EK-FPF11-TM-001

FPF11 Floating-Point Processor Technical Manual

digital

FPF11 Floating-Point Processor Technical Manual

Prepared by Educational Services
of
Digital Equipment Corporation

1st Edition, July 1981

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

9/83-15

CONTENTS

CONTENTS (Cont)

FIGURES

TABLES

PREFACE

The FPF11 floating-point processor is an option designed to operate with a PDP-11/23 central processing unit (or other compatible CPU) in executing floating-point arithmetic operations. This manual provides a detailed technical description of the FPF11 and service information.

- Chapter 1 describes the features of the FPF11 and its architecture.
- Chapter 2 outlines the fundamentals of floating-point arithmetic.
- Chapter 3 describes the three data formats the FPF11 recognizes.
- Chapter 4 describes the floating-point instructions the FPF11 uses.
- Chapter 5 gives a functional description of the FPF11.
- Chapter 6 contains the information necessary to inspect, install, and check out an FPF11 in a PDP-11/23 or other compatible system.
- Chapter 7 describes the diagnostics used to verify the FPF11 is operating properly.

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The FPF11 floating-point processor is a hardware option for use with the PDP-11/23 or other FPF11-compatible central processing unit (CPU). Its function is to execute the entire PDP-11 floating-point instruction set.

The FPF11 is contained in one quad-height module, M8188 (see Figure 1-1), which becomes an integral part of the CPU when installed. The module is installed in the backplane slot adjacent to the CPU (for systems using the PDP-11/23). It connects to the CPU by a ribbon cable that plugs into the socket designated for the optional floating-point processor chip. The FPF11 does not connect to the system bus, and therefore, has no affect on bus loading. Figure 1-2 shows the FPF11 signal interface with the CPU. These signals are discussed in Chapter 5.

The FPF11's dedicated high-speed data path increases the execution speed of floating-point instructions. The 64-bit-wide data path avoids the use of complex arithmetic coding routines that would be required if a 16-bit CPU were to operate on the 32-bit or 64-bit operands. The FPF11 uses its own internal clock to speed up the execution of floating-point instructions. This clock is controlled by the FPF11 microcode and generates variable-length microcycles so that each microword is executed in a minimum amount of time. In addition, the FPF11's operation does not depend on the memory management unit (MMU) for its scratch-pad registers, as the KEF11-A floating-point option does.

The FPF11 features both single- and double-precision (32- or 64-bit) capability. It uses the same addressing modes and memory management (when present) as the CPU. Floating-point processor instructions can reference the floating-point accumulators, the CPU's general registers, or any location in memory.

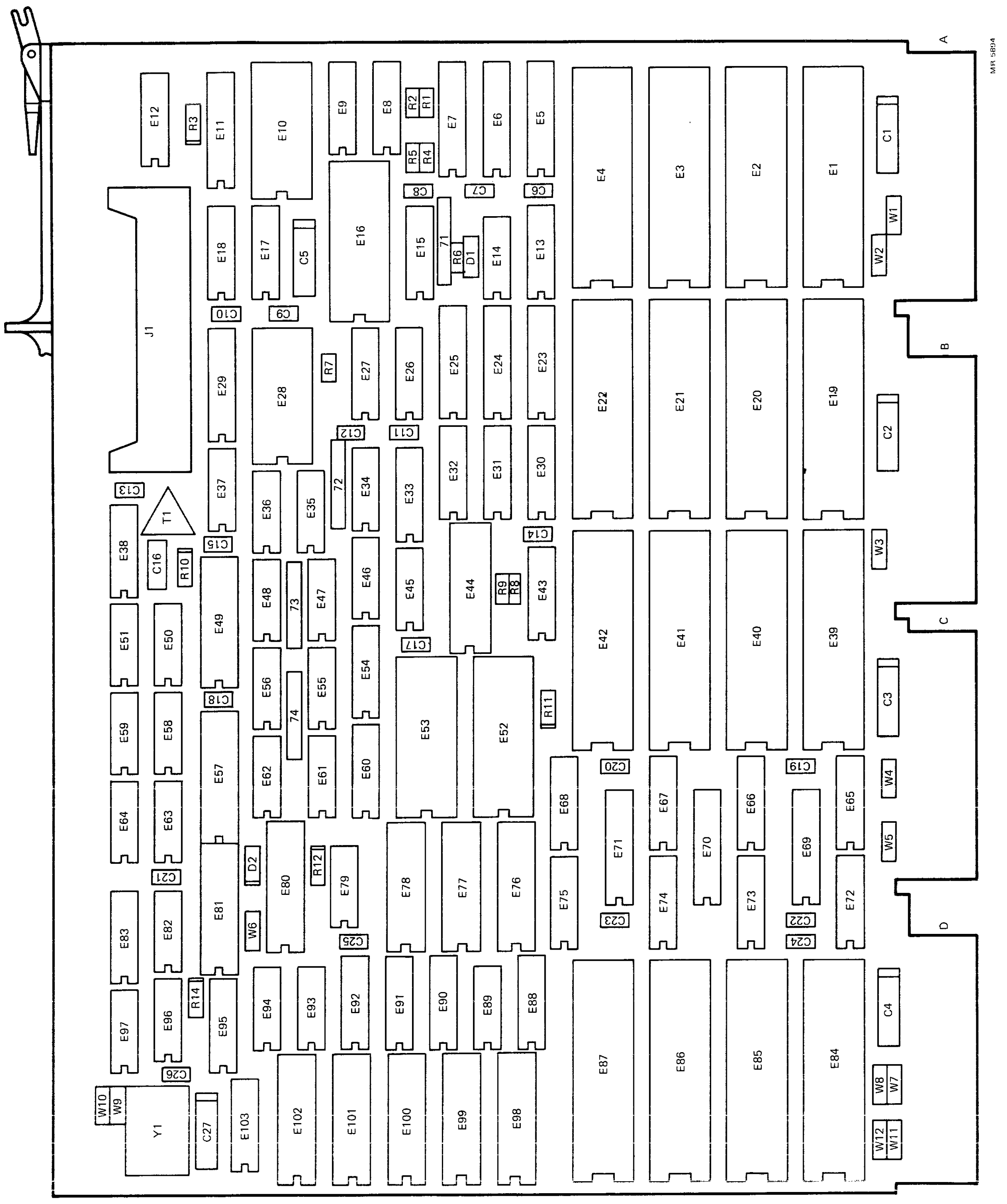

J1
T1
Y1
E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11 E12 E13 E14 E15 E16 E17 E18 E19 E20
E21 E22 E23 E24 E25 E26 E27 E28 E29 E30 E31 E32 E33 E34 E35 E36 E37 E38 E39 E40
E41 E42 E43 E44 E45 E46 E47 E48 E49 E50 E51 E52 E53 E54 E55 E56 E57 E58 E59 E60
E61 E62 E63 E64 E65 E66 E67 E68 E69 E70 E71 E72 E73 E74 E75 E76 E77 E78 E79 E80
E81 E82 E83 E84 E85 E86 E87 E88 E89 E90 E91 E92 E93 E94 E95 E96 E97 E98 E99 E100
E101 E102 E103
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20
C21 C22 C23 C24 C25 C26 C27
R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R14
D1 D2
W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12
71 72 73 74
A
B
C
D
MR 5894

Figure 1-1 FPF11 Module (M8188)

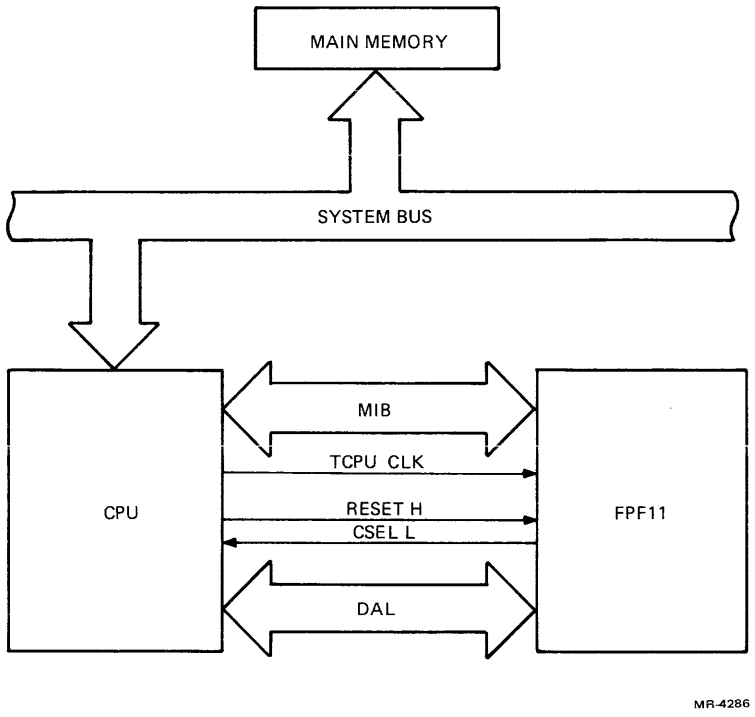


Figure 1-2 FPF11 Floating-Point Processor, Functional Block Diagram

1.2 FEATURES OF THE FPF11

- Performs medium-speed, floating-point operations
- 17_{10}-digit accuracy
- Microprogrammed control store
- Six 64-bit floating-point accumulators
- Error recovery aids
- No affect on system bus loading
- 32-bit (single-precision) and 64-bit (double-precision) data modes
- Addressing modes compatible with existing PDP-11 addressing modes
- Special instructions that improve input/output routines and mathematical subroutines
- Allows execution of in-line code*
- Converts 32- or 64-bit floating-point numbers to 16- or 32-bit integers during store instructions
- Converts 32-bit floating-point numbers to 64-bit floating-point numbers, and vice versa, during load and store instructions

1.3 ARCHITECTURE

The FPF11 contains scratch-pad registers, a floating exception address (FEA) pointer, status and error registers, and six general-purpose accumulators (AC0–AC5).

Each accumulator is interpreted to be either 32 or 64 bits long, depending on the instruction and the status of the floating-point processor. For 32-bit instructions, only the leftmost bits are used. The remaining bits are unaffected.

The six general-purpose accumulators are used in numeric calculations and interaccumulator data transfers. The first four registers (AC0–AC3) are also used for all data transfers between the FPF11 and the central processor's general registers or memory.

1.4 SPECIFICATIONS

Identification	M8188
Type	Quad-size
Height	26.5 cm (10.5 in)
Length	22.8 cm (8.9 in)
Width	1.27 cm (0.5 in)

* That is, floating-point instructions and other instructions can appear in any sequence desired.

Bus Loads	None
Environment	
Operating Temperature	5° C to 50° C (41° F to 122° F)
Humidity	10 to 95 percent (no condensation)
Power Consumption	+5.0 Vdc, 5.5 A

1.5 RELATED DOCUMENTS

The following documents supplement the information contained in this manual.

Document	Number
Microcomputer Interfaces Handbook	EB-17723-18
Microcomputer Processor Handbook	EB-15836-18

These documents can be ordered from:

Digital Equipment Corporation
444 Whitney Street
Northboro, MA 01532

Attention: Printing and Circulating Services (NR2/M15)
Customer Services Section

CHAPTER 2
REVIEW OF FLOATING-POINT NUMBERS

2.1 INTRODUCTION

This chapter briefly outlines the fundamentals of floating-point arithmetic, providing useful background for more advanced topics in later chapters. If you are already familiar with floating-point arithmetic, go directly to Chapter 3 for a discussion of FPF11 data formats.

2.2 INTEGERS

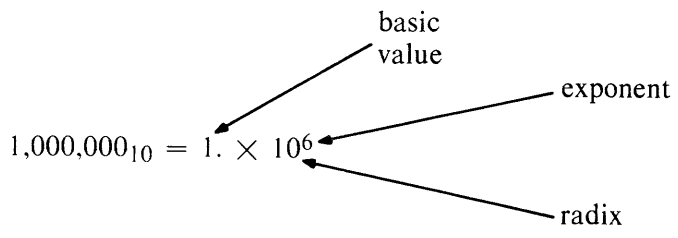
In many cases, data in a computer system is represented by integers. For example, the numbers that could be represented in a 16-bit machine would range from 000000_8 to 177777_8 (0_{10} to $65{,}536_{10}$). However, there are problems with integer representation. A number between 1 and 2, for example, could not be represented. Thus, integer representation imposes an *accuracy* limitation. Furthermore, numbers greater than $65{,}536_{10}$ also could not be represented. This imposes a *range* limitation.

These limitations are caused by the stationary position of the *radix point* (for example, the decimal point in base 10 notation, the binary point in base 2 notation). An integer's radix point is omitted in integer representation since only whole numbers are used. (A defined radix point implies the possibility of fractions in the numbering scheme.) For this reason, an integer is sometimes called a *fixed-point* number.

Integer notation, however, can be modified to overcome the range and accuracy limitations imposed by a fixed radix point. This is done through the use of *floating-point* notation.

2.3 FLOATING-POINT NUMBERS

Floating-point numbers, unlike integers, have no position restrictions on their radix points. A popular type of floating-point representation is called *scientific notation*. In scientific notation a floating-point number is represented by some basic value multiplied by the radix raised to some power.



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

$$\begin{aligned}512_{10} &= 51200. \times 10^{-2}\\ &= 5120. \times 10^{-1}\\ &= 512. \times 10^{0}\\ &= 51.2 \times 10^{1}\\ &= 5.12 \times 10^{2}\\ &= .512 \times 10^{3}\end{aligned}$$

The convention chosen for representing floating-point numbers with scientific notation in the FPF11 requires the radix point to be always to the left of the most significant digit in the basic value (for instance, $.512 \times 10^3$ above). In this way *fractions* are represented. More examples of scientific notation are shown below.

Decimal Number	**Scientific Notation**		
	Decimal	**Octal**	**Binary**
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}	$0. \times 8^{-1}$	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 number whose basic value is greater than (or equal to) 1 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 1. Similarly, establishing the radix point in a number whose basic value is between 1 and 0 (that is, a fraction) is accomplished by shifting the number to the left until all leading 0s are eliminated. Each left shift causes the exponent to be decremented by 1.

To summarize, the value of a number remains constant if its exponent is incremented for each right shift of the basic value and decremented for each left shift. The representation for floating-point fractions is one in which all nonsignificant leading 0s have been removed. The process used to obtain this representation is called *normalization* and is explained in more detail in Paragraph 2.4.

2.4 NORMALIZATION

In digital computers the number of bits in a fraction is limited. Retention of nonsignificant leading 0s decreases accuracy by taking places that could be filled by significant digits. For this reason, the process called normalization is used in the FPF11. Normalization consists of testing a fraction for leading 0s and shifting it left until it is in the form 0.1... . The exponent is accordingly decremented by the number of places of the fraction is shifted left. This ensures that the normalized number retains equivalence with the original number. Since digits to the right of the binary point are weighted with inverse powers of 2 (that is, 1/2, 1/4, 1/8...), the smallest normalized fraction is 0.10000... (1/2), and the largest normalized fraction is 0.11111... . Figure 2-1 shows an unnormalized fraction that must be left-shifted six places to be normalized. The exponent is decremented by six to maintain equivalence with the original number.

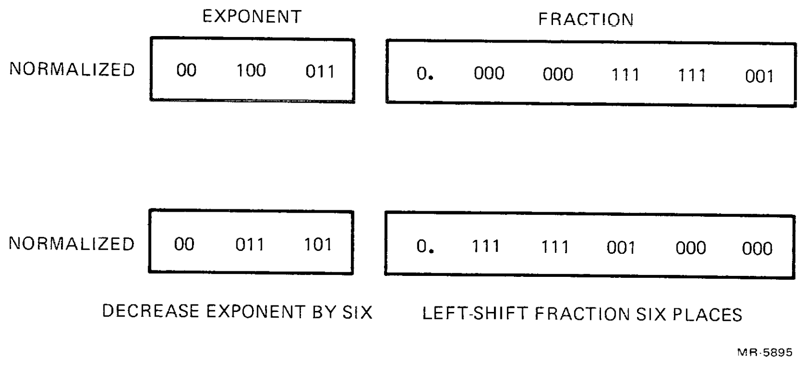


Figure 2-1 Normalization

Problem A – Represent the number 75_{10} as a binary normalized floating-point number.

1. Integer conversion.

   $75_{10} = 1001011_2$

2. Convert to floating-point form.

   $1001011.0 \times 2^0 = 0.1001011 \times 2^7$

   Fraction = 0.1001011
   Exponent = 111

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

1. Integer conversion.

   $0.25_{10} = 0.01_2$

2. Convert to floating-point form.

   $0.01 \times 2^0 = 0.1 \times 2^{-1}$

   Fraction = 0.1
   Exponent = -1

2.5 FLOATING-POINT ADDITION AND SUBTRACTION

In order to perform floating-point addition and subtraction, the exponents of the two floating-point numbers involved 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 exponent. When the exponents are aligned or equal, the fractions can then be added or subtracted. The exponent value indicates the number of places the binary point is to be moved to obtain the integer 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 points.

$$0.101\ 000\ 000\ 000\ 000 \times 2^6 = 50_8 = 40_{10}$$

$$0.111\ 000\ 000\ 000\ 000 \times 2^3 = 7_8 = 7_{10}$$

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

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

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

$$0\ \overset{5}{\overbrace{101}}\ \overset{7}{\overbrace{111}}.000\ 000\ 000 = 57_8 = 47_{10}$$

2.6 FLOATING-POINT MULTIPLICATION AND DIVISION

In floating-point multiplication, a fraction is multiplied by another and their exponents are added. In floating-point division, a fraction is divided by another and their exponents are subtracted. You need not align the binary points in floating-point multiplication or division.

Example: Multiply 7_{10} by 40_{10}.

1.

$$\begin{array}{l}
\quad 0.1110000 \times 2^3 = 7_8 = 7_{10} \\
\times 0.1010000 \times 2^6 = 50_8 = 40_{10} \\
\hline
\qquad 1110000 \\
\quad\ 0000 \\
\quad 11100 \\
\hline
.10001100000000 \times 2^9
\end{array}$$

(Result already in normalized form.)

2. Move the binary point nine places to the right.

$$\overset{4}{\overbrace{100}}\overset{3}{\overbrace{011}}\overset{0}{\overbrace{000}}.00000 = 430_8 = 280_{10}$$

Example: Multiply 7_{10} by 40_{10}.

1. $$\frac{.1111000 \times 2^4}{.1010000 \times 2^3}$$

$$1.010000\overline{\smash{)}\,.1111000 =}$$

$$\begin{array}{r}
1.100000 \\
1010000\overline{\smash{)}\,1111000.000000} \\
\underline{1010000} \\
101000 \\
\underline{101000} \\
0
\end{array}$$

2. Exponent: $4 - 3 = 1$

3. Result: 1.100000×2

   Normalized result: $\underbrace{.1100000}_{\text{normalized fraction}} \times \underbrace{2^2}_{\text{normalized exponent}}$

   Move the binary point two places to the right.

   $11.00000 = 3_8 = 3_{10}$

CHAPTER 3
DATA FORMATS

3.1 INTRODUCTION

The FPF11 requires its input data (operands) to be formatted in one of four ways: short-integer format (I), long-integer format (L), single-precision format (F), and double-precision format (D).

Data output from the FPF11 is also formatted, taking the form of:

- FPF11 status information and FPF11 exception information required by the CPU, or
- Data sent to memory (via the CPU) in I, L, F, or D format.

This chapter describes the FPF11 data formats mentioned above. (It is assumed you are familiar with 2's complement notation.)

3.2 FPF11 INTEGER FORMATS: SHORT AND LONG

The short-integer format (I) is 16 bits long, the long-integer format (L) 32 bits long. Data words (operands) in integer format are represented in 2's complement notation. In the I and L formats the most significant bit of the data word is the sign bit. Figure 3-1 shows the integers 5 and −5 in the I and L formats. Figure 3-2 illustrates the formats in which integers are arranged *in memory*. Integers sent to memory must be in one of the formats shown.

Integers received by the FPF11 are arranged and manipulated according to the type of instruction being executed. Refer to Chapter 4, Paragraphs 4.3.11 and 4.3.12 for descriptions of the ways in which incoming integers are manipulated during the load exponent and load convert integer-to-floating instructions, respectively.

3.3 FPF11 FLOATING-POINT FORMATS: SINGLE- AND DOUBLE-PRECISION

The single-precision format (F) is 32 bits long, the double-precision format (D) 64 bits long. Figure 3-2 shows that the most significant bit is the sign of the fraction (and the floating-point number being represented). The next 8 bits contain the value of the exponent, expressed in *excess 200 notation* (see Paragraph 3.3.1.2). The remaining bits (23 for single-precision, 55 for double-precision) contain the fraction. The fraction and its associated sign bit are expressed in *sign and magnitude notation* (see Paragraph 3.3.1.1).

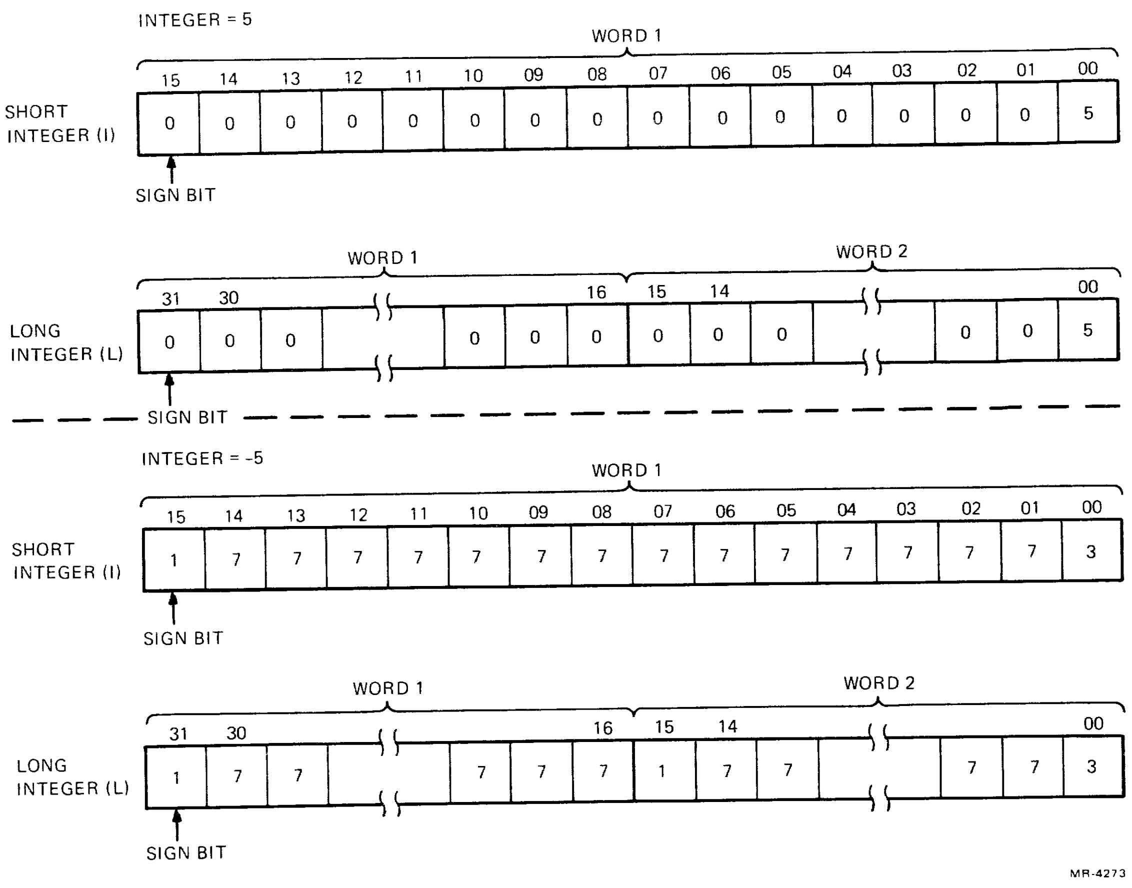


Figure 3-1 Integer Formats

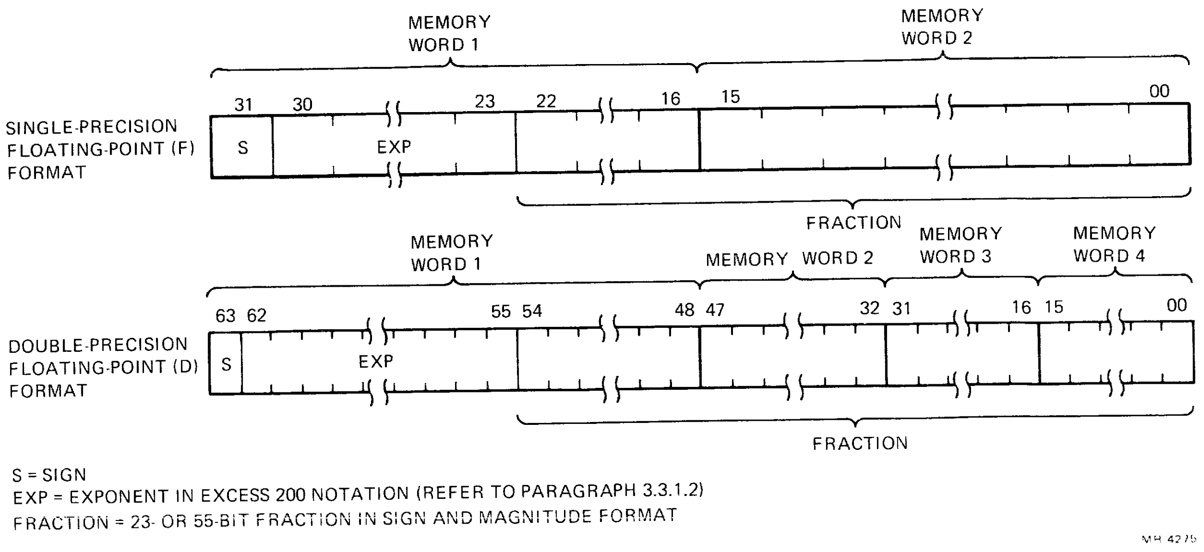
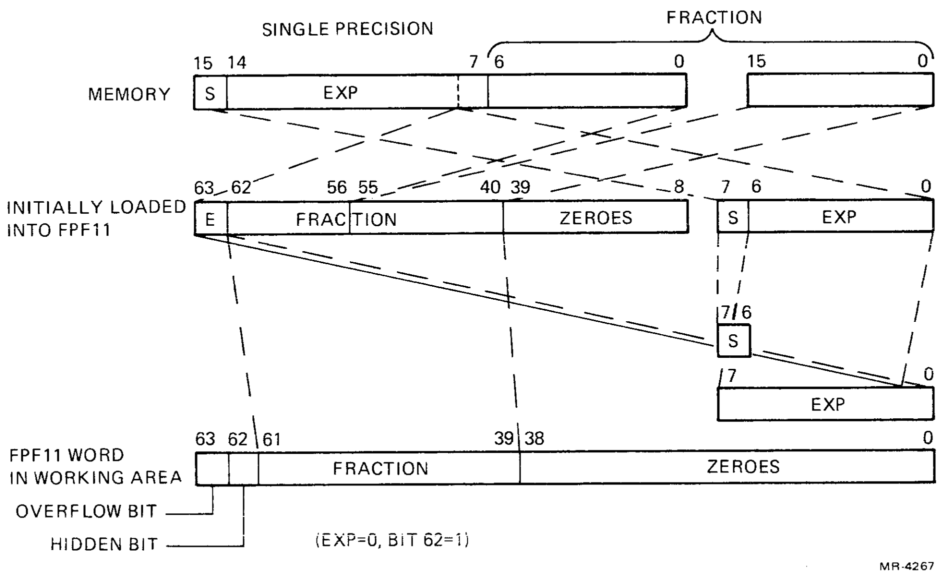

MEMORY WORD 1
MEMORY WORD 2
SINGLE-PRECISION FLOATING-POINT (F) FORMAT
S
EXP
FRACTION
MEMORY WORD 3
MEMORY WORD 4
DOUBLE-PRECISION FLOATING-POINT (D) FORMAT
S = SIGN
EXP = EXPONENT IN EXCESS 200 NOTATION (REFER TO PARAGRAPH 3.3.1.2)
FRACTION = 23- OR 55-BIT FRACTION IN SIGN AND MAGNITUDE FORMAT
MR 4275

Figure 3-2 Floating-Point Data Formats in Memory

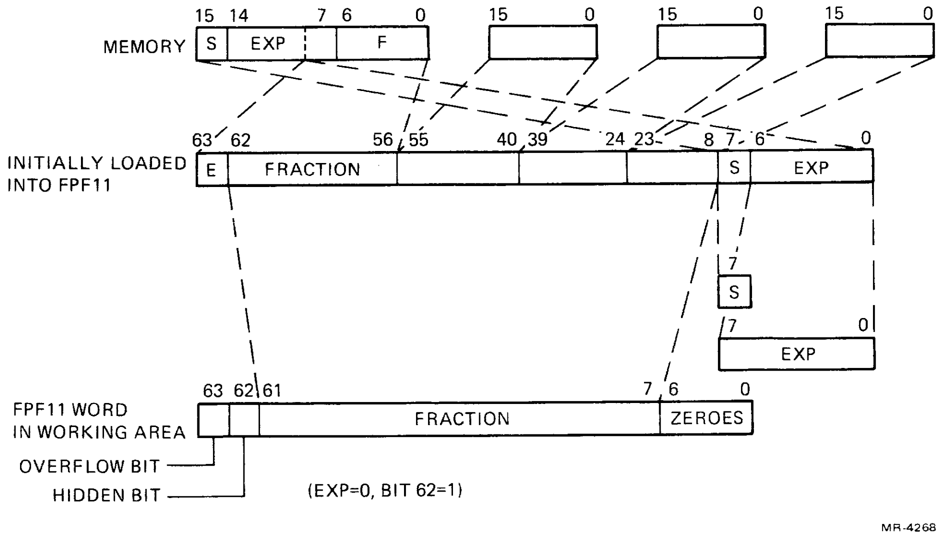
3.3.1 FPF11 Floating-Point Data Word

Figure 3-3 illustrates the formats in which floating-point numbers are arranged *in memory*. Floating-point numbers sent to memory must be in one of the formats shown. Floating-point numbers received by the FPF11 are arranged as illustrated in Figure 3-4.

The sign bit, exponent bits, and fraction bits in an FPF11 data word have the same values as their corresponding data word in memory. Note, however, that the FPF11 data word has more bits than its counterpart in memory. This is so because the FPF11 has provision for generating an overflow bit, a "hidden" bit.



a. Single-Precision Format



b. Double-Precision Format

Figure 3-3 Floating-Point Data Words in Memory

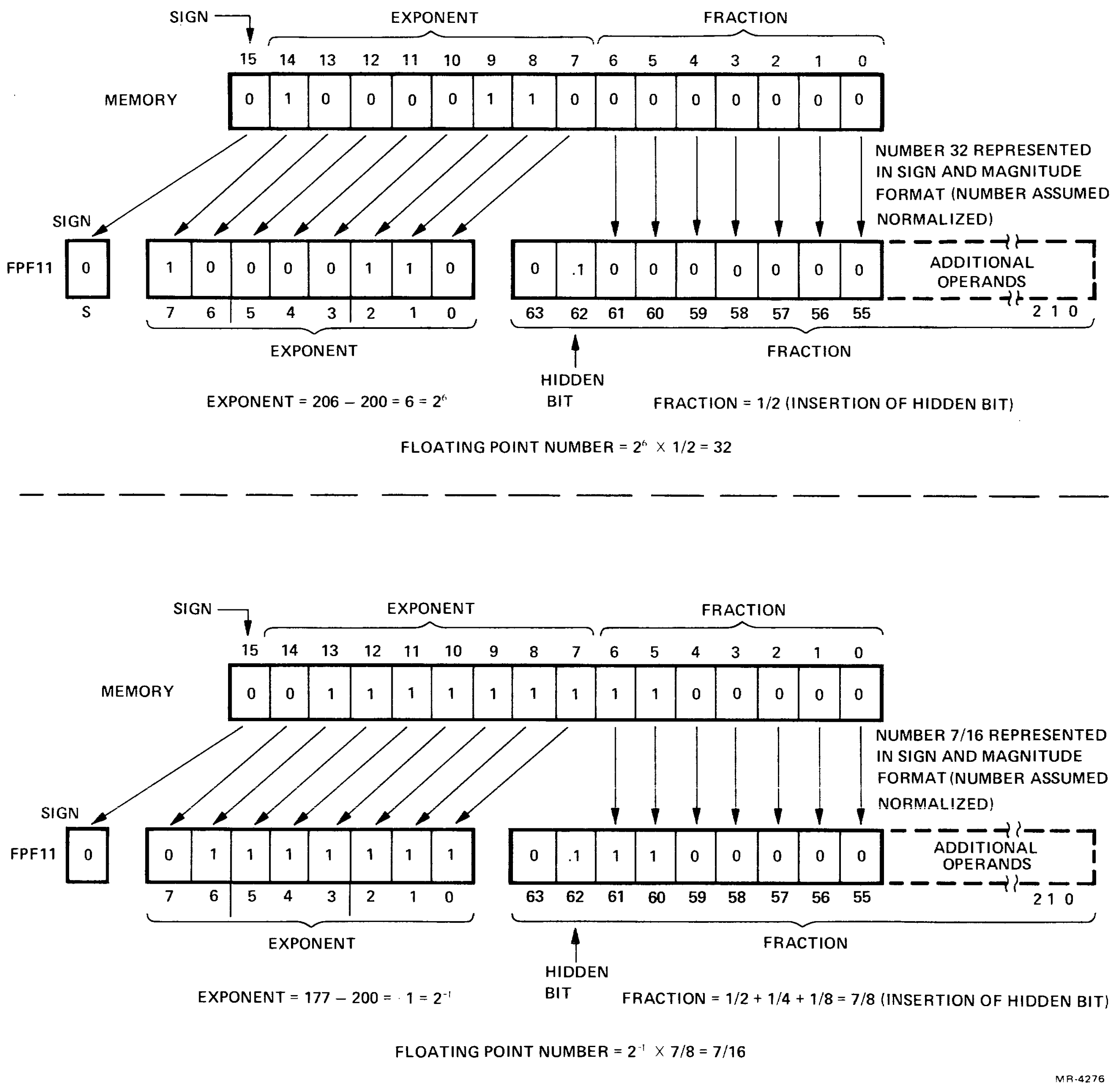


Figure 3-4 Interpretation of Floating-Point Numbers

For this discussion, think of the FPF11 data word as being divided into two major parts:

1. A fraction, with its associated sign bit, hidden bit, and overflow bit.
2. An exponent.

3.3.1.1 Floating-Point Fraction – The fraction is expressed in *sign and magnitude notation.* The following simple example illustrates the idea behind this method of notation.

Integer Notation	2's Complement Notation	Sign and Magnitude Notation
+2	000010	$\underbrace{0}_{\text{sign bit}}\underbrace{00010}_{\text{magnitude}}$
−2	111110	$\underbrace{1}_{\text{sign bit}}\underbrace{00010}_{\text{magnitude}}$

Only a change of sign bit is required to change the sign of a number in sign and magnitude notation. Note that a positive number is the same in both notations.

Unnormalized floating-point fractions have a range of approximately 0 through 2, as shown in Figure 3-5. The FPF11, however, normalizes all unnormalized fractions. That is, the fractions are adjusted so that there is a 0 to the left of the binary point (bit 63) and a 1 to the right of the binary point (bit 62). Thus, normalized fractions range in magnitude from 0.1000 to 0.1111 (1/2 to approximately 1).

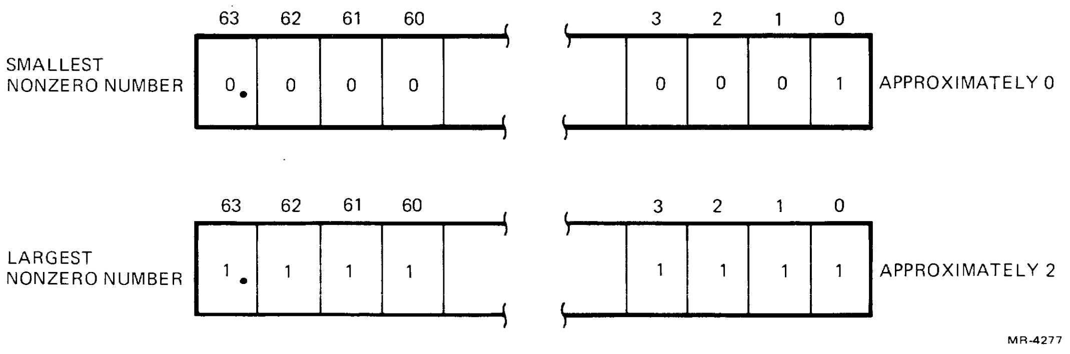


Figure 3-5 Unnormalized Floating-Point Fraction

The fraction overflow bit (bit 63) is set during certain arithmetic operations. For example, during addition certain sums will produce an overflow such as 0.1000... + 0.100... , which yields 1.000... . This result must be normalized, so the FPF11 right-shifts the fraction one place and increases the exponent by one.

Bit 62 is called the *hidden* bit. Recall that since fractions are normalized by the FPF11, the bit immediately to the right of the binary point (bit 62) is always a 1. This bit is dropped when a fraction is sent to memory and appended when a fraction is received from memory. This procedure allows one extra bit of significance in floating-point fraction representation.

3.3.1.2 Floating-Point Exponent – The 8-bit floating-point exponent is expressed in *excess 200 notation.* Figure 3-6 illustrates the relationship between exponents in 2's complement notation and exponents in excess 200 notation. It shows the range of floating-point numbers that can be handled by the FPF11. For simplicity, a fraction length of only three bits is shown.

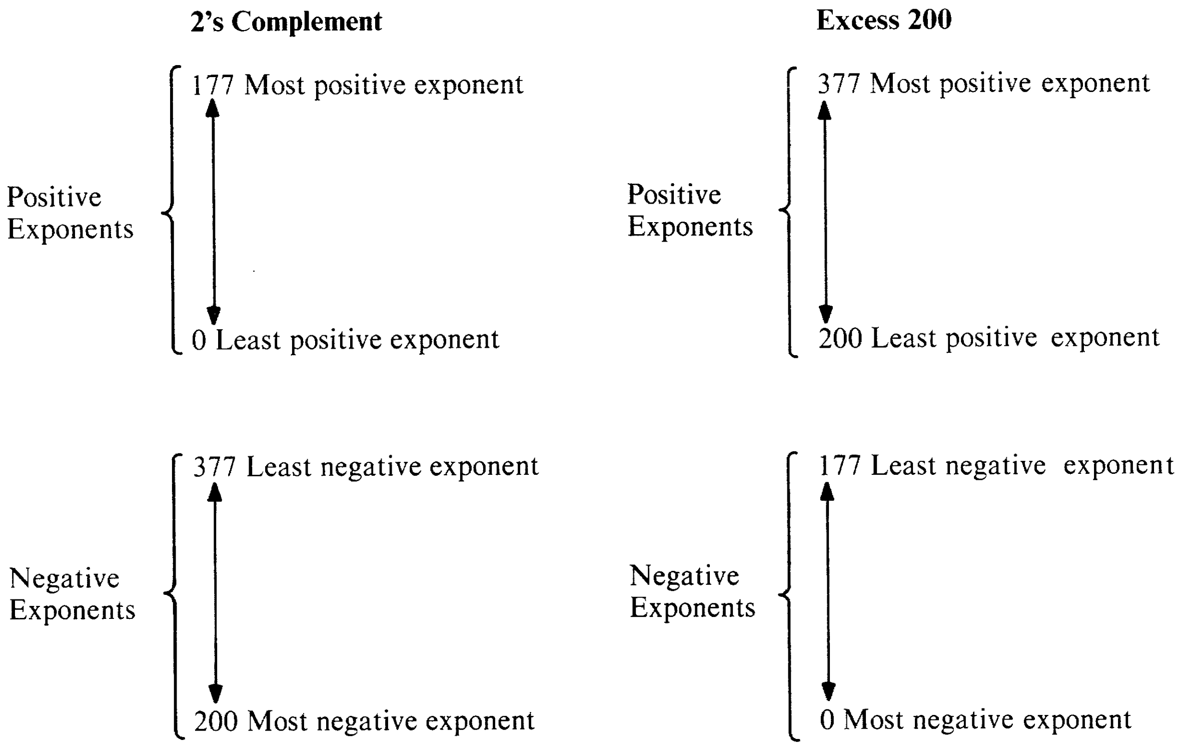

2's Complement
Excess 200
Positive Exponents
177 Most positive exponent
0 Least positive exponent
Positive Exponents
377 Most positive exponent
200 Least positive exponent
Negative Exponents
377 Least negative exponent
200 Most negative exponent
Negative Exponents
177 Least negative exponent
0 Most negative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

Figure 3-6 Floating-Point Exponent Notations

Note that an exponent in excess 200 notation is obtained by simply adding 200 to the exponent in 2's complement notation. Thus, 8-bit exponents in excess 200 notation range from 0 to 377 (−200 to +177). A number with an exponent in excess of −200 is treated by the FPF11 as 0.

The number 0.1_2 is actually 0.1×2^0, and the exponent is represented as 10,000,000 because 200_8 represents an exponent of zero.

3.4 FPF11 PROGRAM STATUS REGISTER

The FPF11 contains a resident program status register that contains the floating-point condition codes (carry, overflow, zero, and negative) that can be copied into the central processing unit. In other words, FN, FZ, FV, and FC can be copied into the CPU's N, Z, V, and C condition codes, respectively. The program status register also contains three mode bits and additional bits to enable various interrupt conditions. Figure 3-7 shows the layout of the program status register. Each bit shown in the figure is described in Table 3-1.

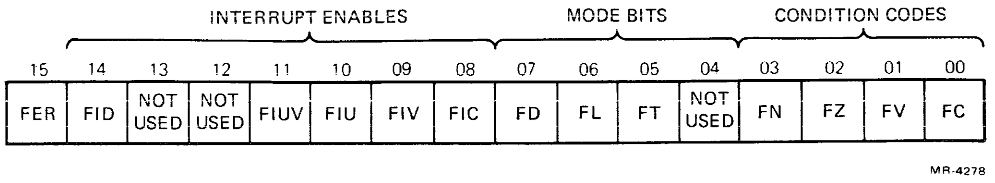


Figure 3-7 FPF11 Status Register Format

Table 3-1 FPF11 Status Register Bit Descriptions

Bit	Name	Function
15	FER	This bit indicates an error condition of the FPF11.
14	FID	Floating interrupt disable – When this bit is on, all interrupts by the FPF11 are disabled; normally clear. Primarily a maintenance feature.
13		Not used.
12		Not used.
11	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, all arithmetic operations treat it as if it were a $+0$.
10	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 this bit is not set and underflow occurs, the result is set to 0.
9	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 this bit is not set, the result of the operation is the same; however, no interrupt occurs.
8	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 occurs, the destination register is cleared and the source register is untouched. When this bit is reset, the result of the operation is the same; however, no interrupt occurs.
7	FD	Double-precision mode bit – When set, this bit specifies double-precision format; when reset, it specifies single-precision format.
6	FL	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.
5	FT	Truncate bit – When set, this bit causes the result of any floating-point operation to be truncated rather than rounded.
3–0	FN, FZ, FV, FC	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 Chapter 4, Table 4-1.

3.5 PROCESSING OF FLOATING-POINT EXCEPTIONS

Location 244_8 is the interrupt vector used to handle all floating-point interrupts. A total of six possible interrupt exceptions can occur; these are encoded in the FPF11 exception code (FEC) register. The interrupt exception codes represent an offset into a dispatch table, which routes the program to the correct error handling routine. (The dispatch table is a function of the software.) The FEC for each exception is briefly described in Table 3-2.

In addition to the FEC register, the CPU 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.

Table 3-2 FPF11 Exception Codes

Exception Code	Definition
2	Floating op code error – The FPF11 causes an interrupt for an erroneous op code.
4	Floating divide by zero – Division by zero causes an interrupt if FID is not set.
6	Floating (or double) integer conversion error.
10	Floating overflow.
12	Floating underflow.
14	Floating undefined variable.

NOTE

The traps for exception codes 6, 10, 12, and 14 can be enabled in the FPF11 program status register. All traps are disabled if FID is set.

CHAPTER 4
FLOATING-POINT INSTRUCTIONS

4.1 FLOATING-POINT ACCUMULATORS

The FPF11 contains six accumulators (AC0–AC5). These accumulators are 64-bit read/write scratch-pad memories with nondestructive readout. Each accumulator is interpreted as being either 32 or 64 bits long, depending upon the instruction and the FPF11 status (refer to Chapter 3). If an accumulator is interpreted as being 64 bits long, 64 bits of data occupy the entire accumulator. If an accumulator is interpreted as being 32 bits long, 32 bits of data occupy only the leftmost 32 bits of an accumulator, as shown in Figure 4-1. The remaining bits are irrelevant.

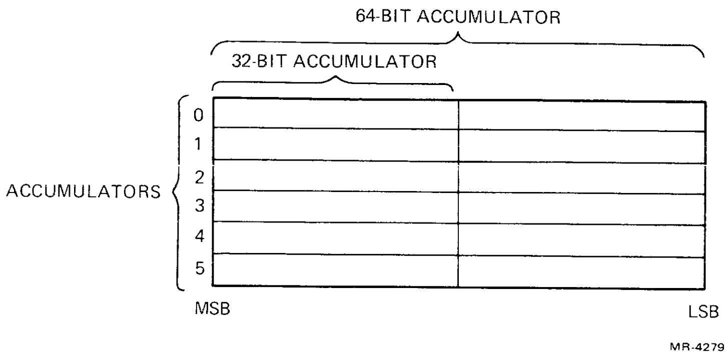


Figure 4-1 Floating-Point Accumulators

The floating-point accumulators are used in numeric calculations and interaccumulator data transfers. Accumulators AC0–AC3 are used for all data transfers between the FPF11 and the CPU or memory.

4.2 INSTRUCTION FORMATS

An FPF11 instruction must be in one of five formats. These formats are summarized in Figure 4-2. The 2-bit AC field (bits 6 and 7) allows selection of scratch-pad accumulators AC0–AC3 only.

If address mode 0 is specified with formats F1 or F2, bits <2:0> are used to select a floating-point accumulator. Only accumulators AC5–AC0 can be specified in mode 0. If AC6 or AC7 is specified in bits <2:0> in mode 0, the FPF11 traps when floating-point interrupts are enabled (FID = 0). The FEC will indicate an illegal op code error (exception code 2). Table 4-1 lists the formats of the FPF11 instructions.

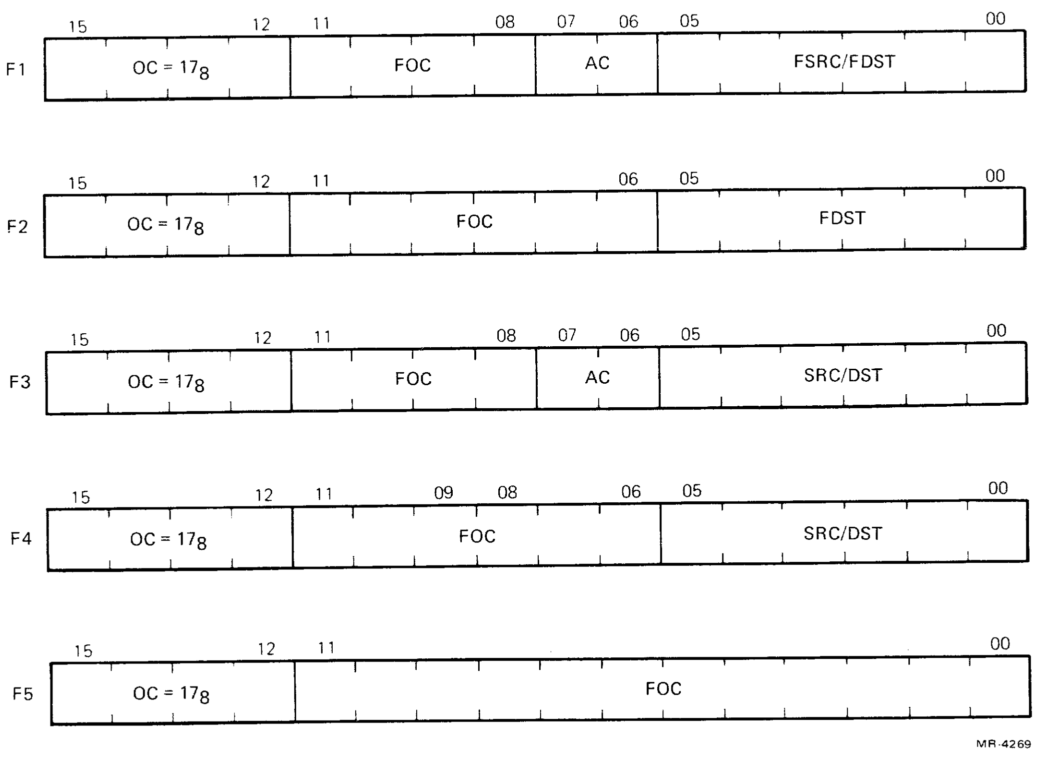


Figure 4-2 FPF11 Instruction Formats

The fields of the various instruction formats listed in Table 4-1 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 PDP-11/23-11/24 instructions.
DST	Destination – A 6-bit destination field identical to that in PDP-11/23-11/24 instructions.
FSRC	Floating source – A 6-bit field used only in format F1. This field is identical to SRC except in mode 0, when it references a floating-point accumulator rather than a CPU general-purpose register.
FDST	Floating destination – A 6-bit field used in formats F1 and F2. This field is identical to DST except in mode 0, when it references a floating-point accumulator instead of a CPU general-purpose register.
AC	Accumulator – A 2-bit field used in formats F1 and F3 to specify FPF11 scratch-pad accumulators AC0–AC3.

Table 4-1 Format of FPF11 Instructions

Instruction Format	Instruction	Mnemonics
F2	ABSOLUTE	ABSF FDST ABSD FDST
F1	ADD	ADDF FSRC, AC ADDD FSRC, AC
F2	CLEAR	CLRF FDST CLRD FDST
F1	COMPARE	CMPF FSRC, AC CMPD FSRC, AC
F5	COPY FLOATING CONDITION CODES	CFCC
F1	DIVIDE	DIVF FSRC, AC DIVD FSRC, AC
F1	LOAD	LDF FSRC, AC LDD FSRC, AC
F1	LOAD CONVERT	LDCFD FSRC, AC FDCDF FSRC, AC
F3	LOAD CONVERT INTEGER-TO-FLOATING	LDCIF SRC, AC LDCID SRC, AC LDCLF SRC, AC LDCLD SRC, AC
F3	LOAD EXPONENT	LDEXP SRC, AC
F4	LOAD FPF11'S PROGRAM STATUS	LDFPS SRC
F1	MODULO	MODF FSRC, AC MODD FSRC, AC
F1	MULTIPLY	MULF FSRC, AC MULD FSRC, AC
F2	NEGATE	NEGF FDST NEGD FDST
F5	SET DOUBLE MODE	SETD
F5	SET FLOATING MODE	SETF
F5	SET INTEGER MODE	SETI
F5	SET LONG-INTEGER MODE	SETL

Table 4-1 Format of FPF11 Instructions (Cont)

Instruction Format	Instruction	Mnemonics
F1	STORE	STF AC, FDST STD AC, FDST
F1	STORE CONVERT	STCFD AC, FDST STCDF AC, FDST
F3	STORE CONVERT FLOATING-TO-INTEGER	STCFI AC, DST STCFL AC, DST STCDI AC, DST STCDL AC, DST
F3	STORE EXPONENT	STEXP AC, DST
F4	STORE FPF11'S PROGRAM STATUS	STFPS DST
F4	STORE FPF11'S STATUS	STST DST
F1	SUBTRACT	SUBF FSRC, AC SUBD FSRC, AC
F2	TEST	TSTF FDST TSTD FDST

4.3 INSTRUCTION SET

Table 4-2 contains the instruction set of the FPF11. Since some of the symbology may not be familiar, a brief explanation if it follows. The information in Table 4-2 is expressed in symbolic notation to provide you with a quick reference to the function of each instruction. The paragraphs following the table supplement its information.

1. A floating-point flip-flop, designated FD, determines whether single- or double-precision floating-point format is specified. If the flip-flop is cleared, single-precision is specified and designated by F. If the flip-flop is set, double-precision is specified and designated by D. Examples are NEGF, NEGD, and SUBD.

   NOTE

   Only the assembler or compiler differentiates between NEGF and NEGD or LDCID and LDCLD instructions. The floating-point does not differentiate between the instructions but depends upon the value of FD and FL as usually controlled by SETD, SETF, SETC, and SETI instructions (that is, LDCID → SETI → SETD → LDCLD).

2. An integer flip-flop, designated FL, determines whether short-integer or long-integer format is specified. If the flip-flop is cleared, short-integer format is specified and designated by I. If the flip-flop is set, long-integer format is specified and designated by L. Examples are SETI and SETL.

3. Several convert instructions use the symbology defined below.

   $C_{IL,FD}$ – Convert long-integer to double-precision floating.

   $C_{FD,IL}$ – Convert double-precision floating to long-integer.

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

4. UPLIM is defined as the largest possible number that can be represented in floating-point format. This number has an exponent of 377 (in excess 200 notation) and a fraction of all 1s. Note that the UPLIM depends on the format specified. LOLIM is defined as the smallest possible number that is not equal to 0. This number has an exponent of 001 and a fraction of all 0s, besides the hidden bit.

5. The following conventions are used when referring to address locations.

(xxxx)	the contents of the location specified by xxxx
ABS (address)	absolute value of (address)
EXP (address)	exponent of (address) in excess 200 notation

6. Some of the octal codes listed are in the form of mathematical expressions. These octal codes can be calculated as shown in the following examples.

   Example 1: LDFPS Instruction

   Mode 3, register 7 specified (F4 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, register 6 specified (F1 instruction format).

   172400 + AC * 100 + FSRC

      AC = 2

      2 * 100 = 200

   172400 + 200 = 172600

      FSRC is equal to 26.

   172600 + 26 + 172626

7. AC v 1 means that the accumulator field (bits 6 and 7 in formats F1 and F3) is logically ORed with 01.

   Example:

      Accumulator field = bits 6 and 7 = AC2 = 10_2, AC v 1 = 11.

Table 4-2 FPF11 Instruction Set

Mnemonics	Instruction Description	Octal Code

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

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 ← if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) < 0; otherwise FN ← 0
		172000+AC*100+FSRC
		F1 Format

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

CFCC	Copy Floating Condition Codes
	C ← FC
	V ← FV
	Z ← FZ
	N ← FN
		170000
		F5 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; otherwise FV ← 0
	FZ ← 1 if EXP (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) < 0; otherwise FN ← 0
		174400+AC*100+FSRC
		F1 Format

Table 4-2 FPF11 Instruction Set (Cont)

Mnemonics	Instruction Description	Octal Code

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

LDCDF FSRC, AC
LDCFD FSRC, AC
	Load Convert Double-to-Floating or Floating-to-Double
	AC ← $C_{F,D}$ or $C_{D,F}$ (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 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

LDCIF SRC, AC
LDCID SRC, AC
LDCLF SRC, AD
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 ← $C_{IL,FD}$ (SRC)
	FC ← 0
	FV ← 0
	FZ ← 1 if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) < 0; otherwise FN ← 0
	
	$C_{IL,FD}$ 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 double specified, and if IL = 1, a 32-bit integer (L) is specified. If floating-point flip-flop FD = 0, a 32-bit floating-point number (F) is specified, and if FD = 1, a 64-bit floating-point number (D) is specified. If a 32-bit integer is specified and addressing mode 0 or immediate mode is used, the 16 bits of the source register are left justified, and the remaining 16 bits are zeroed before the conversion.
		177000+AC*100+SRC
		F3 Format

Table 4-2 FPF11 Instruction Set (Cont)

Mnemonics	Instruction Description	Octal Code

LDEXP SRC, AC	Load Exponent
	AC SIGN ← (AC SIGN)
	AC EXP ← (SRC) + 200 only if ABS (SRC) < 177
	AC FRACTION ← (AC FRACTION)
	FC ← 0
	FV ← 1 if (SRC) > 177; otherwise FV ← 0
	FZ ← 1 if EXP (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) < 0; otherwise FN ← 0
		176400+AC*100+SRC
		F3 Format

LDFPS SRC	Load FPF11's Program Status Word FPS ← (SRC)	170100+SRC F4 Format

MODF FSRC, AC
MODD FSRC, AC
	Floating Modulo
	AC v 1 ← integer part of (AC)*(FSRC)
	AC ← fractional part of (AC)*(FSRC) - (AC v 1) if |(AC)*(FSRC)| ⩾ LOLIM or FIU = 1; 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
	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

MULF FSRC, AC
MULD FSRC, AC
	Floating Multiply
	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
		171000+AC*100+FSRC
		F1 Format

NEGF FDST
NEGD FDST
	Negate
	FDST ← minus (FDST) if EXP (FDST) ≠ 0; otherwise FDST ← 0
	FC ← 0
	FV ← 0
	FZ ← 1 if if EXP (FDST) = 0; otherwise FZ ← 0
	FN ← 1 if (FDST) < 0; otherwise FN ← 0
		170700+FDST
		F2 Format

Table 4-2 FPF11 Instruction Set (Cont)

Mnemonics	Instruction Description	Octal Code
SETD	Set Floating Double Mode FD ← 1	170011 F5 Format
SETF	Set Floating Mode FD ← 0	170001 F5 Format
SETI	Set Integer Mode FL ← 0	170002 F5 Format
SETL	Set Long-Integer Mode FL ← 1	170012 F5 Format

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

STCFD AC, FDST
STCDF AC, FDST
	Store Convert from Floating-to-Double or Double-to-Floating
	FDST ← $C_{F,D}$ or $C_{D,F}$ (AC)
	FC ← 0
	FV ← 1 if $\|AC\| \geqslant$ UPLIM; otherwise FV ← 0
	FZ ← 1 if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) < 0; otherwise FN ← 0
	
	The STCFD instruction is the opposite of the LDCDF instruction; thus, if the current format is single-precision floating-point (FD = 0), the source is assumed to be a single-precision number and is converted to double-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 double-precision number and loaded left-justified in the AC. The lower half of the AC is cleared.
		176000+AC*100+FDST
		F1 Format
		F, D-single-precision to double-precision floating
		D, F-double-precision to single-precision floating

STCFI AC, DST STCFL AC, DST STCDI AC, DST STCDL AC, DST	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.	175400+AC*100+DST F3 Format

Table 4-2 FPF11 Instruction Set (Cont)

Mnemonics	Instruction Description	Octal Code

STCFI = Single Float to Single Integer
STCFL = Single Float to Long Integer
STCDI = Double Float to Single Integer
STCDL = Double Float to Long Integer
	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.	

STEXP AC, DST	Store Exponent
	DST ← AC EXPONENT - 200_8
	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

STFPS DST	Store FPF11's Program Status Word DST ← (FPS)	170200+DST F4 Format
STST DST	Store FPF11's Status DST ← (FEC) DST + 2 ← (FEA) if not mode 0 or not immediate mode	170300+DST F4 Format

SUBF FSRC, AC
SUBD FSRC, AC
	Floating Subtract
	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
		173000+AC*100+FSRC
		F1 Format

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

4.3.1 Arithmetic Instructions
The arithmetic instructions (add, subtract, multiply, divide) require one operand in a source (a floating-point accumulator in mode 0, a memory location otherwise) and one operand in a destination accumulator. The instruction is executed by the FPF11 and the result is stored in the destination accumulator.

The compare instruction also requires one operand in a source and one operand in a destination accumulator. However, the two operands remain in their respective locations after the instruction is executed by the FPF11, and there is no transfer of the result.

4.3.2 Floating Modulo Instruction
The floating modulo (MOD) instruction causes the FPF11 to multiply two floating-point operands, separate the product into integer and fractional parts, and store one or both parts as floating-point numbers. The integer portion goes into an odd-numbered accumulator and the fraction goes into an even-numbered accumulator.

The integer portion of the number, when expressed as a floating-point number, contains an exponent greater than 201 in excess 200 notation. This means the integer has a decimal value of some number greater than 1 and less than UPLIM, where UPLIM is the greatest possible number that can be represented by the FPF11.

The fractional portion of the number, when expressed as a floating-point number, contains an exponent less than or equal to 201 in excess 200 notation. This means the fraction has a value less than 1 and greater than LOLIM, where LOLIM is the smallest possible number that can be represented by the FPF11.

4.3.3 Load Instruction
The load instruction causes the FPF11 (and the CPU, if not in mode 0) to take an operand from a source and copy it into a destination accumulator. The source is a floating-point accumulator in mode 0, a memory location otherwise.

4.3.4 Store Instruction
The store instruction causes the FPF11 (and the CPU, if not in mode 0) to take an operand from a source accumulator and transfer it to a destination. The destination is a floating-point accumulator in mode 0, a memory location otherwise.

4.3.5 Load Convert Double-to-Floating, Floating-to-Double Instructions
The load convert double-to-floating (LDCDF) instruction causes the FPF11 to assume that the source specifies a double-precision floating-point number. The FPF11 then converts that number to single-precision, and places this result in the destination accumulator. If the floating truncate (FT) status bit is set, the number is truncated. If the FT bit is not set, the number is rounded by adding a 1 to the single-precision segment. The MSB of the double-precision segment is a 1 depending on the prior conditions set up the the FD bit (see Figure 4-3). If the MSB of the double-precision segment is 0, the single-precision word remains unchanged after rounding.

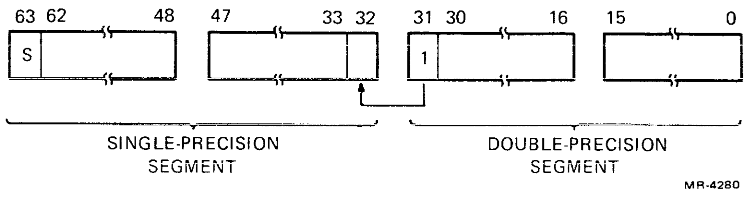


Figure 4-3 Double-to-Single-Precision Rounding

The load convert floating-to-double (LDCFD) instruction causes the FPF11 to assume that the source specifies a single-precision number. The FPF11 then converts that number to double-precision by appending 32 0s to the single-precision word, and places this result in the destination accumulator.

Note that for both load convert instructions, the number to be converted is originally in the source (a floating-point accumulator in mode 0, a memory location otherwise) and is transferred to the destination accumulator after conversion.

4.3.6 Store Convert Double-to-Floating, Floating-to-Double Instructions

The store convert double-to-floating (STCDF) instruction causes the FPF11 to convert a double-precision number located in the source accumulator into a single-precision number. The FPF11 then transfers this result to the specified destination. If the floating truncate bit (FT) is set, the floating-point number is truncated; if the FT bit is not set, the number is rounded. If the MSB (bit 31) of the double-precision segment of the word is a 1, a 1 is added to the single-precision segment of the word. This depends on the prior conditions set up by the FD bit (see Figure 4-3); otherwise, the single-precision segment remains unchanged.

The store convert floating-to-double (STCFD) instruction causes the FPF11 to convert a single-precision number located in the source accumulator into a double-precision number. The FPF11 then transfers this result to the specified destination. The single-to-double precision is obtained by appending the number of 0s equivalent to the double-precision segment of the word (see Figure 4-4).

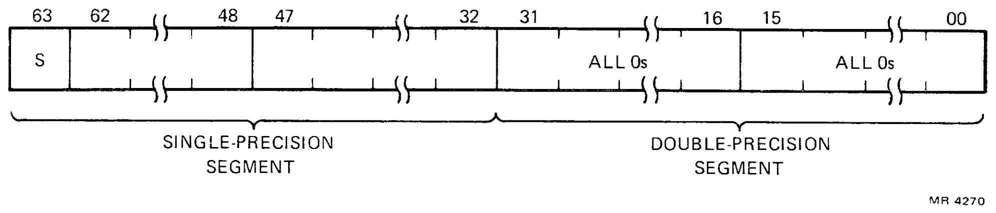


Figure 4-4 Single-to-Double-Precision Appending

Note that for both store convert instructions, the number to be converted is originally in the source accumulator and is transferred to the destination (a floating-point accumulator in mode 0, a memory location otherwise) after conversion.

4.3.7 Clear Instruction

The clear instruction causes the FPF11 (in mode 0) to clear a floating-point number by setting all bits to 0.

4.3.8 Test Instruction

The test instruction causes the FPF11 (in mode 0) to test the sign and exponent of a floating-point number and update the FPF11 status accordingly. The number tested is obtained from the destination (a floating-point accumulator in mode 0, a memory location otherwise). The FC and FV bits are cleared; the FN bit is set only if the destination is negative. The FZ bit is set only if the exponent of the destination is 0. If the FIUV status bit is set, a trap occurs (after the test instruction is executed) when a −0 is encountered.

4.3.9 Absolute Instruction

The absolute instruction causes the FPF11 (in mode 0) to take the absolute value of a floating-point number by forcing its sign bit to 0. If mode 0 is specified, the sign of the number in the floating-point destination accumulator is forced to 0. The exponent of the number is tested, and if it is 0, 0s are written into the accumulator. If the exponent is not 0, the accumulator is unaffected.

If mode 0 is not specified, the sign bit of the specified data word in memory is zeroed. This word is then transferred from memory to a floating-point accumulator. The exponent of this word is tested, and if it is 0, the entire data word is zeroed and transferred back to memory. If the exponent is not 0, the original fraction and exponent are restored to memory.

Absolute and negate instructions are the only instructions that can read and write a memory location.

4.3.10 Negate Instruction

The negate instruction causes the FPF11 (in mode 0) to complement the sign of an operand. If mode 0 is specified, the sign of the number in the floating-point destination accumulator is complemented. The exponent of the number is tested, and if it is 0, 0s are written into the accumulator. If the exponent is not 0, the accumulator is unaffected.

If mode 0 is not specified, the sign bit of the specified data word in memory is complemented. This word is then transferred from memory to a floating-point accumulator. The exponent of this word is tested, and if it is 0, the entire data word is zeroed and transferred back to memory. If the exponent is not 0, the original fraction and exponent are restored to memory.

4.3.11 Load Exponent Instruction

The load exponent instruction causes the FPF11 to load an exponent from the source (a floating-point accumulator in mode 0, a memory location otherwise) into the exponent field of the destination accumulator. In order to do this the 16-bit, 2's complement exponent from the source must be converted (by the FPF11) to an 8-bit number in excess 200 notation. This process is further described below.

Assume that the 16-bit, 2's complement exponent is coming from memory. The possible legal range of 16-bit numbers in memory is 000000_8 to 177777_8. On the other hand, there are two possible legal ranges of exponents in the FPF11.

1. Positive exponents (0_8–177_8) – When 200_8 is added to any of these numbers, the sum stays within the legal 8-bit exponent range (that is, from 200_8 to 377_8).

2. Negative exponents (177601_8–177777_8) – When 200_8 is added to any of these numbers, the sum stays within the legal 8-bit exponent range (that is, from 1_8 to 177_8).

Any number from memory outside these ranges is illegal and will result in either an overflow or an underflow trap condition.

Notice that all legal positive exponents coming from memory have something in common: their nine high-order bits are 0s. Similarly, all legal negative exponents from memory have their nine high-order bits equal to 1. Therefore, to detect a legal exponent, only the nine high-order bits need be examined for all 1s or all 0s.

Example 1: LDEXP 000034

$$\begin{array}{llr|l}
\text{Exponent of 34} & & \overbrace{00000000\;0}^{0}\!\!\!\! & 0\overbrace{011}^{3}\overbrace{100}^{4} \\
\quad 200 & + & & 10000000 \\
\hline
 & & & \underbrace{10}_{2}\underbrace{011}_{3}\underbrace{100}_{4}
\end{array}$$

Each of the nine high-order bits is 0, so this is a legal positive exponent. The number 234 is sent to the 8-bit exponent field of the specified accumulator.

Example 2: LDEXP 201

```
                                    2        0  1
                          ______________ ___ ___
Exponent of 201           00000000    |  10000001
 200                    +        0    |  10000000
                        ----------------------------
                              | 1     |  00000001
                              |_______|
               overflow  <----
```

This is an illegal positive exponent. Notice that an overflow occurs when 200 is added to the exponent.

Example 3: LDEXP 100200

```
                                    2        0  0
                          ______________ ___ ___
Exponent of 100200        10000000    |  10000000
 200                    +             |  10000000
                        ----------------------------
                              | 1     |  00000000
                              |_______|
               underflow <----
```

This is an illegal negative exponent. Notice that when 200 is added to the exponent, a result is produced that is more negative than can be expressed by the 8-bit exponent field. Thus, an underflow occurs.

Example 4: Special Case – Exponent of 0: LDEXP 177600

```
Exponent of 177600

                          11111111    |  10000000
                        +        0    |  10000000
                        ----------------------------
                          00000000    |  00000000
```

This is the one case where the nine high-order bits are all equal, but the exponent is illegal. This is so because 177600 represents an exponent of 0. This exponent causes an underflow condition to exist; that is, it is treated as an illegal negative exponent.

4.3.12 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-precision mode is specified. If long-integer mode is specified, the number from memory is 32 bits and is converted to a single- or double-precision number, depending on whether floating- or double-precision mode is specified. The integer is loaded into bits <55:40> if short-integer mode is specified or into bits <55:24> if long-integer mode is specified. It is then left-shifted eight places so that bit 55 is transferred to bit 63 (see Figure 4-5).

The integer is then assigned an exponent of 217_8 in short-integer mode. This is the result of adding 200_8 to 17_8 (since the exponent is expressed in excess 200 notation), 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.

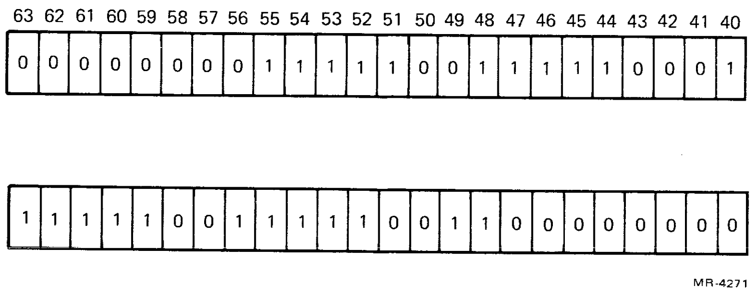


Figure 4-5 Shifting an Integer Left Eight Places

The FPF11 tests the 2's complement integer by examining if bit 63 is a positive or negative number. If it is positive, the number is normalized by left-shifting until bit 63 becomes a 1. If bit 63 is 1 (negative number), the integer is negative, the sign bit is set, the number is 2's complemented, and then normalized.

To normalize a number, bit 63 (MSB) of the fraction must be equal to 0 and bit 62 must be made equal to 1. 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 (see Figure 4-6).

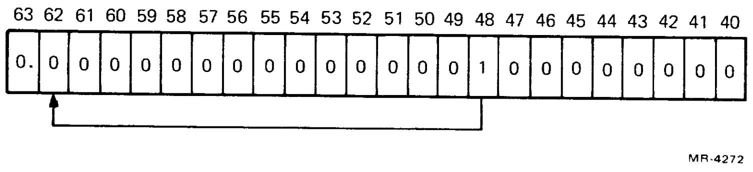


Figure 4-6 Example of a Normalized Integer

EXP =	21_8	Shift integer 15 places to the left to normalize it.
	-17_8	Bit 59 = 0, bit 58 = 1.
	200_8	Decrease the exponent by 15_{10}, which equals 17_8.

Loading a long integer with an FED = 0 and more than 24 significant digits causes the less significant digits to be truncated (with some loss of accuracy).

4.3.13 Store Exponent Instruction

The store exponent (STEXP) instruction causes the CPU to access a floating-point number in the FPF11, extract the 8-bit exponent field from this number, and subtract a constant of 200 (since the exponent is expressed in excess 200 notation). The exponent is then stored in the destination as a 16-bit, 2's complement, right-justified number with the sign of the exponent (bit 07) extended through the eight high-order bits.

The legal range of exponents is 0 to 377_8, expressed in excess 200 notation. This means that the number stored 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.

Two examples that illustrate the process follow. One uses an exponent greater than 200, the other an exponent less than 200.

Example 1: Exponent = 207 (See Figure 4-7.)

$$\begin{array}{ll} \text{Exponent of 207} & 10000111 \\ \text{2's complement of 200} & +10000000 \\ \hline \text{Result} = 7_8 & 00000\underbrace{111}_{7} \end{array}$$

sign bit

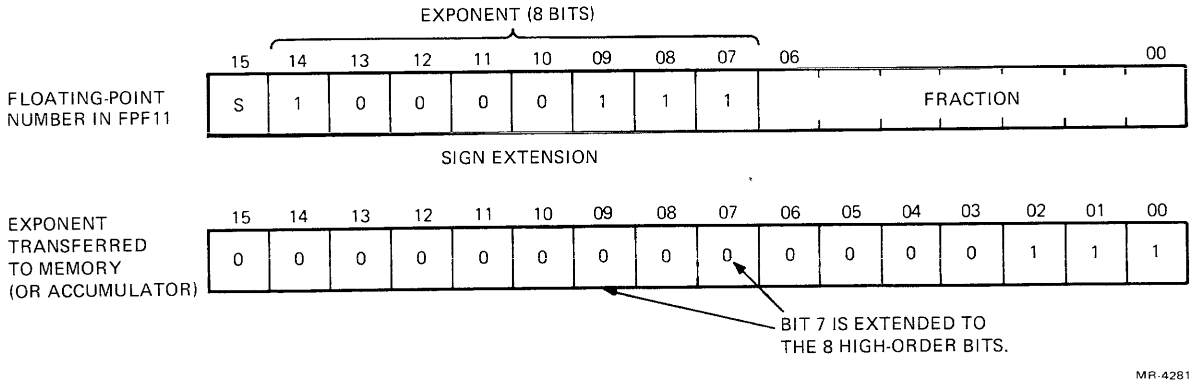


Figure 4-7 Store Exponent (Example 1)

Example 2: Exponent = 42 (See Figure 4-8.)

$$\begin{array}{ll} \text{Exponent of 42} & 00100010 \\ \text{2's complement of 200} & +10000000 \\ \hline \text{Result} = -42 & 10\underbrace{100}_{4}\underbrace{010}_{2} \end{array}$$

sign bit

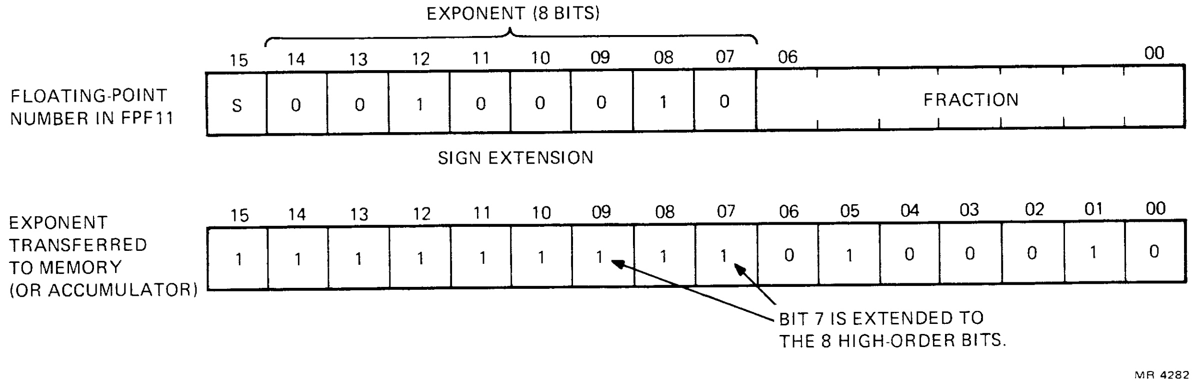


Figure 4-8 Store Exponent (Example 2)

4.3.14 Store Convert Floating-to-Integer Instruction

The store convert floating-to-integer instruction causes the CPU to take a floating-point number and convert it into an integer for transfer to a destination. The four classes of this instruction are as follows.

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

The (normalized) floating-point number to be converted is transferred to the floating-point processor (FPP). The FPP works with the sign bit and either of the following.

1. The 15 MSBs of the fraction for floating-to-integer and double-to-floating conversion.
2. The 31 MSBs of the fraction for double-to-long conversion.
3. The entire fraction for floating-to-long conversion.

The FPP subtracts 201 from the exponent to determine if the floating-point number is a fraction. If the result of the subtraction is negative, the exponent is less than 201, and the absolute value of the floating-point number is less than 1. When converted to an integer, the value of this number is 0; a conversion error occurs, the FZ bit is set, and 0s are sent to the destination. If the result of the subtraction is positive (or 0), the exponent is greater than (or equal to) 201, and the floating-point number can be converted to a nonzero integer (see Figure 4-9).

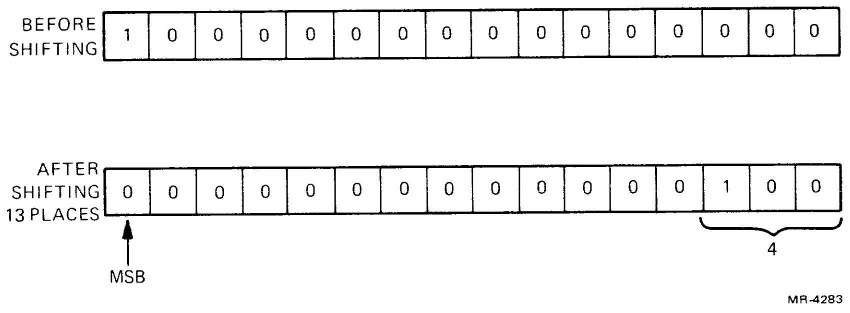


Figure 4-9 Example of a Store Convert Integer

The FPP makes a second test to determine if the floating-point number to be converted is within the range of numbers that can be represented by a 16-bit integer (I format) or 32-bit integer (L format). Consider the range of integers that can be represented in I and L formats and their floating-point equivalents. Refer to Table 4-3.

Table 4-3 I and L Formats and Their Floating-Point Equivalents

	I Format (16 bits)	Floating-point Equivalent	L Format (32 bits)	Floating-point Equivalent
Most Positive Integer	077777	$+.1111... \times 2^{15}$	17777777777	$+.1111... \times 2^{31}$
Least Positive Integer	000001	$+.100... \times 2^{1}$	00000000001	$+.100... \times 2^{1}$
Least Negative Integer	177777	$-.1111... \times 2^{16}$	37777777777	$-.1111... \times 2^{32}$
Most Negative Integer	100000	$-.1000... \times 2^{16}$	20000000000	$-.100... \times 2^{32}$

NOTE: MSB of integer = sign of integer.

Thus, the exponent of a positive floating-point number to be converted must be less than 16_{10} (220 in excess 200 notation) to convert to I format, or 32_{10} (240 in excess 200 notation) to convert to L format. The exponent of a negative number to be converted must be less than or equal to 16_{10} or 32_{10} to convert to I or L format, respectively.

The FPP tests whether the floating-point number to be converted is within the range of integers that can be represented in I or L format by subtracting a constant of 20_8 (for short integers) or 40_8 (for long integers) from the result of the first test. (Result of first test = biased exponent − 201_8 = unbiased exponent − 1.) If the result of the subtraction is positive or 0, the floating-point number is too large to be represented as an integer; a conversion error occurs and 0s are sent to the destination. If the result of the subtraction is a negative number other than −1, the floating-point number can be represented as an integer without causing an overflow condition. If the result of the subtraction is −1, the exponent of the floating-point number is either 220 (for short integers) or 240 (for long integers), and conversion proceeds. However, the floating-point number is within range only if it is negative and its fraction is .100... (that is, if it is the most negative integer; see Table 4-3 above). If, in this case, the number is not the most negative integer, it will be detected by a third conversion error test after conversion (see below).

To convert the fraction to an integer, the FPP shifts it right a number of places as specified by the following algorithms.

Short integer:

Number of right shifts = 20_8 + 201_8 − biased exponent − 1

Long integer:

Number of right shifts = 40_8 + 201_8 − biased exponent − 1

Regardless of the condition of the FT bit, the fractional part of the number is always truncated during this shifting process.

If the floating-point number is positive, the integer conversion is complete after shifting, and the number is transferred to the appropriate destination. However, if the floating-point number is negative, the number must be 2's complemented before it is sent to its destination.

After conversion, the FPP performs a third test for a conversion error by comparing the MSB of the (converted) integer with the sign bit of the original (unconverted) number. If the signs are not the same, a conversion error has occurred and the FPP traps if the FIC bit is set. This test is performed to detect a floating-point number with an exponent of 220 (for short integer) or 240 (for long integer) that has not been converted to the most negative integer.

Example 1: Store Convert Floating-to-Integer (STCFI)

Exponent = 203
Sign = 0
Fraction (24 bits) = .100000000000000000000000
15 MSBs of fraction = .100000000000000
203 (excess 200) = 2
Fraction = 1/2 Integer to be stored = $1/2 \times 2 = 4$

1. Test 1: Is the number to be converted a fraction?

Exponent:	203_8 -201	
No	2	Since this result is positive, the given floating-point number is not a fraction and conversion may proceed without error.

2. Test 2: Is the floating-point number to be converted within range? (We are working with a positive short integer.)

Result of Test 1:	2 -20	
Yes	-16	Indicates that the number to be converted is within range and can be represented as a 16-bit integer. No conversion error occurs.

How many right shifts? Use algorithm:

$$20_8 + 201_8 - 203_8 - 1 = 20_8 - 3_8 = 15_8 = 13_{10}$$

$$= 13 \text{ right shifts}$$

This example involves a positive number, so conversion is complete after 13 right shifts. If the number had been negative, the integer would have been 2's complemented.

3. Test 3: The MSB of the converted integer and the sign bit of the original floating-point number are compared. Since they are equal, no conversion error occurs.

Example 2: Store Convert Floating-to-Integer (STCDL)

Exponent = 240_8
Sign = 0
31 MSBs of fraction = .1000000000000000000000000000000

1. Test 1: Is the number to be converted a fraction?

   Exponent: 240_8 -201

   No 37_8 Since this result is positive, the given floating-point number is not a fraction, and conversion may proceed (i.e., no conversion error occurs).

2. Test 2: Is the floating-point number to be converted within range? (We are working with a positive long integer.)

   Result of Test 1: 37 -40

   -1 We know the number is out of range by examining the sign bit (in fact, this number is one greater than the most positive integer that can be represented). However, the FPP does not know this yet, and conversion proceeds without error at this point.

   How many right shifts? Use algorithm:

   $40_8 + 201_8 - 240_8 - 1 = 0$

   = No right shifts

   Converted 32-bit integer = 20000000000_8

   Since the number is positive, conversion is now complete (i.e., no need for 2's complementing).

3. Test 3: The most significant bit of the converted integer (which is 1) and the sign bit of the original floating-point number (which is 0) are compared. Since they are not equal, a conversion error occurs, which we predicted in Step 2.

4.3.15 Load FPF11's Program Status (LDFPS) Instruction
The load FPF11's program status (LDFPS) instruction causes the FPP to transfer 16 bits from the location specified by the source to the floating-point status (FPS) register. These 16 bits contain status information the FPF11 uses to enable and disable interrupts, set and clear mode bits, and set condition codes. (See Paragraph 3.4.)

4.3.16 Store FPF11's Program Status (STFPS) Instruction
The store FPF11's program status (STFPS) instruction causes the FPP to transfer the 16 bits of the FPS register to the specified destination.

4.3.17 Store FPF11's Status (STST) Instruction
The store FPF11's status (STST) instruction causes the FPP to read the contents of the floating exception code (FEC) and floating exception address (FEA) registers when a floating-point exception (error) occurs.

If mode 0 addressing is enabled, only the FEC is sent to the destination accumulator. If mode 0 addressing is not enabled, the FEC is stored in memory, followed by the FEA. In memory, the FEC data occupies all 16 bits of its memory location, while the FEA data occupies only the lower 4 bits of its location.

When an error occurs and the interrupt trap in the CPU is enabled, the CPU traps to interrupt vector 244_8. The user should issue the STST instruction to determine the type of error that has occurred.

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

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

4.3.19 Set Floating Mode (SETF) Instruction
The set floating mode (SETF) instruction causes the FPP to clear the FD bit (bit 07 of the FPS register) and indicate single-precision operation.

4.3.20 Set Double Mode (SETD) Instruction
The set double mode (SETD) instruction causes the FPP to set the FD bit (bit 07 of the FPS register) and indicate double-precision operation.

4.3.21 Set Integer Mode (SETI) Instruction
The set integer mode (SETI) instruction causes the FPP to clear the IL bit (bit 06 of the FPS) and indicate that short-integer mode (16 bits) is specified.

4.3.22 Set Long-Integer Mode (SETL) Instruction
The set long-integer mode (SETL) instruction causes the FPP to set the IL bit (bit 06 of the FPS) and indicate that long-integer mode (32 bits) is specified.

4.4 FPF11 PROGRAMMING EXAMPLES
What follows are two programming examples that use the FPF11 instruction set. In Example 1, A is added to B, D is subtracted from C, the quantity (A + B) is multiplied by (C − D), the product of this multiplication is divided by X, and the result is stored. Example 2 calculates $DX^3 + CX^2 + BX + A$, which involves a 3-pass loop.

Example 1: [(A + B) * (C − D)] * X

```
SET F
LDF     A,AC0       ;LOAD AC0 FROM A
ADDF    B,AC0       ;AC0 HAS (A + B)
LDF     C,AC1       ;LOAD AC1 FROM C
SUBF    D,AC1       ;AC1 HAS (C − D)
MULF    AC1,AC0     ;AC0 HAS (A + D) * (C − D)
DIVF    X,AC0       ;AC0 HAS (A + D) * (C − D)/X
STF     AC0,Y       ;STORE (A + D) * (C − D)/X IN Y
```

Example 2: $DX^3 + CX^2 + BX + A$

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

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

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

```
                SET F
                MOV #3,%0        ;SET UP LOOP COUNTER
                MOV #D+4,%1      ;SET UP POINTER TO COEFFICIENTS
                LDF (6)+,AC1     ;POP X FROM STACK
                CLRF AC0         ;CLEAR OUT AC0
        LOOP;   ADDF -(4),AC0    ;ADD NEXT COEFFICIENT
                                 ;TO PARTIAL RESULT
                MULF AC1,AC0     ;MULTIPLY PARTIAL RESULT BY X
                SOB %0,LOOP      ;DO LOOP 3 TIMES
                ADDF -(4),AC0    ;ADD X TO GET RESULT
                STF AC0,-(6)     ;PUSH RESULT ON STACK
```

CHAPTER 5
FUNCTIONAL DESCRIPTION

5.1 GENERAL

The functions of the FPF11 floating-point processor are presented in this chapter. First examined, in an overview, is the passing of controls at the interface between the CPU and FPF11. Later a discussion on the control and data functions within the FPF11 provides a basic understanding of how the FPF11 works.

5.2 OVERVIEW

The CPU contains two internal buses, the microinstruction bus (MIB) and the data/address lines (DALs) as illustrated in Figure 5-1. The MIB <15:00> is the control bus, which is used to carry the 16-bit CPU microinstructions. The DAL <15:00> is the data bus, which is used to carry data, addresses and PDP-11 instructions. The FPF11 is connected to these buses by a ribbon cable.

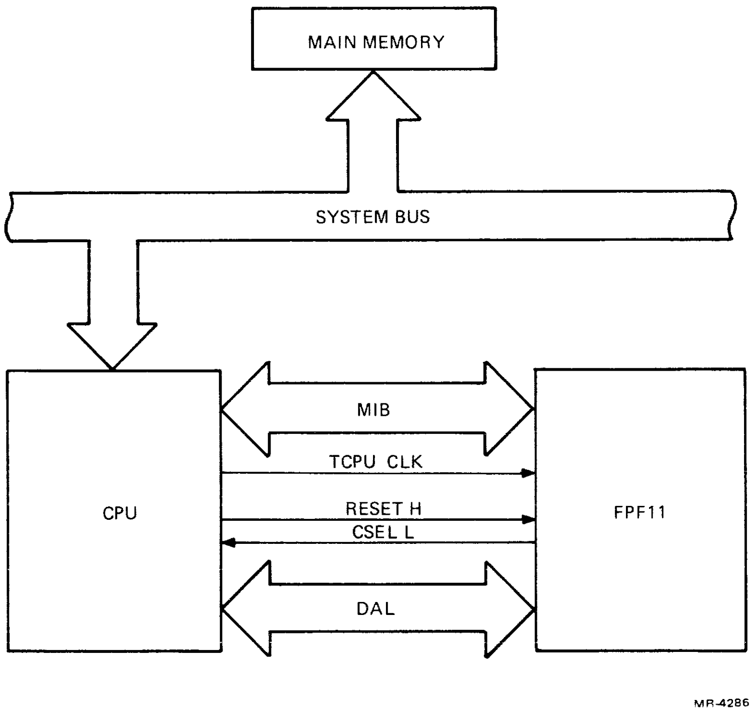


Figure 5-1 FPF11/CPU Interface

The CPU executes PDP-11 code by fetching an instruction and its operands (if any) from memory, performing the operation, and storing the result (if any). It accomplishes this by executing CPU microinstructions. While the CPU is thus involved, the FPF11 monitors the CPU microinstruction flow at its MIB interface (see Paragraph 5.3.2). When the FPF11 decodes the microinstructions to be an instruction fetch (IFETCH), it knows the next piece of information on the DAL is a PDP-11 instruction and it inputs this instruction into its floating-point instruction register (FPIR) in parallel with the CPU. The CPU uses the next two microcycles to examine the instruction. During this interval, the FPF11 sets up to process the instruction in case it turns out to be in the floating-point class (contains an op code of 17XXXX).

At the end of the two microcycles, the CPU microinstruction on the MIB tells the FPF11 whether or not it is in the floating-point class. If it is not, the FPF11 goes back to monitoring the CPU and awaits the next IFETCH. If it is a floating-point instruction, the CPU passes control to the FPF11, which then becomes the microinstruction source on the MIB. At this point the FPF11 has control over the CPU.

The FPF11 issues CPU microinstructions to move operands between main memory and the FPF11 as dictated by the PDP-11 floating-point instruction in the FPIR. The operands are passed on the DAL data bus.

During those times when no CPU actions are required, the CPU receives a no operation (NOP) microinstruction. This occurs while the FPF11 operates on floating-point data. The FPF11 may run at its own clock rate while the CPU receives NOPs at the CPU's clock rate. Resynchronization occurs prior to the returning of control to the CPU, that is, upon successful completion of a PDP-11 floating-point instruction. Control is passed back to the CPU by entering the CPU's SERVICE routine. This causes the FPF11 to return to its monitoring of CPU microinstructions.

If the completion of a floating-point instruction is unsuccessful, control is passed back to the CPU by entering the CPU's TRAP HANDLER microroutine. The trap vector associated with floating-point processor errors (244_8) is sent to the CPU on the MIB. The FPF11 then returns to monitoring CPU microinstructions. The CPU handles the trap as directed by the interrupt service routine at 244_8. The FPF11 stores the error code and address associated with the failing PDP-11 floating-point instruction in its floating error code (FEC) and floating error address (FEA) registers, respectively. Software may use this information to recover from the error.

5.3 FPF11 CONTROL AND DATA FLOW DESCRIPTION

Figure 5-2 is a functional diagram of the FPF11. Similar to the CPU, the FPF11 has both a control path and a data path. The control store outputs form the control path. (The control word is 104 bits wide.)

Data, addresses, and PDP-11 floating-point instructions are passed on the TBus <15:00>, which is the FPF11 internal bus.

Control functions are performed by the sequencer, the control store, the MIB interface, and the interface control and clock logic. Data handling functions are performed by the DAL interface, TBus resources, and the microprocessor data path.

5.3.1 Microcontroller: Sequencer and Control Store

The sequencer and the control store together form the FPF11 microcontroller, shown in Figure 5-3. The microcontroller directs all FPF11 activity by conditionally sequencing through the microcode in the control store.

The sequencer makes the decisions that control microprogram execution. It consists of branch and IR decode programmable logic arrays (PLAs) and the microprogram counter (MPC) logic. Information received from the TBus resources, MIB interface, microprocessor data path, interface control and clock

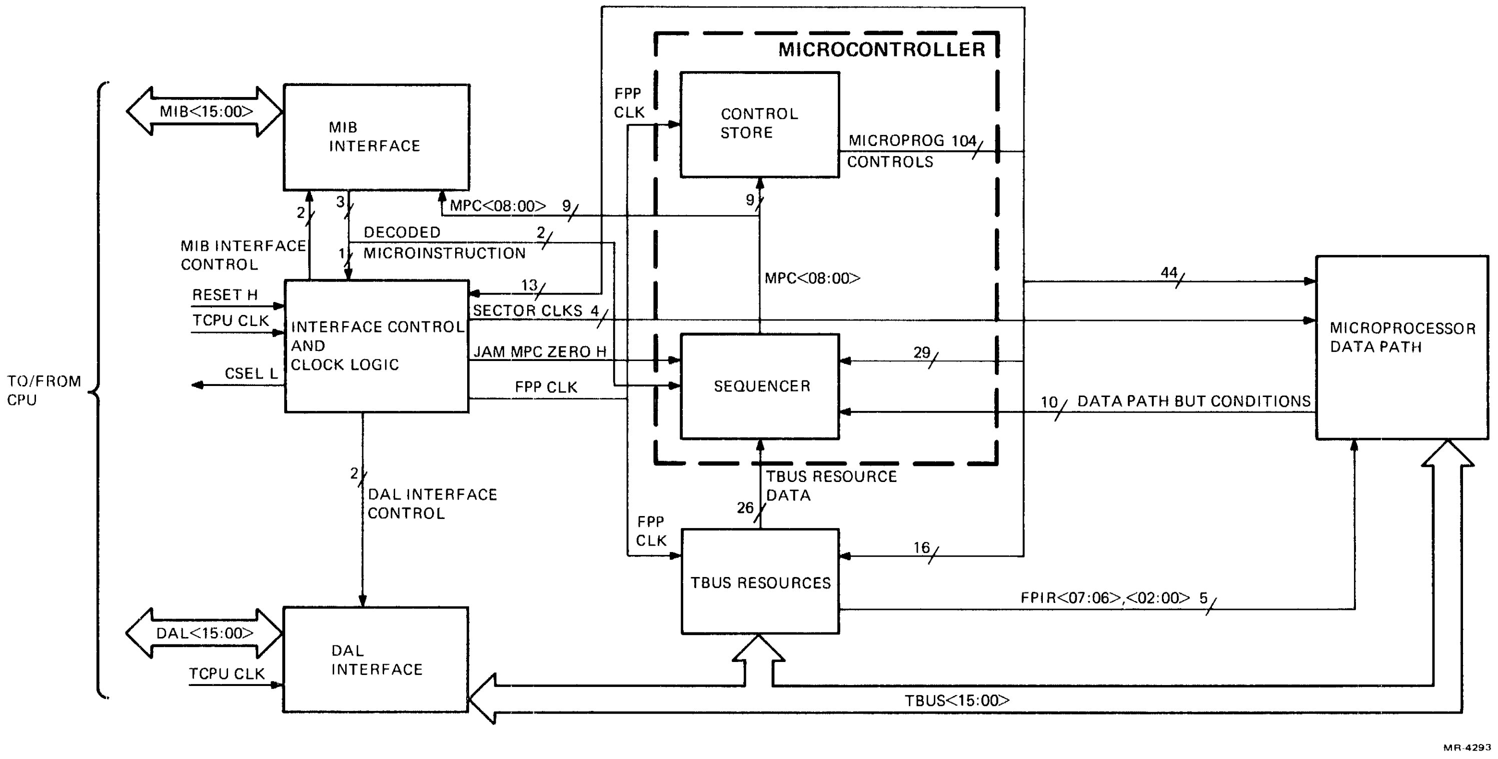
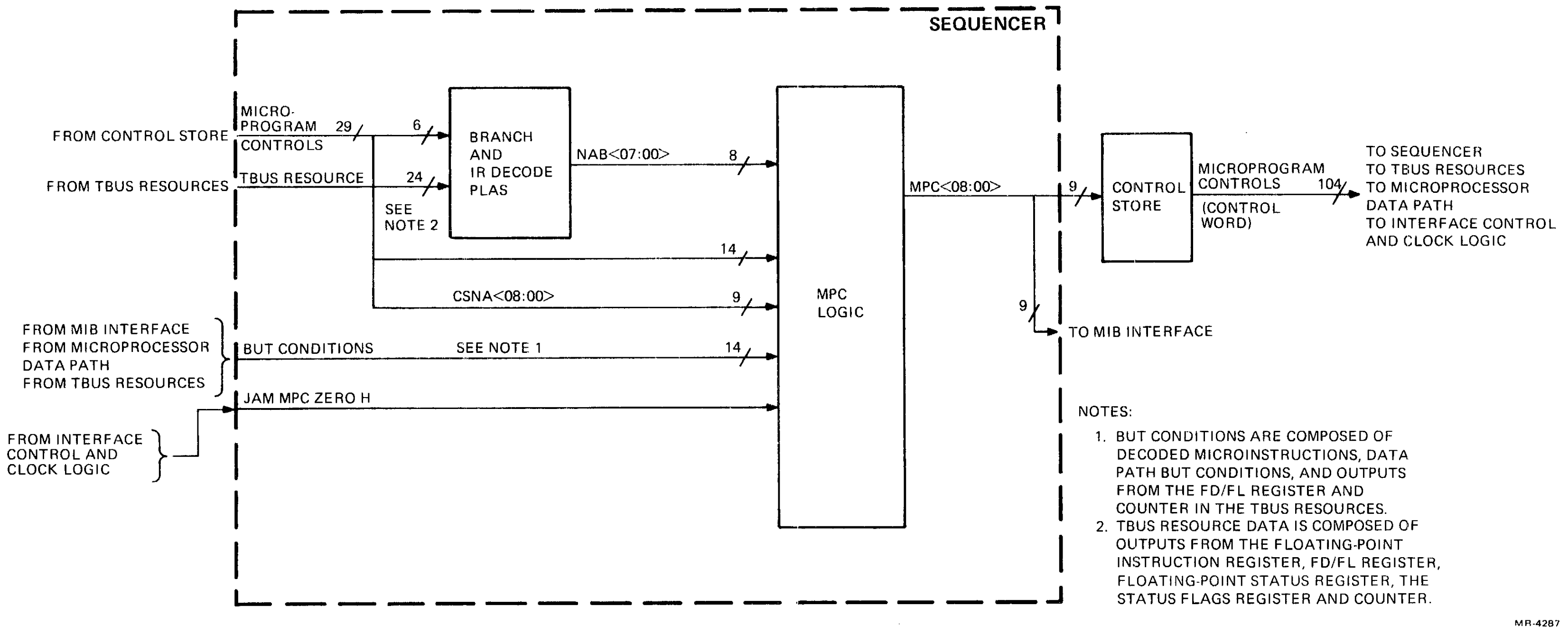


Figure 5-2 FPF11 Floating-Point Processor, Functional Block Diagram



NOTES:

1. BUT CONDITIONS ARE COMPOSED OF DECODED MICROINSTRUCTIONS, DATA PATH BUT CONDITIONS, AND OUTPUTS FROM THE FD/FL REGISTER AND COUNTER IN THE TBUS RESOURCES.
2. TBUS RESOURCE DATA IS COMPOSED OF OUTPUTS FROM THE FLOATING-POINT INSTRUCTION REGISTER, FD/FL REGISTER, FLOATING-POINT STATUS REGISTER, THE STATUS FLAGS REGISTER AND COUNTER.

Figure 5-3 Microcontroller (Sequencer and Control Store), Functional Block Diagram

logic, and bits of the previous control word cause the sequencer to output microaddresses on MPC <08:00>. These microaddresses are sent to the control store to select microinstructions for FPF11 control and to the MIB interface to select CPU microinstructions.

The sequencer uses four types of data:

- Control store next address (CSNA), from the control store. This address gives the sequencer its base microaddress for the next microword.

- Branch micro test (BUT) conditions, from the microprocessor data path (see Paragraph 5.3.6), counter (see Paragraph 5.3.5.6), FD/FL register (see Paragraph 5.3.5.2), and MIB interface (see Paragraph 5.3.2). These signals allow the sequencer to modify the CSNA, this giving the microcontroller its ability to make decisions based on these BUT conditions. The BUT conditions are individually controlled by the BUT field of the control word so that various combinations of conditions can be tested.

- Next address bits (NABs), from the branch and IR decode PLAs. These bits are similar to the BUT conditions except that the branch and IR decode PLAs are used to test for classes of conditions; for example, op code type. Further, they provide more flexibility in modifying the microaddress to the control store.

- JAM MPC ZERO, from the interface control and clock logic. This signal resets the microcode to the CPU monitoring sequence in the event of a CPU RESET or a failure to detect the presence of a PDP-11 floating-point instruction decode.

The control store logic is made up of 13 512 × 8-bit ROMs, which make up the control word (see Figure 5-4 and Table 5-1). In addition, two more ROMs are used to hold the CPU microinstruction. Thus, the microcontroller directs the execution of the current PDP-11 floating-point instruction.

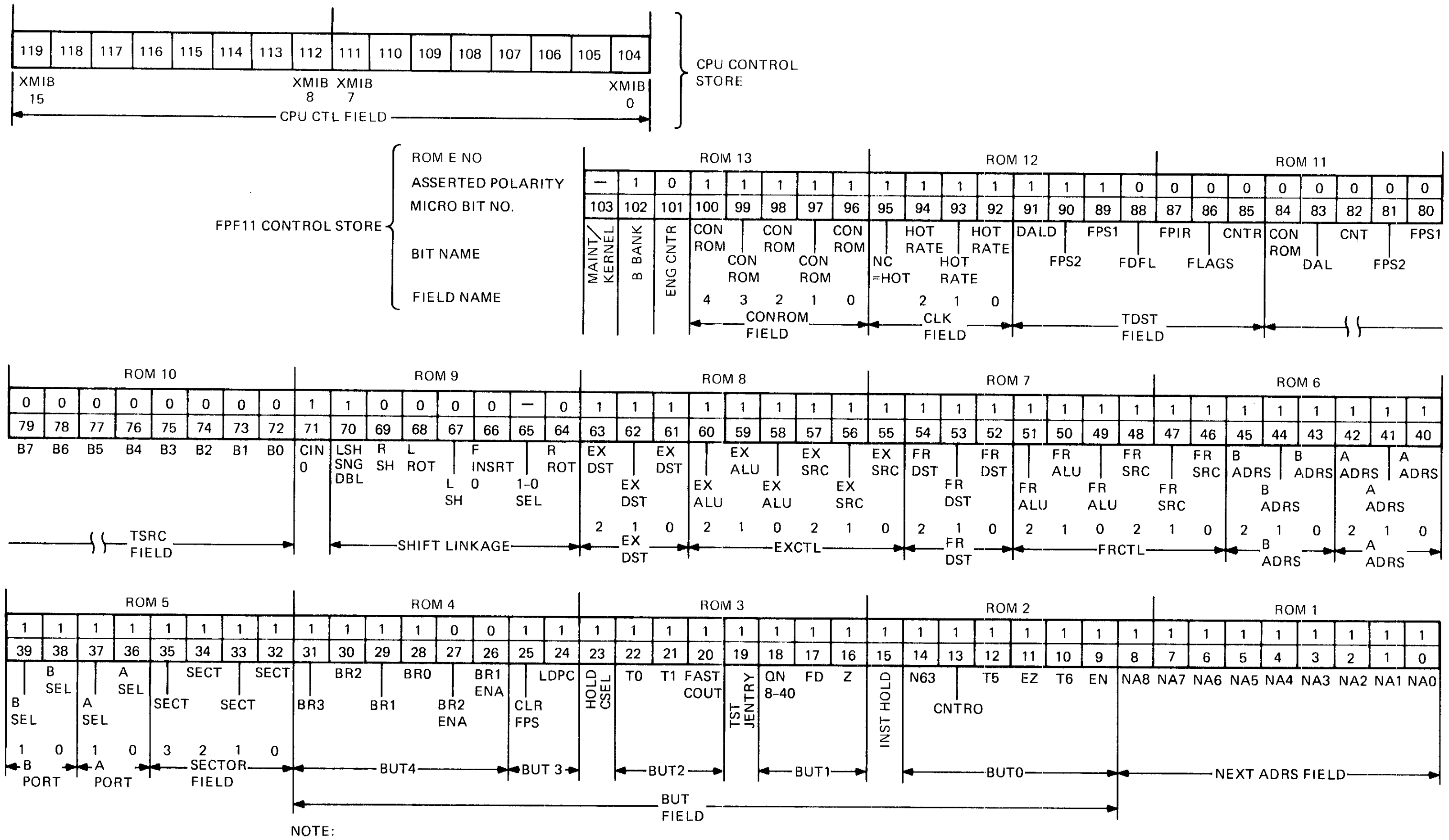
5.3.2 MIB Interface
The FPF11 uses the MIB interface to either monitor or control the CPU, as shown in Figure 5-5.

5.3.2.1 Receiving Logic (from the CPU) - While the CPU is executing PDP-11 instructions, this logic monitors the MIB, waiting to decode the CPU microinstruction associated with IFETCH. This decoded microinstruction is sent to the sequencer. This logic also decodes the CPU microinstruction, which indicates the start of a PDP-11 floating-point instruction.

5.3.2.2 Transmitting Logic (to the CPU) - While the FPF11 is executing PDP-11 floating-point instructions, the microinstruction ROMs located here are addressed by MPC <08:00> from the sequencer. This causes the microinstruction ROMs to output CPU microinstructions on the MIB <15:00> and makes the CPU available to the FPF11 for use in accessing main memory.

The direction of information flow out of the MIB interface is controlled by the microcontroller (see Figure 5-3) and timed by the interface control and clock logic (see Figure 5-6).

5.3.3 Interface Control and Clock Logic
The interface control and clock logic is responsible for maintaining proper timing between the FPF11 and the CPU, for both data (DAL) and control (MIB). (See Figure 5-6.) This logic provides the CPU with the signal that allows the FPF11 to be the CPU microinstruction source on the CSEL line (MIB), and also receives the RESET line from the CPU. In addition, internal FPF11 processing is timed here.



NOTE:
(ON TBUS BUT CONDITIONS)
T0 = Y8
T1 = Y9
T5 = Y61
T6 = Y62

Figure 5-4 FPF11 Control Word

Table 5-1 Control Word Bit Descriptions

Bits	Field	Description
00:08	Next Address	Provides the base address for the next control word.
09:14	BUT0	Enables for branch micro test conditions.
15	Instruction Hold	Controls the latching of MPC <08:00> into microinstruction ROMs at the MIB interface.
16:18	BUT1	Enables for branch micro test conditions.
19	Test JENTRY	Controls the test for passing MIB mastership from the base CPU to the FPF11.
20:22	BUT2	Enables for branch micro test conditions.
23	Hold CSEL	Controls the assertion of FPF11 "CSEL" (chip select) to indicate FPF11 MIB mastership.
24:31	BUT4	Enables for branch micro test conditions.
32:35	Sector Control	Enables for the sector clocks.
36:37	A Port	Enables for A-port addresses.
38:39	B Port	Enables for B-port addresses.
40:42	A Address	Selects address for the A port.
43:45	B Address	Selects address for the B port.
46:51	Fraction Control	Selects the data source and logical function to be performed in the ALU.
52:54	Fraction Destination	Selects the ALU data destination.
55:60	Exponent Control	Selects the data source and logical function to be performed in the ALU.
61:63	Exponent Destination	Selects the ALU data destination.
64:70	Shift Linkage	Controls for the shift linkage logic.
71	Carry In	Control for the carry logic.
72:84	TBus Source	Controls for placing information on the TBus.
85:91	TBus Destination	Controls for removing information from the TBus.
92:95	Clock Control	Controls for the internal clock rate.
96:100	Constant ROM Control	Selects the proper constants from the constant ROMs.
101	Enable Counter	Enables the counter to count.
102	B Bank	Selects high or low bank of the B-port address at 2901s.
103	Maintenance/Kernel	Spare for DIGITAL's use. (Future microcode BUT condition.)

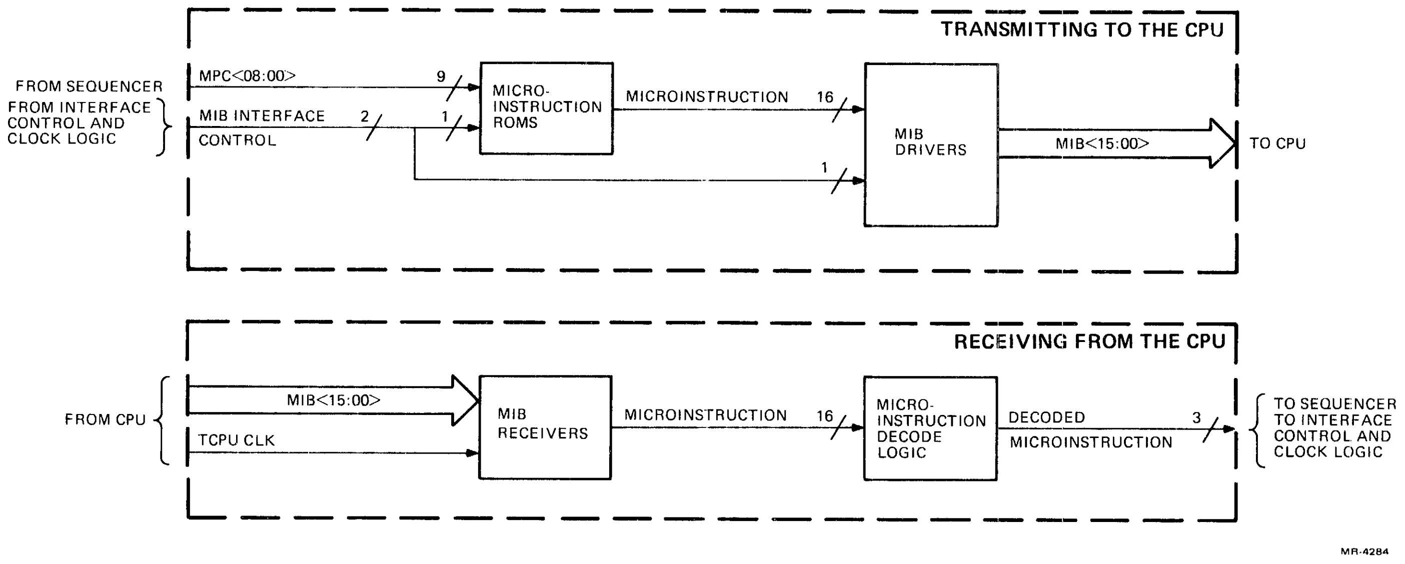


Figure 5-5 MIB Interface, Functional Diagram

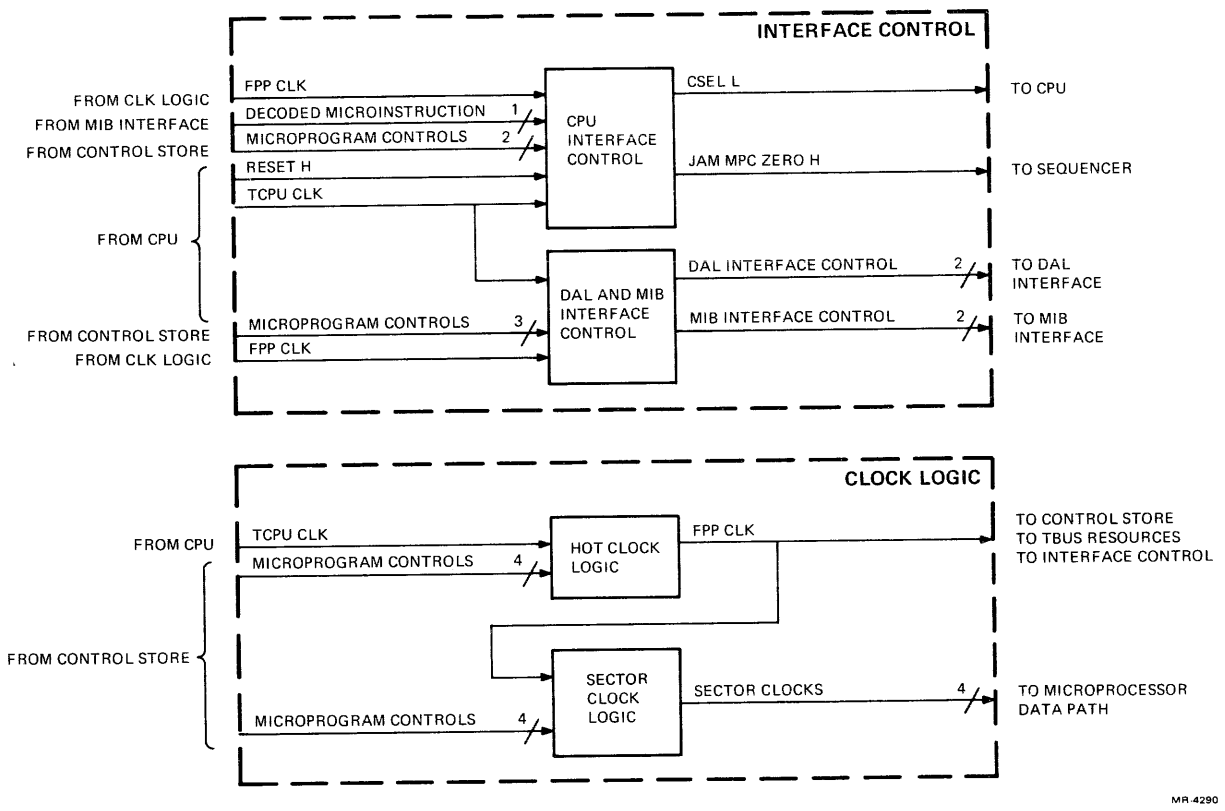


Figure 5-6 Interface Control and Clock Logic, Functional Diagram

5.3.3.1 Interface Control Logic – When decoded microinstructions from the MIB interface inform the interface control logic that a floating-point instruction has been decoded by the CPU, the signal CSEL L is sent to the CPU to inform it that the FPF11 is taking control. Controls or data entering or leaving the FPF11 at the MIB interface or DAL interface are timed by signals from this logic. When the CPU issues a RESET or when the test for a floating-point instruction decode from the CPU fails, the FPF11 microcode is reset to the CPU monitoring sequence. This is accomplished by the signal JAM MPC ZERO H from this logic to the sequencer.

5.3.3.2 Clock Logic – Normally, the FPF11 runs at the speed of the CPU. However, during certain arithmetic operations, the clock logic generates a faster clock (HOT clock) for timing FPF11 operations while the MIB interface runs at the speed of the CPU. The clock is controlled by the microcontroller and is used to run each FPF11 microcycle at its fastest possible rate.

The clock logic is also responsible for resynchronizing the FPF11 with the CPU prior to returning control to it. In addition, four clocks are generated in the sector clock logic. One of four sectors (sectors 0, 1, 2, and 3 CLK L) are selected. A sector is a 16-bit slice of the microprocessor data path (see Paragraph 5.3.6). Each sector clock causes data to be loaded into a random access memory (RAM), or a Q register, in its respective sector.

5.3.4 DAL Interface

The DAL interface buffers the information received from the CPU's DAL and transfers the information from the TBus onto the DAL for use by the CPU or storage in main memory. (See Figure 5-7.) Information received is in the form of operands and a PDP-11 floating-point instruction. Information passed back to the CPU consists of processed operands and associated addresses and error codes. The direction of information flow is controlled by the microcontroller and timed by the interface control and clock logic.

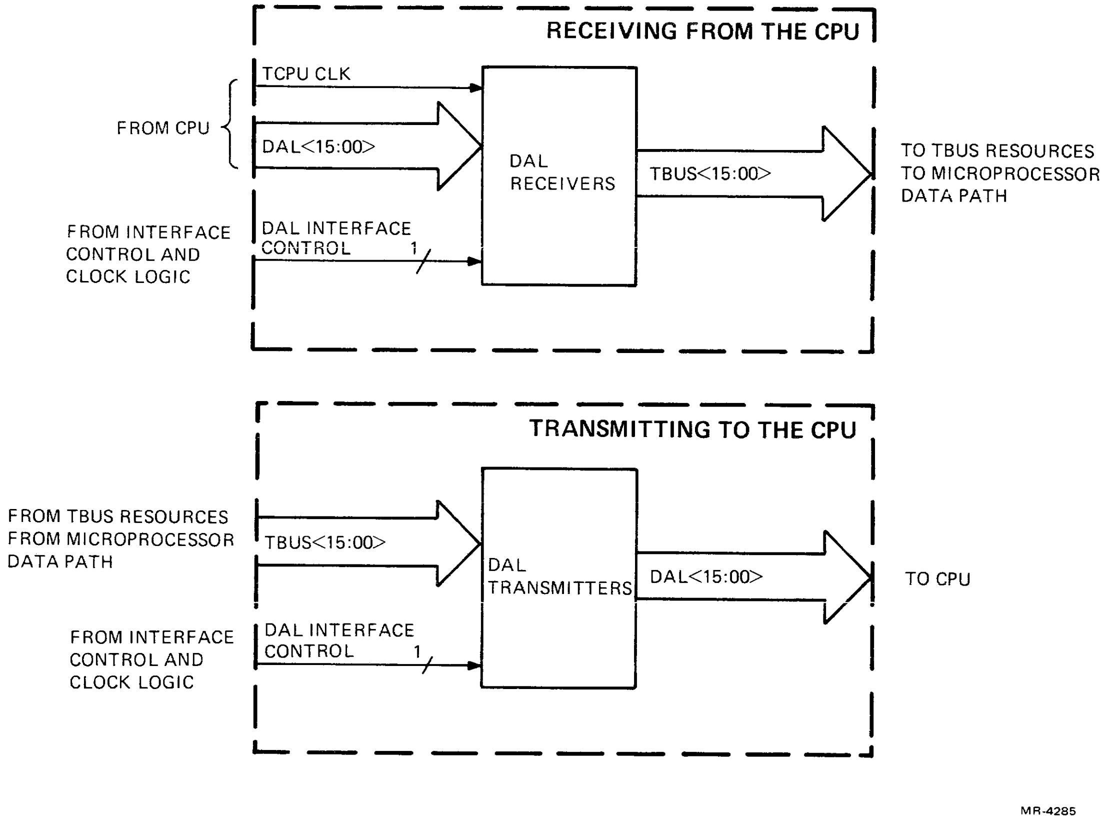


Figure 5-7 DAL Interface, Functional Diagram

5.3.5 TBus Resources

The TBus resources supply the sequencer and the microprocessor data path with information necessary for data and control processing. (See Figure 5-8.) The TBus resources are five registers, a counter and pair of constant ROMs. TBus resource data is passed to/from the TBus under the direction of the microcontroller.

5.3.5.1 Floating-Point Instruction Register (FPIR) - The floating-point instruction register (12 bits) receives the floating-point instruction from the CPU by way of the TBus and DAL interface. FPIR <11:00>, which contain op code, accumulator, source/destination, and floating source/floating destination information, are sent to the sequencer. The sequencer examines these bits at various times dictated by the microcontroller during the execution of a PDP-11 floating-point instruction.

In addition, FPIR bits <02:00>, part of the source/destination field, and FPIR bits <07:06>, the accumulator field, go to the address multiplexer in the microprocessor data path. These signals are used to select the RAM scratch-pad registers in the microprocessor data path. Bits <15:12>, part of the op code, of the floating-point instruction always contain 1s (op code = 17_8) and therefore are not stored.

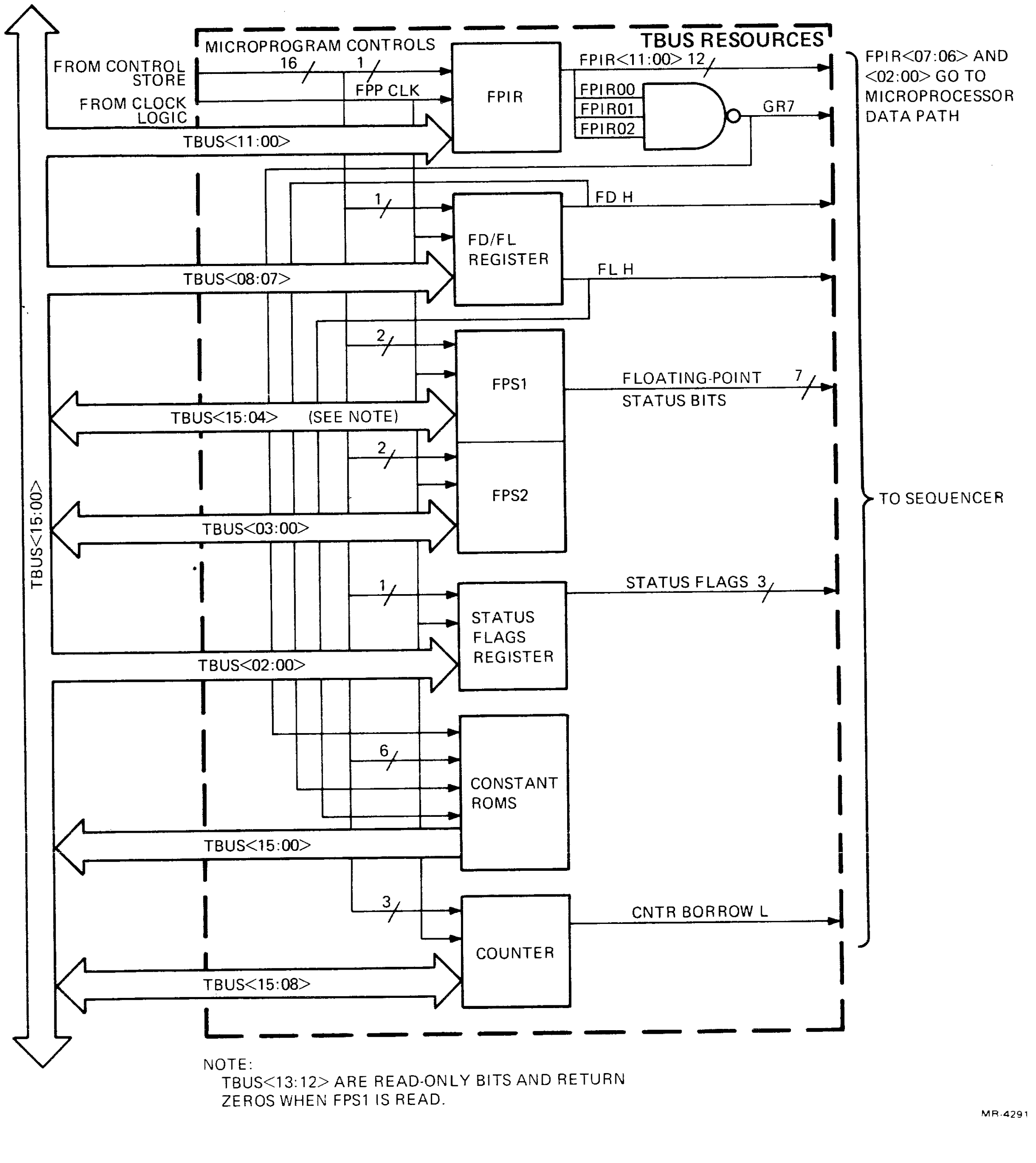


Figure 5-8 TBus Resources, Functional Diagram

5.3.5.2 FD/FL Register – The FD/FL register (2 bits) designates whether the mode is single- or double-precision floating-point format (FD), or single- or double-precision integer format (FL). FD/FL bits are modified during the execution of certain floating-point instructions.

This register is used as a scratch area for the FD/FL bits so that the original copy, stored in FPS bits <07:06>, will not be lost. The signals FD H and FL H are sent to the sequencer, which tests them in order to control microprogram flow. These signals also go to the constant ROMs to enable the output of the proper constants in single- or double-precision and short- and long-integer operations.

5.3.5.3 Floating-Point Status (FPS) Register – The floating-point status register consists of two segments, FPS1 and FPS2, which can receive or drive the TBus. Two segments are used so that FPS2 can be changed without affecting FPS1. The microcontroller tests the bits in FPS1 in order to control microprogram flow. (The FPS format and bits are described in Chapter 3.)

Floating-point condition codes are loaded into the 4-bit FPS2. These codes reflect the condition of the last arithmetic operation (that affected condition codes) performed in the microprocessor data path.

5.3.5.4 Status Flags Register – The microcode stores special microcode conditions as status bits in the status flags register. The microcontroller uses these bits to control microprogram flow. They allow a limited amount of microcode sharing to occur, similar to subroutining.

5.3.5.5 Constant ROMs – Two 256 $\times$ 8-bit constant ROMs contain fixed-value numbers required for certain floating-point functions. The magnitude of some of the constants depends on whether the floating-point numbers are single- or double-precision and short- or long-integer. This is why, as discussed earlier, FD/FL signals are used as ROM-gating signals. The ROMs are addressed by the microcontroller.

As an example, the constants include such items as the number 200_8, which must be added to or subtracted from the exponent during arithmetic operations. (Exponents are stored in excess 200 notation.) The constants are output to the TBus for use in the microprocessor data path.

5.3.5.6 Counter – Data is loaded into the counter from the TBus. The counter is used as a loop counter for microcode iterations, such as the multiply microroutine. The sequencer uses the output CNTR BORROW L to control microprogram flow. The counter outputs are also available on the TBus for use in the microprocessor data path.

5.3.6 Microprocessor Data Path Logic

It is in the microprocessor data path that the floating-point arithmetic operations and data manipulations take place. This logic consists of 16 AM2901 bipolar microprocessors and support circuitry. The support circuitry consists of fast carry logic, shift linkage logic, and address multiplexers for the scratch-pad registers. (See Figure 5-9.)

Operands are moved to and from an AM2901 bipolar microprocessor in 16-bit words by way of the TBus. The microcontroller controls the location of the operands to their designated places in a scratch-pad register. Once the operands are loaded, they are manipulated by the microcontroller to arrive at resulting operands. The resulting operands are stored in the scratch-pad register until they are moved out onto the TBus.

5.3.6.1 Address Multiplexers – It is the function of the address multiplexers (see Figure 5-9) to supply the A- and B-port addresses to a RAM. The addresses are selected from data input from the FPIR and A-address/B-address fields of the control word, and controlled by the A-port/B-port fields of the control word.

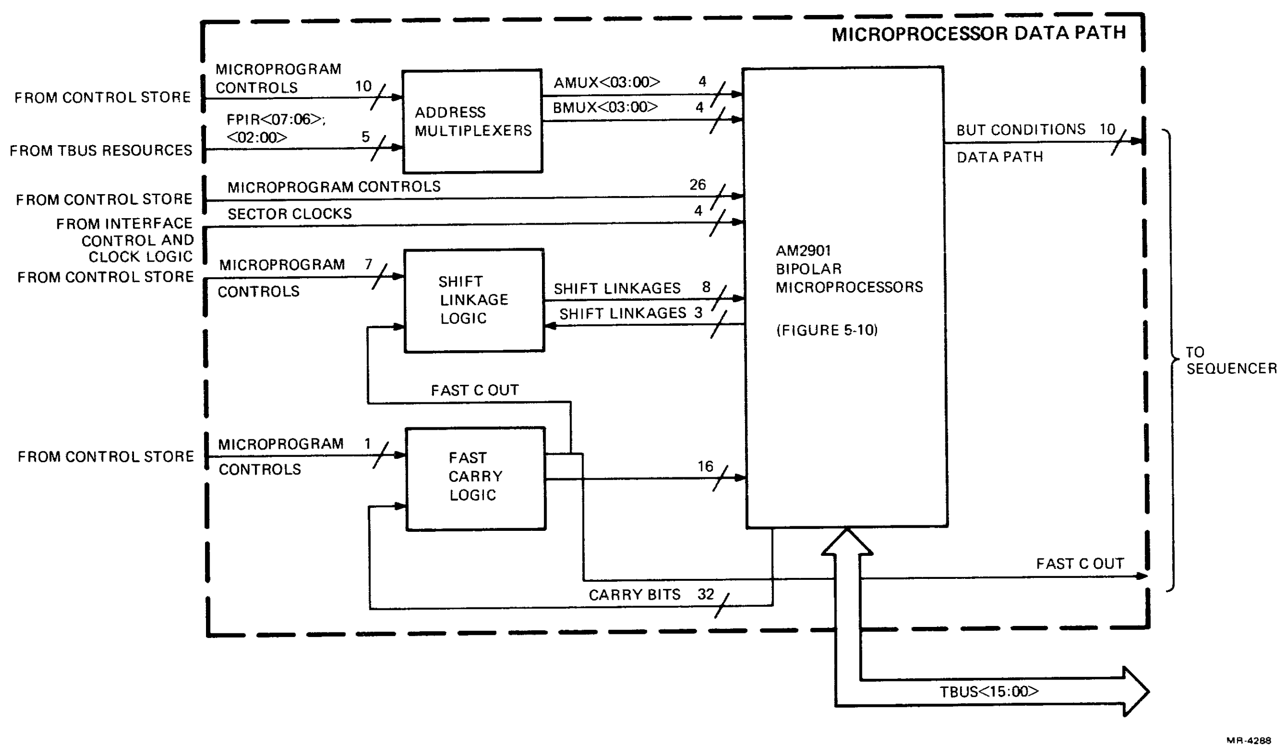


Figure 5-9 Microprocessor Data Path, Functional Diagram

5.3.6.2 Shift Linkage Logic – The shift linkage logic (see Figure 5-9) provides the data path for shift or rotate operations. It allows special shift operations, as well as the basic shift and rotate functions for the RAM, and the shift function for the Q register. (The Q register has no rotate function.) The RAM shift functions include provision for rotating an entire 64-bit input word for initial setup and for inserting 1s and 0s into the bit stream as required during a rotate or shift operation. A shift function performed on the Q register also causes a corresponding shift in the RAM. The RAM may be shifted by itself but the Q register may not. In addition, the shift linkage logic puts the carry bit (FAST C OUT) in its proper place.

5.3.6.3 Fast Carry Logic – The fast carry logic (see Figure 5-9) provides the FPF11 the ability to do two levels of carry look-ahead for maximum performance in addition and subtraction. This allows the FPF11 to function with full 64-bit look-ahead carry generation.

5.3.6.4 AM2901 Bipolar Microprocessor – Operands are moved into the AM2901 bipolar microprocessor from the TBus and stored in the RAM register file. (See Figure 5-10. Also, see Figure 5-11 for the RAM register file layout.) The RAM register file is the scratch-pad area where the results of arithmetic and logical operations are temporarily stored. As directed by the microcode, data loaded into the RAM (see Figure 5-10) may be shifted left or right, or remain unshifted. The dual-port RAM (A-port, B-port) consists of 16 64-bit words (each of the 16 AM2901s contains a 16 $\times$ 4-bit RAM).

Six of the 64-bit registers are allocated for the accumulators and are accessible to the programmer by way of the FPF11 instruction register. Registers 6 and 7 are unused, while registers 10–17 are set aside for special functions. (Registers 10–17 are accessed only by the microcontroller, with registers 10–14 constituting a working storage area for the FPF11 microcode.) Other sections of the RAM register file contain a temporary copy of the floating-point status register (FPS TEMP) and the condition codes (FCCR). In addition, data containing the floating error address (FEA, FEA TEMP), floating error code (FEC), and special working registers (FWR16, FWR17) are implemented in locations 15, 16, and 17. The contents of the RAM are either read into the arithmetic logic unit (ALU) or passed directly to the TBus under microprogram control.

The ALU is the data path component that performs the arithmetic/logical operation under command of the microcode. ALU output data may be routed to the Q register or RAM, or may be multiplexed with the RAM output. The ALU function control fields in the microword (EX CTL, FR CTL – refer to Table 5-1) determine the data source and arithmetic or logical function to be performed. The ALU destination field (EX DST, FR DST – refer to Table 5-1) determines which of the indicated registers is to receive the data or if the data is to be output on the TBus.

The Q register is used during multiply and divide operations to store multiplier or product operators. Its contents may be shifted left or right, or remain unshifted. The register may route data to the ALU or receive input from it.

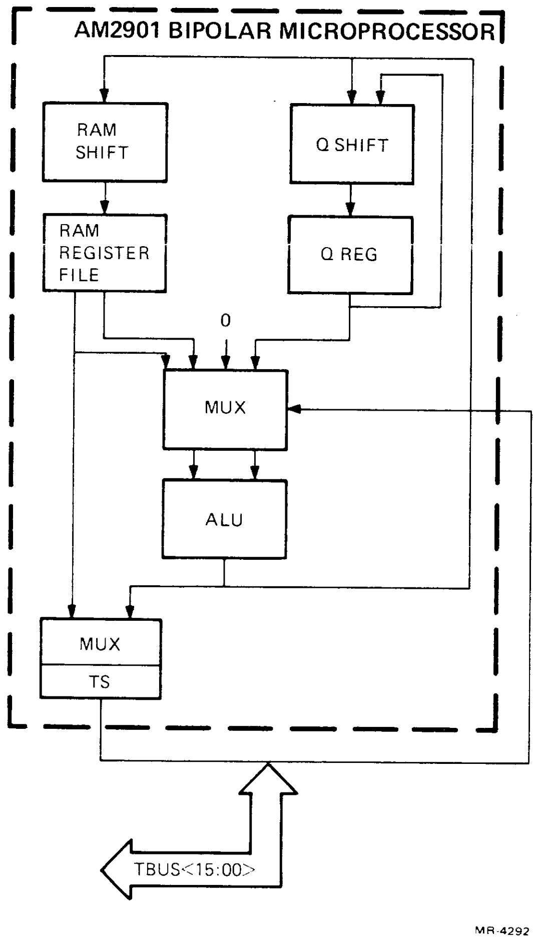


Figure 5-10 AM2901 Bipolar Microprocessor

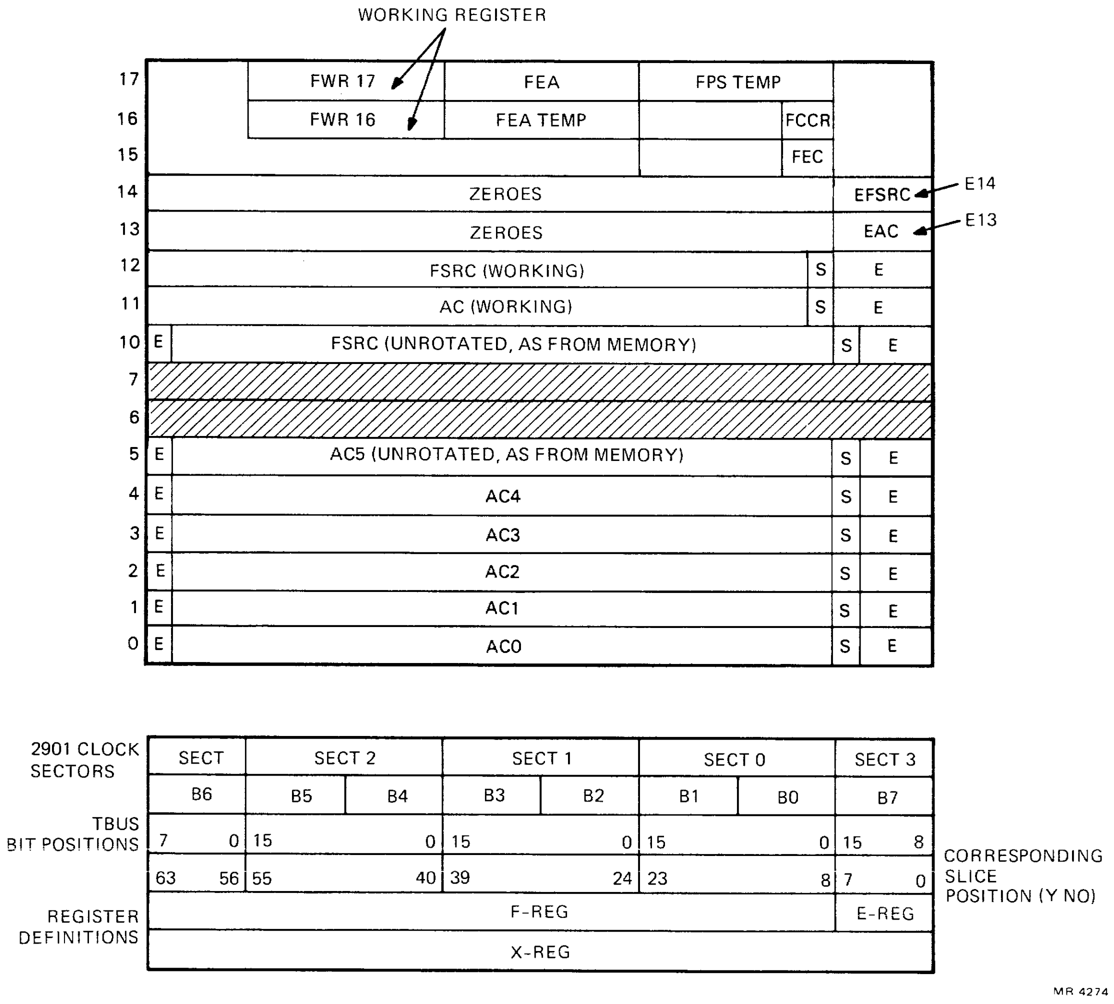


Figure 5-11 RAM Register File Layout

CHAPTER 6
INSPECTION AND INSTALLATION

6.1 GENERAL
This chapter contains information needed to inspect and install an FPF11 floating-point processor option used in a system containing a PDP-11/23 or other compatible central processing unit.

6.2 INSPECTION
The FPF11 option consists of one quad module (M8188) and a ribbon cable. Remove the module from its shipping carton and inspect it for loose components or cracks in the etch. Inspect the cable for loose connections.

NOTE
Return damaged goods to Digital Equipment Corporation, Material Repair Center (MRC), 36 Cabot Road, Woburn, MA, 01801.

6.3 INSTALLATION PROCEDURE
The following is a general installation procedure for adding an FPF11 to a PDP-11/23 or other compatible system. The FPF11 is installed in the backplane slot as illustrated in Figures 6-1 and 6-2. It connects to the CPU by a cable that plugs into the floating-point chip socket on the CPU. Refer to the *Microcomputer Processor Handbook* (EB-15836-18/80) for special requirements.

1. Before installing the FPF11, run system diagnostics to verify that the system receiving the option is working properly.
2. Turn the power off and reconfigure the system. Refer to Figures 6-1 and 6-2.

WARNING
To prevent damage to components, use the special handling procedures for MOS devices when performing the following steps.

3. If present, remove the floating-point processor chip from the CPU module. Refer to Figure 6-2 for its location.
4. To ensure proper bus grant continuity, configure the jumpers as indicated in Table 6-1. Refer to Figure 6-3 for the locations of the jumpers on the FPF11.
5. Insert the Berg connector on the ribbon cable into J1 of the FPF11 module.
6. Install the M8188 module in the vacant slot (see Figure 6-1):

   a) PDP-11/23 system – slot 2 (adjacent to CPU)
   b) PDP-11/24 system – slot 7
   c) MINC and DECLAB-11/MNC – slot 2 (adjacent to CPU)

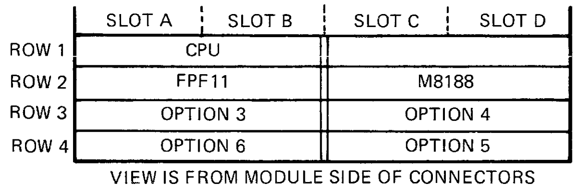
7. Fold the ribbon cable as shown in Figure 6-2 for the above systems. (For installation in systems other than the above, refer to the documentation supplied with those systems.)

8. Insert the 40-pin DIP plug of the ribbon cable into the floating-point socket on the CPU. Note the position of pin 1 in Figure 6-2.

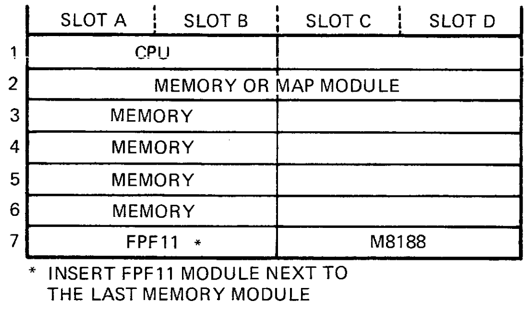
NOTE
Check for possible power supply overload before restoring power. The FPF11 module draws 4 A to 6 A.

9. Turn the power on and run the FPF11 diagnostics to verify proper operation. Refer to Chapter 7, Maintenance, for diagnostic information.

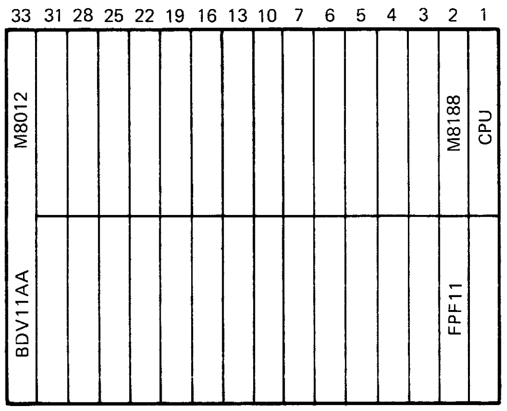
10. Run DEC-X11 to verify the entire system (including the FPF11) is operating properly.



a. PDP-11/23 System



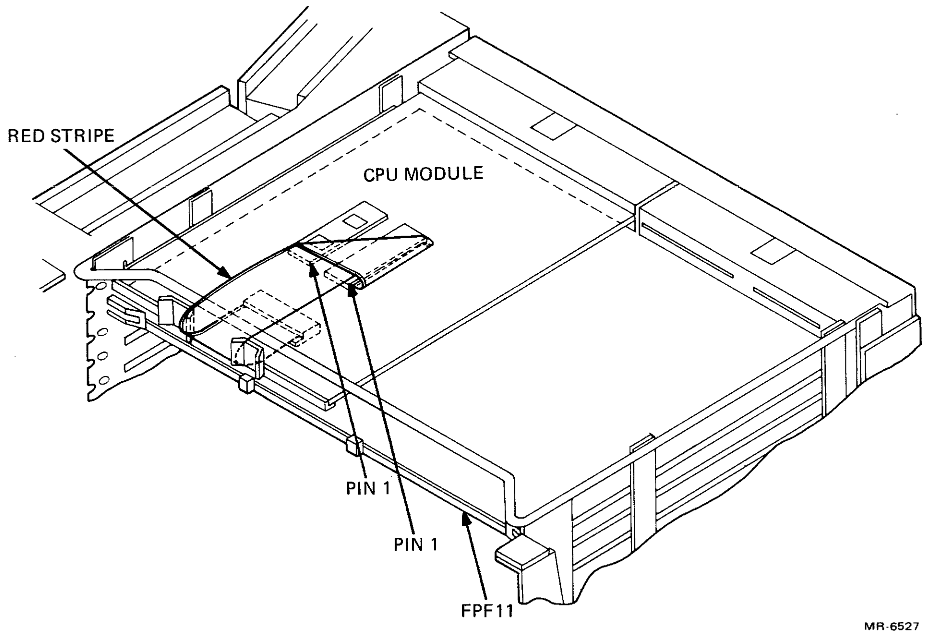
b. PDP-11/24 System



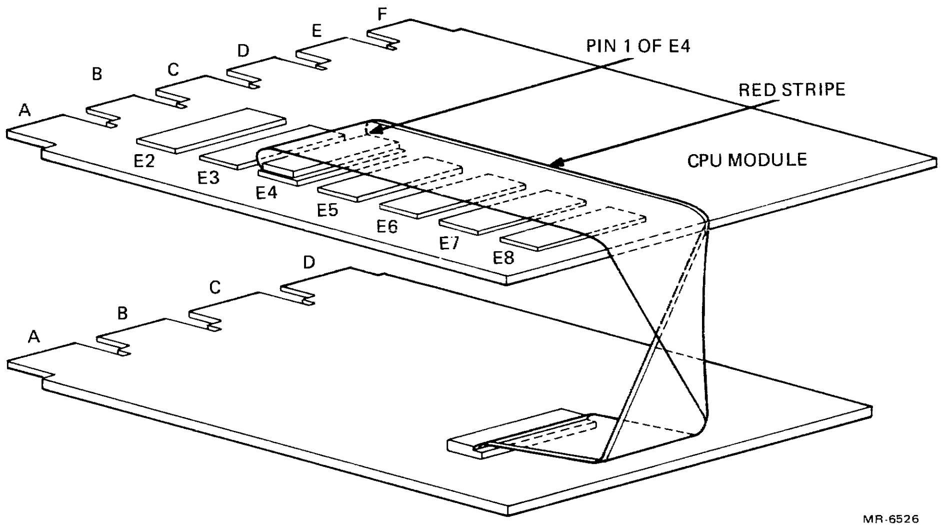
c. MINC and DECLAB-11/MNC Systems

MR-6525

Figure 6-1 FPF11 Module in Various Configurations



a. PDP-11/23 System



b. PDP-11/24 System

Figure 6-2 FPF11 Cable Layout

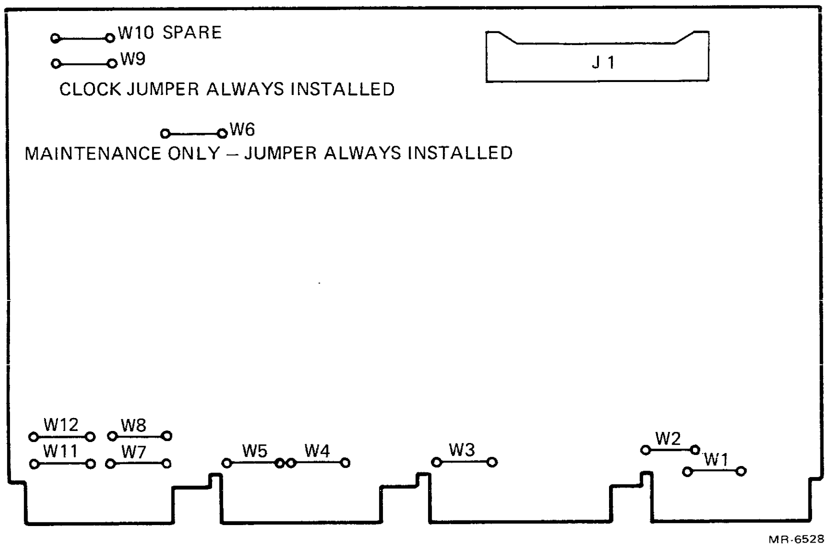

W10 SPARE
W9
CLOCK JUMPER ALWAYS INSTALLED
W6
MAINTENANCE ONLY – JUMPER ALWAYS INSTALLED
J1
W12
W8
W11
W7
W5
W4
W3
W2
W1
MR-6528

Figure 6-3 FPF11 Jumper Locations

Table 6-1 FPF11 Jumper Configurations

	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12
Unibus	R	R	I	R	R	I	I	I	I	R	I	I
QBus	I	I	R	I	I	I	R	R	I	I	R	R

NOTE: R = Jumper removed, I = Jumper installed.

CHAPTER 7
MAINTENANCE

7.1 FFP11 DIAGNOSTICS

Two diagnostics are available to validate and diagnose the FPF11 option. The CPU tests should be run prior to running floating-point diagnostics if there is any doubt about the CPU. Successful running of CPU tests does not rule out the possibility that a failure may cause only floating-point instructions to fail. The two FPF11 diagnostics are listed below. These diagnostics must be run in the order listed because each test requires that the one preceding it was faultless. Otherwise, you may not identify correctly a failed microstep and the location of its cause.

7.1.1 MAINDEC CJFPAA (FPF11, No. 1)

This diagnostic tests the following floating-point instructions.

LDFPS
STFPS
CFCC
SETF, SETD, SETI, and SETL
STST
LDF and LDD (all source modes)
STD (mode 0 and 1)
ADDF, ADDD, and SUBD (most conditions)
CMPD and CMPF
DIVD and DIVF
MULD and MULF
MODD and MODF

7.1.2 MAINDEC CJFPBA (FPF11, No. 2)

This diagnostic tests the following floating-point instructions.

STF and STD (all modes)
STCFD and STCDF
CLRD and CLRF
NEGF and NEGD
ABSF and ABSD
TSTF and TSTD
NEGF, ABSF, and TSTF (all source modes)
LDFBS (all source modes)
LDCIF, LDCLF, LDCID, and LDCLD
LDEXP
STFPS (all destination modes)
STCFL, STCFI, STCDL, and STCDI
STEXP
STST

FPF11 Floating-Point Processor
Technical Manual
EK-FPF11-TM-001

Reader's Comment

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

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

____

____

____

What features are most useful? ____________________

____

____

____

What faults or errors have you found in the manual? ____________________

____

____

____

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

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

____

____

____

☐ Please send me the current copy of the *Technical Documentation Catalog*, which contains information (the remainder of DIGITAL's technical documentation.

Name ____________________	Street ____________________
Title ____________________	City ____________________
Company ____________________	State/Country ____________________
Department ____________________	Zip ____________________

Additional copies of this document are available from:

Digital Equipment Corporation
444 Whitney Street
Northboro, MA 01532
Attention: Printing and Circulating Service (NR2/M15)
Customer Services Section

Order No. EK-FPF11-TM-001

Fold Here

Do Not Tear – Fold Here and Staple

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD, MA.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Educational Services Development and Publishing
200 Forest Street (MR1-2/T17)
Marlboro, MA 01752

Digital Equipment Corporation·Bedford, MA 01730