraction in the QR represent the final quotient.

5.4.7 Example 2 of Division Algorithm

$.1000001100 \times 2_{10} \div .1000000000 \times 2^3 = 1.0000011 \times 2^7 = 10000011.$

$524_{10} \div 4_{10} = 131_{10}$

Initial Conditions:

ER = Exponent difference + number of bits in answer + 1. Therefore, ER = 18; FMX ← divisor; AR ← dividend; QR = 0 and forms quotient; divisor and dividend are positive normalized numbers; SWR N - 1 is shift within range, previous cycle.

Exponent	**Fraction**
210	.1000001100
203	.1000000000

Cycle 1: Subtract Divisor from Dividend

AR =	.1000001100	
FMX =	.1000000000	
AR =	.0000001100	(Positive)
QR =	.0000000000	
QR EXT =	1000000	

Positive result; normalize AR; SWR true; QR EXT is loaded with 1000000. AR NORM = 6; QR NORM = 10; AR NORM < QR NORM. Shift control set to 6. Shift QR and AR left 6 in next cycle.

Cycle 2

Shift AR left by 6	0.1100000000	QR EXT
Shift QR left by 6	0.0000100000	1000000
ER = ER - 6	18 - 6 = 12	
Subtract divisor from dividend	0.1100000000	
	0.1000000000	
AR =	0.0100000000	
QR =	0.0000100000	
QR EXT =	1000000	

Positive result; normalize AR; SWR true; QR EXT is loaded with 1000000. AR NORM = 1; QR NORM = 4; AR NORM < QR NORM. Shift control set to 1. Shift QR and AR left 1 in next cycle.

Cycle 3

Shift AR left by 1	0.1000000000	QR EXT
Shift QR left by 1	0.0001000001	0000000
ER = ER - 1	12 - 1 = 11	
Subtract divisor from dividend	0.1000000000	
	0.1000000000	
AR =	0.0000000000	
QR =	0.0001000001	
QR EXT =	1000000	

Positive result; SWR true; QR EXT is loaded with 00000000. AR NORM = infinity; QR NORM = 3; QR NORM ⩽ AR NORM. Set Divide Done. Shift QR and AR left by 3 in next cycle.

Cycle 4

Shift AR left by 3	0.0000000000	QR EXT
Shift QR left by 3	0.1000001100	xxxxxxx
ER = ER - 3	11 - 3 = 8	
QR normalized; set Divide Done flag	Divide Done	
Add bias to exponent	200 + 8 = 208	
Fraction =	$.10000011 \times 2^8$	
	= 10000011.	
	$= 131_{10}$	

Example 2 shows the decimal number 524 divided by decimal 4 to yield a quotient of 131. Initially, the dividend is loaded in the AR, the divisor at the input to FMX, and the QR which forms the quotient is cleared. The ER is loaded with the exponent difference plus the number of bits in the quotient plus 1. In this example, the number is 7 + 10 + 1 or 18. In the first cycle, the divisor is subtracted from the dividend since positive normalized numbers are assumed.

The result of the subtraction is positive and the SWR flag is asserted since the AR can be normalized in six shifts. The shift control is set up to shift by six.

In the next cycle, the AR and QR are left-shifted by six and the ER is decremented by six. The QR extension was loaded with 1000000 since the result of the subtraction was positive and SWR from the previous cycle is asserted. Since the result of the last arithmetic operation produced a positive number in the AR, the divisor will be subtracted from the dividend to try to make the AR go negative. There-sult of this subtraction produces a positive number in the AR which means that the divisor will be subtracted from the dividend in the next cycle in order to try to make the AR go negative. Also, the hardware calculates the number of shifts required to normalize the AR; in this cycle, one left shift will cause the AR to be normalized.

In the third cycle, the AR and QR are left-shifted by 1, the ER is decremented by 1 to 11, and the QR EXT is loaded with 1000000 since the AR is positive and SWR from the previous cycle is asserted. The divisor is now subtracted from the dividend, yielding all 0s, which is a positive number. Upon examining the QR and AR it is seen that the AR cannot be normalized since it contains all 0s. Since the QR requires three left shifts to be normalized, the hardware sets the Divide Done flag in this cycle. The FP11-C will then perform the next cycle by shifting the AR and QR left by 3, decrementing the ER from 11 to 8, and performing the arithmetic operation of subtracting the divisor from the dividend. Finally, the bias is added to the exponent and the resulting exponent (208) and fraction (.10000011)· represent the final quotient.

5.4.8 Example 3 of Division Algorithm

$.1000000000 \times 2_{10} \div .1000000000 \times 2_{10} = 1$

$2_{10} \div 2_{10} = 1$

Initial Conditions:

ER = Exponent difference + number of bits in answer + 1. Therefore, ER = (10 − 10) + 10 + 1 = 11; FMX ←; AR ← dividend; QR = 0 and forms quotient; divisor and dividend are positive normalized numbers.

Exponent	Fraction
210	.1000000000
210	.1000000000

Cycle 1: Subtract divisor from dividend

AR =	.1000000000
FMX =	.1000000000
AR =	.0000000000
QR =	.0000000000
QR EXT =	1000000

Positive result; QR EXT is loaded with 1000000. AR NORM ← ∞, QR NORM = 10; therefore, SWR is negated and FP11-C shifts QR and AR by 7.

Shift AR by 7	.0000000000	QR EXT
Shift QR by 7	.0001000000	0000000
ER = ER − 7	11 − 7 = 4	

AR NORM = infinity; QR NORM = 3; therefore, QR NORM ⩽ AR NORM. Shift control set to 3. Shift AR and QR left by 3 in next cycle. Set Divide Done flag.

Shift AR left by 3	1.0000000000	QR EXT
Shift QR left by 3	0.1000000000	xxxxx
ER = ER − 3	4 − 3 = 1	

QR normalized

Add bias to exponent (200_8)	200 + 1 = 201
Fraction =	0.1000000000
Answer =	$.10_{10} \times 2^1 = 1_{10}$

Example 3 shows the result of dividing 2 by 2, using 10-bit operands. The ER is initially loaded with the exponent difference plus the number of bits in the quotient plus 1, the dividend is loaded in the AR, the divisor is present at the input to FMX, and the QR, which forms the quotient, is initially cleared. Since the divisor and dividend are assumed to be positive normalized numbers, the divisor is first subtracted from the dividend. Examination of the AR and QR reveals that an infinite number of left shifts are required to normalize the AR or QR. In this case, the SWR flag is negated since more than seven shifts are required. When this flag is negated, the arithmetic operation, which would normally be performed in the next cycle, is inhibited.

In the second cycle, the AR and QR are left-shifted by seven and the QR EXT is loaded with 0s since the AR is positive and the SWR flag from the previous cycle is negated. The ER is decremented by 7. The QR and AR are now examined and the hardware determines that QR NORM is less than or equal to AR NORM and the QR will be normalized by a left shift of 3. The Divide Done flag is asserted which indicates completion of the operation in the next cycle.

Consequently, in the last cycle the FP11-C shifts the AR and QR left by 3, decrements the ER by 3, and adds the bias (200_8) to the exponent. The resulting normalized quotient (0.1000000000) with an exponent of 201 yields the number 1, which is the correct result.

CHAPTER 6
FP11-C LOGIC DIAGRAM DESCRIPTIONS

6.1 INTRODUCTION

This chapter describes the logic diagrams associated with the FP11-C Floating-Point Processor.

6.2 DETAILED LOGIC DIAGRAM DESCRIPTIONS

The FP11-C logic diagrams are divided into four groups of prints each group corresponding to one of the four FP11-C hex modules. The prints are designated by a 4-letter code where the first three letters of the code are defined as follows:

FXP	Floating-Point Exponent Data Path	M8129-0-01
FRM	FP ROM and ROM Control	M8128-0-01
FRH	Fraction Data Path, High Order	M8126-0-01
FRL	Fraction Data Path, Low Order	M8127-0-01

NOTE

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

The FXP group of prints contains the following logic:

EALU
AMX and BMX
Step Counter
Floating Instruction Registers A and B
Exponent Register
Microbreak Register
Data In Multiplexer
Control ROM
Control ROM Data Buffer
FPA, FEA, and FDR Registers
Data Out Multiplexer
IR Decode Logic
OBUF
Scratchpad Addressing

The FRM group of prints contains the following logic:

Control ROM
Control ROM Address Register
ALU Control
Register Clocking
FP Status Control
Main Timing Control
ROM Multiplexers
ROM Data Buffer
Interface Logic

The FRH group of prints contains the following logic:

Upper Half of FALU
Upper Half of Fraction Scratchpad
Upper Half of ACMX
Upper Half of FMX, QMX, QSHFR, ASHFR, QR, and AR
Multiply/Divide Control Logic
Shift Control Logic

The FRL group of prints contains the following logic:

Lower Half of FALU
Lower Half of Fraction Scratchpad
Lower Half of ACMX
Lower Half of FMX, QMX, QSHFR, ASHFR, QR, and AR
Floating-Point Status
Double-Precision Multiply Control
Quotient Formation Logic (for use in division)

6.3 FXPA LOGIC DIAGRAM

This diagram contains the FPA register, the FDR, and a ROM which, based on the instruction specified, generates a constant used to update the contents of the general register.

6.3.1 Floating-Point Address Register (FPA)

The FPA register is a 16-bit register consisting of three 74174 chips. The register is loaded by the CPU with an FP control field of 3 if FP READ is not asserted (indicating a write operation). Consequently, with a field of 3, the FPA is loaded via the address lines of the CPU at time state 3. This register contains the address of the instruction currently being fetched by the CPU.

6.3.2 Floating Data Register (FDR)

The FDR is a 16-bit register consisting of four 74175 chips. It is loaded from the BR in the CPU when the CPU issues FP ATTN or by a LOAD FDR signal (which is a decode of 1 in the FP control field) if FP READ is not asserted. The signal is enabled at time state 3, which causes the data from the BR to be loaded into the FDR. The data in this case represents the contents of the general register.

6.3.3 ROM with AD1, AD2 Constants

The FP11-C contains a 256-word by 4-bit ROM that is used to generate constants AD1 and AD2, which are employed to update the contents of the general register. The general register can be updated by 0, 2, 4, or 10, depending on the state of AD1 and AD2. For example, if the CPU was executing a mode 2 instruction, where a memory reference is performed and the address is incremented at the end of the instruction, the general register would be updated by 2. However, if the instruction were a floating-point instruction, 2- and 4-word operands are utilized so that the decoding of AD1 and AD2 is necessary in order to update the general register by the correct amount. As an example, if the FP11-C is performing a double-precision floating-point instruction with mode 2 (auto-increment), starting at location 1000, four memory references are required (1000, 1002, 1004, and 1006). Consequently, at the end of the instruction, the general register should be pointing to location 1010 and the ROM would have outputs AD1 and AD2 both low, as indicated by the table below.

AD2	AD1	ADDED TO INSTRUCTION
L	L	10
L	H	4
H	L	2
H	H	0

Mode 0 and Mode 1 Addressing – For mode 0 and mode 1, IR bits 05 and 04 are 0. In this case, the update constant of the general register is 0 since it is desired to inhibit updating the general register for these modes.

6.4 FXPB LOGIC DIAGRAM

This diagram contains the FIRA, two ROMs, a multiplexer for IR decoding, a multiplexer to decode immediate mode, and several miscellaneous gates to decode certain instructions.

6.4.1 Floating Instruction Register A (FIRA)

The FIRA is loaded from the BR in the CPU. The register is enabled by an FP control field of 2 and the negation of FP READ (indicating the register is to be loaded) at time state 3. The register is loaded with the floating-point instruction. Note that the register is only 12 bits long since the 4 upper bits of the floating-point instruction contain the op code and are not transferred to the FP11-C.

6.4.2 IR Decode

The IR decode logic consists of two 256 × 4 ROMs, multiplexer E86, and flip-flop E26. The multiplexer decodes the classes of FP instruction that are specified.

If FIRA 11 (0) H, FIRA 10 (0) H, and FIRA 9 (0) H are all asserted, the following instructions are decoded:

LOAD FP STATUS
STORE FP STATUS
STORE STATUS
CLEAR
TEST
ABSOLUTE
NEGATE

If any of the above three bits is not 0, the following instructions are decoded:

MULTIPLY
MODULO
ADD
LOAD
SUBTRACT
COMPARE
STORE
DIVIDE
STORE EXPONENT
STORE CONVERT INTEGER TO FLOATING
STORE CONVERT FLOATING TO INTEGER
LOAD EXPONENT
LOAD CONVERT INTEGER TO FLOATING
LOAD CONVERT FLOATING TO INTEGER

If IR bits 11–06 are decoded as 0, IR bits 03–00 are decoded to specify one of the following instructions. This selection is also accomplished by the multiplexer on logic diagram FXPC.

COPY FLOATING CONDITION CODES
SET FLOATING
SET INTEGER
LOAD MICROBREAK
MAINTENANCE SHIFT BY N
SET DOUBLE
SET LONG

The output from the multiplexer is applied to the ROM together with other inputs to ascertain if the instruction is mode 0 (M0 H), integer long (IL (1) H), or double precision (FD (1) H). Two ROMs are required since more than four outputs are required. The outputs are FIRD 0 through FIRD 5 and WL1 and WL0. FIRD 0 through FIRD 5 represent the instruction decode and WL1 and WL0 specify the word length to be transferred. The word length determines how long FP REQ remains asserted. These ROMs, which decode the instruction, are enabled when the FP11-C is in the Ready state.

Mode 0 Decoder – Two NAND gates are used for decoding mode 0 (non-memory reference) operation. If FIRA 03 (0) H, FIRA 04 (0) H, and FIRA 05 (0) H are asserted, mode 0 is specified. Also, if FIRA 11 (0) H through FIRA 06 (0) H are all zeros (which occurs during SETL, SETD, SETI, SETF, and Copy Condition Codes instructions), mode 0 is specified, since these instructions are also non-memory reference instructions.

6.4.3 Immediate Mode Decoder

Immediate mode specifies register 7, which uses the program counter (PC) for the address calculation, if mode 1, 2, or 4 is asserted.

The immediate mode decoder is multiplexer E84. The strobe inputs [FXPB FIRA 05 (1) H, FXPB FIRA 04 (1) H, and FXPB FIRA 03 (1) H] select mode 1, 2, or 4 if asserted and FXPB FIRA 00 (1) H, FXPB FIRA 01 (1) H, and FXPB FIRA 02 (1) H are implemented as an AND gate and specify register 7 if asserted. Consequently, if all inputs are asserted, mode 1, register 7; mode 2, register 7; or mode 4, register 7 is specified. These modes are immediate and cause IMMEDIATE H to be asserted. If mode 0, 3, 5, 6, or 7 is specified, immediate mode is negated.

6.4.4 Miscellaneous Instructions

Two NAND gates are shown on this diagram and merely decode the subtract instruction (FXPB SUB L) and the Store Convert Floating to Integer instruction (FXPB STCFI L).

6.5 FXPC LOGIC DIAGRAM

This logic diagram contains the FIRB, the IR decode ROMs, and the Microbreak register.

6.5.1 Floating Instruction Register B (FIRB)

The FIRB is a 10-bit register consisting of two 74175 chips. The register is loaded with the contents of the FIRA which is the current instruction being executed. Not all bits from the FIRA are loaded into the FIRB. (Bits 05 and 04 are not required for scratchpad addressing or for branching and are, therefore, not loaded into the FIRB.) The FIRB is clocked by IR CLK at time state 3, which occurs every time the FP11-C sequences through the Ready state.

6.5.2 IR Decode ROMs

The IR decode ROMs on this diagram are enabled after the FP11-C leaves the Ready state and determines the branching on the IR decode. For add, subtract, multiply, and divide instructions in mode 0 (register-to-register), this ROM calculates the branching conditions. For add, subtract, multiply, and divide instructions in memory-to-register operations, the ROM determines the branching conditions after the operand has been transferred to accumulator 6 in the FP11-C. The ROM also decodes the non-memory reference instructions such as SETI, SETL, SETF, SETD, and Copy Floating Condition Codes.

Note that the ROM outputs are designated with FXPB prefixes. These outputs are connected to the IR decode ROMs shown on FXPB. The ROMs employ tristate logic, which means that the ROMs on FXPB are chip-selected when the FP11-C is in the Ready state and the ROMs on FXPC are chip-selected when the FP11-C is not in the Ready state.

6.5.3 Illegal Accumulator

If mode 0 is specified and bits FXPB FIRA 01 (1) H and FXPB FIRA 02 (1) H are asserted (indicating source accumulator 6 or accumulator 7), FXPC ILLEGAL AC L is asserted indicating an illegal accumulator is specified.

6.5.4 Illegal Op Code

An illegal op code address is generated when the control ROM output creates an address of 52_8 (Refer to address 52 on the flow diagram.) This occurs when FXPB IR (11:06) 0 L is asserted and either FIRA 05 or FIRA 04 is set.

6.5.5 Floating Condition Code Load Enable

The FCLD EN L signal, when asserted, tells the CPU to copy the condition code. This signal is also used for the Store Convert Integer to Floating instruction and the Store Exponent instruction.

6.5.6 Microbreak Register

The Microbreak register consists of two 74175 registers connected to two 7485 comparators. The register is loaded with the 8-bit microbreak address via DIMX and the comparators are loaded with the 8-bit ROM address. When both addresses match, FXPC UMATCH H is asserted. This is a maintenance feature which can be used for sync pulse, scope loops, or microbreak traps.

6.6 FXPD LOGIC DIAGRAM

This diagram contains the DOMX and associated gating logic to select the various inputs.

6.6.1 Data Out Multiplexer (DOMX)

The DOMX is a dual 4-line to 1-line multiplexer which selects the FPA register, the FDR, the ACOMX, or the OBUF. The FPA register reads the address back to the CPU during interrupts, the FDR reads back the contents of the general register to the CPU during interrupts, the ACOMX stores the floating-point status information, and the OBUF stores the data to be transferred to the CPU.

6.6.2 DOMX Select Logic

The DOMX is selected by the control field (FPC1, FPC0) from the CPU and is enabled by the FP READ signal. The control field bits are applied to a combinational logic network which specifies outputs of FXPD S0 H, FXPD S1 H, and FXPD SEL FPS L. The various combinations of S0 and S1 specify one of four inputs to the multiplexer in accordance with the truth table shown on the logic diagram. For example, if FXPD S0 H is negated and FXPD S1 H is asserted, the FDR inputs to DOMX are enabled which allows the general register to be read back to the CPU.

6.6.3 Store FP Status

If the FP11-C executes a Store FP Status instruction, the CPU expects to transfer data into memory or into the CPU general register, and issues READ DATA, not knowing that the data is status information. In this case, the FRMB DOMX MOD L signal modifies the select lines to DOMX and forces DOMX to transfer floating-point status to the data lines rather than the output of OBUF.

6.7 FXPE LOGIC DIAGRAM

This diagram contains the OBUF, FEA registers, and the clocking logic associated with loading these registers.

6.7.1 OBUF Register

The OBUF register contains the data to be transferred back to memory. The register is loaded initially by the control ROM signal (FRMB OBUF CLK), which designates the loading of OBUF. This causes the first 16 bits of data to be loaded into OBUF. Subsequent data words for the specified operation are clocked into OBUF by FRMF FP ATTN (1) L. For example, in a double-precision instruction, the first 16 bits are loaded into OBUF by the FRMB OBUF CLK signal at FRMC TS3 H and FRHK CLK D H time. The remaining three 16-bit words are clocked into OBUF by the FP ATTN signal.

6.7.2 Floating Exception Address (FEA)

The FEA register is loaded from the Floating-Point Address (FPA) register every time the FP11-C sequences through the Ready state. The signal used to clock the FEA register is FXPC LD FIRB H, which occurs at time state 3 of the Ready state. If an error is flagged, the contents of the FEA will subsequently be transferred to accumulator 7.

6.8 LOGIC DIAGRAM FXPF

This diagram contains the DIMX and the clocking signals used to enable the DIMX.

6.8.1 Data In Multiplexer (DIMX)

The DIMX is normally enabled to accept data from the FDR. It can also accept inputs from the FEA register and from the EALU. In the event of a trap, the DIMX is enabled to accept the FEA. The DIMX accepts the EALU outputs when the FEC (floating exception code) is to be transferred to AC7 (1), or during the Store Exponent instruction when the 200_8 exponent bias is subtracted from the operand and routed into AC6 (2). If the subtraction causes EALU 08 to be a 1 (indicating a negative quantity), the 1 is sign-extended to all the upper bits of the instruction which produces a 16-bit integer. This is accomplished by AND gate E72, which asserts FXPF ESXT H. This signal is applied to bits 15 through 08 of the DIMX, in order to sign extend EALU bit 08 from bits 15 to 09.

6.8.2 DIMX Select Logic

The inputs to DIMX are selected by the FXPM DIMX C0 (1) H and FXPM DIMX C1 (1) H signals from bits 15 and 14 of the control ROM.

6.9 LOGIC DIAGRAM FXPH

This diagram contains the lower bits of the EALU, the lower bits of the BMX, and the carry look-ahead circuit.

6.9.1 BMX

The BMX consists of four 74S153 4-input dual multiplexers and a 74S11 AND gate (see FXPJ). It can be selected to output the EXPB scratchpad, the 8-bit constant field, the DIMX used for special instructions, or the shift control constant obtained from the shift control logic on the FRH module. The BMX select lines are controlled by the ROM via the ROM data buffer on logic diagram FXPM.

6.9.2 EALU

The AMX inputs are connected to the A side of the EALU and the BMX inputs are connected to the B side of the EALU. The select lines for the EALU are controlled by the control logic on logic diagram FRMF.

6.9.3 Carry Look-Ahead Circuitry

Each EALU chip is connected to a carry look-ahead chip (74S182) which is used to anticipate a carry rather than implementing the ripple carry which ripples from stage to stage.

6.10 LOGIC DIAGRAM FXPJ

This diagram contains the 10 bits of the AMX, the upper 2 bits of the EALU, a gate to sign extend the shift control, and the shift within range circuit.

6.10.1 A Multiplexer (AMX)

The AMX is a 10-bit, 4-input dual multiplexer consisting of five 74S153 chips. It can be selected to enable the ER, SC, ABS VAL, or EXPA scratchpad. The AMX is selected by FXPM AMXC1 (1) H and FXPM AMXC0 (1) H from the control ROM (on sheet FXPM) via the ROM output buffer.

6.10.2 EALU

The AMX inputs are connected to the A side of the EALU and the BMX inputs are connected to the B side of the EALU. FXPJ EALU 08 H is used to detect overflow or underflow and FXPJ EALU 09 H determines whether the condition is overflow or underflow. If FXPJ EALU 08 H is asserted (logic 1) and FXPJ EALU 09 H is negated (logic 0) FXPJ OVF H is asserted, designating an overflow condition.

6.10.3 Shift Control

When the BMX control field is set to 3, the shift control circuit is enabled. SHIFT CNT 03 H is the sign bit associated with the 4-bit shift control field. If this bit is asserted with the BMX selecting the shift control, the sign bit is extended from BMX 13 through BMX 09.

6.10.4 Shift Within Range

The four 8242 Exclusive-OR gates comprise a comparator circuit to determine if the operation can be completed in the next cycle. The circuit examines bits 06 through 03 and if they are all the same (all 1s or all 0s), FXPJ EALU SWR H is asserted, meaning that seven shifts or less are required to complete the operation in the next cycle. The number of shifts, consequently, is determined by the state of bits 02, 01, and 00.

6.11 LOGIC DIAGRAM FXPK

This diagram contains the ACMX and the EXPA and EXPB scratchpads for bits 07 through 00.

6.11.1 ACMX

The ACMX multiplexes the DIMX or the EALU which allows the FP11-C to write the output of the EALU into the scratchpads or write the data from memory into the scratchpads. The selection of the DIMX or EALU into the ACMX is controlled by a 1-bit field (bit 08) in the control ROM. If ACMXC 8 is a 0, the EALU input is selected and if ACMXC 8 is a 1, the DIMX input is enabled.

6.11.2 EXPA and EXPB Scratchpads

The EXPA and EXPB scratchpads are 74S189 random access memories (RAMs) which accept the DIMX or EALU inputs from the ACMX. The three select lines (FXPN ACS2 H through FXPN ACS0 H) select one of seven source accumulators to be loaded into the EXPA scratchpad. The three select lines (FXPN ACD2 H through FXPN ACD0 H) select one of seven destination accumulators to be loaded into scratchpad EXPB. If quadrant 3 of the scratchpad is to be written (which occurs when the exponents are to be written), FRMH WRITE ACD3 L is asserted, which causes both scratchpads to be written with the same exponents.

6.12 FXPL LOGIC DIAGRAM

This diagram contains the zero checkers, the decode of ACMX, branch condition negative (BCN), and the sign scratchpads.

6.12.1 Zero Checkers

The zero checkers consist of eight 74S05 inverter gates for exponent A and eight for exponent B. The inverters function as an 8-input AND gate. If FXPK EXPA 00 H through FXPK EXPA 07 H are all 0s, then FXPL EXPA EQ 0 H is asserted. A similar condition exists for FXPK EXPB 00 H through FXPK EXPB 07 H. If all these signals are 0s, then FXPL EXPB EQ 0 H is asserted. These signals are transferred to flip-flops on logic diagram FRMJ which indicate that source or destination is equal to zero.

6.12.2 Decode of ACMX

If FXPK ACMX 00 L through FXPK ACMX 07 L are all 0s, then FXPL ACMX EQ 0 L is asserted, which indicates an exponent field of 0. This signal is used by the BZ (branch zero) logic and by the FM0 (floating minus zero) trap.

6.12.3 Branch Condition Negative

The Branch Condition Negative (FXPL BCN L) signal is asserted if the DIMX is selected and bit 15 of the DIMX (sign bit) is a 1, or if the EALU is selected and bit 09 of the EALU (sign bit) is a 1. FXPL BCN L is applied to the 74S175 flip-flop on logic diagram FRMB. This flip-flop is clocked when the ER or SC is clocked and asserts FRMB BN (1) H if BCN is asserted. The assertion of BN (branch on negative) indicates that the arithmetic result in the EALU is negative or the data coming from DIMX is negative.

6.12.4 Sign Scratchpads

The two 74S189 random access memory chips serve as the sign scratchpad and are an extension of the exponent scratchpads shown on diagram FXPK. The sign A (sign of source) scratchpad is selected for AC0 through 7 by select lines FXPN ACS0 H through FXPN ACS2 H. The sign B (sign of destination) scratchpad is selected by select lines FXPN ACD0 H through FXPN ACD2 H. The outputs of the sign scratchpads are applied to the sign flip-flops on logic diagram FRMF. The SS and SD flip-flops can be loaded with various inputs such as the SS flip-flop being loaded with the sign of destination, etc.

6.13 LOGIC DIAGRAM FXPM

This diagram contains part of the control ROM and the ROM buffer register. This part of the ROM is located on the FXP module rather than the FRM module to minimize backplane connections.

6.13.1 Control ROM

The ROM, in total, is a 256 word by 76-bit wide read-only memory. Any of the 256 words can be addressed by FRMB RARB 00 H through FRMB RARB 07 H. The portion of the ROM on FXPM contains the AMX, BMX, DIMX, ER, SC, CONST, ACF, and ACMX fields.

6.13.2 ROM Buffer Register

Every time the FP11-C sequences through TS3, the ROM address is clocked into the ROM buffer register. The clock signal is FXPM CLK RB L, which is asserted every TS3.

The FXPM EN BRANCH COND H signal is asserted if the SC or ER is to be loaded and causes the branch condition codes to be loaded.

6.14 LOGIC DIAGRAM FXPN

This diagram contains the combinational logic used to address the source scratchpad and the destination scratchpad. This logic represents the addresses of accumulators 0 through 7. The logic on the left can be used to address source or destination accumulators even though the output designators are labeled ACD (destination accumulator). The logic on the right can be used to address source or destination accumulators even though the output designators are labeled ACS (source accumulator). Either logic network can generate addresses for AC6 or AC7.

6.14.1 Source Scratchpad

The combinational logic on the right of the diagram determines how the source address is generated. The inputs to the combinational network are:

1. Bits 11 through 09 of the ROM, which represent the ACF field.
2. FIR bits 00, 01, 02 to address a source scratchpad.
3. Mode 0.

The outputs are the three select lines (FXPN ACS0 through FXPN ACS2). If mode 0 is asserted (non-memory reference), FIR 0, 1, and 2 specify the source accumulator. If mode 0 is negated, the source accumulator is forced to be accumulator 6. The output of the combinational network yields various combinations of FXPN ACS0 H through FXPN ACS2 H, depending on the various inputs.

The source accumulator is specified by FIR bits 00, 01, and 02 as shown below.

FIR 2	FIR 1	FIR 0	Accumulator Specified
0	0	0	AC0
0	0	1	AC1
0	1	0	AC2
0	1	1	AC3
1	0	0	AC4
1	0	1	AC5
1	1	0	AC6 (illegal AC for ACS)
1	1	1	AC7 (illegal AC for ACS)

AC6 and AC7 cannot be designated as source accumulators.

6.14.2 Destination Scratchpad

The combinational logic on the left of FXPN is used to determine how a destination addrss is generated. The destination scratchpad is similar to that used for the source scratchpad. However, in this instance FIR bits 07 and 06 specify the destination accumulator as shown below:

FIR 7	FIR 6	Accumulator Specified
0	0	AC0
0	1	AC1
1	0	AC2
1	1	AC3

Because of the structure of the instruction, only two bits (FIR 6 and FIR 7) are available to specify a destination accumulator. (See F1 and F3 formats in Figure 3-4 of this manual.)

The outputs of the combinational logic for the destination scratchpad are select lines FXPN ACD 0 through FXPN ACD 2.

6.15 FXPP LOGIC DIAGRAM

This diagram contains the ER, the step counter, the negative absolute value ROM, and the out-of-range circuit.

6.15.1 Exponent Register (ER)

The ER is a 10-bit exponent register which accepts 10-bit inputs from the EALU. The ER holds the exponent of the result and is clocked by the FXPM CLK ER H signal (from the ROM buffer register) at TS3 if CLKD H is asserted. The FRHM CLKD H signal is a 60-ns signal obtained from the 30-ns CPU clock via a 2 to 1 frequency divider. (See logic diagram FRHM.)

6.15.2 Step Counter (SC)

The SC is a 10-bit step counter which accepts 10 bits from the EALU that represents the shift count. The register is clocked by the FXPM CLK SC (1) H signal from the ROM buffer register at TS3, and CLKD H is asserted. The FRHM CLKD H signal is a 60-ns signal obtained from the 30-ns CPU clock via a 2 to 1 frequency divider. (See logic diagram FRHM.)

6.15.3 Negative Absolute Value ROM

The negative absolute value ROM is a read-only memory which takes the negative absolute value of the step counter. This value is used in the add and subtract instructions to determine the number of shifts required during fraction alignment.

This causes the shift count to right shift the smaller operand; this is the proper shift direction for aligning exponents.

6.16 LOGIC DIAGRAM FRMA

This diagram contains the branching multiplexers, the gating to form the address that gets clocked into the ROM address register, and special branch and trap conditions.

6.16.1 Branching Multiplexers

The branching multiplexers are selected by the UAF field (bits 61 and 60 of control ROM), as shown below.

UAF		Multiplexer(s)
61	60	Enabled
0	0	0
0	1	3 and 2
1	0	1 and 0
1	1	3 through 0

The inputs to the multiplexers are selected by the UBR field (bits 67 through 65 of control ROM). If the UBR field is a 0, no branch conditions are enabled. For a UBR field of 1, the D1 inputs to the multiplexers are enabled; for a UBR field of 2, the D2 inputs to the multiplexers are enabled, etc. A complete description of the branching logic is given in Chapter 4.

6.16.2 ROM Address Gating

The gating on the bottom of the diagram provides the ROM address signals (FRMA RAD 00 H through FRMA RAD 07 H) which are applied to the D inputs of the ROM address register on FRMB. The inputs to these gates are the next address bits, the branching conditions, and the fact that no trap has occurred. This gating logic, in addition to the branching multiplexers, determines the 8-bit next address which can be 1 of a possible 256 addresses.

If a UBR field of 1 is specified, bits FXPB FIRD 4 H and FXPB FIRD 5 H are decoded for branch conditions. The only branch condition for these bits occurs with a UBR field of 1. If any of the trap conditions (INIT, Floating Minus Zero, Microbreak, or Microjump) are present, FRMA RAD 04 H and FRMA RAD 05 H are negated via pin 6 of the 74S20 gate. In addition, the existence of one of these traps causes bits FRMA RAD 07 H, FRMA RAD 06 H, FRMA RAD 03 H, and FRMA RAD 02 H to be negated.

6.16.3 Branch and Trap Conditions

The following trap conditions will cause bits 07 through 02 of the next address bits to go to 0. Bits 00 and 01 will go to the states shown below.

Bit 01	Bit 00	Trap Condition
0	0	INIT
0	1	Floating Minus 0
1	0	Microbreak Trap
1	1	Microjump Trap (to Ready state)

The two 74S64 NOR-AND gates decode these trap conditions except for the microjump trap. This trap is ROM state 3, which forces the FP11-C to the Ready state. The Ready state is decoded by NAND gate 74S10, pin 12, which is asserted when FRMA RAD 00 H and FRMA RAD 01 H are asserted and the microjump flip-flop (see logic diagram FRMD) is set.

Several additional gates are shown to AND several branch conditions and to form the right signal levels such as FRMA BOU + BZ L, which combines the Branch on Overflow or Underflow signal with the Branch on Zero signal.

6.17 LOGIC DIAGRAM FRMB

This diagram contains the ROM address register, part of the ROM and ROM control logic, and additional trap and branch logic.

6.17.1 ROM Address Register

The three 74S174 chips on the top of the diagram comprise the ROM address register. The register contains two sets of inputs and two sets of outputs because of loading requirements. One set of outputs has pin designations such as DU1, DS1, DF2, etc. This set is applied to indicators on the console and also to the ROM on the FXP module. The other set of outputs is applied to the ROM on the FRM module. The ROM address register is clocked at the leading edge of T2 if FRHM CLKC H is asserted. This clock has a 60-ns period which is equal to half the CPU frequency.

6.17.2 Floating Minus 0 Trap

The floating minus 0 trap is asserted [FRMB FM0 (1) H] under the following conditions:

1. FXPL ACMX EQ 0 L is asserted, indicating the operand has an exponent of 0.

2. The sign bit [FXPF DIMX 15 H] is negative.

3. The status bit [FRLP FIUV (1) H] that allows the FP11-C to trap on minus 0 is enabled.

4. The ROM bit [FRMB EN FM0 (1) H] that monitors the minus 0 trap is asserted.

5. No aborts [FRMC INIT (0) H] are occurring.

6. All memory cycles have been completed [designated by FRMF R2 (1) H] or the FP11-C is in immediate mode (FRMA IMMEDIATE L). R2 is part of a register on diagram FRMF which counts memory cycles. Single-precision requires two memory cycles and double-precision requires four memory cycles.

When all of the above conditions are met, the floating minus 0 trap is enabled and forces the ROM address to 1.

6.17.3 Microbreak Trap

A microbreak trap will occur during maintenance mode [FRLP FMM (1) H] if FXPC UMATCH is asserted and the interrupt logic is not disabled (FRMD DIS INTR H). The UMATCH signal indicates that the contents of the Microbreak register compares with the contents of the ROM address register. The microbreak trap flip-flop (UBT) is clocked by FRME CLK RB L from the control ROM. The DIS INTR H signal does not allow the programmer to microtrap out of states in the interface flow so the FP11-C will remain synchronized to the CPU. The microbreak trap forces the ROM address to state 2.

6.17.4 Branch Condition Logic

The branch condition logic is contained on the 74S175 quad flip-flop chip. If FXPL ACMX EQ 0 L is asserted (indicating an exponent of 0), the FRMB BZ (1) H signal is asserted, causing a branch on zero. If FXPL BCN L (branch condition negative) is asserted, the FRMB BN (1) H signal is asserted, causing a branch or negative. If FXPJ EALU 08 H is asserted (indicating overflow or underflow), FRMB BOU (1) H is asserted, causing a branch, overflow, or underflow. The 74S175 flip-flops are clocked on the trailing edge of FRMC TS3 A H when FRMH CLKC H and FXPM EN BRANCH COND H are asserted. The CLKC H signal is a 60-ns period which is one-half of the CPU frequency. The EN BRANCH COND signal is asserted when the ER or the SC is clocked.

6.17.5 ROM Buffer Register

Bits 75 through 68 of the control ROM are shown on this diagram and are addressed via the ROM address register on this sheet. The output of the ROM is applied to the ROM buffer register which is two 74S175 flip-flop register chips. The ROM buffer register is clocked by FRME CLK RB L at the trailing edge of T3. The outputs of the ROM buffer register are summarized below.

1. FRMB FP SYNC L – A signal indicating that the FP11-C is ready to accept or transmit data.

2. FRMB EN FM0 (1) H – A test to detect a 0 exponent and a negative sign.

3. FRMB FPC1 (1) H – A signal sent to the CPU which indicates a DATI or DATO cycle.

4. FRMB REG WR (1) H – A signal sent to the CPU to indicate that the FP11-C wishes to write data into a 16-bit CPU register during mode 0.

5. FRMB IRC (1) H – An IR clock signal which clocks the IR register.

The miscellaneous control field of the ROM (bits 27 through 25) is decoded off three bits of the 74S175 flip-flops by BCD decoder 8251-1. The miscellaneous signals are:

1. FRMB FP CLASS L – Used during read-modify-write for the Absolute and Negate instructions.
2. FRMB LD REQ L – Clocks the FP REQ counter which counts the number of requests.
3. FRMB CLR QR L – Clears the QR register.
4. FRMB OBUF CLK L – Loads the OBUF to prime the first word transfer to memory during the Store class of instructions.
5. FRMB DOMX MOD L – Used during store FP status and allows status rather than data to be supplied to the CPU.
6. FRMB UBRK CLK H – Clocks the Microbreak register.
7. FRMB LD FPS L – Loads the FP status word into the FP11-C status register.

6.18 LOGIC DIAGRAM FRMC

This diagram contains the time state generator, the interrupt clear logic, the INIT synchronizer, and restart conditons.

6.18.1 Time State Generator

The time state generator consists of three 74S112 J-K flip-flops and associated gating and buffer logic. The generator produces three 60-ns times states (Figure 6-1). The gating network lengthens the total time from 180 ns to 240 ns if long cycle is specified (Figure 6-2). The extra 60 ns occurs between FRMC T1 (1) H and T2 (1) H. In addition, the time state generator can be made to wait for external conditions such as FP START, FP ATTN, FP ACKN (during interrupts), or during certain maintenance functions (Figure 6-3). The 74S64 OR-AND gate ORs the following conditions and disables T2 of the time state generator if one of the conditions exist.

1. If FRMC T1 (1) H and FRME LONG (1) H are asserted, it indicates that a long ROM state of 240 ns is required.
2. If the FP11-C is waiting for FP START or FP ATTN, the 3-input gate of E54 will be enabled to restart the time state generator provided no aborts are occuring [FRMC INIT (0) H]. The FRMD EN ATTN (1) H indicates that the time state generator is enabled to wait for FP ATTN. As a result, the generator will not be restarted until FP ATTN or FP START is received, since the two signals are ORed through the restart synchronizer output from flip-flop E107, pin 6.
3. The FRMC MSTOP (0) H, FRMC T3 (0) H, and FRMC T2 (0) H signals are used to ensure that no double time states occur (T2 cannot be set if T2 or T3 is set) and also ensures that the maintenance stop flip-flop cannot be on. This flip-flop is used during the single ROM state stepping.
4. If the FP11-C is waiting for FP ACKN FRMD ACKN WAIT (1) H, the interrupt is not disabled [FRMJ FID (0) H], and no aborts are occurring [FRMC INIT (0) H], the timing sequence is paused prior to time state 2. When FP ACKN occurs, FRMC ACKN REST (1) L is asserted, which indicates the trap has occurred.

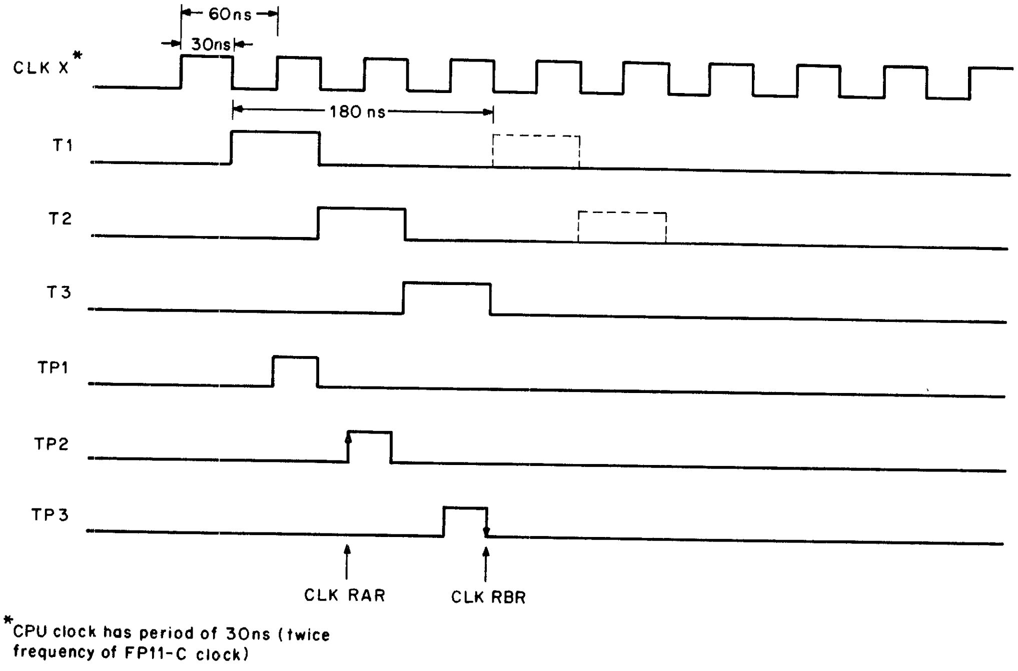

60ns
30ns
CLK X*
180 ns
T1
T2
T3
TP1
TP2
TP3
CLK RAR
CLK RBR
*CPU clock has period of 30ns (twice frequency of FP11-C clock)
11-3735

Figure 6-1 Time State Generator (Normal Cycle)

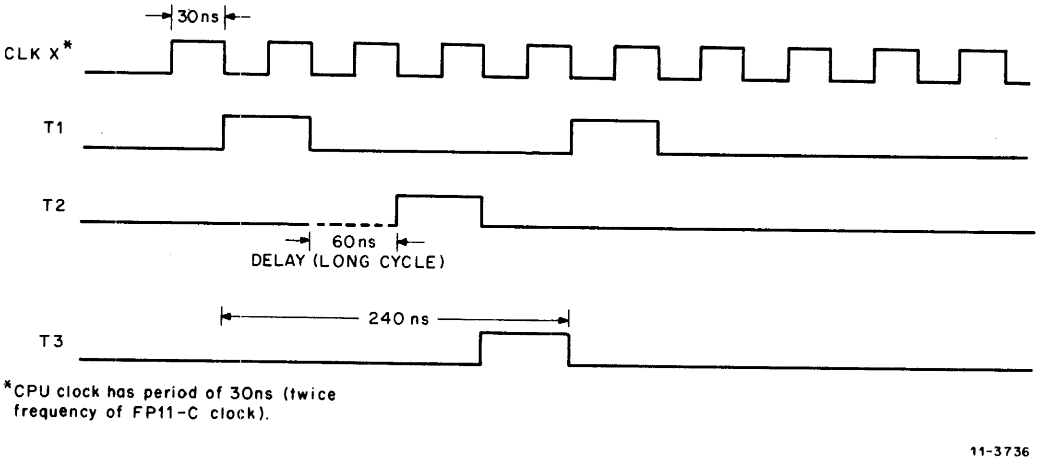

30ns
CLK X*
T1
T2
60ns
DELAY (LONG CYCLE)
240 ns
T3
*CPU clock has period of 30ns (twice frequency of FP11-C clock).
11-3736

Figure 6-2 Time State Generator (Long Cycle)

6.18.2 INIT Synchronizer

The INIT synchronizer consists of two 74S74 flip-flops with associated gating and is used for power-up sequences to initialize the FP11-C. When the CPU issues an INIT, it is applied to OR gate E98 (pin 8), forcing flip-flop E105 (pin 9) to 0. On the next clock pulse, INIT flip-flop E104 (pin 9) is set, asserting FRMC INIT (1) H. This signal is applied to the ROM address, forcing the ROM address to 0. It is also applied to OR-AND gate E54 (pin 8) which terminates the FP11-C having to wait for FP ACKN, FP ATTN, or FP START. Also, INIT forces the FP11-C to ROM state 0, which causes the FP11-C to go to the Ready state.

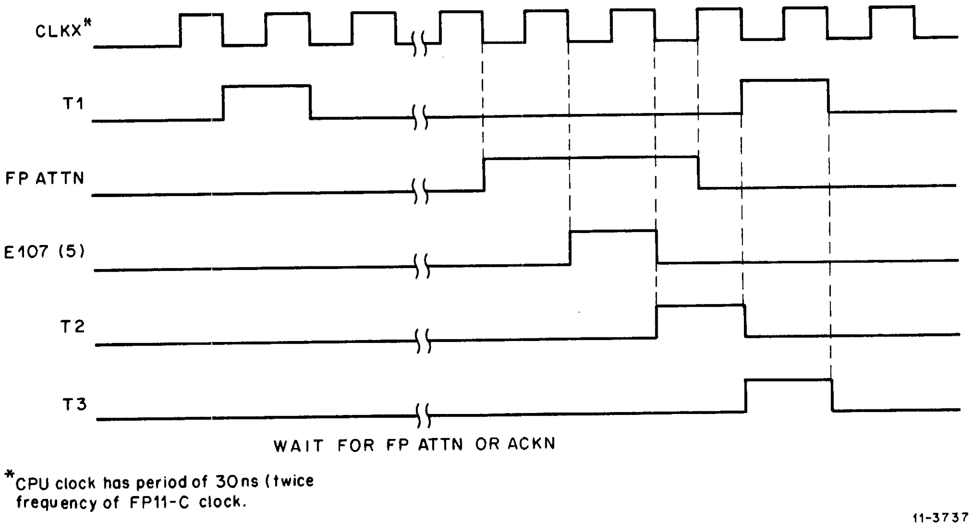

CLKX*
T1
FP ATTN
E107 (5)
T2
T3
WAIT FOR FP ATTN OR ACKN
*CPU clock has period of 30ns (twice frequency of FP11-C clock.
11-3737

Figure 6-3 Time State Generator (Indefinite Waits)

A second function performed by the INIT synchronizer is to get the FP11-C back to the Ready state after the CPU has issued an instruction and aborted the instruction to service an interrupt. In this instance, the INIT synchronizer prevents the FP11-C from hanging. This function is performed by flip-flop E105 (pin 2). The flip-flop is set if the interrupts are not being disabled (set input) and an TMCE INTR CLR is issued by the CPU. After several clock pulses, the INIT flip-flop is set, forcing the FP11-C to ROM state 0.

6.18.3 Restart Logic

The two 74S74 flip-flops (E107, pin 8 and E107, pin 5) are used for synchronizing RACB FP START (1) L and RACH FP ATTN (1) L to the FP11-C. When either signal is present, E107 is clocked set. This, in turn, sets the second E107 flip-flop. The FP START and FP ATTN signals are a function of the CP clock, while the second E107 flip-flop is clocked by the FP11-C clock.

The first E107 flip-flop is cleared by:

1. FRMC INIT (0) H, which clears any FP START signal that may have been present during power-up.

2. TMCE INTR CLR L, which clears any FP START signal that may have been asserted during an interrupt sequence.

3. FRMD EN ATTN (1) H and FRMC T2 (1) H, which clears the FP ATTN and sequences the FP11-C into T2.

The two 74S74 flip-flops (E63, pin 11 and E63, pin 2) are used for the floating-point exception trap. If the FP11-C wants to interrupt the CPU, it pauses the time state generator and issues FRMC FP EXC TRAP L, which is asserted while the FP11-C is waiting for FP ACKN if the interrupt is not disabled. The FP EXC TRAP is sent to the CPU which arbitrates it with other requests.

When the CPU accepts the trap, it issues FRHM FP ACKN L, indicating that the trap is accepted, and traps to interrupt vector 244. The FP ACKN signal is stored in flip-flop E63 (pin 12) and is synchronized with the FP11-C clock in flip-flop E63 (pin 6). This restarts the time state generator via gate E54 (pin 8).

6.18.4 Maintenance Stop Flip-Flop

If the FP11-C is performing single ROM states, the MSTOP flip-flop (E81, pin 5) is set at T1. A single ROM state is denoted by FRMC CPU S2 H and XMAA S1 L being asserted. This inhibits the time state generator from entering T2 until the MSTOP flip-flop is cleared.

This flip-flop is cleared by depressing stepper switch S4 on the maintenance card, which sets J-K flip-flop E83. This in turn resets the MSTOP flip-flop, which sequences the time state generator through T2, T3, and back to T1, where the sequence is again stopped by the MSTOP flip-flop.

6.19 LOGIC DIAGRAMS FRMD, FRME

These diagrams contain part of the control ROM and the ROM buffer register. FRMB RAR 00 H through FRMB RAR 07 H are the next address lines which select 1 of 256 addresses.

At T2, the ROM is read and at the trailing edge of T3, the output of the ROM (except the next address) is clocked into the ROM buffer register. This output represents the various control fields which specify the particular functions performed in the various microstates. The next address is clocked into the ROM address register on logic diagram FRMB.

The ROM buffer register is clocked by the FRME CLK RB L signal which occurs at TS3 during the assertion of FRHM CLKC H. The CLKC signal has a period of 60 ns which is half the frequency of the CPU clock (30-ns period).

6.20 LOGIC DIAGRAM FRMF

This diagram contains the FP REQ control, sign processor, EALU control, and miscellaneous branch conditions.

6.20.1 FP REQ Control

The FP REQ control contains a 74193 counter (E16) which counts the number of FP ATTN signals that is equivalent to the number of memory cycles to be performed. The counter is shut off when it reaches a count of four, which occurs when FRMF R2 (1) H is asserted. At this time, FRMF FP REQ L at the output of the counter becomes negated. The counter is loaded at TS3 by the FRMB LD REQ L signal. If the FP11-C is in the Ready state, the counter is loaded from WL1 and WL0 (which are the word length ROM designating the number of words to be transferred).

If the FP11-C is not in the Ready state, the counter is loaded from FRMJ FD (1) H. This occurs in the case of the ABS or NEG instruction where a zero exponent is encountered. When this happens, the fraction in memory must be zeroed and, if double-precision is specified [FRMJ FD (1) H], the counter is preloaded with a 0. For single-precision, the counter is preloaded with 2.

The other inputs to NAND gate E18 (pin 12) cause FRMF FPREQ L to be negated if the FP11-C is in immediate mode or if a floating minus 0 trap occurs.

6.20.2 Sign Processor

The sign processor consists of two flip-flops and two multiplexers. The flip-flops are loaded from their respective multiplexers which are controlled by the sign control field (ROM bits 28, 29, and 42). For the default case, the flip-flops are loaded with what was previously stored in them. If it is desired to zero the signs, a sign control field of 6 is selected. This selects the D6 input to the multiplexer which is grounded.

Note that the SS (sign of source) flip-flop follows the sign of the source while the SD (sign of destination) flip-flop can be selected for various inputs such as SS XOR SD, SS (0) H, SC09 (1) H, etc.

The sign flip-flops are clocked by FRME CLK RBL, which is the same signal used to clock the ROM buffer.

6.20.3 Branch Conditions

FRMF ADD * SC<8 L and FRMF SUB * SC<8 L are used to define branching conditions for add and subtract operations. They establish whether shift-within-range and the proper sign are asserted.

6.20.4 FN Circuitry

The FRMF SD MUX L signal is applied to the D input of the FN (floating negative) flip-flop and is asserted if bit 15 of the DIMX is being monitored. This occurs when FXPM ACMXC (1) H is asserted. If ACMXC (1) H is negated, the FP11-C monitors the ALUs. In this case, the SD being asserted causes FRMF SD MUX L to be asserted. A buffered version and the ACMXC (1) H signal is developed and is sent to the fraction processor.

6.20.5 EALU Control

The EALU control is a combinational network which accepts the EALU control bits and produces the EALU select lines which are applied to the EALU on logic diagrams FXPH and FXPJ to specify the particular EALU function (A + B, A + B + 1, etc.) required. The chart below shows the state of the select lines for the various combinational inputs. For example, to have the EALU perform an A + B operation, EALUC2, EALUC1, and EALUC0 are all 0s which forces S3 and S0 to be 1s, S2 and S1 to be 0s, M to be a 0 (logic mode), and C (carry in) to be a 1 (no carry).

Function	EALUC2	EALUC1	EALUC0	S3	S2	S1	S0	M	C*
A plus B	0	0	0	1	0	0	1	0	1
A plus B plus 1	0	0	1	1	0	0	1	0	0
A minus B minus 1	0	1	0	0	1	1	0	0	1
A minus B	0	1	1	0	1	1	0	0	0
A	1	0	0	1	1	1	1	1	1
B	1	0	1	1	0	1	0	1	0
Not used	1	1	0	0	1	1	0	1	1
Not used	1	1	1	0	1	1	0	1	0

*1 = no carry
0 = carry

6.21 LOGIC DIAGRAM FRMH

This diagram contains the scratchpad write pulse logic, the register clocking, and the FALU control.

6.21.1 Scratchpad Write Pulse Logic

The scratchpad write pulse logic consists of a combinational logic network and a multiplexer. The table on the diagram labeled Scratchpad Write Pulse Control designates the output pulses generated based on the various input combinations. The output pulses are the FRMH WRITE ACD 0 L through FRMH WRITE ACD 3 L signals, which enable the four scratchpad quadrants (ACD 0 through 3) to be written. The WRITE ACD signals are asserted at TS3 when FRHM CLKC H is asserted. The CLKC signal is derived from the CPU clock and has a period of 60 ns (one-half the frequency of the CPU clock).

As an example of how the combinational network asserts the various scratchpad write signals, assume an ACC field of 4 (ACC 2 = 1, ACC 1 = 0, ACC 0 = 0). According to the chart, the FRMH WRITE ACD 2 L signal should be asserted and all other signals negated.

AND gate E88 (pin 6) is inhibited due to FRME ACC 0 (1) H and FRME ACC 1 (1) H, so FRMH WRITE ACD 0 L is negated. AND gate E79 (pin 8) is negated due to the f output of multiplexer 74151 going low. The ACC field of 4 enables the D4 input of the multiplexer. Since this input is ground, the output is low and inhibits the assertion of FRMH WRITE ACD 1 L. AND gate E79 (pin 6) is asserted since FRME ACC 2 (1) H is asserted at TS3 when FRHM CLKC H occurs. AND gate E88 (pin 8) is inhibited through AND-OR gate E97 (pin 8) since FRME ACC 0 (0) H and FRME ACC1 (0) H drive E97 (pin 8) low to negate the FRMH WRITE ACD 3 L signal. Consequently, FRMH WRITE ACD 2 L is the only signal asserted, which coincides with the listing in the chart.

Note that an ACC field of 7 will write ACD 3 through 0 if FD (floating double) is set or will only write quadrants 3 and 2 if FD is reset.

6.21.2 Register Clocking

A group of register clocking signals is generated on this sheet. The signals are:

1. FRMH CLK QR H – Asserted at TS3 and CLKC H time when the ROM specifies a QRC field of 1.
2. FRMH CLK SHIFT CONT H – Asserted every TS3 and CLKC H time.
3. FRMH CLK AR H – Asserted every TS3 and CLKC H time when the ROM specifies an ARC field of 1.
4. FRMH CLR QR L – Asserted by the decoder of the miscellaneous ROM control field at the output of the ROM buffer and is merely buffered on this diagram.
5. FRMH CLR AR (59:35) L – Asserted at T2 when the ROM specifies 3 in the ARC control field.
6. FRMH CLR AR (34:00) L – Asserted at T2 when the ROM specifies an ARC field of 2.
7. FRMH FMXC0 (1) B H – Buffered version of FRME FMXC0 (0) H.
8. FRMH QMXC0 (1) B H – Buffered version of FRME QMXC0 (0) H.

6.21.3 FALU Control

The FALU control consists of a combinational logic network which controls the operations performed by the FALU. The FALU is controlled by a 4-bit ROM control field which provides conditional outputs (conditional add, for example) of the FALU as well as the normal outputs. The outputs of the FALU are FRMH FALU S0 H through FRMH FALU S3 H, FRMH FALU M1 H, and FRMH C1N H. Both A and B versions of FALU S0 through S3 are available due to loading. FALU M H is the mode signal and specifies logic mode when asserted, or arithmetic mode if not asserted. FALU CIN H is the carry in signal.

6.22 LOGIC DIAGRAM FRMJ

This diagram contains the clocking logic for the FP status (clocking FCC bits, FD flip-flop, IL flip-flop, and FPS register), part of the FPS register (FID and FER flip-flops), the branch logic for the 0 exponent test, and a multiplexer to establish the proper levels to the D inputs of the IL and FD flip-flops.

6.22.1 FCC Clock

The floating condition codes (FZ, FV, FC, and FN) are clocked into the FPS register upon assertion of the FRMT CLK FCC H signal. This signal is asserted on the trailing edge of time state 3 with a floating condition code of 0, 1, or 2. If the FCC field is 3 (NOP), both FRME FCC0 (1) H and FRME FCC1 (1) H are false forcing the output of OR gate E80 (pin 8) low and negating the FRMJ CLK FCC H signal. The associated control logic for the FCC bits is shown on logic diagram FRLN.

The FRMJ CLK FCC signal is also asserted at the trailing edge of time state 3 when FRMJ LOAD STATUS L is asserted during a LD FPS instruction.

6.22.2 FD Clock, IL Clock

FRMJ CLK FD H clocks the FD flip-flop with the level at its D input. This clock signal is generated at the trailing edge of time state 3 when a LD FPS instruction is specified or when a SETF or SETD instruction is specified. FRMJ CLK IL H clocks the IL flip-flop with the level at its D input. This clock signal is generated at the trailing edge of time state 3 when a LD FPS instruction is specified or when a SETI or SETL instruction is specified.

6.22.3 FPS Register Clock

When a LD FPS instruction is specified, the FPS register is clocked with memory data at the trailing of time state 3. During execution of the LD FPS instruction, it is desired to clock the entire FPS register including bit 06 (IL) and bit 07 (FD). During execution of the SETI or SETL instruction, it is desired to clock only bit 06 of the FPS register and during execution of the SETF or SETD instruction, it is desired to clock only bit 07 of the FPS register.

6.22.4 Multiplexer

If the miscellaneous control field (bits 27 through 25) of the control ROM is a 1 (FPS clock), a LD FPS instruction or a SET mode instruction is performed, depending on the state of IR bit 06. If IR bit 06 is a 1, a LD FPS instruction is executed; if bit 06 is a 0, a SET mode instruction is executed. The SET mode instruction may be a SETI, SETL, SETF, or SETD instruction. The instruction executed will be a function of IR bits 0 and 1. If IR bit 03 is a 1, it designates a SETL or SETD instruction depending on whether IR bit 00 or IR bit 01 is set. With IR bit 00 on a 1, a SETD instruction is performed and with IR bit 01 on a 1, a SETL instruction is performed.

If IR bit 03 is a 0, a SETI or SETF instruction is performed depending on the state of IR bits 00 and 01. If IR bit 00 is a 1, a SETF instruction will be performed and if IR bit 01 is a 1, a SETI instruction will be performed.

	IR Bit 0	IR Bit 1	IR Bit 3	IR Bit 6	MSC = 1 (MCODE LDFPS)
SETI	X	1	0	0	1
SETL	X	1	1	0	1
SETF	1	X	0	0	1
SETD	1	X	1	0	1
LDFDS	X	X	X	1	1

X = Don't Care

6.22.5 FID and FER Flip-Flops

The FID (floating interrupt disable) and FER (floating error) flip-flops are clocked when the LD FPS instruction is executed. Data bit 14 is loaded into the FID flip-flop and data bit 15 is loaded into the FER flip-flop. The FER bit is set as a result of any of the following trap conditions occurring:

Illegal op code
Microbreak trap
Overflow
Underflow
Divide by 0
Floating minus 0
Conversion error

The FER bit will be set regardless of the setting of the FID bit.

6.22.6 Zero Checkers

The EXPA and EXPB flip-flops sample the zero checkers on the exponent scratchpads for one microstate and are used during arithmetic operations to discard exponent arguments containing zeros.

6.22.7 FP PRESENT, FPADR INC Signals

When the FP11-C is plugged into the KB11-C backplane, the FRMJ FP PRESENT L signal is grounded. This indicates to the CPU that instructions with a 17xxxx format will be processed by the FP11-C. If the FP PRESENT signal is high, the CPU does reserved instruction traps on these instructions. The FRMJ FPADR INC signal indicates to the CPU to increment the address by two for floating-point cycles.

6.23 LOGIC DIAGRAM FRHA

For a better understanding of the FRH and FRL modules, the reader should make frequent reference to the data path diagram in this manual. Diagram FRHA contains parts of the ACMX, fraction scratchpads, QMX, and FMX.

6.23.1 ACMX

The ACMX on this diagram consists of three 74S158 multiplexer chips (E33, E37 and E35). E33 and E37 are associated with quadrant 3 of the specified accumulator and accept inputs from the DIMX if the B inputs to ACMX are selected or accept inputs from the FALU if the A inputs are selected. Note that only the low-order seven bits of quadrant 3 are present as the upper nine bits represent the exponent and sign which are routed to the exponent and sign scratchpads. E35 processes the upper four bits of quadrant 2. The remaining 12 bits of this quadrant are shown on logic diagram FRHB. The ACMX is selected for FALU or DIMX inputs via the ACMX control field in the control ROM. If this bit (bit 08) is a 0, FALU inputs are selected and if the bit is a 1, the DIMX inputs are selected.

6.23.2 Fraction Scratchpads

The output of the ACMX is applied to the fraction scratchpads which route the data to the FMX, QMX, or to the ACOMX for transfer to memory. Bits 58 thorugh 47 of the scratchpad are shown on this sheet. The scratchpads consist of 74S189 random access memory chips with each chip consisting of a 64-bit memory matrix organized as 16 memory locations with 4 inputs. The four select lines applied to each chip allow the four data inputs to be routed to 1 of 16 locations, if the write pulse (pin 3) is low. When the scratchpad is selected and the write pulse is high, the contents of the scratchpad are read. This diagram contains the scratchpad outputs for FRHA SCR OUT 58 H through FRHA SCR OUT 47 H.

6.23.3 QMX

Bits FRHA QMX 59 H through FRHA QMX 47 H are shown on this sheet. The QMX consists of 74S157 quad 2 input multiplexers and accepts inputs from the QSHFR or from the fraction scratchpads (SCR OUT signals). Note that only FRMH QMX C0 (1) B H is applied to the select input of the chip, which means that only a QMX field of 0 (SCROUT) or 1 (QSHFR) can be specified. The output of QMX is applied to QREG, which is wrapped around to the QSHFR inputs.

6.23.4 FMX

If the STB input to QMX is low, the output follows the selected input. If the STB input is high, the outputs are driven to 0.

The FMX accepts inputs from the fraction scratchpads or from the output of QSHFR. The FMX consists of 74S157 quad 2-input multiplexer chips. Note that FRMH FMXCO (1) B H is the signal that selects the QSHFR or SCROUT. This diagram contains bits FRHA FMX 59 H through FRHA FRMX 48 H. The FRHD DISABLE H1 FMX A H signal is only used when the FMX field of 3, which designates rounding, is specified. This signal clears the high-order bits of FMX (bits 59 through 32).

6.24 LOGIC DIAGRAM FRHB

This diagram contains additional ACMX chips, scratchpad chips, QMX chips, and FMX chips. The DIMX inputs to ACMX represent the remaining bits of quadrant 2. The description of these chips is similar to that described for logic diagram FRHA.

6.25 LOGIC DIAGRAM FRHC

This diagram contains the low-order bits (bits 34 through 31) of the ACMX, scratchpad, QMX, and FMX on the FRH module. The description of these circuits is described in the FRHA logic diagram description. The diagram also contains the round and integer increment gating and bits 59 through 44 of the FALU.

6.25.1 Round Logic

When rounding is specified (a code of three in the FMX field of the control ROM), the select inputs to multiplexer E112 are enabled to select the round function (pin 13). If single-precision mode is designated and truncate mode is inhibited, bit 34 of the FMX is forced to a 1 and the other bits of the FMX are 0s. The FMX is applied to the B input to FALU. This bit is added to bit 34 of the AR, which is applied to the A input to FALU. If AR 34 is a 1, a carry is propagated. If AR 34 is a 0, it is forced to a 1. When double-precision mode is specified, rounding will occur on bit 02 and this is shown on diagram FRLC.

When it is desired to convert a double-precision 56-bit fraction to a single-precision 24-bit fraction by issuing a STCDF (store convert double to floating) instruction, the FMX is forced to round mode. The STCDF instruction simulates single-precision mode to the hardware which wants to force bit 02 to a 1 when it is really desired to force bit 34 to a 1 (double-precision to single-precision).

NOTE

Bit 02 is guaranteed to be a 0 because as the 56-bit fraction, which is loaded, is loaded in bits 57 through 03.

Bit 34 is forced to a 1 by FXPB STCF L, which enables AND gate E107 (pin 8) and allows FRHC FMX 34 H to go to a 1. For the STCFD instruction, which converts a 24-bit fraction to a 56-bit fraction, the round hardware is not selected and no rounding occurs.

When the STCFI (store convert floating to integer) instruction is executed and short integer mode is specified, the 16-bit integer is transferred to quadrant 2 (bits 50 through 35) of the scratchpad accumulator. If the floating-point number is negative, FMX is selected for rounding and the integer has to be complemented and 1 added to bit 35. This is accomplished by the FRMJ INTEGER INC L signal, which enables OR-AND gate E108 (pin 6).

For long integer mode, a 1 is forced into bit 19 if the STCFL instruction is executed and a negative integer is encountered. This is accomplished by the FXPB INTEGER INC L signal, which forces FRLC FMX 19 H to a 1 during long integer mode.

6.25.2 FALU

Four FALU chips are shown on this diagram together with 74182 carry look-ahead generators. The carry look-ahead is accomplished in two levels. One level of carry look-ahead anticipates the carry across a group of four FALU chips or a 16-bit segment while the second level of carry look-ahead anticipates the carry across the first level of carry look-ahead. Therefore, there are 16 carry look-ahead chips for the first level of look-ahead and four carry look-ahead chips for the second level of look-ahead.

The FALU accepts inputs from FMX or the ASHFR and the outputs of the FALU are routed to the scratchpads via the ACMX. The FALU is selected by the FALU control field (FRMH FALU S3B through FRMH FALU S0B and FRMH FALU M H), which is generated by the FALU control logic based on inputs from the control ROM.

6.26 LOGIC DIAGRAM FRHD

This diagram contains additional bits of the FALU and combinational logic to disable the high- or low-order parts of the FMX and QMX.

6.26.1 FALU

Bits 43 through 32 of the FALU are shown on this sheet and are merely an extension of the FALU on diagram FRHC. The carry look-ahead circuits are also shown: one for each 4-bit FALU chip.

6.26.2 Disable FMX, QMX Logic

The FRHD DISABLE HI FMX A H and FRHD DISABLE LOW FMX H signals are asserted during rounding designated by an FMX control field of 3 [FRMH FMX C0 (1) B H and FRME FMX C1 (1) H both asserted]. The disable signals are applied to the STB inputs of FMX which cause the FMX output to go to 0, except for the round bit. During the conditional SCROUT, the low-order portions of the FMX (bits 34 through 00) and the QMX are disabled. Disabling the QMX allows a single-precision fraction to be read into the QR.

The FRHD DISABLE ROUND FMX H signal is asserted when a 2 is specified by the FMX field in the control ROM. This signal is used to inhibit the round bit (bit 34) from being a 1 if single-precision is specified.

6.27 LOGIC DIAGRAMS FRHE, FRHF

These diagrams contain the A register, the ASHFR, and the AR fill logic.

6.27.1 A Register

Bits 59 through 38 of the A register are shown on FRHE and bits 37 through 32 are shown on FRHF. The A register accepts the output of FALU and serves as a holding register before routing the output to the ASHFR. The A register is clocked by FRMH CLK AR H, which is a field of 1 in ROM AR control field (bits 41, 40).

6.27.2 ASHFR

The ASHFR network is separated into two levels of shift. The first level of shift is accomplished by E55, E63, E72, and E80 on FRHE and by E116, E105, E95, and E87 on FRHF. These chips are set up for a shift of 0, 4, 8, or 12. The second level (E54, E62, E71, E79, E86, E94, and E104) performs a shift of 0, 1, 2, or 3. For example, if a shift of five is desired, the first level of shift is set to shift by 4 and the second level of shift is set to shift by 1. The ASI signals out of the first level of shifting and feeding into the second level of shifting merely represent the wiring connections and have no correlation to the actual bits being shifted. The reader should disregard them to avoid confusion.

The ASHFR is wired up so that the inputs are offset right by 8 in order to provide the ASHFR with the the capability of left- or right-shifting. Without the offset, the ASHFR would merely be able to left-shift from 0 through 15 places. Consequently, if an operand is to be routed through the ASHFR with no shifting of the operand desired, as far as the output is concerned, the hardware will actually shift the operand left by 8 to compensate for the offset of the inputs. Table 6-1 shows the functions performed by the ASHFR control and how they are implemented. Consider FRHF A 40 H as an example and assume that the signal is to be fed through the ASHFR network with no shift relative to the output. Since the input is offset by 8, the ASHFR select code would be all 0s which indicates a right shift of 8 (Table 6-1) and the output would be the equivalent of the input. For a code of 00 in ASHF CONT 3A and ASHF CONT 2A, I0 is connected to O0, I1 to O1, I2 to O2, and I3 to O3. Consequently, AR 40 is routed from I0 to O0 and appears on the output of the first level shifter as FRHF ASI 32 H.

This signal is applied to second level shifter E104 on FRHF. Since a second level code of 00 is specified [ASHF CONT 1A (1) H and ASHF CONT 0A (1) H both low], the I0 input is connected to output O0, I1 to O1, I2 to O2, and I3 to O3. This means that FRHF ASI 32 H is routed to the output of the second level shifter as FRHF ASHFR 40 H.

As another example, assume A40 is to be shifted by 5. According to Table 6-1, this means that a right shift of 8 followed by a left shift of 3 is to occur. The code specified is ASHF CONT 3A (4) and ASHF CONT 2A (1) on a 0 and ASHF CONT 1A (1) and ASHF CONT 0A (1) H on a 1. Consequently, the I0 input (which A40 is connected to) is connected to output O0 and is designated FRHF ASI 32 H. This signal is applied to the second level shifter having a select code of 11, meaning that the I0 input is connected to the O3 output (designated FRHF ASHFR 35 H.) AR 40, if right-shifted by 5, appears on the output as ASHFR 35 H.

6.27.3 AR FILL Logic

The FRHE AR FILL H signal is used only with the multiplication algorithm. The number in the AR is right-shifted and then the normalized multiplicand of the output of FMX via the scratchpad is subtracted from the AR. The result of the subtraction will be a negative number. Therefore, in the next cycle when the right shift is performed, it is necessary to sign-extend the negative numbers by extending 1s in the most significant bit positions. The MPY SIGN EXTEND flip-flop stores the fact that the last operation was a subtraction and causes the assertion of FRHE AR FILL H when the FILL control field of the control ROM is a 3, which specifies RS·AR59·AR. The AR FILL signal, in turn, causes bits 67 through 60 of the ASHFR input to be 1s.

6.28 LOGIC DIAGRAM FRHH, FRHJ

These diagrams contain the QR, the QSHFR, and the Q FILL logic.

6.28.1 QR

The QR is composed of 74S174 chips which accept the QMX inputs and feed the output to the QSHFR. Bits FRHH QR 59 H through FRHH QR 42 H are shown on FRHH and FRHJ QR 41 H through FRHJ QR 32 H are shown on FRHJ.

Table 6-1 Shifter IC Truth Table

ASHFR Control						
First 3*	Level 2*	Second 1*	Level 0*	Function Performed by First Level Shift	Function Performed by Second Level Shift	Overall Function Performed
0	0	0	0	RS 8	LS 0	RS8
0	0	0	1	RS 8	LS 1	RS7
0	0	1	0	RS 8	LS 2	RS6
0	0	1	1	RS 8	LS 3	RS5
0	1	0	0	RS 4	LS 0	RS4
0	1	0	1	RS 4	LS 1	RS3
0	1	1	0	RS 4	LS 2	RS2
0	1	1	1	RS 4	LS 3	RS1
1	0	0	0	RS 0	LS 0	RS0
1	0	0	1	RS 0	LS 1	LS1
1	0	1	0	RS 0	LS 2	LS2
1	0	1	1	RS 0	LS 3	LS3
1	1	0	0	LS 4	LS 0	LS4
1	1	0	1	LS 4	LS 1	LS5
1	1	1	0	LS 4	LS 2	LS6
1	1	1	1	LS 4	LS 3	LS7

* These inputs are the ASHF CONT 3A (1) H through ASHF CONT 0A (1) H which select a group of four inputs to the shifter, as shown below.

25S10 Inputs (Select)		Outputs				
ASHF CONT 3A (1) H	ASHF CONT 2A (1) H	00	01	02	03	
0	0	I0	I1	I2	I3	
0	1	I-1	I0	I1	I2	Data
1	0	I-2	I-1	I0	I1	Inputs
1	1	I-3	I-2	I-1	I0	

6.28.2 QSHFR

The QSHFR operates in exactly the same fashion as the ASHFR previously explained on logic diagrams FRHE and FRHF.

6.28.3 QFILL Logic

The FRHH Q FILL H signal is asserted for the MOD and STCFI instructions. During the execution of these instructions, the QR is right-shifted and 1s are shifted into the upper end of the QR to form a mask as described in the MOD and STCFI instructions in Chapter 3. The FILL field of 1 from the control field must be asserted; it specifies RS·1·QR which allows the QR to be right-shifted and 1s to be shifted into the upper bits.

6.29 LOGIC DIAGRAM FRHK

This diagram contains the priority encoders, the multiply shift control, and divide termination logic. This logic is used in conjunction with the logic on diagram FRHL to provide the shift control for the AR and QR.

6.29.1 Priority Encoders

Priority encoder E45 is a 74148 chip which monitors the output of the FALU and outputs a signal determining how many shifts are required to normalize the FALU output. The priority scheme is implemented from the high-order bit to the low-order bit. For example, if FRHC FALU 55 H and FRHC FALU 54 H are asserted, bit 55 assumes priority. In this case, then, bits 58, 57, and 56 would be 0s and three right shifts would be necessary to normalize the number. With FRHC FALU 55 H asserted, the input to the priority encoder at pin 4 is low. The priority encoder output is such that FRHK NORM POS 2 H is high and FRHK NORM POS 1 H and FRHK NORM POS 0 H are low, yielding an output of 011, when one would think the output should be 100. This is due to an extra inversion through the encoder. Consequently, the encoder output of 011 represents three of the four-bits of shift control and designates a left shift of 3, which is the number of shifts required to transfer the 1 in bit 55 to bit 58 in order to normalize the number. If FRHC FALU 59 H is asserted, the EI input to the encoder is high, which forces the output to be all 1s at the NORM POS 2, NORM POS 1, and NORM POS 0 outputs, which is equivalent to a right shift of 1. This is the special case where a 1 occurs in bit 59 and right shift of 1 is required to normalize. If all the inputs to the priority encoder are 0, the FRHK NORM POS SWR H signal is asserted, indicating that more than seven shifts are required to normalize the number.

To summarize, the priority encoder processes shifting over 0s so a 1 can be shifted into bit 58. It also implements the special case of right-shifting the operand when bit 59 is a 1, and encodes the fact that the number of shifts are more or less then seven.

The second priority encoder shown is only used during division. Note that the inputs are non-inverted. This encoder causes shifting over 1s and is used when a negative number is incurred, which means that bit 59 is a 1 and bit 58 must be a 0. Its operation is similar to the other encoder. Note that the EI input is grounded, which indicates that it does not have a special case such as the previous encoder. If all the inputs to the encoder are 1s, the FRHK NORM NEG SWR H signal is asserted, indicating that more than seven shifts are required to normalize the number.

6.29.2 Multiply Encoder

The multiply encoder logic consists of a multiply shift encoder ROM and combinational logic to determine whether FRHK MUL SWR H is asserted. This signal is asserted during multiply if less than six shifts are required or if six shifts are required to normalize the fraction and it is known whether the number is a string of 1s or string of 0s. For single-precision, bits 40 and 39 are exclusively ORed. If these bits are both 1s or both 0s and the ROM shift encoder E96 decodes a shift count of 6 from the QSHFR inputs, AND gate E107 (pin 12) is qualified forcing the FRHK MUL SWR H signal low to indicate a shift count of six and to indicate that the hardware does not know if it is in a string of 1s or string of 0s. A similar situation occurs for double-precision operation except that bits 40 and 39 are exclusively ORed and are then ANDed with a shift count of 6. If bits 40 and 39 are the same and a shift count of six is encountered, the FRHK MUL SWR H signal is negated.

The FRHK and FRLH MUL SHF 2 H through FRHK and FRLK MUL SHF 0 H signals dictate how many shifts are to occur. These signals are applied to the shift control logic on FRHL.

The FRHK and FRL H STRING H signal, when asserted, causes the multiplicand to be subtracted from the partial product in the AR during the next cycle. If the STRING signal is negated, the multiplicand is added to the partial product in the AR during the next cycle.

6.29.3 Divide Termination Logic

The divide termination logic consists of norm shift ROM E39, comparator E12, and multiplexers E17 and E7 on diagram FRHL. The norm shift ROM examines the upper bits of the QR to determine the number of shifts required to align the QR. This value is compared with the output of multiplexer E17 on diagram FRHL. E17 contains the number of shifts necessary to align the AR. Note that multiplexer E17 selects the AR for either positive numbers (bit 59 = 0) or negative numbers (bit 59 = 1). If the AR is positive, priority encoder E45 on FRHK will contain the count required to normalize the AR and if the AR is negative, priority encoder E42 will contain the count required to normalize the AR. Consequently, the output of E17 (FRHL DIV NORM 2 H through FRHL DIV NORM 0 H) represents the number of shifts to normalize the AR regardless of its sign. This output is applied to comparator E12 on FRHK together with the ROM output containing the count necessary to normalize the QR. When the number of shifts required to align the QR is less than the number of shifts required to align the AR, the comparator asserts FRHK DIV DONE L, indicating that the QR will be normalized in the next cycle.

6.30 LOGIC DIAGRAM FRHL

This diagram contains the shift control logic for the QSHFR and ASHFR and contains part of the divide termination logic.

6.30.1 Divide Termination Logic

The divide termination logic is used in conjunction with the divide termination logic on sheet FRHK (Paragraph 6.29).

6.30.2 Shift Control Logic

The QSHFR shift logic consists of two 74S153 multiplexers (E16 and 14) and two 74S174 flip-flop chips (E4 and E5).

Note that only half of E16 is used for the QSHFR control; the other half is used to generate FRHL SHIFT WITHIN RANGE H, and a separate combinational network is used to generate FRHL Q CONTROL 3 H. This was necessary since a 4-input multiplexer was needed for FRHL SHIFT WITHIN RANGE H. Also, only half of E5 is used for QSHFR control and the other half is used for ASHFR control.

The multiplexed inputs to E14 and E16 are associated with EALU shift, divide shift, or multiply shift. The input is selected by FRME SHFC 1 (0) H and FRME SHFC 0 (0) H in accordance with chart shown on FRHL. The SHFC signals applied to the multiplexers are asserted on a (1) L. Thus, there is an inversion between the two. For example, if SHFC 1 (1) H and SHFC 0 (1) H are both false, EALU is designated in the chart. Consequently, the EALU inputs are multiplexed to the output (FRHL Q CONTROL 0 H through FRHL Q CONTROL 3 H). The Q CONTROL signals are applied to E4 and half of E5 to generate a count of the number of shifts necessary. Note that two versions (one designated with an A in the signal name and one with a B) are provided due to loading limitations. The flip-flops are clocked by FRMH CLK SHF CONT H from the control ROM.

The ASHFR shift control logic consists of two 74S153 multiplexers (E15 and E13), a 74S174 flip-flop chip (E3), and one-half of 74S175 flip-flop chip E5. The ASHFR shift logic is similar to the QSHFR logic except that a fourth input (FRHK NORM POS 2 H through FRHK NORM POS 0 H) is applied to the multiplexer. This input is necessary to normalize the AR. Note that the multiplexer select signals [FRME SHFC 1 (0) H and FRME SHFC 0 (0) H] are the same as used for the QSHFR logic.

6.30.3 FORCE ZERO AR SHIFT, FORCE ZERO QR SHIFT

During addition or subtraction, the operand with the smaller exponent is aligned with the operand with the larger exponent by right-shifting the fraction and incrementing the exponent until the two exponents are aligned. Since the operand with the smaller exponent may be in the QR or AR, a means is provided to inhibit the QR if this register contains the fraction associated with the larger exponent or to inhibit the AR if this register contains the fraction associated with the larger exponent. The signals to accomplish this are FRHL FORCE ZERO QR SHIFT H and FRHL FORCE ZERO AR SHIFT H, respectively. The FRME SHFC 2 (1) H, FRME SHFC 0 (0) H, and FXPP SC09 (1) H signals are applied to both circuits. The value of the SC09 signal determines which signal is asserted. If SC09 (1) H is asserted, it indicates that the AR has the fraction with the smaller exponent and consequently, FRHL FORCE ZERO QR SHIFT H is asserted. If SC09 (1) H is negated, the reverse is true.

6.30.4 Miscellaneous Latch Logic

The FRHL MPY/DIV ADD (1) L signal, when asserted, forces a subtract operation and, when negated, forces an add operation. This feature is used in multiplication and division operations. In a multiplication, if FRHK + FRLH STRING H is asserted, it indicates that a string of 1s is encountered and this string is terminated by an add. Consequently, pin 12 of the 74S175 flip-flop is high and, when the flip-flop is clocked, FRHL MPY/DIV ADD (1) H is asserted forcing an add operation to occur. If the STRING signal is negated, FRHL MPY/DIV ADD (1) l is asserted causing a subtraction operation to occur.

6.30.4.1 Shift Within Range - The FRHL SHIFT WITHIN RANGE H signal is latched in the 74S175 and is asserted as FRHL SWR (1) H. This is a branching condition and is routed to the branching multiplexers on diagram FRMA.

6.30.4.2 Divide Done - The FRHK DIV DONE L signal is latched in the 74S175 flip-flop and is asserted as FRHL DIV DONE (1) H. This signal is a branching condition and is also sent to the branching logic on diagram FRMA.

6.30.4.3 Sign Extend During Multiply - The FRHL MPY/DIV ADD (1) H signal is latched up in the 74S175 flip-flop and is asserted as FRHL MPY SIGN EXTEND (1) H. This signal stores the last operation (add or subtract). If the operation was a subtraction, the result is assumed to be negative since the multiplicand, which is subtracted from the AR, is normalized. In this case, 1s must be sign-extended into the most significant bits of the answer.

6.31 LOGIC DIAGRAMS FRLA, FRLB

These diagrams contain the low-order bits of the fraction scratchpad and the low-order bits of the FMX and QMX which were described in Paragraph 6.23. The FRLA DISABLE LOW QMX signal is used to disable the low-order bits of the QMX during single-precision operations when the control ROM specifies a field of 3.

6.32 LOGIC DIAGRAM FRLC

This diagram contains the remaining bits of the fraction scratchpad and QSHFR bits 23 through 16. The FRLC DISABLE LOW FMX H is asserted during rounding (FMX field of 3) if single-precision operation is specified. The FRLC DISABLE FMX H signal is asserted during COND AC SCROUT and disables the FMX during single-precision mode.

The circuitry that generates FRLC FMX 19 H and FRLC FMX 02 H are generated in a manner similar to that described for FMX 34 and FMX 35 on diagram FRHC. FMX 19 H is the integer increment bit which is asserted during a STCFI instruction when long integer mode is specified while FMX 35 H is the integer increment bit asserted during STCFI instruction when short integer mode is specified. FMX 02 H is the bit asserted during rounding when double-precision mode is specified.

6.33 LOGIC DIAGRAMS FLRD, FRLE

These diagrams contain the low-order portion of the AR and the ASHFR which is similar to the upper portion described on logic diagrams FRHE and FRHF.

6.34 LOGIC DIAGRAMS FRLF, FRLH

These diagrams contain the low-order portion of the QR and the QSHFR which are similar to the high-order portion of the QR and QSHFR described on diagram FRHH and FRHJ. One difference, however, is the quotient generator (E18 and E19) which generates eight bits of quotient during division. In this instance, the control ROM specifies a QMX of 3, which turns on the quotient generator connected to bits 24 through 03 and shuts off QSHFR outputs 31 through 24.

The FRLH HI QUOT GEN BIT H is asserted in a divide operation if the FRHL SWR (1) H signal (shift within range) is not asserted, indicating that more than seven shifts are required. In this case, the QR extension is a 1 from bits QR31 through QR24.

The FRLH XOR QSHFR 7:8 L signal is used with double-precision multiplication. You may recall that if bits 41 through 35 (single-precision) or bits 09 through 03 (double-precision) are all 0s and in a string of 0s, the arithmetic operation is inhibited and a shift of six is encountered. Also, if the bits are all 1s and in a string of 1s, the arithmetic operation is inhibited and a shift of six is encountered. For all other combinations, the hardware determines the number of shifts and what arithmetic operation (add or subtract) is to be performed.

6.35 LOGIC DIAGRAMS FRLJ, FRLK, FRLL

These diagrams contain the low-order portion of the FALU which is similar to the FALU on logic diagrams FRHC and FRHD. Bits 31 through 24 of the FALU are shown on diagram FRLJ, bits 23 through 12 on diagram FRLK, and bits 11 through 00 on diagram FRLL.

Logic diagram FRLJ also contains the two 74S153 multiplexers used during shifting operations. For example, to right shift once, the first stage of the QSHFR right-shifts 4 and the second stage left-shifts by 3. The purpose of the multiplexers is to feed these bits into the shift network. The multiplexers are also used during double-precision division to monitor the QR extension.

6.36 LOGIC DIAGRAM FRLM

This diagram contains the ACOMX which multiplexes the exponent B scratchpad inputs, the fraction scratchpad outputs, and the FPS MX inputs. Note that the ACO PREMUX inputs are scratchpad signals which have been already multiplexed. For example, FRHM ACO PREMUX 13 H represents FRHA SCROUT 48 H or FRHC SCROUT 32 H. The ACO PREMUX signals range from ACO PREMUX 15 through ACO PREMUX 12 and from ACO PREMUX 6 through ACO PREMUX 0. The ACO MX is enabled by a code of 3 in the ACO MX field of the control ROM.

6.37 LOGIC DIAGRAM FRLN

This diagram contains the FPS MX (floating-point status multiplexer) and the floating-point status. The FPS MX is utilized to minimize backplane wiring and multiplexes quadrant 1 of the scratchpad with the floating-point status bits.

The 74157 multiplexer multiplexes the DIMX (bits 03 through 00) with the status bits. The status bits are stored in the 74175 and 74174 flip-flop chips (E56 and E54, respectively). The function of the status bits is described in Chapter 3.

CHAPTER 7
MAINTENANCE

7.1 INTRODUCTION

This chapter describes some of the maintenance techniques and tools available for maintenance of the FP11-C. A description of the use of the maintenance card and the diagnostic program is also provided.

7.2 MAINTENANCE MODULE

The maintenance module consists of an indicator switch board (W131) and a driver board (W133) mounted piggy-back in slot B1 of the KB11-C mainframe. This maintenance module is used by the FP11-C and the CPU. The indicator board contains a series of indicators related to the CPU since the maintenance module is used by both the FP11-C and CPU. The driver board contains a series of drivers to drive the indicators on the indicator board.

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 (CPU microstate only)
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.

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 switch (S4) is depressed. The cycle will stop between TS1 and TS2. This feature can be used where maintenance personnel suspect that a specified instruction is not sequencing through the proper branches. Maintenance personnel can operate in a 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 and S1 are both on, a clock transition occurs every other time the MAINT STPR switch is depressed. This allows the FP11-C 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.

During normal operation the FP11-C clock is derived from the CPU clock. By margining the FP11-C speed, the CPU is also margined. This is also true for the single ROM cycle and single time pulse functions. For example, if the CPU is selected for single ROM cycle, the FP11-C is also selected for single ROM cycle since the same maintenance module is employed and the FP11-C clock is synchronized to the CPU clock. Consequently, when the stepper switch is advanced, both the FP11-C and CPU will sequence to the next ROM state.

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 R162 on the M8139 module. The limits are from 27 ns minimum to 450 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 due to IC degradation.

7.3 SPECIAL MAINTENANCE INSTRUCTIONS

A set of four 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 04) 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 Kernal 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 dis~~e~~nable bit (bit 14) of the floating-point processor status word is ~~set~~ CLEARD, 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 signal is present. This output is pin DB1 (slot 5 in the FXP module) and is used as a scope sync to allow visual observation of events that occur during a particular ROM state. Note that match occurs at T2 of the previous state and is negated at T2 of the selected state.

7.3.2 STA0 - Store AR in AC0 (170005)

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

AR (57:35) ← AC0 (57:35)if FD = 0

AR (57:3) ← AC0 (57:3) if FD = 1

7.3.3 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.

7.3.4 MSN – Maintenance Shift by N (170004)

This instruction transfers the contents of register R4 to the shift control logic and causes the contents of the AR and QR to be right- or left-shifted by N. A negative number in R4 causes a right shift by that number and a positive number in R4 causes a left shift by that number.

7.4 POWER SEQUENCE

The H744 power supply regulator for the FP11-C is located in the upper H7420 bulk supply. This regulator should be adjusted for +5 V.

The –15 V needed for the time state operator on the FRH module is supplied by regulator E which is included as part of the Central Processor Regulator Set. The –15 V needed in the PDP-11/70 comes from the lower H7420 power supply.

7.5 DIAGNOSTICS

In the event a system malfunction occurs, the CPU diagnostics will check out the CPU and the FP11-C diagnostics will check out the FP11-C. The FP11-C diagnostics are:

MAINDEC-11-DEFPA	PDP-11/45/55/70 FP11-C Diagnostic Part 1,
MAINDEC-11-DEFPB	PDP-11/45/55/70 FP11-C Diagnostic Part 2.

7.5.1 MAINDEC-11-DEFPA, B

These diagnostic programs are designed to detect logic malfunctions in the FP11-C Floating-Point Processor. DEFPA consists of a set of instructions designed to test instructions, logic operations, and data paths used by the more complex instructions such as multiplication and division.

DEFPB consists of a set of tests for:

1. Multiplication, modulo, and division instructions.
2. Memory management ROMs.
3. Locations of the A-branch, No-Mem branch, and ADX ROMs that have not previously been tested. This test is also used to verify the Disable Interrupt bit in the FP11-C Control Store ROM.

Fault detection is provided by an error service routine which prints the error condition on the console terminal. The error service routine also facilitates user control of the program sequence via console switch register options. After the error is reported, the program continues on its normal sequence unless modified by the user activating the halt on error switch option.

The error reports in these programs assure there are no previous errors and that there is only one failure in the processor. This means that if the programs are not run in sequence, the error message may be invalid.

7.6 USE OF MAINTENANCE MODULE FOR DEBUGGING

The maintenance module is useful for debugging when the hardware is malfunctioning to the point where the diagnostic will not print error messages. The most likely cause for this type of malfunction is interface problems between the FP11-C and the CPU, since the interface is synchronous and interdependent. Because of the interdependent nature of the interface, a malfunction could cause the system to hang up as a result of the FP11-C waiting for the CPU or the CPU waiting for the FP11-C. Once a malfunction has been recognized, maintenance personnel should find the last floating-point instruction that was properly executed. This is accomplished by examining the test number in the console display register and PC. This will point to a section of the diagnostic listing associated with the last properly executed instruction in a given subtest. When this instruction has been found, maintenance personnel should use the console to single step from the beginning of a subtest to determine the instruction which caused the malfunction.

The system can be single-stepped by instruction, memory cycle, ROM state, or time pulse. These are briefly described below. The single instruction represents the largest time interval while the single time pulse represents the smallest time interval.

Single Instruction – This operation is useful for bypassing individual instructions that are not related to the problem in question and allows maintenance personnel to quickly go to the instruction that is not properly executing.

Single Memory Cycle – This operation is useful in determining that the proper number of memory cycles has been accomplished. For example, in a double-precision load instruction, four memory cycles are expected to occur. It is also useful for being able to see data transferred from memory to the FP11-C or from the FP11-C to memory. This data can be seen via the BR in the console.

Single ROM State – This operation is most useful in that maintenance personnel can sequence through the instruction to determine whether the proper microcode sequence is being executed or to see if the branching is occuring properly. If the microcode branches to an incorrect microstate, maintenance personnel can, with the aid of the flow diagrams, determine the section (fraction processor, data interface, or ROM control) of the FP11-C that is malfunctioning.

Another useful feature of the single ROM state occurs when the control ROM is sequencing correctly and an incorrect result still occurs. In this case, the FP11-C can be stopped at any point with respect to the flow diagram which allows maintenance personnel to probe elements of the control or data path functions to determine partial results or to determine if the data path is set up correctly.

Single Time Pulse – This operation is useful to sequence into a specific time state to probe the data path. For example, the single ROM state causes the FP11-C to stop just prior to T2 of the indicated microstate. If it is desired to stop the FP11-C at T3 to dc check the data path, it would be necessary to use the single time pulse, which stops the FP11-C after each time pulse.

7.7 USE OF MICROBREAK REGISTER

In the event that an error message occurs while the diagnostic is running but the FP11-C does not hang up, maintenance personnel can loop on the diagnostic subtest that is failing. This is accomplished by examining the FP11-C flow diagrams to determine the microstate that it is desired to examine. The Microbreak register is then loaded with this microstate, and a probe is placed at FXPC MICROMATCH H (pin DB2) on the backplane to provide a suitable sync point.

Refer to diagnostic listing DEFPA, B for details of the error loop.

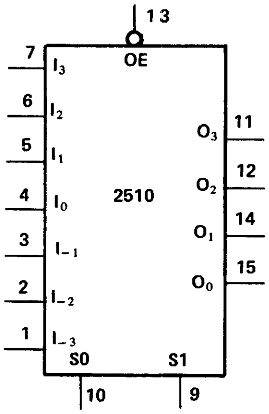
APPENDIX A
INTEGRATED CIRCUIT DATA

The following integrated circuits (ICs) are included in this appendix:

2510	4-Bit Shifter
7474	Dual Flip-Flop
7485	4-Bit Comparator
8251	4 to 10 Decoder
74112	Dual J-K Flip-Flop
74151	8 to 1 Multiplexer
74153	Dual 4 to 1 Multiplexer
74157	Quad 2 to 1 Multiplexer
74158	Quad 2 to 1 Multiplexer
74174	Hex D Flip-Flop Register
74175	Quad Storage Register
74181	4-Bit Arithmetic Logic Unit, Active High Data
74182	Look-Ahead Carry Generator
74189	64-Bit Random Access Memory

2510 FOUR BIT SHIFTER

The 2510 accepts a 4-bit data word and shifts the word 0, 1, 2, or 3 places. The number of places to be shifted is determined by S0 and S1. Active low enable input OE controls the three state outputs. Using appropriate interconnections, 2510s can be used to shift any number of bits any number of places up or down. Shifting can be logical, with logic 0s pulled in at either or both ends of the shifting field; arithmetic, where the sign bit is repeated during a shift down; or end-around, where the data word forms a continuous loop.



V_{CC} = Pin 16
GND = Pin 8

IC-2510

TRUTH TABLE

Inputs			Outputs			
OE	S1	SO	O_0	O_1	O_2	O_3
0	0	0	I_0	I_1	I_2	I_3
0	0	1	I_{-1}	I_0	I_1	I_2
0	1	0	I_{-2}	I_{-1}	I_0	I_1
0	1	1	I_{-3}	I_{-2}	I_{-1}	I_0
1	X	X	Z	Z	Z	Z

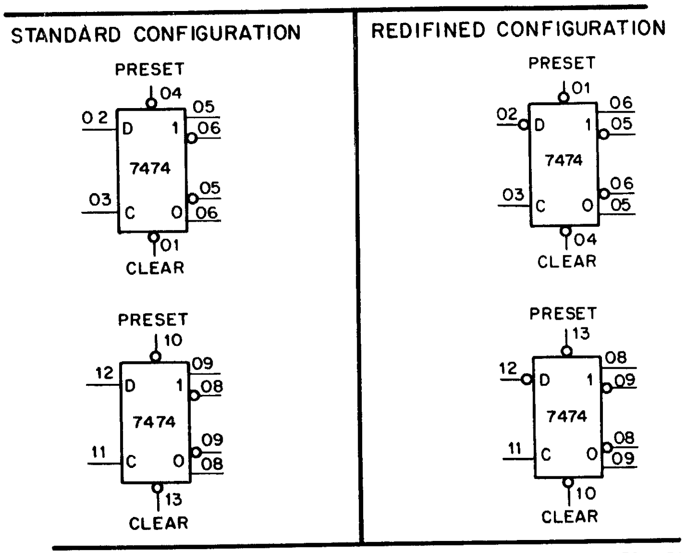
X = Irrelevant
Z = OFF = In high Impedance State

7474 DUAL FLIP-FLOP

TRUTH TABLE FOR
7474 STANDARD CONFIGURATION
(EACH FLIP-FLOP)

t_n			t_n+1	
Preset Pin 4(10)	Clear Pin 1(13)	D Input Pin 2(12)	1 Side Pin 5	0 Side Pin 6
High	High	Low	Low	High
High	High	High	High	Low
High	Low	X	Low	High
Low	High	X	High	Low
Low	Low	X	High	High

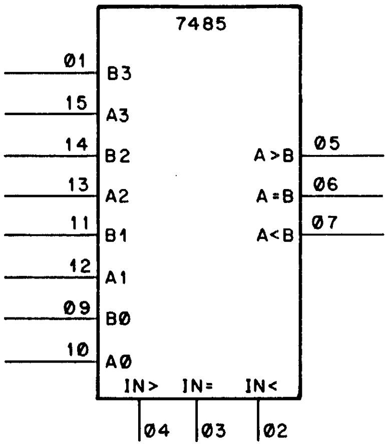
t_n = bit time before clock pulse.
t_n+1 = bit time after clock pulse.
X = irrelevant



V_{CC}= PIN 14
GND= PIN 07

IC-7474

7485 4-BIT COMPARATOR



VCC = PIN 16
OND= PIN Ø8

TRUTH TABLE

COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A3, B3	A2, B2	A1, B1	A0, B0	IN >	IN <	IN =	A > B	A < B	A = B
A3 > B3	X	X	X	X	X	X	H	L	L
A3 < B3	X	X	X	X	X	X	L	H	L
A3 = B3	A2 > B2	X	X	X	X	X	H	L	L
A3 = B3	A2 < B2	X	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 > B1	X	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 < B1	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 > B0	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 < B0	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	L	L	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	H	L	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	H	L	L	H

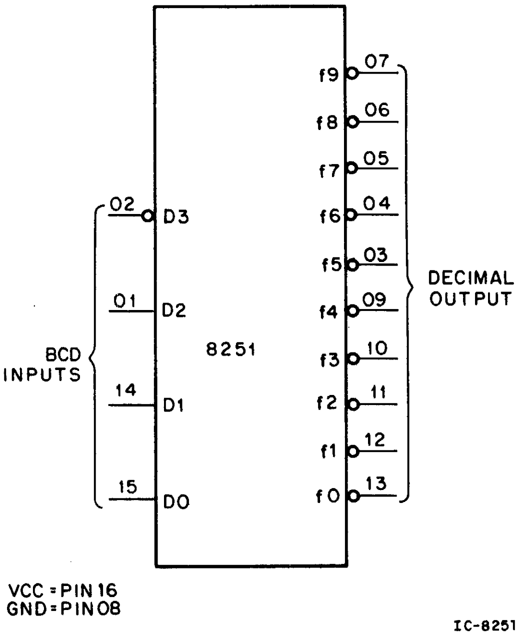
NOTE: H = high level, L = low level, X = irrelevant

8251 4 TO 10 DECODER

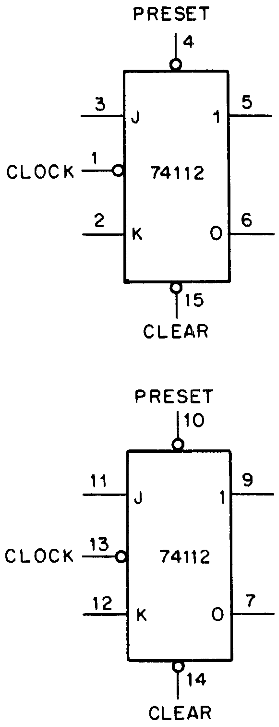
8751 TRUTH TABLE

INPUT				f OUTPUT									
D0	D1	D2	D3	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1	1	1	1	1	1	1
0	1	0	0	1	1	0	1	1	1	1	1	1	1
1	1	0	0	1	1	1	0	1	1	1	1	1	1
0	0	1	0	1	1	1	1	0	1	1	1	1	1
1	0	1	0	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
1	1	1	0	1	1	1	1	1	1	1	0	1	1
0	0	0	1	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
0	1	0	1	1	1	1	1	1	1	1	1	0	1
1	1	0	1	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	0

1 = High
0 = Low



74112 DUAL J-K FLIP-FLOP



74112 Truth Table

t_n		t_{n+1}
J	K	Pin 5 or 9
L	L	No change
L	H	L
H	L	H
H	H	Complement

t_n = Bit time before clock pulse.
t_{n+1} = Bit time after clock pulse.

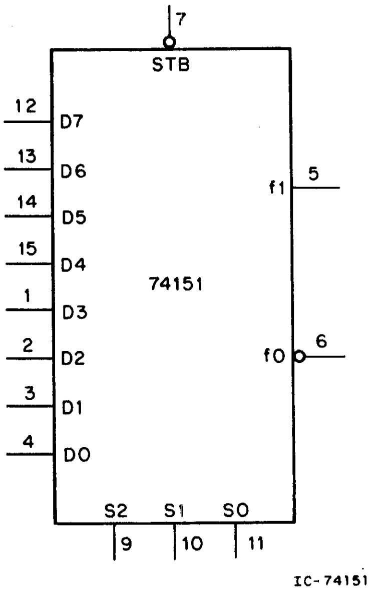
IC-74112

74151 8 TO 1 MULTIPLEXER

74151 TRUTH TABLE

Inputs												Outputs	
S2	S1	S0	STB	D0	D1	D2	D3	D4	D5	D6	D7	f1	f0
X	X	X	1	X	X	X	X	X	X	X	X	0	1
0	0	0	0	0	X	X	X	X	X	X	X	0	1
0	0	0	0	1	X	X	X	X	X	X	X	1	0
0	0	1	0	X	0	X	X	X	X	X	X	0	1
0	0	1	0	X	1	X	X	X	X	X	X	1	0
0	1	0	0	X	X	0	X	X	X	X	X	0	1
0	1	0	0	X	X	1	X	X	X	X	X	1	0
0	1	1	0	X	X	X	0	X	X	X	X	0	1
0	1	1	0	X	X	X	1	X	X	X	X	1	0
1	0	0	0	X	X	X	X	0	X	X	X	0	1
1	0	0	0	X	X	X	X	1	X	X	X	1	0
1	0	1	0	X	X	X	X	X	0	X	X	0	1
1	0	1	0	X	X	X	X	X	1	X	X	1	0
1	1	0	0	X	X	X	X	X	X	0	X	0	1
1	1	0	0	X	X	X	X	X	X	1	X	1	0
1	1	1	0	X	X	X	X	X	X	X	0	0	1
1	1	1	0	X	X	X	X	X	X	X	1	1	0

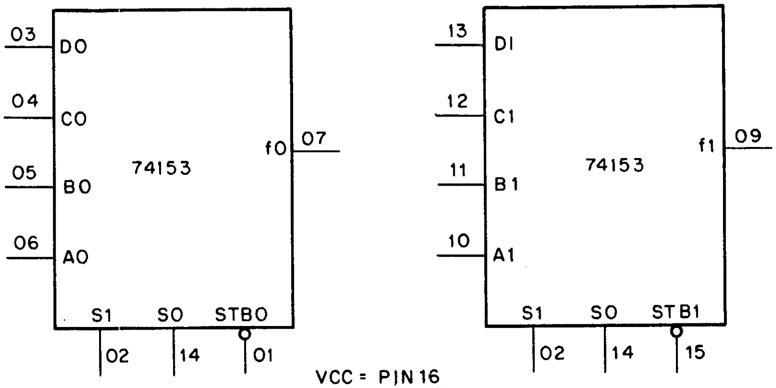
When used to indicate an input, X = irrelevant.



74153 DUAL 4 TO 1 MULTIPLEXER

ADDRESS INPUTS		DATA INPUTS				STROBE	OUTPUT
S1	S0	A	B	C	D	STB	f
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
L	H	X	L	X	X	L	L
L	H	X	H	X	X	L	H
H	L	X	X	L	X	L	L
H	L	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

Address inputs S0 and S1 are common to both sections.
H = high level, L = low level, X = irrelevant.



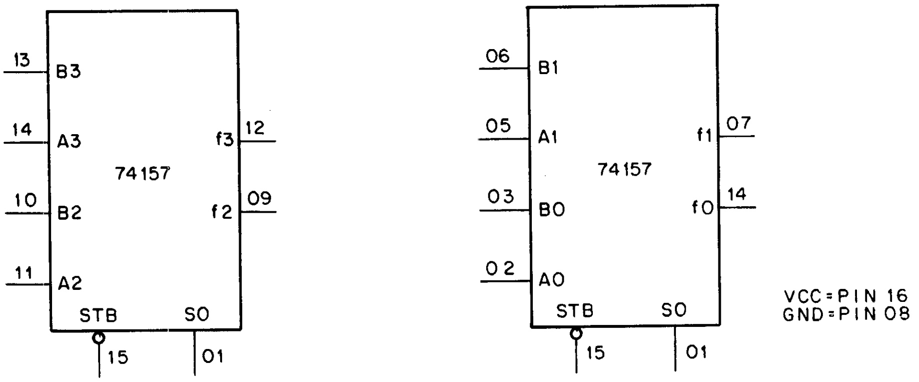
VCC = PIN16
GND = PIN08

IC-74153

74157 QUAD 2 TO 1 MULTIPLEXER

INPUTS				OUTPUT
STB	S0	A	B	f
H	X	X	X	L
L	L	L	X	L
L	L	H	X	H
L	H	X	L	L
L	H	X	H	H

H = high level, L = low level, X = irrelevant.

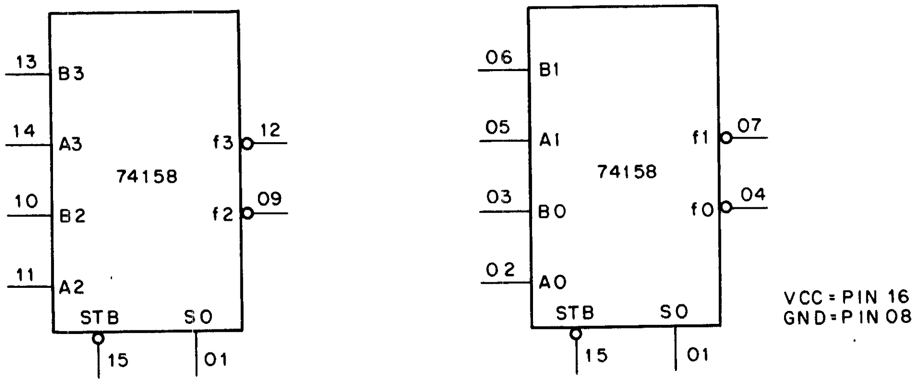


IC-74157

74158 QUAD 2 TO 1 MULTIPLEXER

INPUTS				OUTPUT
STB	S0	A	B	f
H	X	X	X	L
L	L	L	X	L
L	L	H	X	H
L	H	X	L	L
L	H	X	H	H

H = high level, L = low level, X = irrelevant.



IC-74158

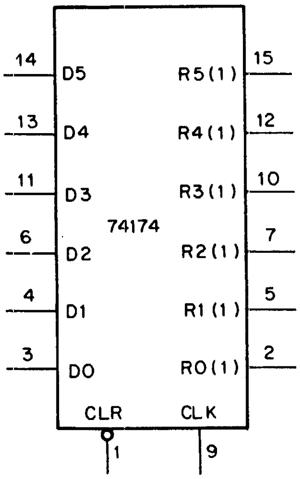
74174 HEX D FLIP-FLOP REGISTER

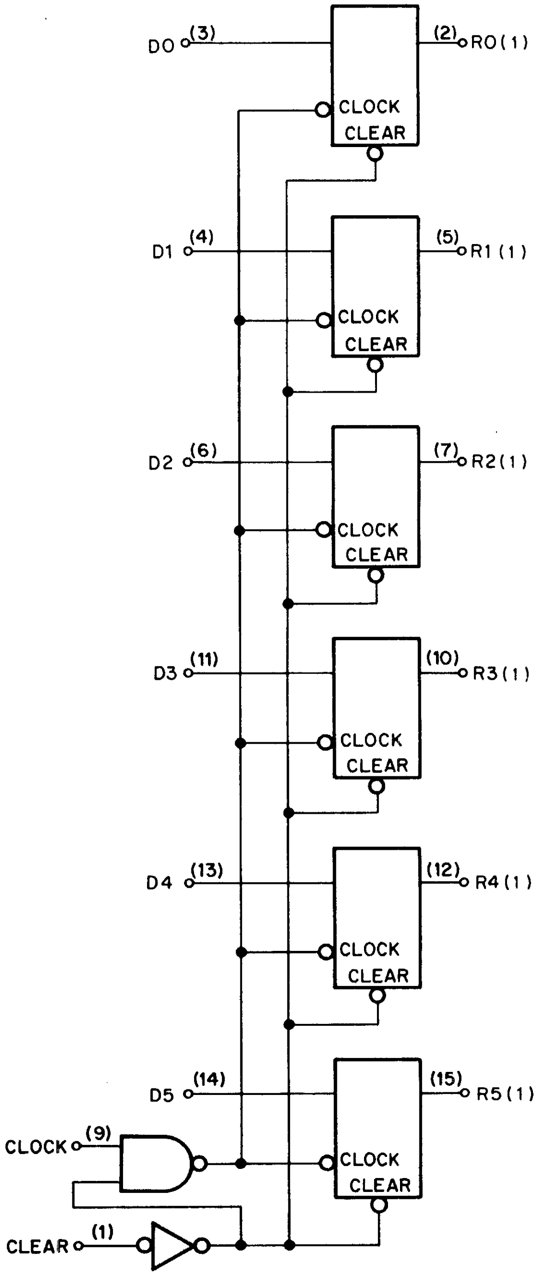
TRUTH TABLE

INPUT t_n	OUTPUT t_n+1
D	R(1)
H	H
L	L

t_n = Bit time before clock pulse.

t_n+1 = Bit time after clock pulse.





Pin (16) = V_{CC}, Pin (8) = GND

IC-74174

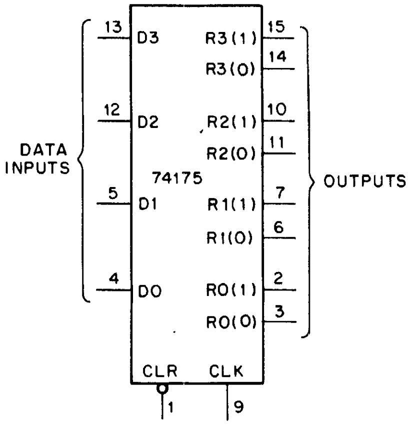
74175 QUAD STORAGE REGISTER

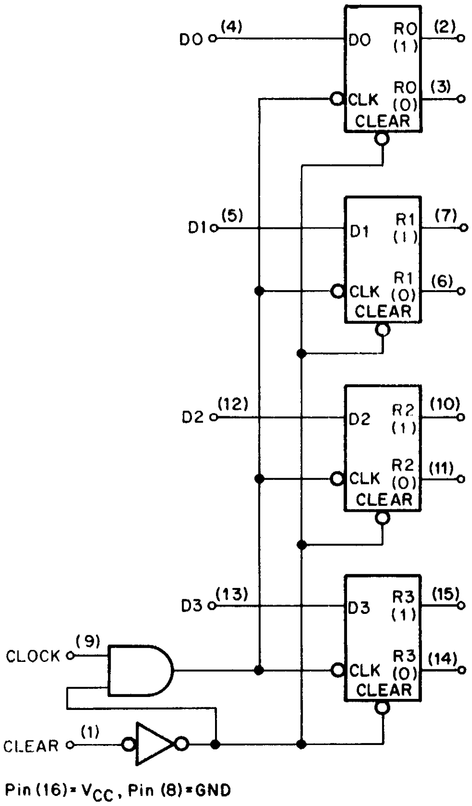
TRUTH TABLE

INPUT t_n	OUTPUTS t_{n+1}	
D	R(1)	R(0)
H	H	L
L	L	H

t_n = Bit time before clock pulse.

t_{n+1} = Bit time after clock pulse.





Pin (16) = V_{CC}, Pin (8) = GND

IC-74175

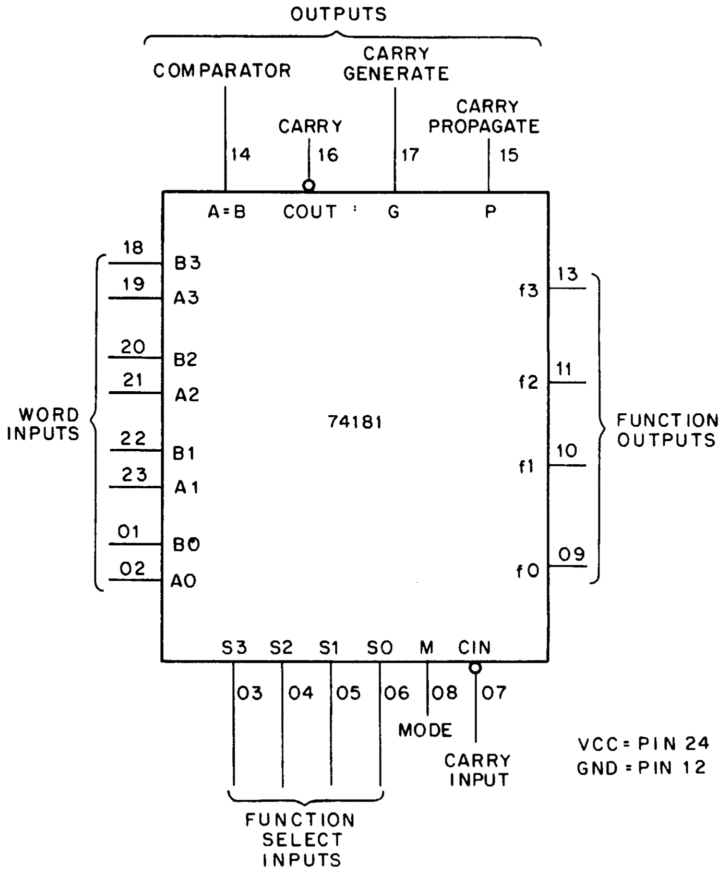
74181 4-BIT ARITHMETIC LOGIC UNIT, ACTIVE HIGH DATA

The 74181 performs up to 16 arithmetic and 16 logic functions. Arithmetic operations are selected by four function-select lines (S0, S1, S2, and S3) with a low-level voltage at the mode control input (M), and a low-level carry input. Logical operations are selected by the same four function-select lines except that the mode control input (M) must be high to disable the carry input.

74181
TABLE OF LOGIC FUNCTIONS

Function Select				Output Function	
S3	S2	S1	S0	Negative Logic	Positive Logic
L	L	L	L	$f = \overline{A}$	$f = \overline{A}$
L	L	L	H	$f = \overline{AB}$	$f = \overline{A + B}$
L	L	H	L	$f = \overline{A} + B$	$f = \overline{A}B$
L	L	H	H	f = Logical 1	f = Logical 0
L	H	L	L	$f = \overline{A + B}$	$f = \overline{AB}$
L	H	L	H	$f = \overline{B}$	$f = \overline{B}$
L	H	H	L	$f = \overline{A \oplus B}$	$f = A \oplus B$
L	H	H	H	$f = A + \overline{B}$	$f = A\overline{B}$
H	L	L	L	$f = \overline{A}B$	$f = \overline{A} + B$
H	L	L	H	$f = A \oplus B$	$f = \overline{A \oplus B}$
H	L	H	L	f = B	f = B
H	L	H	H	f = A + B	f = AB
H	H	L	L	f = Logical 0	f = Logical 1
H	H	L	H	$f = A\overline{B}$	$f = A + \overline{B}$
H	H	H	L	f = AB	f = A + B
H	H	H	H	f = A	f = A

With mode control (M) high: C_{in} irrelevant
For positive logic: logical 1 = high voltage
logical 0 = low voltage
For negative logic: logical 1 = low voltage
logical 0 = high voltage



IC-74181

74181
TABLE OF ARITHMETIC OPERATIONS

Function Select				Output Function	
S3	S2	S1	S0	Low Levels Active	High Levels Active
L	L	L	L	f = A minus 1	f = A
L	L	L	H	f = AB minus 1	f = A + B
L	L	H	L	$f = A\overline{B}$ minus 1	$f = A + \overline{B}$
L	L	H	H	f = minus 1 (2's complement)	f = minus 1 (2's complement)
L	H	L	L	$f = A$ plus $[A + \overline{B}]$	$f = A$ plus $A\overline{B}$
L	H	L	H	$f = AB$ plus $[A + \overline{B}]$	$f = [A + B]$ plus $A\overline{B}$
L	H	H	L	f = A minus B minus 1	f = A minus B minus 1
L	H	H	H	$f = A + \overline{B}$	$f = A\overline{B}$ minus 1
H	L	L	L	f = A plus [A + B]	f = A plus AB
H	L	L	H	f = A plus B	f = A plus B
H	L	H	L	$f = A\overline{B}$ plus $[A + B]$	$f = [A + \overline{B}]$ plus AB
H	L	H	H	f = A + B	f = AB minus 1
H	H	L	L	f = A plus A†	f = A plus A†
H	H	L	H	f = AB plus A	f = [A + B] plus A
H	H	H	L	$f = A\overline{B}$ plus A	$f = [A + \overline{B}]$ plus A
H	H	H	H	f = A	f = A minus 1

With mode control (M) and C_{in} low
† Each bit is shifted to the next more significant position.

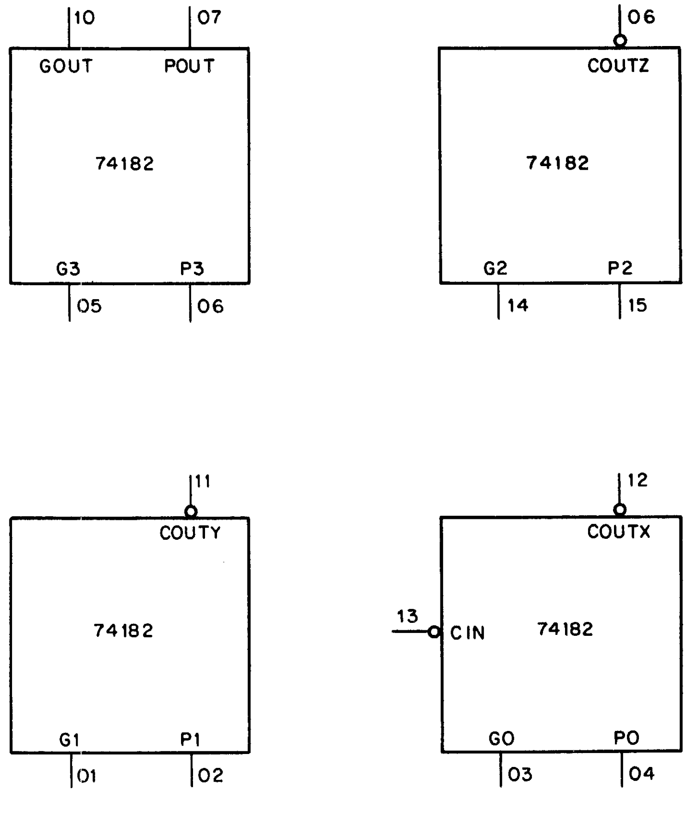
74182 LOOK-AHEAD CARRY GENERATOR

The 74182 Look-Ahead Carry Generator, when used with the 74181 ALU, provides carry look-ahead capability for up to n-bit words. Each 74182 generates the look-ahead (anticipated carry) across a group of four ALUs and, in addition, other carry look-ahead circuits may be employed to anticipate carry across sections of four look-ahead packages up to n-bits.

Carry inputs and outputs of the 74181 ALU are in their true form, and the carry propagate (POUT) and carry generate (GOUT) are in negated form.

PIN DESIGNATIONS

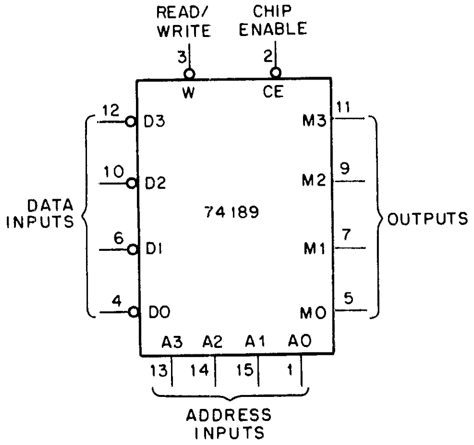
Designation	Pin No.	Function
G0, G1, G2, G3	3, 1, 14, 5	ACTIVE-LOW CARRY GENERATE INPUTS
P0, P1, P2, P3	4, 2, 15, 6	ACTIVE-LOW CARRY PROPAGATE INPUTS
CIN	13	CARRY INPUT
COUTX, COUTY, COUTZ	12, 11, 9	CARRY OUTPUTS
GOUT	10	ACTIVE-LOW CARRY GENERATE OUTPUT
POUT	7	ACTIVE-LOW CARRY PROPAGATE OUTPUT
V_{CC}	16	SUPPLY VOLTAGE
GND	8	GROUND



VCC = PIN 16
GND = PIN 08

IC-74182

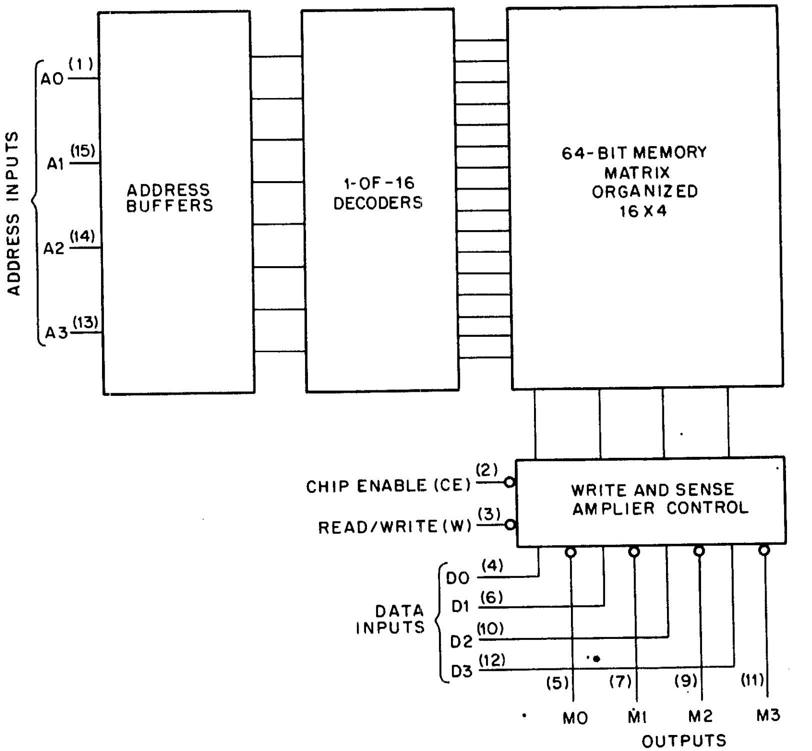
74189 64-BIT RANDOM ACCESS MEMORY



FUNCTION TABLE

FUNCTION	INPUTS		OUTPUT
	CHIP ENABLE	READ/ WRITE	
Write (Store Complement of Data)	L	L	High Impedance
Read	L	H	Stored Data
Inhibit	H	X	High Impedance

H = high level, L = low level, X = irrelevant.



IC-74189

Reader's Comments

FP11-C FLOATING-POINT PROCESSOR
MAINTENANCE MANUAL
EK-FP11C-MM-001

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

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

What features are most useful? ______________________________

What faults do you find with the manual? ______________________________

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

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

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

Please describe your position. ______________________________

Name ______________ Organization ______________

Street ______________ Department ______________

City ______________ State ______________ Zip or Country ______________

CUT OUT ON DOTTED LINE

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

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

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