EK-FP11A-UG-001

FP11-A floating-point processor user's guide

digital equipment corporation • maynard, massachusetts

1st Edition, May 1978

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

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

Printed in U.S.A.

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

6/79-14

CONTENTS

Page

CHAPTER 1 INTRODUCTION

CHAPTER 2 INSTALLATION AND CHECKOUT

CHAPTER 3 REVIEW OF FLOATING-POINT NUMBERS

CHAPTER 4 DATA FORMATS

CONTENTS (Cont)

CONTENTS (Cont)

Page

CHAPTER 7 MAINTENANCE

APPENDIX A OPTION POWER SPECIFICATIONS

FIGURES

Figure No. Title Page

TABLES

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The FP11-A Floating-Point Processor is a hardware option that enables the PDP-11/34A central processor to execute floating-point arithmetic operations. The FP11-A performs all floating-point arithmetic operations and converts data between integer and floating-point formats. Floating-point representation permits a greater range of number values than is possible with the conventional integer mode. Thus, the FP11-A option provides a speedier alternative to the use of software floating-point routines, and system speed is increased without complex arithmetic coding routines that consume valuable CPU time. The FP11-A features both single- and double-precision (32- or 64-bit) capability and floating-point modes.

The FP11-A is an integral part of the central processor. It operates using similar address modes, and the same memory management facilities as the central processor. Floating-point processor instructions can reference the floating-point accumulators, the central processor's general registers, or any location in memory.

1.2 FEATURES

The following paragraphs summarize the features of the PDP-11/34A floating-point instruction set and the FP11-A.

1.2.1 Floating-Point Instruction Set Features

- 32-bit (single-precision) and 64-bit (double-precision) data modes
- Addressing modes compatible with existing PDP-11 addressing modes
- Special instructions that can improve input/output routines and mathematical subroutines
- Allows execution of in-line code (i.e., floating-point instructions and other instructions can appear in any sequence desired)
- Multiple accumulators for ease of data handling
- Can convert 32- or 64-bit floating-point numbers to 16- or 32-bit integers during the Store class of instructions
- Can convert 32-bit floating-point numbers to 64-bit floating-point numbers and vice-versa during the Load or Store class of instructions.

1.2.2 FP11-A Features

- Performs medium-speed, floating-point operations on single- and double-precision data
- Has 17 (decimal) digit accuracy
- Contains its own microprogrammed control store
- Contains six 64-bit floating-point accumulators
- Contains error recovery aids

1.3 ARCHITECTURE

The FP11-A contains scratchpad registers, a floating exception address pointer (FEA), status and error registers, and six general-purpose accumulators (AC0–AC5).

Each accumulator is interpreted to be 32 or 64 bits long depending on the instruction and the status of the floating-point processor. For 32-bit instructions, only the left-most 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 FP11-A and the central processor's general registers or memory.

1.4 PHYSICAL DESCRIPTION

The FP11-A consists of a single hex board [M8267 for the PDP-11/34A (KD11-EA)] and modifications to the M7265 and M7266 boards used in the PDP-11/34 central processor. (The modified boards are designated M8265 and M8266, and the modified processor is designated as the KD11-EA). Figure 1-1 shows the basic signal paths between the central processor and the FP11-A. The bidirectional data bus transfers instructions and data between the processors. An expanded control store in the KD11-EA accommodates floating-point requirements.

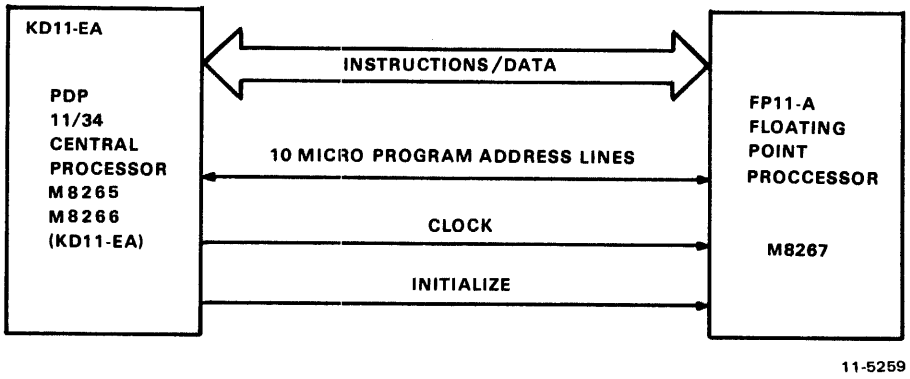


Figure 1-1 KD11-EA/FP11-A Signal Interface

1.5 RELATED DOCUMENTATION

The following documents supplement this user's guide on the FP11-A Floating-Point Processor.

Manual	Document Number
BA11-K Mounting Box Manual	EK-BA11K-MM
BA11-L Mounting Box Manual	EK-BA11L-MM
DL11-W Maintenance Manual	EK-DL11W-MM
KD11-E Processor Manual (PDP-11/34)	EK-KD11E-TM
M9301 Bootstrap Terminator Maintenance Manual	EK-M9301-MM
MM11-C/CP Core Memory Manual	EK-MM11B-TM
MM11-D/DP Core Memory Manual	EK-MM11D-TM
MS11-E-J MOS Memory Maintenance Manual	EK-MS11E-MM
PDP-11 Peripherals Handbook	EP-PDP11-HB
PDP-11/04, 34, 45, 55 Processors Handbook	EP-PDP11/04-HB
PDP-11/34 Processor Handbook	EP-11034-HB
KD11-EA Processor Manual (PDP-11/34A)	EK-KD1EA-MM

CHAPTER 2
INSTALLATION AND CHECKOUT

2.1 SCOPE

This chapter provides the information necessary for unpacking, inspection, installation, and checkout of the FP11-A and FP11-AU Floating-Point Processors.

2.2 FP11-A FLOATING-POINT PROCESSOR INSTALLATION

The FP11-A Floating-Point Processor option for the PDP-11/34A CPU consists of the following parts:

1. M8267 – Floating-point module
2. H8821 – 20-pin over-the-top connector
3. 54-12416 – 10-pin over-the-top connector
4. W9042 – Bus extender module

Prior to the installation of the FP11-A option, the +5 Vdc current available to the PDP-11/34A CPU backplane must be calculated. The following procedure is designed to help you calculate +5 Vdc current drain and system configuration.

2.2.1 FP11-A Add-On Installation Procedure

1. Verify system integrity by running the following diagnostics in the order given.

PDP-11/34	CPU Test	DFKAA
	Traps Test (at least Rev. C)	DFKAB
	EIS Test	FDKAC
0–124K memory exerciser		DZQMC

2. Is CPU a PDP-11/34A? (See serial name tag.)

Yes ↓ **No**

An FP11-A cannot be installed on a PDP-11/34. To upgrade a PDP-11/34 to use an FP11-A, an FP11-AU kit must be used. Refer to Paragraph 2.3.

3. Is CPU box 26.7cm (10.5 in)?

Yes ↓ **No**

Refer to Paragraph 2.2.2, BA11-L Box.

4. Calculate +5 Vdc current drain in the CPU backplane. Calculate +5 Vdc current drain for all other backplanes in box (Figure 2-1).

5. Is the total current drain (all backplanes) greater than 57 A? (Does not include M8267 current.)

   No **Yes**

   Is expander box available with room and current?

   Yes **No**

   Refer to step 17.

   Refer to step 6.

6. Is the battery backup (BBU) option present?

   No **Yes**

   All jumpers must be out of all backplanes in box. Refer to Figure 2-2 and step 9.

7. Do backplane jumpers check as follows?

   CPU backplane (DD11-PK).

+5 VB to +5 V jumper	In	See Figure 2-2.
+15 VB to +15 V jumper	In	
-15 VB to -15 V jumper	In	

   Refer to step 8.

8. All other backplanes in box (DD11-DK, CK).

+5 VB to +5 V jumper	Out	See Figure 2-2.
+15 VB to +15 V jumper	In	
-15 VB to -15 V jumper	In	

   Refer to step 9.

9. Is slot 3 open in CPU backplane?

   Yes **No**

   Is cache (M8268) in slot 3?

   No **Yes**

   Cache is placed in slot 3 only if FP is not present. When FP is added, cache is moved to slot 5. Slots 4A and B are reserved for M9301/M9312 (Figure 2-1). Remove the over-the-top (OTT) connectors and move cache module (M8268) to slot 5. H8822 (OTT) is necessary to complete the installation (Figure 2-1). Refer to step 10.

   Refer to step 10.

10. Is the MOS memory installed in any backplane other than the CPU backplane?

    No **Yes**

    Add 0.5 A at +5 Vdc for each MOS board not installed in the CPU backplane (CPU box only) to the current drain total for the CPU backplane calculated in step 4.

    Refer to step 11.

11. Is the CPU backplane current drain less than 25 A at +5 Vdc?

Yes **No**

The devices must be moved from the CPU backplane to some other backplane in the box in order to vacate slot 3 and to reduce the current drain at +5 Vdc to 25 A or less. This must be done without overloading the second +5 Vdc regulator in the box.

Is reconfiguring within the box possible?

Yes **No**

Can the devices be moved to an expander box without overloading the expander box?

No **Yes**

Reconfigure the system until the current in the CPU backplane is less than 25 A at +5 Vdc.

Refer to step 17.

Reconfigure within the box until the CPU backplane current drain at +5 Vdc is less than 25 A.

Refer to step 12.

12. Install the FP11-A module (M8267) in slot 3 of the CPU backplane.

13. Is KY11-LB (M7859) present?

No **Yes**

Remove the two 10-pin maintenance cables, if necessary, from the CPU (M8266) and install them in the FP module (M8267) as shown in Figure 2-1.

Refer to step 14.

14. Install the two over-the-top (OTT) connectors as shown in Figure 2-1. Use H8822 if the cache and FP are both present.

15. Turn power on and run the following diagnostics in the order given.

PDP-11/34	CPU Test		DFKAA
PDP-11/34	Traps Test	At least Rev. C	DFKAB
PDP-11/34	EIS Test		DFKAC
PDP-11/34	FPP Diagnostic	Part 1	DFFPA
PDP-11/34	FPP Diagnostic	Part 2	DFFPB
PDP-11/34	FPP Diagnostic	Part 3	DFFPC

16. End

17. When it is impossible to reconfigure the box to accommodate the FP11-A (M8267) without overloading the +5 V regulator, one alternative is to move some devices to an expander box. If an expander box is not present on the system, then there are two ways to proceed.

    a. Remove some number of devices from the box to compensate for the 7 A at +5 Vdc used by the FP11-A, and leave these devices out of the system.

    b. Postpone installation until an expansion box can be added to the system.

    Refer to step 16.

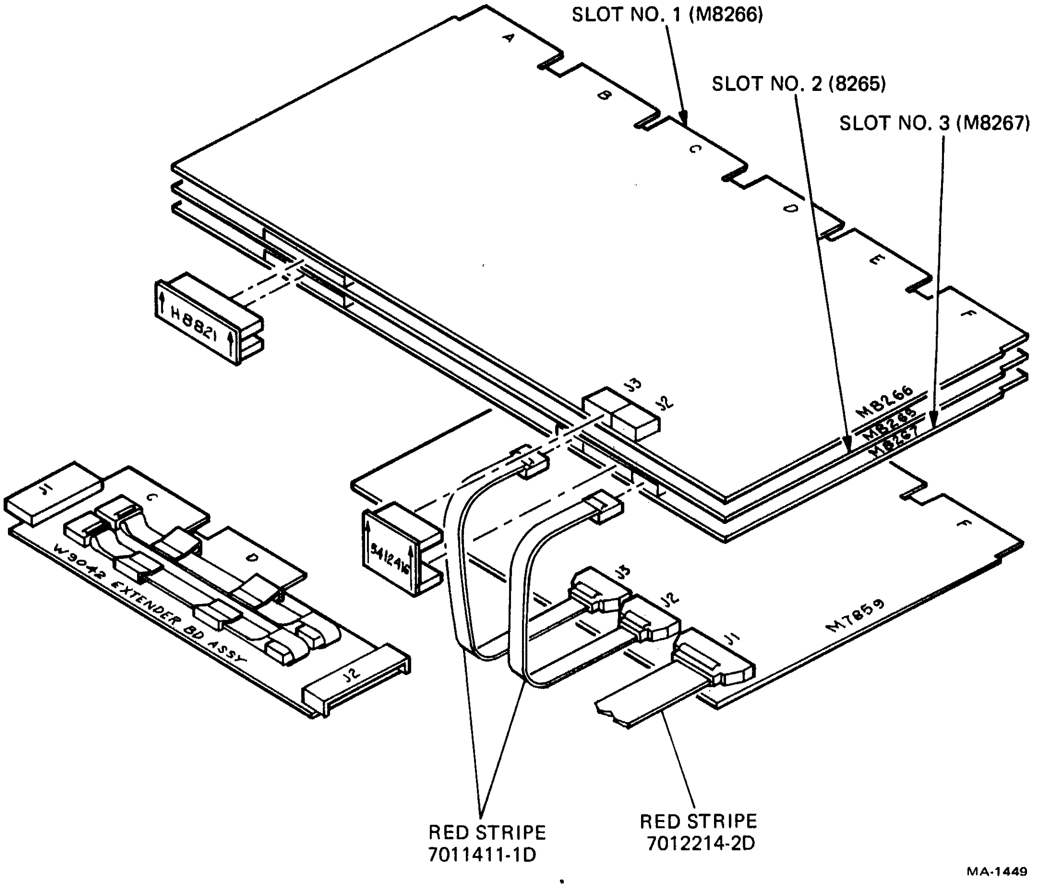
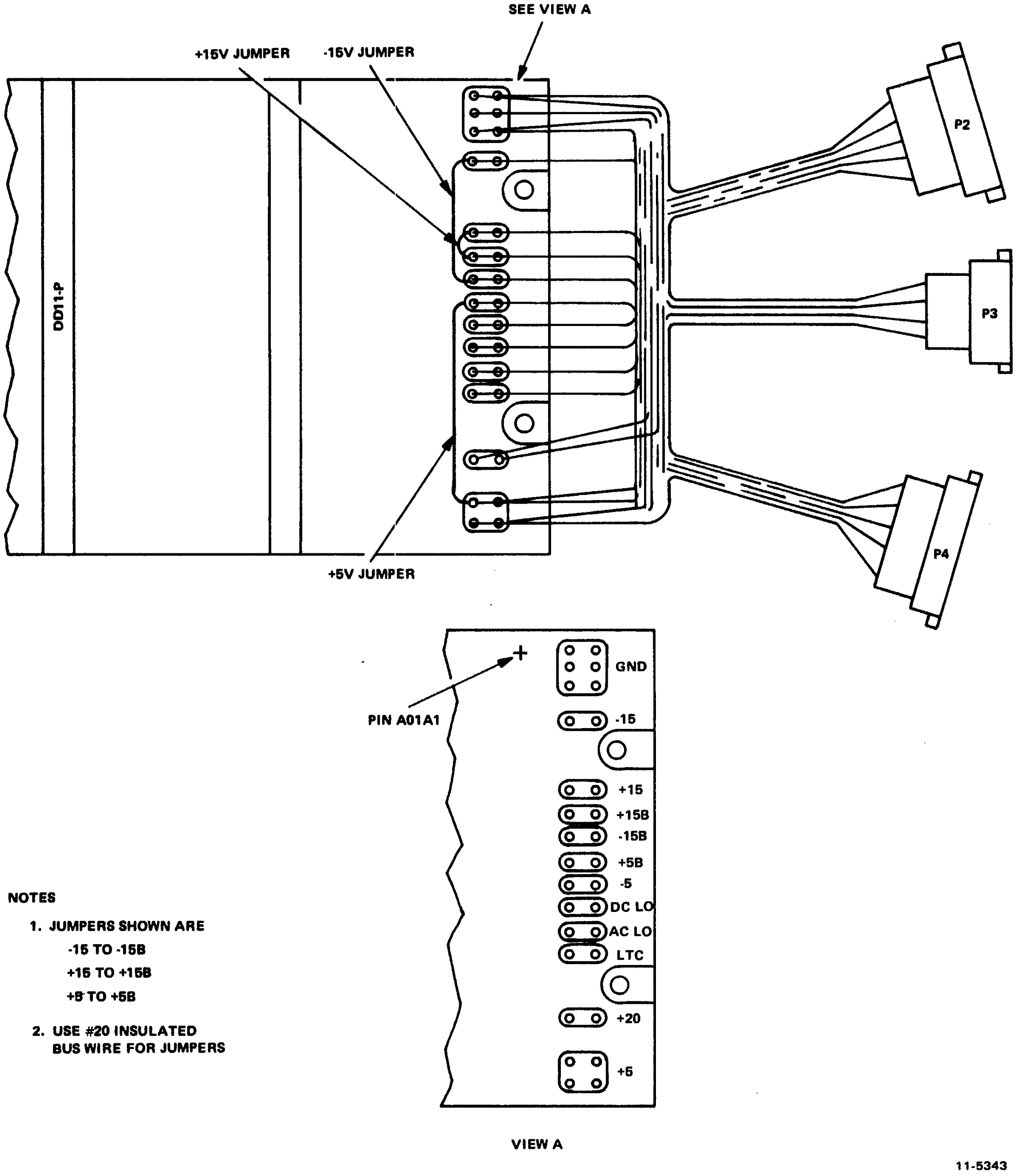


Figure 2-1 Maintenance Cable Installation



NOTES

1. JUMPERS SHOWN ARE
   - -15 TO -15B
   - +15 TO +15B
   - +5 TO +5B
2. USE #20 INSULATED BUS WIRE FOR JUMPERS

Figure 2-2 Backplane Jumpers

2.3 FP11-AU UPGRADE KIT

The FP11-AU upgrade kit contains power supply components necessary to increase the +5 Vdc current levels available from the 26.7 cm (10.5 in) mounting box. The purpose of the upgrade kit is to provide a method by which PDP-11/34 CPUs can use the floating-point (FP11-A) option. Since the FP11-A is an option for the PDP-11/34A CPU, additional hardware is required to upgrade the PDP-11/34 models to include the floating-point option. The following parts are required.

1. M8265 - Data path module
2. M8267 - FP module
3. M8266 - Control module
4. H7441 - Regulator module
5. H8821 - 20-pin over-the-top connector
6. 54-12416 - 10-pin over-the-top connector
7. W9042 - Bus extender module
8. 54-10834YA - Power distribution board

The tools required are:

1. Phillips screwdriver (medium and large)
2. Slot screwdriver (large)
3. 90 degree offset Phillips screwdriver.

2.3.1 FP11-AU Power Components Installation

CAUTION

Turn off computer system and disconnect it from power source before performing installation procedure.

1. Slide BA11-K mounting box out of the cabinet assembly to the limits of the chassis slides.
2. Release and remove mounting box top cover to gain access to H765 power supply assembly.
3. Loosen and remove cable clamp that secures the cables that are routed across the top of the power supply.
4. Loosen and remove power supply cover.
5. Rotate the mounting box in such a manner that the bottom of the mounting box faces away from the cabinet (box rotated 90 degrees).
6. Loosen and remove mounting box bottom cover to gain access to the power distribution board located between the power supply and the backplane.

CAUTION

Do not remove the hinge screws (one on each side) located at the junction of the power supply and the module enclosure near the top side of the mounting box.

7. Remove four flat-head screws (no washers) located approximately 10 cm (4 in) from the bottom of the mounting box and at the junction of the power supply and the module enclosure assembly. The power supply can now be swiveled away from the module enclosure.

8. Locate the H744 +5 Vdc regulator assembly. This regulator is the second module from the right when viewing the bottom of the mounting box from the wire-wrap side of the backplane.

9. Locate and remove the two mounting screws and washers located just to the left of the H744 Mate-N-Lok connector. There will be a green safety wire secured by one of these screws.

NOTE

A 90 degree offset Phillips-head screwdriver is required to remove these screws and attached hardware.

10. Locate and remove the last retaining screw and washer located on the back of the power supply and to the left of the H744 decal.

11. Release and remove the H744 Mate-N-Lok connector.

12. Remove the H744 regulator by sliding it out through the top of the power supply assembly. (Note that the mounting box may have to be rotated to accomplish removal.)

13. Replace the H744 regulator with the H7441 regulator (included in the upgrade kit).

14. Replace mounting hardware removed in steps 9 and 10. Do not connect the H7441 Mate-N-Lok connector at this time.

15. Release and remove the three Mate-N-Lok connectors connecting the remaining regulators to the power distribution board assembly.

16. Locate and remove the black ground wire soldered to the power distribution board (located near J16). Remove this ground wire from the power supply and the module enclosure assembly.

17. Remove the +5 V and ground fastons from the power distribution board (located near J14).

18. Release and disconnect Mate-N-Lok connector from J8 on the power distribution board.

19. Locate and remove four flat-head screws securing the power distribution board to the module enclosure assembly. These screws are located (two on each side) 5 cm (2 in) from the bottom of the mounting box and near the junction of the power supply and the module enclosure.

NOTE

The removal of these four screws will allow the removal of the power distribution panel which in turn will allow removal of all backplane Mate-N-Lok connectors.

20. Release and disconnect all backplane Mate-N-Lok connectors.

21. Release and disconnect two Mate-N-Lok connectors connecting the power distribution board to the power supply.

22. Remove the power distribution board.

23. Replace the power distribution board with the new 5410834-YA power distribution board (included in upgrade kit).

24. Reverse procedure (steps 22–14 and steps 7–1) to install the new power distribution board and to return system to normal.

2.3.2 FP11-AU Logic Installation

Refer to Paragraph 2.2.1 and calculate +5 Vdc current drain and system configuration.

CHAPTER 3
REVIEW OF FLOATING-POINT NUMBERS

3.1 INTRODUCTION

This chapter briefly outlines some fundamentals of floating-point arithmetic. It provides useful background for more advanced topics in later chapters. The reader already familiar with floating-point numbers and arithmetic may skip this chapter and continue to Chapter 4 for a discussion of FP11-A data formats.

3.2 INTEGERS

All data within a computer system can be represented in integer form. The numbers that can be represented in a 16-bit machine range in magnitude from 000000_8 to 177777_8 (or from 0_{10} to $65{,}536_{10}$). However, this presents problems with integer representation. A number between 1 and 2 (for example) or numbers greater than $65{,}536_{10}$ can not be represented. Thus, integer representation imposes an *accuracy* and a *range* limitation.

These limitations are imposed by the stationary position of the *radix point* (e.g., the decimal point in base 10 notation or the binary point in base 2 notation). An integer's radix point is usually omitted in integer representation because it always marks the integer's least significant place. That is, there are never any digits to the right of an integer's radix point. 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 the fixed radix point. This is accomplished through the use of *floating-point* notation.

3.3 FLOATING-POINT NUMBERS

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

Example

$$1{,}000{,}000_{10} = \overset{\text{Basic value}}{1.} \times \underset{\text{Radix}}{10}^{\overset{\text{Exponent}}{6}}$$

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

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

The convention chosen for representing floating-point numbers with scientific notation in the FP11-A requires the radix point to always be to the left of the most significant digit in the basic value (e.g., .512 × 10^3 in the above example). This modified basic value is called a fraction.

More examples of scientific notation are shown below.

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

Note that in each of the examples above, only significant digits are retained in the final result and the radix point is always (by convention) 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 (i.e., 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 the 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 in the FP11-A is one in which all nonsignificant leading 0s have been removed. The process used to obtain this representation is called *normalization*, which is explained in more detail in Paragraph 3.4.

3.4 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, a process called normalization is used in the FP11-A. The normalization process consists of testing the fraction for leading 0s and left-shifting it until it is in the form 0.1 . . . . The exponent is accordingly decremented by the number of left shifts of the fraction. 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 (i.e., 1/2, 1/4, 1/8 . . .), the smallest normalized fraction is 1/2 (0.10000 . . .). The largest normalized fraction is 0.11111 . . . . Figure 3-1 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.

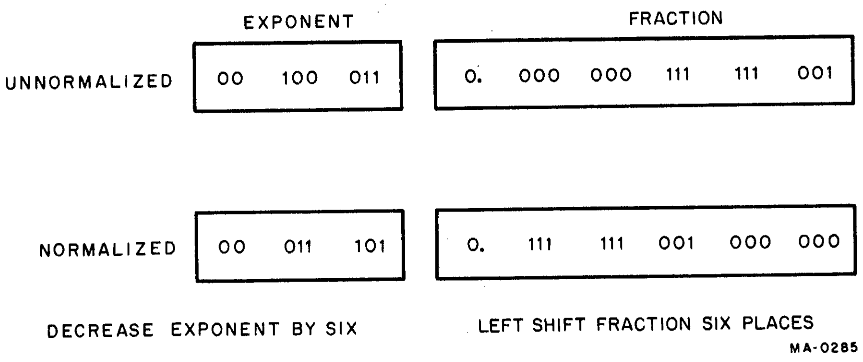


Figure 3-1 Normalization

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 $= -1$

3.5 FLOATING-POINT ADDITION AND SUBTRACTION

In order to perform floating-point addition or 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 associated 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.

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

$+0.111\ 000\ 000\ 000\ 000 \times 2^6 = 7_8 = 7_{10}$

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

$$\begin{array}{r}
+0.101\ 000\ 000\ 000\ 000 \times 2^{6} = 50_{8} = 40_{10} \\
+0.000\ 111\ 000\ 000\ 000 \times 2^{6} = 7_{8} = 7_{10} \\
\hline
+0.101\ 111\ 000\ 000\ 000 \times 2^{6} = 57_{8} = 47_{10}
\end{array}$$

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

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

3.6 FLOATING-POINT MULTIPLICATION AND DIVISION

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

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

Example:

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

1.

$$\begin{array}{r}
0.1110000 \times 2^{3} = 7_{8} = 7_{10} \\
\times 0.1010000 \times 2^{6} = 50_{8} = 40_{10} \\
\hline
111 \\
0000 \\
11100 \\
\hline
.10001100000000 \times 2^{9}
\end{array}$$

(Result already in normalized form.)

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

$$.\overbrace{100}^{4}\overbrace{011}^{3}\overbrace{000}^{0}.00000 = 430_{8} = 280_{10}$$

Example:

Divide 15_{10} by 5_{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: $.1100000 \times 2^2$

   Normalized Fraction   Normalized Exponent

   Move binary point two places to the right.

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

CHAPTER 4
DATA FORMATS

4.1 INTRODUCTION

The FP11-A requires its input data (operands) to be formatted. Formatting allows the FP11-A to process operands in a meaningful way and produce correct results. There are four different formats for operands input to the FP11-A: short-integer format (I), long-integer format (L), single-precision format (F), and double-precision format (D).

Output data from the FP11-A is also formatted. This output data is in the form of:

1. FP11-A status information and FP11-A exception information required by the CPU
2. Data sent to memory (via the CPU), which must be in I, L, F, or D format.

This chapter describes the FP11-A data formats. It is assumed that the reader is familiar with 2's complement notation.

4.2 FP11-A INTEGER FORMATS

There are two integer formats, short (I) and long (L). The short-integer format is 16 bits long and the long-integer format is 32 bits long. Data words (operands) in integer format are represented in 2's complement notation. In both I and L formats, the most significant bit of the data word is the sign bit. Figure 4-1 shows the integers 5 and –5 in both I and L formats.

Figure 4-2 illustrates the formats in which integers are arranged *in memory*. Integers sent to memory must be in one of these formats. Integers received by the FP11-A are arranged and manipulated according to the type of instruction being executed. Refer to Paragraphs 5.3.11 and 5.3.12 for descriptions of the ways in which incoming integers are manipulated during the load exponent and load convert integer-to-floating instructions, respectively.

4.3 FP11-A FLOATING-POINT FORMATS

There are two floating-point formats, single-precision (F) and double-precision (D). The single-precision format is 32 bits long and the double-precision format is 64 bits long. Figure 4-2 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 (Paragraph 4.3.1.2). 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 (Paragraph 4.3.1.1).

4.3.1 FP11-A Floating-Point Data Word

Figure 4-3 illustrates the formats in which floating-point numbers are arranged *in memory*. Floating-point numbers sent to memory must be in one of these formats. Floating-point numbers received by the FP11-A are arranged as illustrated in Figure 4-4.

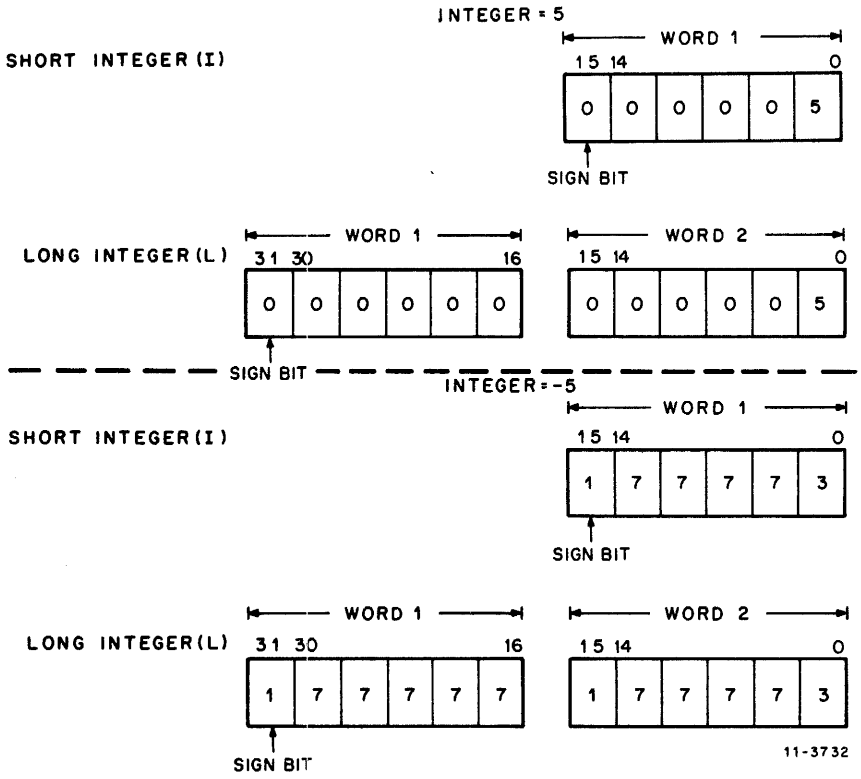
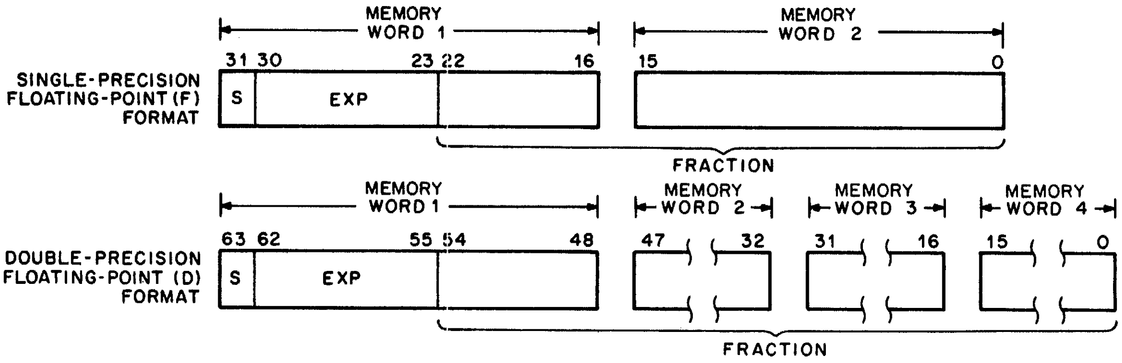


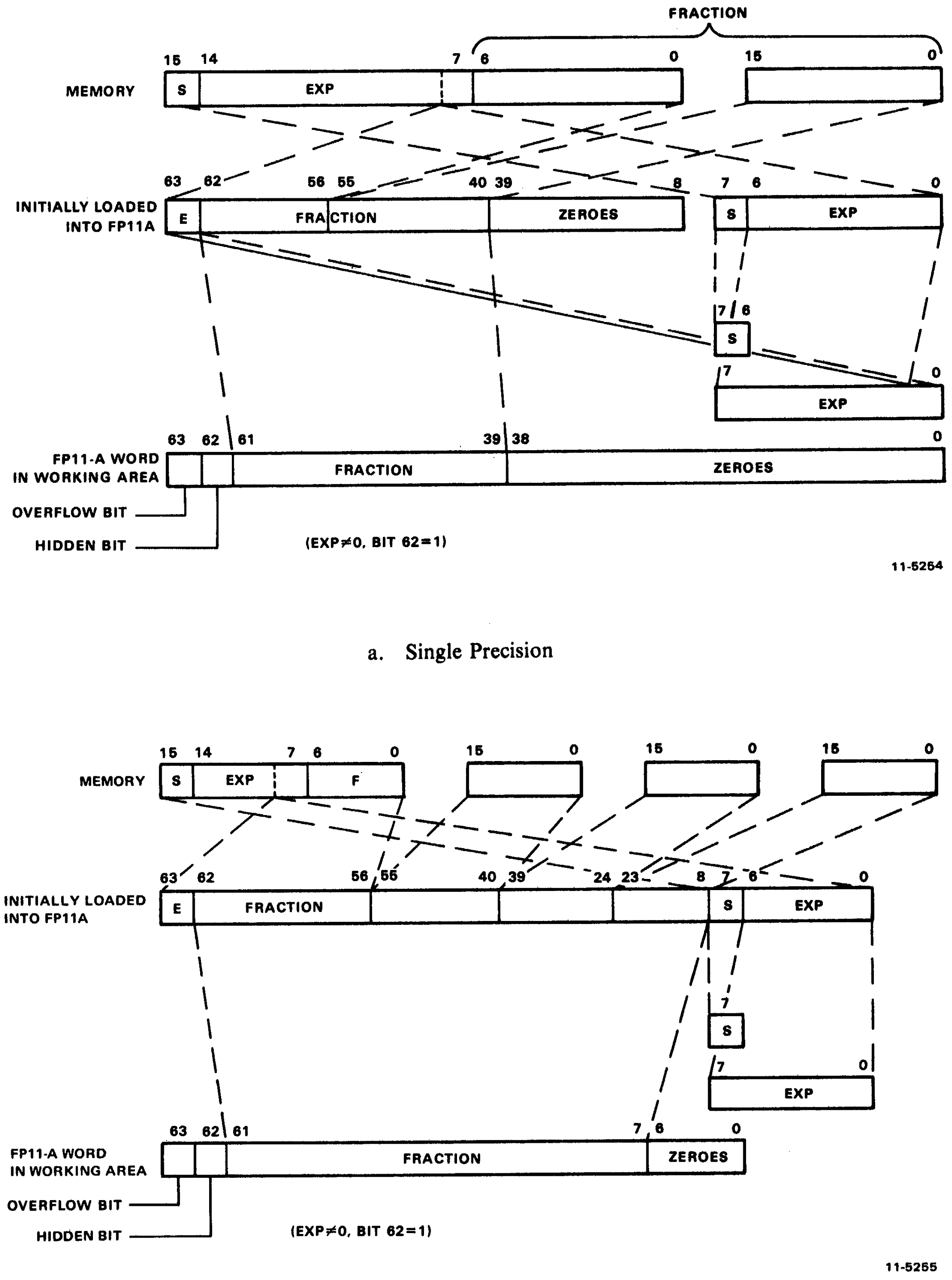
Figure 4-1 Integer Formats



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.

MA-0280

Figure 4-2 Floating-Point Data Formats



a. Single Precision

b. Double Precision

Figure 4-3 Floating-Point Data Words

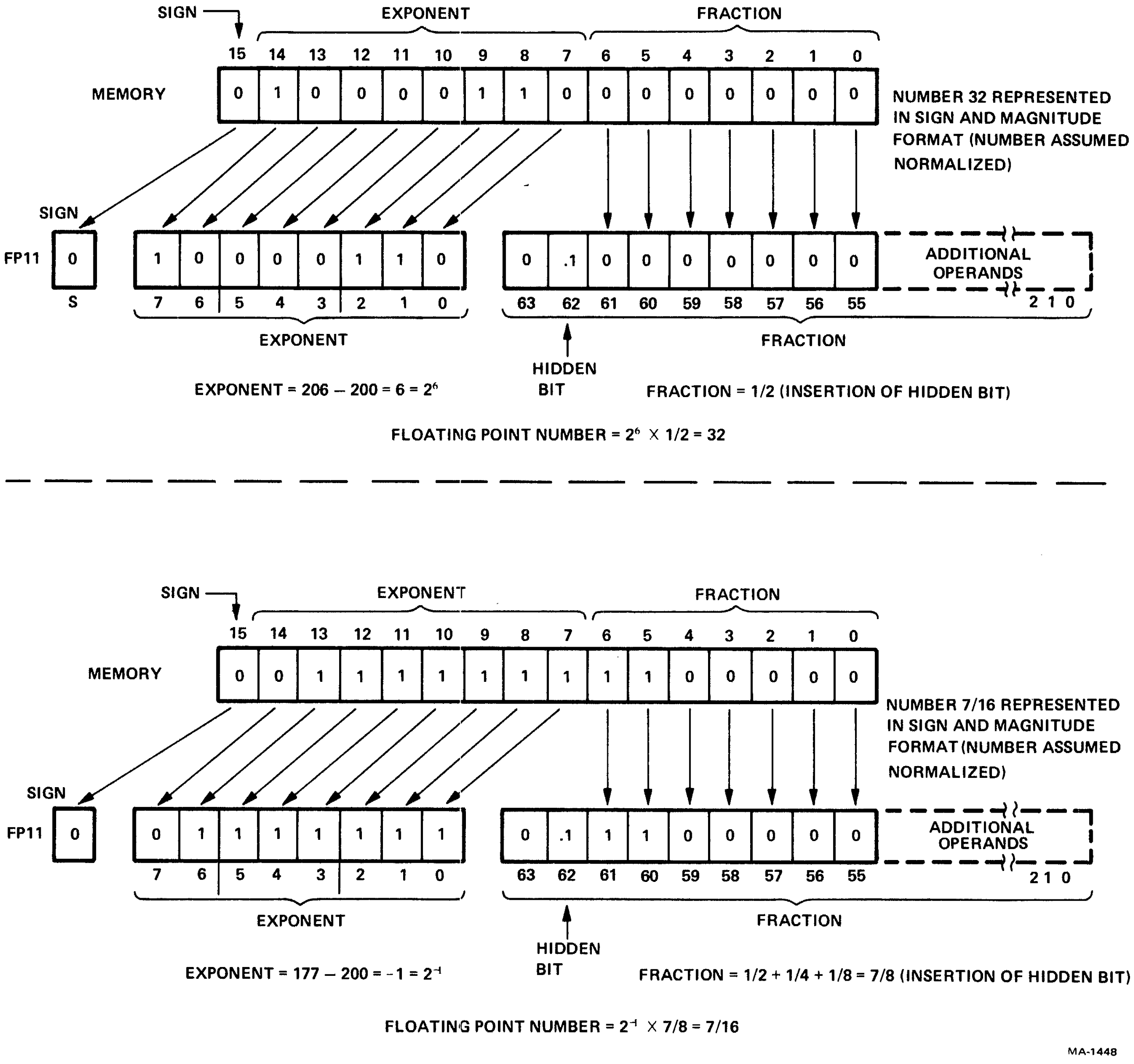


Figure 4-4 Interpretation of Floating-Point Numbers

The sign bit, exponent bits, and fraction bits in the FP11-A data word have the same values as the data word in memory. Note, however, that the FP11-A data word has more bits than its counterpart in memory. This is because the FP11-A has provisions for generating an overflow bit and a "hidden" bit.

For purposes of discussion, the FP11-A data word can be thought of as being divided into two major parts:

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

2. An exponent.

4.3.1.1 Floating-Point Fraction

4.3.1.1 Floating-Point Fraction - The fraction is expressed in sign and magnitude notation. The following simple example illustrates the idea behind sign and magnitude notation.

	2's Complement Notation	Sign and Magnitude Notation
+2	000010	000010 Sign Magnitude
−2	111110	100010 Sign 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 from approximately 0 through 2 as shown in Figure 4-5. The FP11-A, however, normalizes all unnormalized fractions. That is, the fractions are adjusted such 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 or from 1/2 to approximately 1.

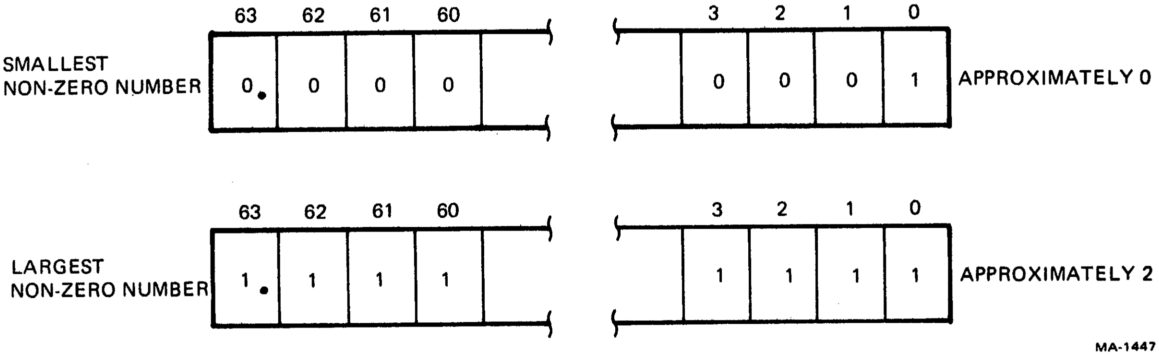
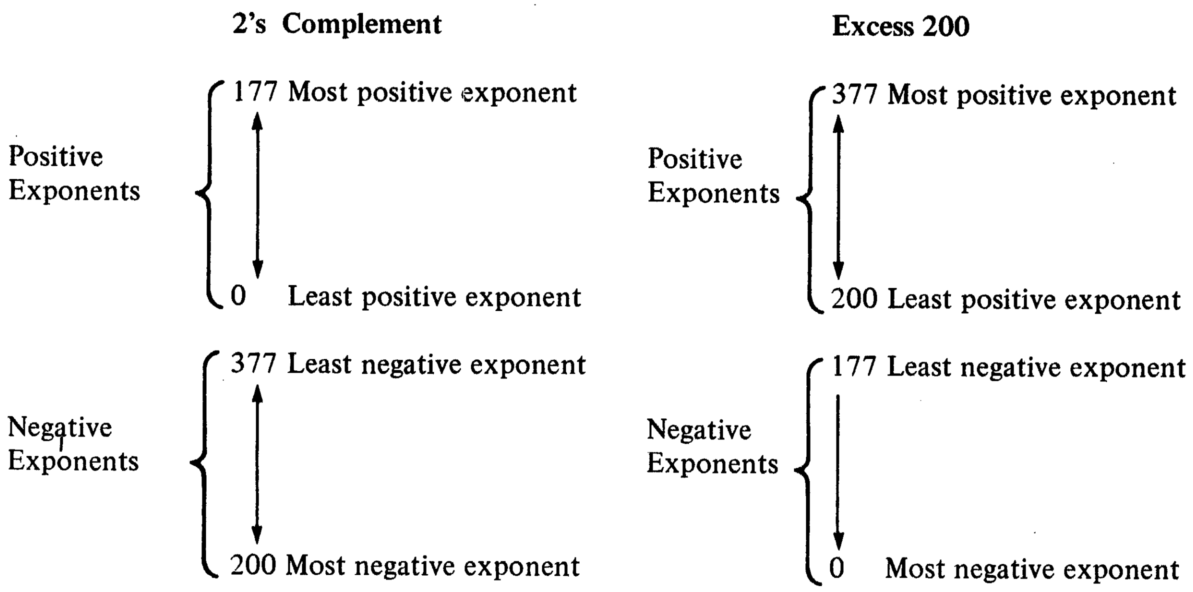


Figure 4-5 Unnormalized Floating-Point Fraction

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 FP11-A 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 FP11-A, 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.

4.3.1.2 Floating-Point Exponent – The 8-bit floating-point exponent is expressed in excess 200 notation. The chart below illustrates the relationship between exponents in 2's complement notation and exponents in excess 200 notation.


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

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 (or from –200 to +177). A number with an exponent of –200 is treated by the FP11-A as 0.

For example, 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. Figure 3-5 illustrates the range of floating-point numbers that can be handled by the FP11-A. For simplicity, a fraction length of only three bits is shown.

4.4 FP11-A PROGRAM STATUS REGISTER

The FP11-A 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 processor. 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 3 mode bits and additional bits to enable various interrupt conditions. Figure 4-6 shows the layout of the program status register. Each bit shown in the figure is described in Table 4-1.

NOTE

The FP11-A has no Unibus addresses. All FP11-A registers are accessed by floating-point instructions only.

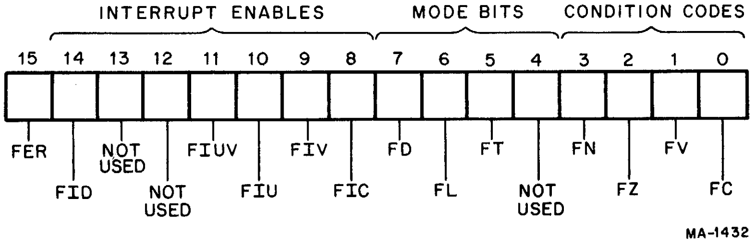


Figure 4-6 FP11-A Status Register Format

Table 4-1 FP11-A Status Register

Bit	Name	Function
15	FER	This bit indicates an error condition of the FP11-A.
14	FID	Floating Interrupt Disable – All interrupts by the FP11-A are disabled when this bit is on. Primarily for maintenance use. Normally clear.
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, any arithmetic operation treats it as if it were a positive 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 the FIU is not set and underflow occurs, the result is set to zero.

Table 4-1 FP11-A Status Register (Cont)

Bit	Name	Function
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 the FIV bit is not set, the result of the operation is the same; the only difference is that the interrupt does not occur.
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 FIC is reset, the result of the operation is the same; however, no interrupt occurs.
7	FD	Double-Precision Mode Bit – This bit, when set, specifies double-precision format and, when reset, 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 – This bit, when set, causes the result of any floating-point operation to be truncated rather than rounded.
4	Not Used	
3–0	FN, FZ, FV, and FC	These bits are the four floating-point condition codes, which can be loaded in the CPU's N, Z, V, and C 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 Table 5-1.

4.5 PROCESSING OF FLOATING-POINT EXCEPTIONS

Location 244 is the interrupt vector used to handle all floating-point interrupts. A total of six possible interrupts can occur. These possible interrupt exceptions are encoded in the FP11-A exception code (FEC) register. The interrupt exception codes represent an offset into a dispatch table, which routes the program to the right error handling routine. The dispatch table is a function of the software. The FEC for each exception is briefly described in Table 4-2.

Refer to the *PDP-11/04, 34, 45, 45 Processor Handbook* for further details concerning FP11-A exceptions.

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 4-2 FP11-A Exception Codes

FP11-A Exception Code	Definition
2	Floating Op Code Error - The FP11-A 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 FP11-A program status register. All traps are disabled if FID is set.

CHAPTER 5
FLOATING-POINT INSTRUCTIONS

5.1 FLOATING-POINT ACCUMULATORS

The FP11-A contains six general-purpose accumulators (AC0–AC5). These accumulators are 64-bit read/write scratchpad memories with non-destructive readout.

Each accumulator is interpreted as being either 32 or 64 bits long, depending on the instruction and the FP11-A status (Chapter 4). 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 left-most 32 bits of an accumulator as shown in Figure 5-1.

The floating-point accumulators are used in numeric calculations and interaccumulator data transfers. AC0–AC3 are used for all data transfers between the FP11-A and the CPU or memory.

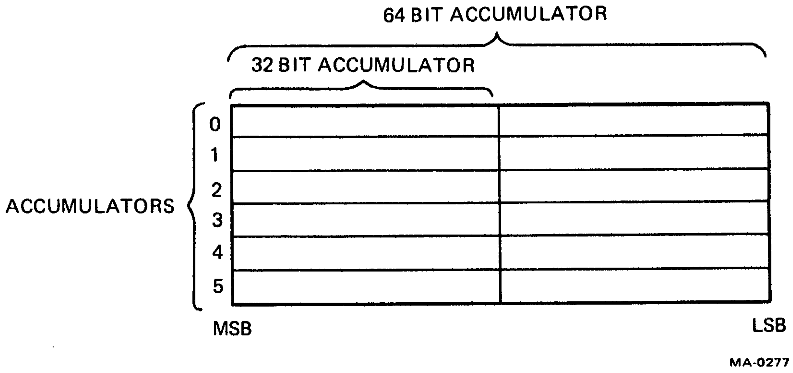


Figure 5-1 Floating-Point Accumulators

5.2 INSTRUCTION FORMATS

An FP11-A instruction must be in one of five formats. These formats are summarized in Figure 5-2.

The 2-bit AC field (bits 6 and 7) allows selection of scratchpad accumulators 0 through 3 only.

If address mode 0 is specified with formats F1 or F2, bits 2–0 are used to select a floating-point accumulator. Only accumulators 5–0 can be specified in mode 0. If 6 or 7 is specified in bits 2–0 in mode 0, the FP11-A traps if floating-point interrupts are enabled (FID = 0). The FEC will indicate an illegal op code error (exception code 2).

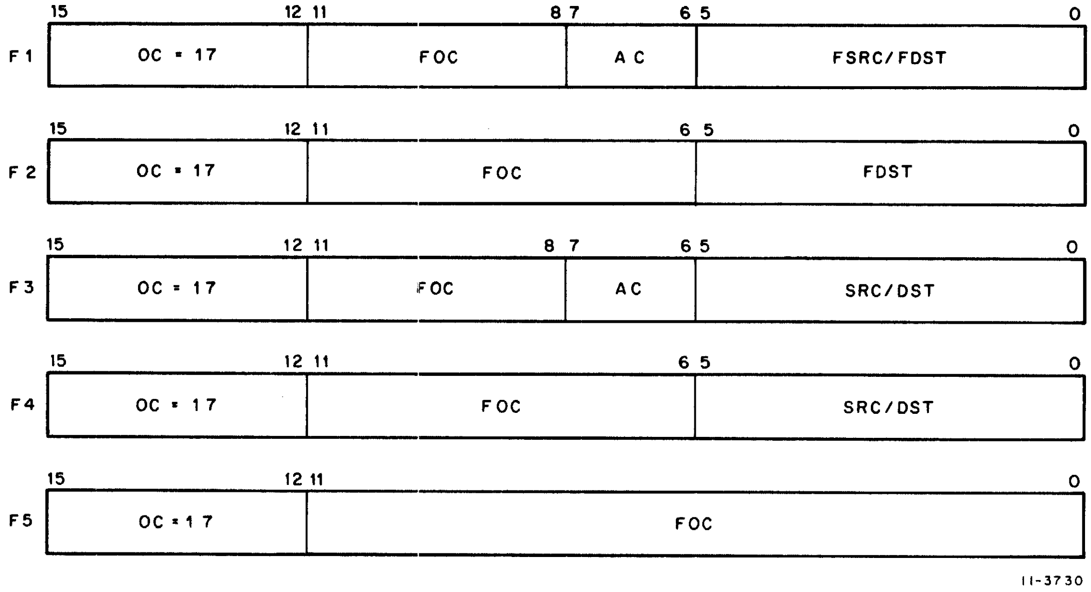


Figure 5-2 Instruction Formats

The fields of the various instruction formats (as summarized in Table 5-1) 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 Operating 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 the PDP-11 instruction.

DST Destination – A 6-bit destination field identical to that in a PDP-11 instruction.

FSRC Floating Source – A 6-bit field used only in format F1. It is identical to SRC, except in mode 0 when it references a floating-point accumulator rather than a CPU general register.

FDST Floating Destination – A 6-bit field used in formats F1 and F2. It is identical to DST, except in mode 0 when it references a floating-point accumulator instead of a CPU general register.

AC Accumulator – A 2-bit field used in formats F1 and F3 to specify FP11-A scratchpad accumulators 0–3.

Table 5-1 Format of FP11-A Instructions

Instruction Format	Instruction	Mnemonic
F2	ABSOLUTE	ABSF FDST
		ABSD FDST
F1	ADD	ADDF FSRC, AC
		ADD FSRC, AC
F2	CLEAR	CLRF FDST
		CLRD FDST
F4	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	LDCIF SRC, AC
		LDCID SRC, AC
		LDCLF SRC, AC
		LDCLD SRC, AC
F3	LOAD EXPONENT	LDEXP SRC, AC
F4	LOAD FP11'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
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 FP11'S PROGRAM STATUS	STFPS DST
F4	STORE FP11'S STATUS	STST DST
F1	SUBTRACT	SUBF FSRC, AC
		SUBD FSRC, AC
F2	TEST	TSTF FDST
		TSTD FDST

5.3 INSTRUCTION SET

Table 5-2 contains the instruction set of the FP11-A. Some of the symbology may not be familiar. Therefore, a brief description follows.

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 is designated by F. If the flip-flop is set, double-precision is specified and is designated by D. Examples are NEGF, NEGD, and SUBD.

   NOTE

   Only the assembler or compiler differentiates between NEGF and NEGD or LDCID or 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 (i.e., 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 is designated by I. If the flip-flop is set, long-integer format is specified and is designated by L. Examples are SETI and SETL.

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

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

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

   $C_{F,D}$ or $C_{D,F}$ - Convert single-floating to double-floating or convert 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 (excess 200 notation) and a fraction of all 1s. Note the UPLIM is dependent on the format specified. LOLIM is defined as the smallest possible number that is not identically 0. This number has an exponent of 001 and a fraction of all 0s except for 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 in Table 5-2 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 (F instruction format)

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

Example 2: LDF Instruction

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

172400 + C * 100 + FSRC

AC = 2

2 * 100 = 200

172400 + 200 = 172600
FSRC is equal to 26

172600 + 26 + 172626

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. AC v 1 = 11.

The information in Table 5-2 is expressed in symbolic notation to provide the reader with a quick reference to the function of each instruction. The following paragraphs supplement the information in Table 5-2.

Table 5-2 FP11-A Instruction Set

Mnemonic	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

Table 5-2 FP11-A Instruction (Cont)

Mnemonic	Instruction Description	Octal Code

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

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

Table 5-2 FP11-A Instruction (Cont)

Mnemonic	Instruction Description	Octal Code

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

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 FP11-A'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

Table 5-2 FP11-A Instruction (Cont)

Mnemonic	Instruction Description	Octal Code

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*100FSRC
		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

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

Table 5-2 FP11-A Instruction (Cont)

Mnemonic	Instruction Description	Octal Code

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

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.	

Table 5-2 FP11-A Instruction (Cont)

Mnemonic	Instruction Description	Octal Code

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 FP11-A's Program Status Word DST ← (FPS)	170200+DST F4 Format
STST DST	Store FP11-A'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

5.3.1 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 FP11-A 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 FP11-A, and there is no transfer of the result.

5.3.2 Floating-Modulo Instruction

The Floating-Modulo (MOD) instruction causes the FP11-A 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 whole-number portion goes into an odd-numbered accumulator and the fraction goes into an even-numbered accumulator.

The whole-number portion of the number, when expressed as a floating-point number, contains an exponent greater than 201 in excess 200 notation, which means that the whole number has a decimal value of some number greater than one and less than UPLIM, where UPLIM is the greatest possible number that can be represented by the FP11-A.

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 that the fraction has a value less than one and greater than LOLIM, where LOLIM is the smallest possible number that can be represented by the FP11-A.

5.3.3 Load Instruction

The Load instruction causes the FP11-A to take an operand from a source and copy it into a destination accumulator. The source is a floating-point accumulator in mode 0 and a memory location otherwise.

5.3.4 Store Instruction

The Store instruction causes the FP11-A to take an operand from a source accumulator and transfer it to a destination. This destination is a floating-point accumulator in mode 0 and a memory location otherwise.

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

The Load Convert Double-to-Floating (LDCDF) instruction causes the FP11-A to assume that the source specifies a double-precision floating-point number. The FP11-A 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 if the MSB of the double-precision segment is a 1 depending on the prior conditions set up by the FD bit (Figure 5-3). If the MSB of the double-precision segment is 0, the single-precision word remains unchanged after rounding.

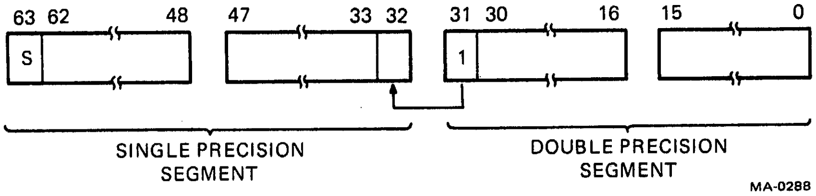


Figure 5-3 Double-to-Single Precision Rounding

The Load Convert Floating-to-Double (LDCFD) instruction causes the FP11-A to assume that the source specifies a single-precision number. The FP11-A then converts that number to double-precision by appending 32 zeros 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.

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

The Store Convert Double-to-Floating (STCDF) instruction causes the FP11-A to convert a double-precision number located in the source accumulator to a single-precision number. The FP11-A then transfers this result to the specified destination. If the floating-truncate (FT) bit 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, 1 is added to the single-precision segment of the word, depending on the prior conditions set up by the FD bit (Figure 5-3); otherwise, the single-precision segment remains unchanged.

The Store Convert Floating-to-Double (STCFD) instruction causes the FP11-A to convert a single-precision number located in the source accumulator to a double-precision number. The FP11-A then transfers this result to the specified destination. The single-to-double precision is obtained by appending zeros equivalent to the double-precision segment of the word (Figure 5-4).

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.

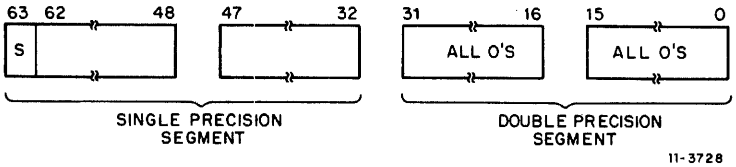


Figure 5-4 Single-to-Double Precision Appending

5.3.7 Clear Instruction

The Clear instruction causes the FP11-A to clear a floating-point number by setting all its bits to 0.

5.3.8 Test Instruction

The Test instruction causes the FP11-A to test the sign and exponent of a floating-point number and update the FP11-A 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 zero. If the FIUV status bit is set, a trap occurs (after the test instruction is executed) if a minus zero is encountered.

5.3.9 Absolute Instruction

The Absolute instruction causes the FP11-A 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, zeros are written into the accumulator. If the exponent is non-zero, the accumulator is unaffected.

If mode 0 is not specified, the sign bit of the specified data word in memory is zeroed. The exponent of this word is tested, and if it is 0, the entire data word in memory is zeroed. If the exponent is non-zero, the integer exponent is restored to memory.

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

5.3.10 Negate Instruction

The Negate instruction causes the CPU (or the FP11-A, 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, zeros are written into the accumulator. If the exponent is non-zero, 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 non-zero, the original fraction and exponent are restored to memory.

5.3.11 Load Exponent Instruction

The Load Exponent instruction causes the floating-point processor (FPP) 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 to an 8-bit number in excess 200 notation. This process is described further 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 from 000000 to 177777_8. On the other hand, the possible legal range of exponents in the FP11-A falls into two classes.

1. Positive exponents (0 through 177) - When 200 is added to any of these numbers, the sum stays within the legal 8-bit exponent field (i.e., from 200 through 377).
2. Negative exponents (177601 through 177777) - When 200 is added to any of these numbers, the sum stays within the legal 8-bit exponent field (i.e., from 1 through 177).

Notice that all legal positive exponents coming from memory have something in common: their 9 high-order bits are all 0s. Similarly, all legal negative exponents from memory have their 9 high-order bits equal to 1. Therefore, to detect a legal exponent, only the 9 high-order bits need be examined for all 1s or all 0s.

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

Example 1: LDEXP 000034

```
Exponent of 34          00000000  |  00011100
 200                  +           |  10000000
                      ------------+----------
                                  |  10011100
                                  |   2  3  4
```

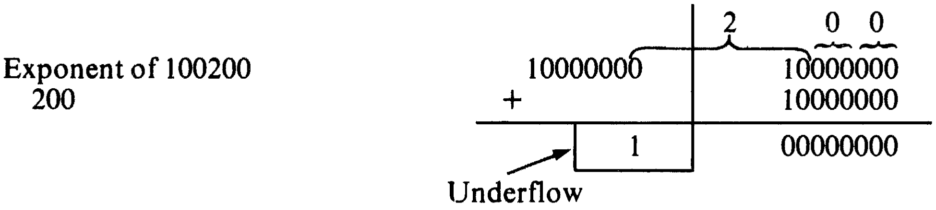
The upper 9 bits all equal 0, so this is a legal positive exponent. The number 234 is sent to the 8-bit exponent field of the specified accumulator.

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 when 200 is added to the exponent, an overflow occurs.

Example 3: LDEXP 100200



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

$$\begin{array}{r|l} 11111111 & 10000000 \\ +\quad 0 & 10000000 \\ \hline 00000000 & 00000000 \end{array}$$

This is the one case where the 9 high-order bits are all equal, but the exponent is illegal. This is 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.

5.3.12 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 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 mode is specified. The integer is loaded into bits 55–40 if short integer is specified or into bits 55–24 if long integer is specified. It is then left-shifted eight places so that bit 55 is transferred to bit 63 (Figure 5-5).

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40
0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	1
1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0

11-5307

Figure 5-5 Integer Left-Shift Example

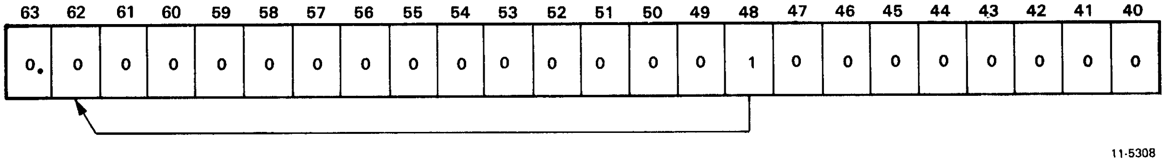
The integer is then assigned an exponent of 217_8 short integer. This is the result of adding 200_8 (since the exponent is expressed in excess 200 notation) to 17_8 which represents 15_{10} shifts. This number of shifts is the maximum number required to normalize a number. If long-integer mode is specified,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

The 2's complement integer is tested by examination of bit 63 to see if it is a positive or negative number. The number is then 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 (Figure 5-6).

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

When loading a long integer with an FD = 0, if the long integer contains more than 24 significant digits, then less significant digits will be truncated with some loss of accuracy.



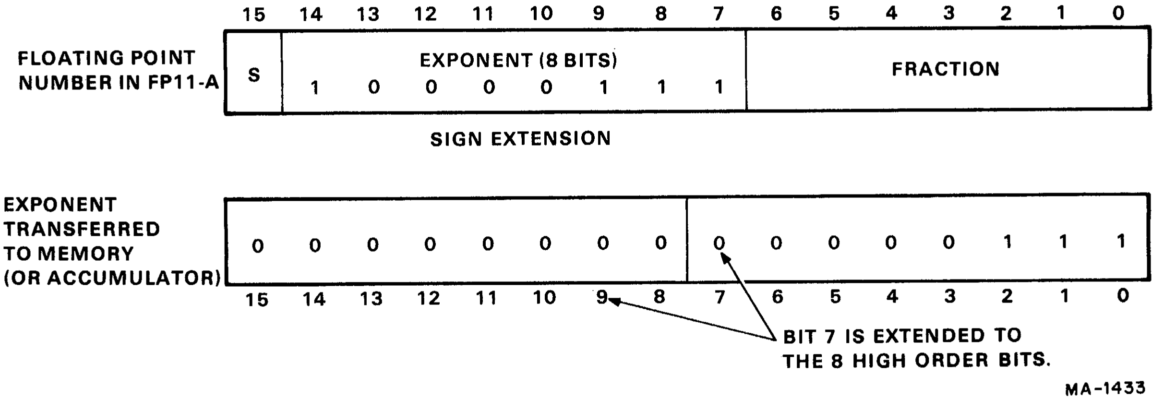
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40
0.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 5-6 Normalized Integer Example

5.3.13 Store Exponent Instruction

The Store Exponent (STEXP) instruction causes the CPU to access a floating-point number in the FP11-A, 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 8 high-order bits.

The legal range of exponents is from 0 to 377, 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 (Figures 5-7 and 5-8).



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLOATING POINT NUMBER IN FP11-A	S	EXPONENT (8 BITS) 1	0	0	0	0	1	1	1	FRACTION						
EXPONENT TRANSFERRED TO MEMORY (OR ACCUMULATOR)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Figure 5-7 Store Exponent Example No. 1

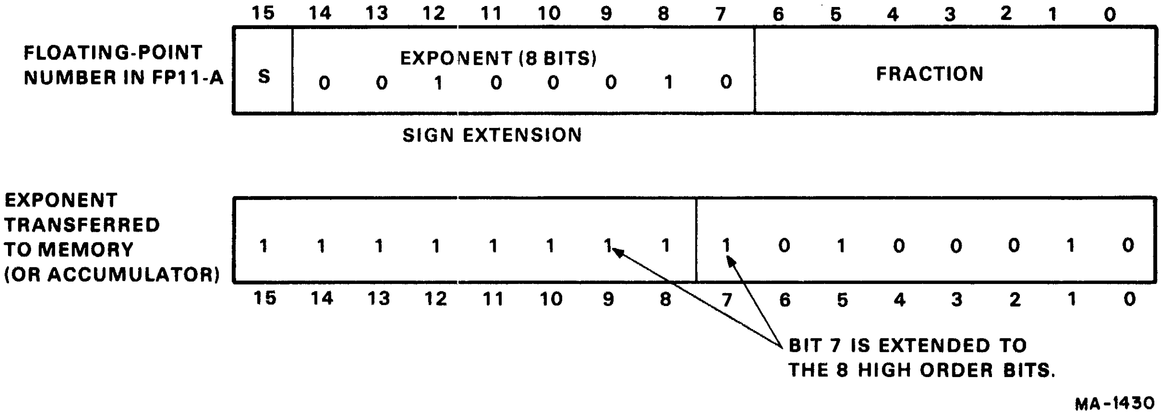


Figure 5-8 Store Exponent Example No. 2

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

Example 1: Exponent = 207

$$\begin{array}{lr} \text{Exponent of 207} & 10000111 \\ \text{2's Complement of 200} & +10000000 \\ \hline \text{Result} = 7 & \underset{\text{Sign Bit}}{0}0000\underbrace{111}_{7} \end{array}$$

Example 2: Exponent = 42

$$\begin{array}{lr} \text{Exponent of 42} & 00100010 \\ \text{2's Complement of 200} & +10000000 \\ \hline \text{Result} = -42 & \underset{\text{Sign Bit}}{1}0\underbrace{100}_{4}\underbrace{010}_{2} \end{array}$$

5.3.14 Store Convert Floating-to-Integer Instruction

The Store Convert Floating-to-Integer instruction causes the FPP to take a floating-point number and convert it to an integer for transfer to a destination.

The four classes of this instruction are as follows.

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

The (normalized) floating-point number to be converted is transferred to the FPP. The FPP works with the sign bit and one 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 zero), it indicates that the exponent is greater than (or equal to) 201, and the floating-point number can be converted to a non-zero integer (Figure 4-9).

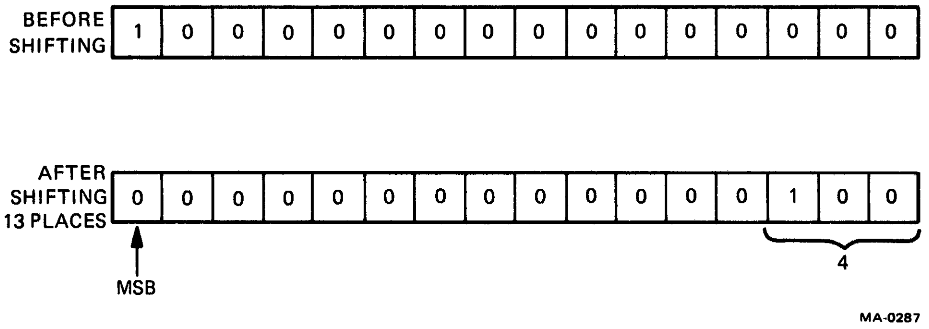


Figure 5-9 Store Convert Integer Example

A second test is made by the FPP 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.

	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-formats, 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, it indicates that the floating-point number is too large to be represented as an integer. In that case, 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 (short) or 240 (long), and converson proceeds. However, the floating-point number is within range only if its sign is negative and its fraction is .100 . . . (i.e., if it is the most negative integer; see table above). If, in this case, the number is not the most negative integer, it will be detected by a third conversion error test (see below) after conversion.

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

Short integer:	No. of right shifts = $20_8 + 201_8$ - biased exponent - 1
Long integer:	No. 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. If, however, the floating-point number is negative, the integer must be 2's complemented before being 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 these signs are not equal, there has been a conversion error 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 (short) or 240 (long) 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 X 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.

5.3.15 Load FP11's Program Status

This 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 for use by the FP11-A in order to enable and disable interrupts, set and clear mode bits, and set condition codes (Paragraph 4.4).

5.3.16 Store FP11's Program Status

This instruction causes the FPP to transfer the 16 bits of the FPS register to the specified destination.

5.3.17 Store FP11's Status

The Store FP11'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 FEA data occupies all 16 bits of its memory location, while the FEC 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 and issues the STST instruction to determine the type of error.

NOTE

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

5.3.18 Copy Floating Condition Codes

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

5.3.19 Set Floating Mode

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.

5.3.20 Set Double Mode

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.

5.3.21 Set Integer Mode

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.

5.3.22 Set Long-Integer Mode

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.

5.4 FP11-A PROGRAMMING EXAMPLES

This paragraph contains two programming examples using the FP11-A 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 6
PROCESSOR ORGANIZATION

6.1 INTRODUCTION

The FP11-A Floating-Point Processor connects to the KD11-EA central processor via a tri-state bus (Figure 6-1). This interface allows addressing of floating-point memory utilizing the memory management option.

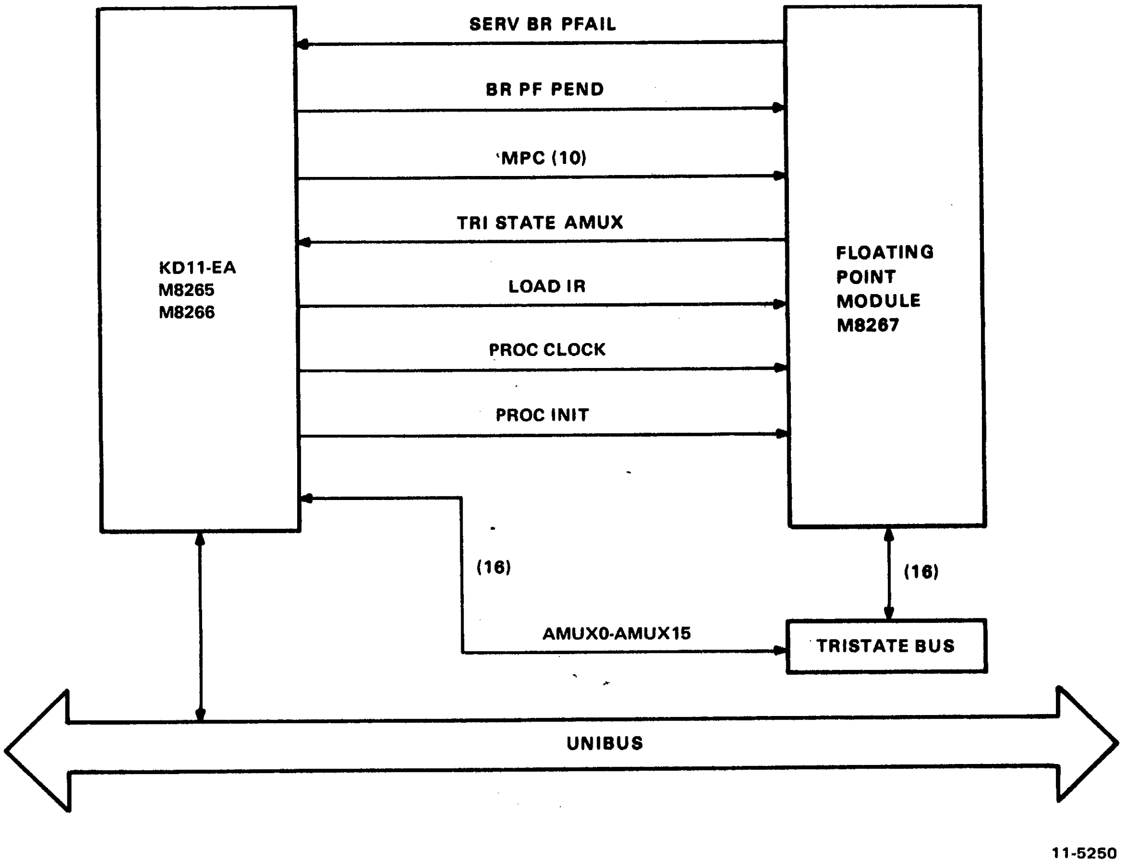


Figure 6-1 KD11-EA/FP11-A Data Flow

The CPU software must initiate floating-point operation and originate addresses and data since control of the FP11-A resides in the CPU.

The FP11-A depends on the CPU to fetch instructions and data from memory in order to initiate floating-point operations. If the instruction is not a floating-point instruction, it is ignored by the FP11-A. If the decoded instruction is a floating-point instruction (i.e., contains an op code of 17XXXX), the FP11-A causes the CPU to branch to the FP11-A ROM (read-only memory) microstates associated with floating-point instructions.

The simplified block diagram illustrated in Figure 6-2 shows the major functions of the FP11-A.

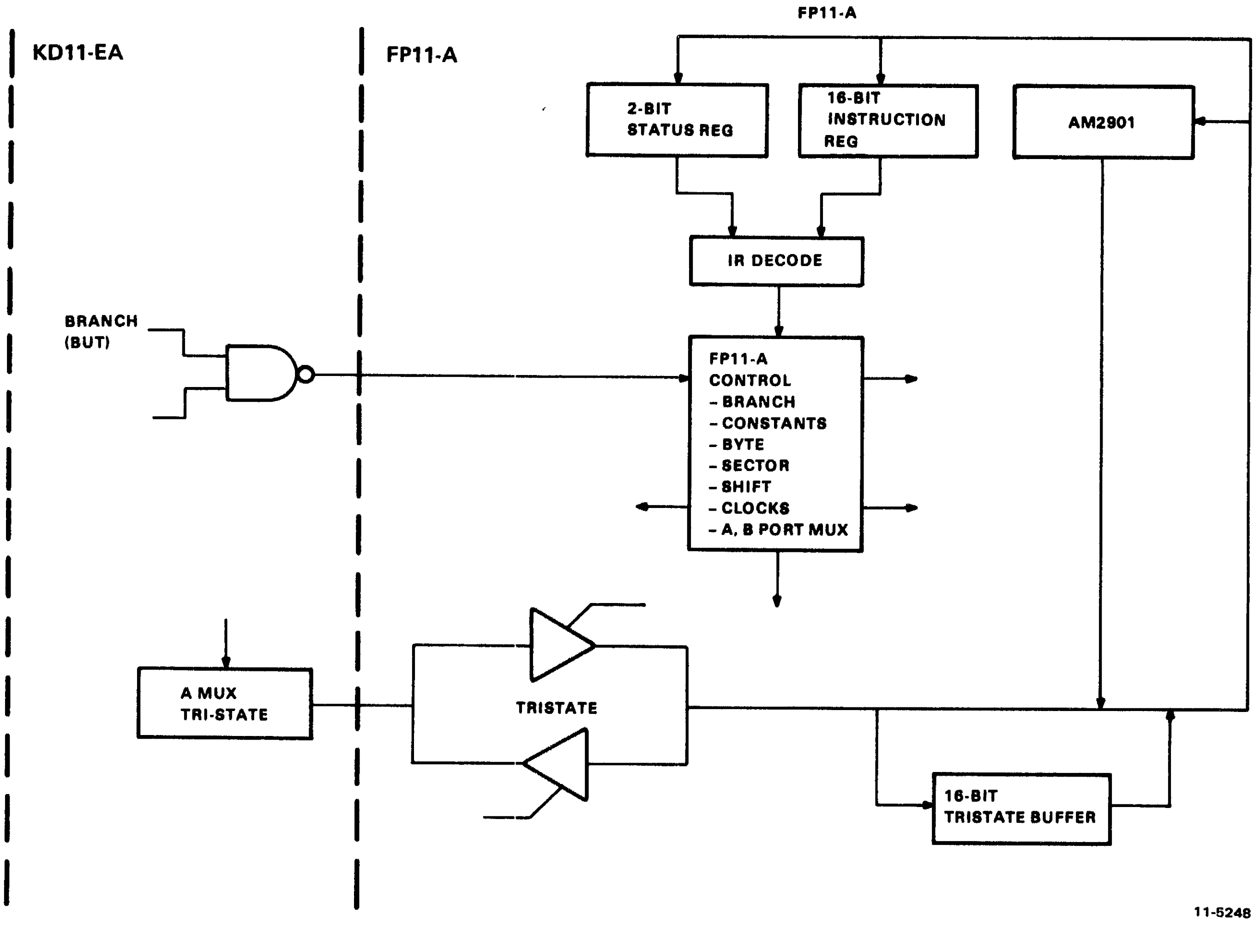
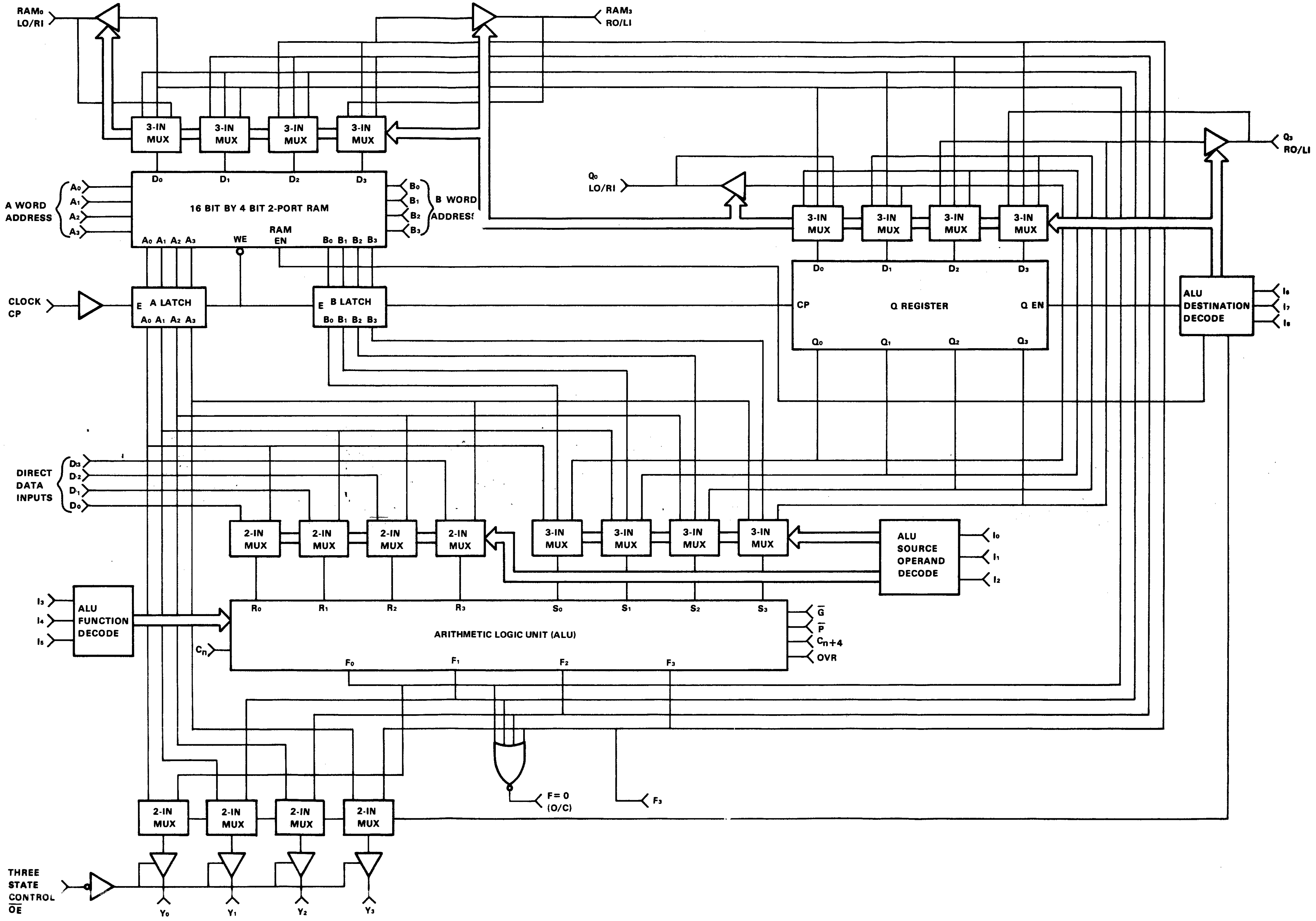


Figure 6-2 Simplified FP11-A Block Diagram

6.2 MICROPROCESSOR DESCRIPTION

The principal data manipulation element in the floating-point processor is the AM2901 microprocessor (Figure 6-3). The basic microprocessor is 4 bits wide and 16 of these units are cascaded to make up a 16 by 64-bit word for the FP11-A. A general discussion of the microprocessor followed by a description of its integration into the overall floating-point processor is given in the following paragraphs.



11-5267

Figure 6-3 Microprocessor (AM2901) Block Diagram

6.2.1 Microprocessor Organization

As shown in Figure 6-3, the major components of the microprocessor are the RAM, the arithmetic logic unit (ALU), and the Q-register.

Information contained in any of the 16 64-bit words of the RAM may be read from the A-port as controlled by the 4-bit A-word address (A_0–A_3) field. Similarly, data in any of the 16 words of the RAM as defined by the B-address (B_0–B_3) field input may be simultaneously read from the B-port of the RAM. It is also possible to apply the same address code to both the A and B select fields, in which case the identical file data will appear at both the RAM A-port and B-port simultaneously.

New data is always written into the file word specified by the B-address field of the RAM. The RAM data input field is driven by a 3-input multiplexer which permits shifting of the ALU output. The 3-input multiplexer allows data to be shifted right 1 bit position, left 1 bit position, or not shifted in either direction.

6.2.2 Arithmetic/Logical Operations

The arithmetic logic unit (ALU) is capable of performing three arithmetic and five logical operations on the two 4-bit input words R_0–R_3 and S_0–S_3. The R-input field to the ALU receives its input from a 2-input multiplexer while S receives its signals from a 3-input multiplexer. The 2- and 3-input multiplexers both have an inhibit capability. This is the equivalent of an "O" source operand.

If the five data inputs to the ALU are combined into pairs, 10 combinations of registers are possible, i.e., AB, DA, AQ, 0A, 0B, BQ, BD, D0, DQ, and 0Q, as illustrated in Table 6-1. The microprocessor uses eight of these operand pairs. Selection of the ALU source operand pairs is accomplished by the microinstruction inputs I0, I1, and I2.

Table 6-1 ALU Source Operand Contest

Microcode				ALU Source Operands	
I_2	I_1	I_0	Octal Code	R	S
L	L	L	0	A	Q
L	L	H	1	A	B
L	H	L	2	O	Q
L	H	H	3	O	B
H	L	L	4	O	A
H	L	H	5	D	A
H	H	L	6	D	Q
H	H	H	7	D	O

The direct-data (D) source-operand input port is used to insert all data into the working registers inside the 2901 microprocessor. The D-input can also be used by the ALU to modify any of the internal data files of the RAM via the F outputs of the ALU.

The Q-register is a separate 4-bit register intended primarily for multiplication and division routines. This register can also be used as an accumulator or buffer register for certain applications.

The ALU performs three arithmetic and five logical functions as directed by the three control bits I3, I4, and I5 (Table 6-2).

Table 6-2 ALU Function Control

Microcode					
I_5	I_4	I_3	Octal Code	ALU Function	Symbol
L	L	L	0	R Plus S	$R + S$
L	L	H	1	S Minus R	$S - R$
L	H	L	2	R Minus S	$R - S$
L	H	H	3	R OR S	$R \vee S$
H	L	L	4	$\underline{R}$ AND S	$\underline{R} \wedge S$
H	L	H	5	$\bar{R}$ AND S	$\bar{R} \wedge S$
H	H	L	6	R EX OR S	$R \forall S$
H	H	H	7	R EX NOR S	$\overline{R \forall S}$

ALU output data may be routed to one of eight possible destinations as defined by control bits I6, I7, and I8. ALU output data may be a data output from the device or it may be stored in the RAM or the Q-register.

The data output of the microprocessor uses a 2:1 multiplexer whose inputs are the A-port of the RAM or the ALU outputs (F). Selection of these outputs is controlled by bits I6, I7, and I8 of the microinstruction control input (Table 6-3). *Note that the left- and right-shift functions in Table 6-3 are reversed for the FP11-A application.*

The FP11-A uses 16 AM2901 units connected in cascade with 3 levels of look-ahead carry logic. This configuration results in a 64-bit word. Carry generate (G), and carry propagate (P), are unit outputs for use with a look-ahead carry generator. Carry out (C_{n+4}) is also generated by the microprocessor and is available as an output carry flag in a status register. C_n and C_{n+4} are both active high.

Three additional outputs are generated by the ALU. These are F3, F = 0, and overflow (OVR). F3 represents the most significant bit (sign) of the ALU and can be used to determine positive or negative results without enabling the 3-state outputs or while enabling the A-port to output. F3 is non-inverted with respect to the sign bit output Y3. F = 0 output is used for zero detect and is an open-collector output that can be wire ORed between microprocessor slices. F = 0 is high when all F outputs are low. OVR is a flag indicating an arithmetic operation exceeds the available range and is high when the overflow condition exists, i.e., when C_n and C_{n+4} are not of the same polarity.

Inputs to the RAM are via a 3-input multiplexer. The multiplexer allows input data to be entered into the RAM in three modes:

- Shifted left one place
- Shifted right one place
- Unshifted.

The shifting is accomplished by two ports: RAM-LO/RI and RAM-RO/LI. Both ports consist of a buffer driver with a tri-state output and an input to the multiplexer. In the shift-up (X2) mode, the RO buffer is enabled and the RI multiplexer input is enabled. In the shift-down (-2) mode, the LO buffer and LI input are enabled. In the no-shift mode, both the LO and RO buffers are in the high-impedance state and the multiplexer inputs are not selected. The microinstruction control bits I_6, I_7, and I_8 operate the shifter as shown in Table 6-3.

Table 6-3 ALU Destination Control

Microcode				RAM Function		Q-Register Function			RAM Shifter		Q Shifter	
I_8	I_7	I_6	Octal Code	Shift	Load	Shift	Load	Y Output	RAM_0 LO/RI	RAM_3 LI/RO	Q_0 LO/RI	Q_3 LI/RO
L	L	L	0	—	—	None	ALU (F_i)	F	X	X	X	X
L	L	H	1	—	—	—	—	F	X	X	X	X
L	H	L	2	None	ALU (F_i)	—	—	A	X	X	X	X
L	H	H	3	None	ALU (F_i)	—	—	F	X	X	X	X
H	L	L	4	Left (Down)	ALU (F_{i+1})	Left (Down)	Q-Reg (Q_{i+1})	F	F_0	IN_3	Q_0	IN_3
H	L	H	5	Left (Down)	ALU (F_{i+1})	—	—	F	F_0	IN_3	Q_0	X
H	H	L	6	Right (Up)	ALU (F_{i-1})	Right (Up)	Q-Reg (Q_{i-1})	F	IN_0	F_3	IN_0	Q_3
H	H	H	7	Right (Up)	ALU (F_{i-1})	—	—	F	IN_0	F_3	X	Q_3

X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three state output which is in the high impedence state.

The Q-register is also driven from a 3-input multiplexer. In the no-shift mode, the multiplexer enters ALU data into the Q-register. Operation for the shift-up or shift-down modes is the same as for the RAM as indicated in Table 6-3.

The RAM, Q-register, and the A and B data latches are controlled by the clock input. When enabled, data latches are also controlled by the clock input and data is clocked into the Q-register on the positive-going transition of the clock. When the clock input is high, the A and B data latches are open and will pass any data that is present at the RAM outputs. When the clock is low, the latches are closed and will retain the last data entered. If the RAM-EN is enabled, new data is entered into the RAM file (word) defined by the B-address field when the clock input is low.

6.2.3 RAM

The FP11-A RAM register usage is shown in Figure 6-4. This unit, located in the microprocessor, is the scratchpad area where the results of arithmetic and logical operations are temporarily stored. The contents of the RAM are read into the ALU under control of the FP11-A microcode. It consists of 16 64-bit words (each of the 16 microprocessors (AM2901) contains a 16- X 4-bit RAM).

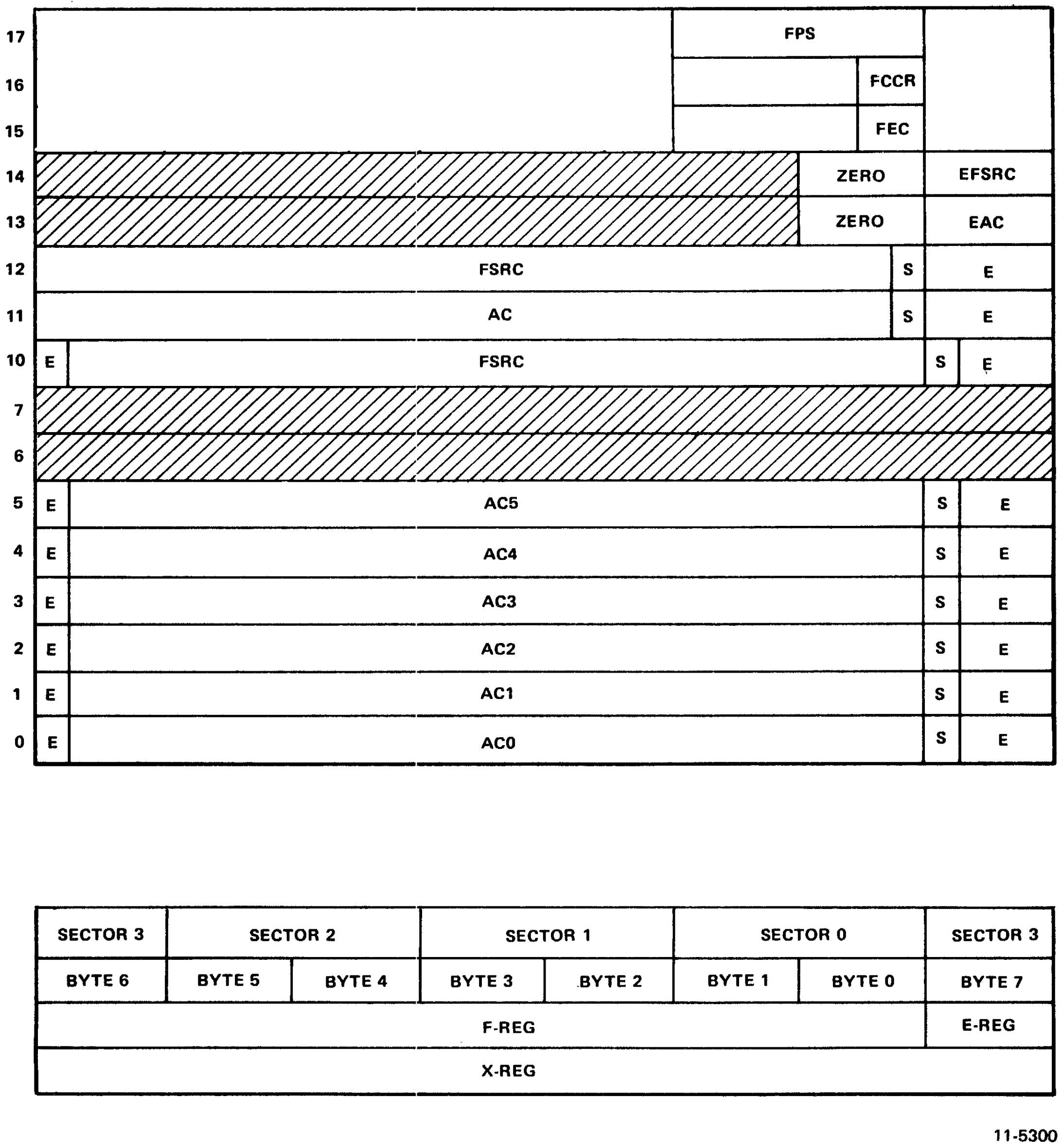


Figure 6-4 RAM Register Usage

Six of the 64-bit registers are allocated for the accumulators and are accessible to the programmer via the FP11-A 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 control ROM. Registers 10–14 constitute a working storage area for the FP11-A microcode. Other functions included are the floating-point status register, condition codes, and exception codes.

6.2.4 Arithmetic Logic Unit (ALU)

The ALU is the data path component that actually performs the arithmetic/logical operation under command of the microcode (Table 6-4). R-inputs are fed in via a 2-input multiplexer whose inputs are the direct data (D) inputs and the output of the A-port of the RAM. The S-inputs include the A- and B-ports of the RAM and the Q-register outputs.

Table 6-4 Source Operand and ALU Function Matrix

	$I_{2\,1\,0}$ Octal	0	1	2	3	4	5	6	7
$I_{5\,4\,3}$ Octal	ALU Source ALU Function	A,Q	A,B	O,Q	O,B	O,A	D,A	D,Q	D,O
	C_n=L	A+Q	A+B	Q	B	A	D+A	D+Q	D
0	R Plus S								
	C_n=H	A+Q+1	A+B+1	Q+1	B+1	A+1	D+A+1	D+Q+1	D+1
	C_n=L	Q-A-1	B-A-1	Q-1	B 1	A 1	A-D-1	Q-D-1	-D-1
1	S Minus R								
	C_n=H	Q-A	B-A	Q	B	A	A-D	Q-D	-D
	C_n=L	A-Q-1	A-B-1	-Q-1	-B-1	-A-1	D-A-1	D-Q-1	D-1
2	R Minus S								
	C_n=H	A-Q	A-B	-Q	B	-A	D-A	D-Q	D
3	R OR S	A∨Q	A∨B	Q	B	A	D-A	D∨Q	D
4	R AND S	A∧Q	A∧B	0	0	0	D∧A	D∧Q	0
5	R AND S	$\overline{A}\wedge Q$	$\overline{A}\wedge B$	Q	B	A	$\overline{D}\wedge A$	$\overline{D}\wedge Q$	0
6	R EX-OR S	A∀Q	A∀B	Q	B	A	D∀A	D∀Q	D
7	R EX-NOR S	$\overline{A\forall Q}$	$\overline{A\forall B}$	Q	B	A	$\overline{D\forall A}$	$\overline{D\forall Q}$	D

+ = Plus; – = Minus; ∨= OR, ∧ = AND; ∀ = EX OR

ALU output data (F) may be routed to the Q-register or RAM, or may be multiplexed with the A-port output data from the RAM as Y_0–Y_3. The ALU function decode determines the arithmetic or logical function to be performed, while the ALU destination decode determines which of the indicated registers the data is routed to or whether it will be a. data output of the device itself.

The ALU source operand decode performs the actual register selection. All three of these functions are controlled by bits I0–I8 of the control word.

6.2.5 Q-Register

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

6.2.6 Source Operands and ALU Functions

This paragraph summarizes the arithmetic and logic functions performed by the ALU and presents ALU logic and arithmetic functions in separate tabulations.

6.2.6.1 Logical and Arithmetic Functions – The ALU performs five logical and three arithmetic functions on eight source operand pairs. ALU logic functions and appropriate control bit values (I0–I5) are shown in Table 6-5. The carry input (C_n), has no effect in logic mode but does affect operations in arithmetic mode (Table 6-6). Both carry-in LOW (C_n = 0) and carry-in HIGH (C_n = 1) are defined.

6.2.6.2 Logical Functions for G, P, C_{n+4} , and OVR – The four signals, G, P, C_{n+4}, and OVR, as described in Paragraph 6.2 are designed to indicate carry and overflow conditions when the microprocessor is in the add or subtract mode. Table 6-7 indicates the logic equations for these four signals for each of the eight ALU functions. The R- and S-inputs are the two inputs selected according to Table 6-8.

6.2.7 Summary of Pin Definitions

The AM2901 pin definitions are summarized in Table 6-7. Pin assignments for the AM2901-40-pin dual in-line package are shown in Figure 6-5.

6.3 INSTRUCTION STATUS REGISTERS AND DECODE

The FP11-A contains a 12-bit instruction register and two flip-flop's that are bits of the status registers (FD;FL). The possible FP11-A instruction formats are presented in Chapter 2. One bit of the status register (FD) specifies double- or single-precision format and the other (FL) designates single- or double-precision integer format. The outputs of these registers are fed to the floating-point instruction decode register which consists of two 512- × 4-bit ROMS. The ROM outputs then generate microprocessor control (MPC) outputs to control the microprogram.

6.4 TRI-STATE TRANSCEIVERS AND BUFFER

The 8097 tri-state status gales are arranged as tri-state transceivers to communicate between the AMUX bus (KD11-EA) and the T-bus (FP11-A), which are 16-bit tri-state buses.

74S173s, which are tri-state flip-flops, are used to buffer Unibus data being passed to the AM2901s.

6.5 BRANCH LOGIC AND TRI-STATE CONTROL

This logic is controlled by condition code and T-bus branch ROMs in the FP11-A. Branch (BUT) bits are routed to the ROMs whose outputs condition a group of gates. These gates are enabled by the various branch conditions that may arise during floating-point operations and direct (point) the microprogram counter to the appropriate code in the microprogram to service the branch function. The BUT conditions include the condition codes, exponent negative, exponent zero, carry, overflow, and T-bus bits 0, 1, 5, 6, 8, 9, 10, 11, and 14. These functions include items such as fraction Z-bit, fraction negative, bus request, and the like.

Table 6-5 ALU Logic Mode Functions

Octal I_{543}, I_{210}	Group	Function
4 0	AND	$A \wedge Q$
4 1		$A \wedge B$
4 5		$D \wedge A$
4 6		$D \wedge Q$
3 0	OR	$A \vee Q$
3 1		$A \vee B$
3 5		$D \vee A$
3 6		$D \vee Q$
6 0	EX OR	$A \forall Q$
6 1		$A \forall B$
6 5		$D \forall A$
6 6		$D \forall Q$
7 0	EX NOR	$\overline{A \vee Q}$
7 1		$\overline{A \vee B}$
7 5		$\overline{D \vee A}$
7 6		$\overline{D \vee Q}$
7 2	Invert	$\overline{Q}$
7 3		$\overline{B}$
7 4		$\overline{A}$
7 7		$\overline{D}$
6 2	Pass	Q
6 3		B
6 4		A
6 7		D
3 2	Pass	Q
3 3		B
3 4		A
3 7		D
4 2	"Zero"	0
4 3		0
4 4		0
4 7		0
5 0	Mask	$\overline{A} \wedge Q$
5 1		$\overline{A} \wedge B$
5 5		$\overline{D} \wedge A$
5 6		$\overline{D} \wedge Q$

Table 6-6 ALU Arithmetic Mode Functions

Octal	$C_n = 0$ (Low)		$C_n = 1$ (High)	
I_{543}, I_{210}	Group	Function	Group	Function
0 0	ADD	A+Q	ADD plus one	A+Q+1
0 1		A+B		A+B+1
0 5		D+A		D+A+1
0 6		D+Q		D+Q+1
0 2	PASS	Q	Increment	Q+1
0 3		B		B+1
0 4		A		A+1
0 7		D		D+1
1 2	Decrement	Q-1	Pass	Q
1 3		B-1		B
1 4		A-1		A
2 7		D-1		D
2 2	1's Comp.	-Q-1	2's Comp. (Negate)	-Q
2 3		-B-1		-B
2 4		-A-1		-A
1 7		-D-1		-D
1 0	Subtract (1's Comp.)	Q-A-1	Subtract (2's Comp.)	Q-A
1 1		B-A-1		B-A
1 5		A-D-1		A-D
1 6		Q-D-1		Q-D
2 0		A-Q-1		A-Q
2 1		A-B-1		A-B
2 5		D-A-1		D-A
2 6		D-Q-1		D-Q

Table 6-7 Logic Equations for ALU Functions

Definitions (+ =OR)

$P_0 = R_0 + S_0$ $\quad$ $G_0 = R_0 S_0$

$P_1 = R_1 + S_1$ $\quad$ $G_1 = R_1 S_1$

$P_2 = R_2 + S_2$ $\quad$ $G_2 = R_2 S_2$

$P_3 = R_3 + S_3$ $\quad$ $G_3 = R_3 S_3$

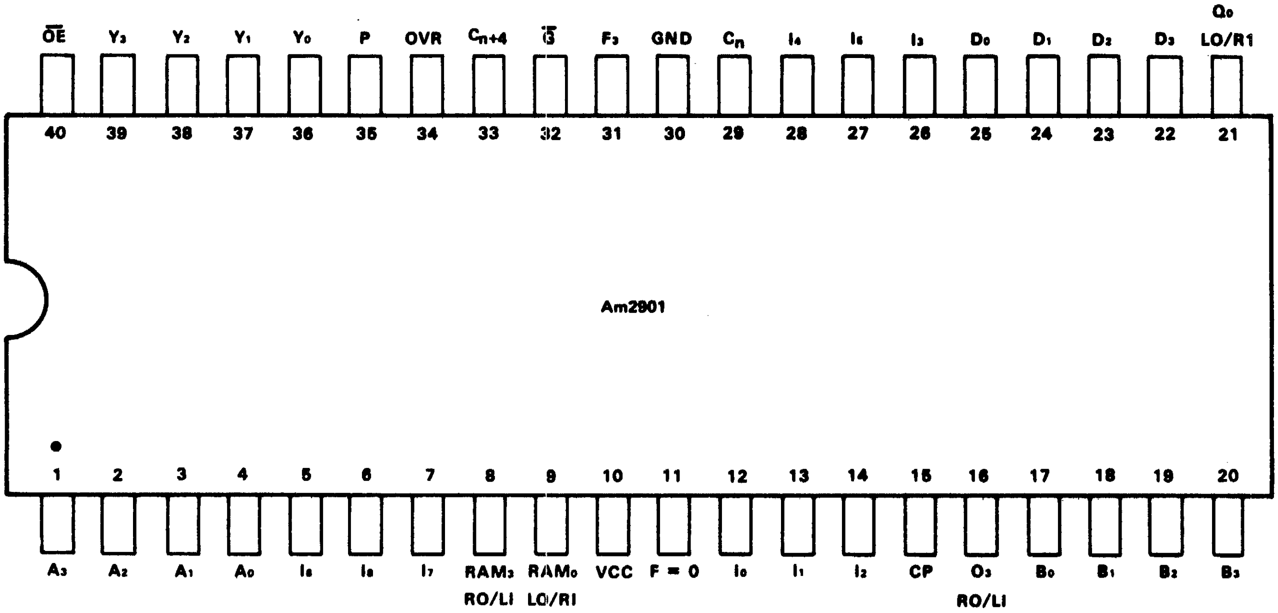
$$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_n$$

$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_n$$

Table 6-8 P, G, C_{N+4}, OVR Functions

I_{543}	Function	P	G	C_{n+4}	OVR
0	R + S	$\overline{P_3 P_2 P_1 P_0}$	$\overline{G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0}$	C_4	$C_3 \quad C_4$
1	S − R	← Same as R + S equations, but substitute $\overline{R}_1$ for R_1 in definitions →			
2	R − S	← Same as R + S equations, but substitute $\overline{S}_1$ for S_1 in definitions →			
3	R ∨ S	Low	$P_3 P_2 P_1 P_0$	$\overline{P_3 P_2 P_1 P_0} + C_n$	$\overline{P_3 P_2 P_1 P_0} + C_n$
4	R ∧ S	Low	$\overline{G_3 + G_2 + G_1 + G_0}$	$G_3 + G_2 + G_1 + G_0 + C_n$	$\overline{G_3 + G_2 + G_1 + G_0} + \overline{C}_n$
5	$\overline{R}$ ∧ S	Low	← Same as R ∧ S equations, but substitute $\overline{R}_1$ for R_1 in definitions →		
6	R ∀ S	← Same as $\overline{R \forall S}$, but substitute $\overline{R}_1$ for R_1 in definitions →			
7	$\overline{R \forall S}$	$G_3 + G_2 + G_1 + G_0$	$P_3 G_3 + P_3 P_2 G_2 + P_3 P_2 P_1 G_1 + P_3 P_2 P_1 P_0$	$P_3 G_3 + P_3 P_2 G_2 + P_3 P_2 P_1 G_1$ $+ P_3 P_2 P_{S1} P_0 (G_0 + \overline{C}_n)$	Complement of C_{n+4} at left

+ = OR



NOTE: PIN 1 IS MARKED FOR ORIENTATION.

11-5247

Figure 6-5 AM2901 Pin Connections

6.6 CONSTANTS, BYTE AND SECTOR CONTROL, SHIFT CONTROL

The constant ROMs contain the 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-precisionend short or long integer. Thus, FD and FL are used as ROM-gating signals for proper constant selection.

The BYTE control lines enable AM2901 outputs onto the T-bus by 8-bit bytes. Any high-byte/low-byte combination may be enabled.

Sector control is used to independently select one of four 16-bit sectors of the AM2901. Each sector clock clocks four AM2901 slices (16 bits) which are used internally to load the RAM or Q-register.

Shift and rotate signals are generated to operate the input multiplexers to the RAM and Q-registers of the AM2901. These registers may be left and right shifted and controls are provided for injection of 1s or 0s into the bit stream as shifts are carried out.

CHAPTER 7
MAINTENANCE

7.1 INTRODUCTION

This chapter describes some of the maintenance tools and techniques available for maintenance of the FP11-A floating-point option. Descriptions of the diagnostics, programmer's console, display features, and documentation aids are also included.

7.2 FP11-A DIAGNOSTICS

Three diagnostics are available to validate and diagnose the FP11-A. However, since the KD11-EA data path is used extensively on floating-point instructions, 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 KD11-EA failure may cause only floating-point instructions to fail. The three FP11-A diagnostics are listed below with a short description of each. The diagnostics should be run in the same order as they are listed because succeeding diagnostics have been run successfully. Otherwise, faulty diagnosis of the failed micro-step and where the problem is located may result.

7.2.1 MAINDEC DFFPAA

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)

7.2.2 MAINDEC DFFPBA

This diagnostic tests the following floating-point instructions.

ADDF, ADDD, and SUBD (all conditions not listed in DFFPAA)
CMPD and CMPF
DIVD and DIVF
MULD and MULF
MODD and MODF

This diagnostic also makes use of a special testing module (M8267-TA), which allows the diagnostic to check the ability of the floating-point to abort an ADD, SUB, MVC, DIV, or MOD instruction if an interrupt request occurs during the initial portion of one of these instructions. The extra hardware tested using the special test module is minimal and it is expected to be used only during manufacturing for more complete testing. The diagnostic automatically checks for the test module, and only if present, performs the special instruction abort test. A message at the beginning of the program indicates the presence of the test module and its use by the diagnostic. If the module is not present, no message is generated.

7.2.3 MAINDEC DFFPCA

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

7.3 KY11-LB PROGRAMMER'S CONSOLE

Normal console and maintenance features provided by the KY11-LB programmer's console to debug and diagnose the KD11-EA processor are directly extendable in use to the FP11-A floating-point option. These features include the normal console functions of examining and depositing into memory and general registers, single-instruction stepping, the console maintenance features of single micro-instruction stepping, and displaying MPC lines, Unibus data, and Unibus address lines.

The KY11-LB displays the additional MPC line (MPC 09 L) if the proper cable connections between the KY11-LB and FP11-A modules are made. Thus, single micro-stepping the machine through floating-point micro-code is possible.

A change in the KD11-EA processor from the KD11-E processor enables the AMUX lines onto the Unibus data lines in the manual clock mode. (KY11-LB maintenance cables are attached, the console is in MAINT mode, and the HLT/SS key has been depressed.) The AMUX to Unibus drivers are not enabled, however, if the current micro-step is a DATI, at which time some other device (memory, I/O) will be driving the Unibus data lines. Since the console can display the Unibus data lines (EXAM key in MAINT mode), the AMUX lines are being indirectly displayed most of the time. This new feature is directly extendable to the FP11-A in that the AMUX lines are the data path link between the KD11-EA and the FP11-A. At any micro-step, the AMUX lines may be displayed and while running floating-point micro-code, the T-bus lines of the FP11-A are defaulted onto the AMUX lines. This means that if the AMUX lines are not specifically being used in a floating-point micro-instruction, the T-bus will be enabled onto the AMUX, allowing the T-bus to be displayed. Also, whenever the T-bus is not being explicitly used, 2 bytes of the 64-bit data path are enabled onto the T-bus. The actual source of the data on the AMUX lines at any micro-step may be determined from the FP11-A flow diagrams. Refer to the *KY11-LB Programmer's Console Maintenance Manual* for more information on the use and operation of the KY11-LB for maintenance. Refer to the FP11-A print set for information regarding the proper installation of the FP11-A and KY11-B.

7.4 FP11-A FLOW DIAGRAMS

Each micro-step in the FP11-A flow diagrams denotes what will be displayed on the Unibus data lines when the manual clock is enabled. This information is given just below the dotted line in each block.

The information may be a constant (such as 100000) or may be defined in a general way such as Q(B7:B0), which indicates that bytes 7 and 0 of the Q-register will be displayed. Refer to Figure 7-1.

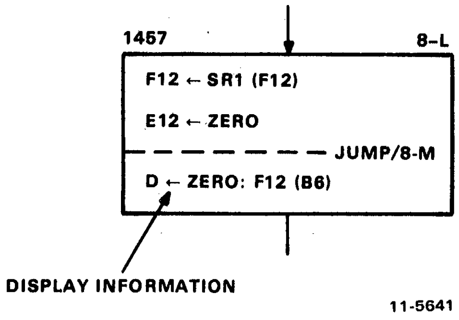


Figure 7-1 Display Information

7.5 EXTENDER BOARD

A special extender board (W9042) and two extender cables are included with the FP11-A module on a hex extender module. The FP11-A print set shows the correct methods of using the W9042 extender board and the included cables.

APPENDIX A
OPTION POWER SPECIFICATIONS

Table A-1 PDP-11 Family Models and Options Power Requirements

Model/Option	Description	Current Needed (Amperes)						AC Line Current (Amperes)
		+5 V (CPU)	+5 V (Options)	-15 V	+20 V	-5 V	+15 V	
H765	Power Supply							
(115/230 Vac)								
Regulator Units	**							
15 V Regulator	Power line monitor						4.	
(5411086)								
11/05-S	KD11-B	8.0		0.25			0.05	
	MM11-U	5.4			4.4	0.51		
	3 SPC	6.0						
	2 M930s	2.5						
	Total Amperes	16.6		0.25	4.4	0.51	0.05	5.0
11/35-S	KD11-A	10.5						
	KE11-F	2.0						
	KE11-E	3.0						
	KJ11-A (optional)	0.5						
	KT11-D	2.5						
	KW11-L	0.5						
	SPC	2.0						
	M981		1.25					
	MF11-U (16K)		6.1					
	M930		1.25					
	Total Amperes	21.	8.6		4.4	0.51		6.0
MF11-U/MM11-U*	16K sense							
(Active)	core memory		6.1		4.4	0.51		2.2
(Standby)	(double SU)		5.4		0.56	0.41		0.8
MF11-UP/MM11-UP	16K sense							
(Active)	core with parity		7.3		4.4	0.51		2.3
(Standby)	(double SU)		5.4		0.56	0.41		0.8
MF11-L (MM11-L)	8K core							
(Active)	memory		3.4	6.0				1.8
(Standby)	(double SU)		1.7	0.5				0.3
MF11-LP (MM11-LP)	8K parity							
(Active)	core memory		4.9	6.0				2.
(Standby)	(double SU)		1.7	0.5				0.3

*Noninterleaved.

**Refer to appropriate appendix for regulator unit output current.

Table A-1 PDP-11 Family Models and Options Power Requirements (Cont)

Model/Option	Description	Current Needed (Amperes)						AC Line Current (Amperes)
		+5 V (CPU)	+5 V (Options)	-15 V	+20 V	-5 V	+15 V	
MM11-S	Same as MM11-L except in SU configuration (1 SU)		Same as MF11-L					
PDP-11/04	KD11-D	5.0						7.0
	M9301	2.0						
	M9302	1.2						
	Memory: See individual memory listings.							
	DL11-W (optional)	2.0		0.15			0.05	
	M7850 (optional)	1.0						
	KY11-LA	0.1						
	KY11-LB (optional)	3.0		0.06				
PDP-11/34A								9.0
PDP-11/34A	KD11-EA	11.5						9.0
	M9301	2.0						
	M9302	1.2						
	Memory: See individual memory listings.							
	DL11-W (optional)	2.0		0.15			0.05	
	M7850	1.0						
	KY11-LA	0.1						
	KY11-LB (optional)	3.0		0.06				
FP11A	M8267		7.0					
MM11-CP	8K core memory		3.0		3.5	0.2		
MM11-DP	16K core memory		3.0		4.0	0.5		
MM11-WP	32K parity							
(Active)	core memory		6.1		3.4	0.74		2.1
(Standby)	(double SU)		5.5		0.6	0.64		0.8
MM11-YP	32K parity							
(Active)	core memory		5.0		3.5	0.4		2.0
(Standby)			5.0		0.6	0.4		0.8

Table A-1 PDP-11 Family Models and Options Power Requirements (Cont)

Model/Option	Description	Current Needed (Amperes)						AC Line Current (Amperes)
		+5 V (CPU)	+5 V (Options)	-15 V	+20 V	-5 V	+15 V	
MS11-EP	4K MOS MUD memory		1.5 (+5) 0.5 (+5B)***	0.1			0.34	
MS11-FP	8K MOS MUD memory		1.5 (+5) 0.5 (+5B)***	0.1			0.36	
MS11-JP	16K MOS MUD memory		1.5 (+5) 0.5 (+5B)***	0.1			0.4	
M7850	Parity control for MUD memories		1.0					

***Current from +5 Vb rail if Battery Backup Option is used. If there is no Battery Backup Option, then 2.0 A is drawn from +5 V.

Table A-2 PDP-11 Family Options Power Requirements

Option	Mounting Code	Description	Power Harness	Current Needed (Amperes)*				AC Line Current (Amperes)
				+5 V	-15 V	-5 V	+15 V	
AA11-D	1 SU	D/A converter subsystem	7009562	3.0				0.3
AR-11	SPC	ADC and DACs	N/A	5.0				0.5
BA614	(AA11-D)	D/A converter		3.0				0.3
BM792-Y	SPC	Bootstrap loader		0.3				0.3
CD11-A/B	1 SU	1000 cpm, 80-col. card reader controller	7010117	2.5				0.25
CD11-E	1 SU	1200 cpm, 80-col. card reader controller	7010117	2.5				0.25
CM11	SPC	200 cpm, 80-col. card reader controller		1.5				0.15

*+20 V not used in this configuration.

Table A-2 PDP-11 Family Options Power Requirements (Cont)

Option	Mounting Code	Description	Power Harness	Current Needed (Amperes)*				AC Line Current (Amperes)
				+5 V	-15 V	-5 V	+15 V	
CR11	SPC	300 cpm, 80-col. card reader controller		1.5				0.15
DA11-DB	1 SU	Unibus link		4.0				0.4
DA11-F	1 SU	Unibus window	7010117	5.0				0.5
DB11-A†	1 SU	Bus repeater	7009562	3.2				0.31
DC11-A	1 SU	Dual clock and system unit	7010117	0.2				0.02
DC11-DA	(DC11-A)	Full duplex module set		2.0	0.2		0.2	0.2
DD11-B	1 SU	Peripheral mounting panel	7010117					
DH11-AA	DLB SU	Prog. async 16-line multiplexer	7010118	8.4	0.42			0.9
DH11-AD	DLB SU	@ Modern control	~~7010118~~	10.8	0.665		0.4	1.33
DJ11-A	1 SU	Async 16-line MUX	7010117	4.7	0.25		0.25	0.6
DJ11-AC	1 SU	Async 16-line MUX			1.0			0.25
DL11	SPC	Async interface		1.8	.15		.016	0.21
DM11-B	(DH11)	16-line modem control	(DH11)	2.4				0.24
DN11-A	1 SU	Auto calling system unit	7009562	2.6				2.5
DP11-D	1 SU	Half/full duplex sync interface	7009562	2.56	0.07		0.04	0.28
DP11-C	(DP11-D)	Data/sync register extender		0.77				0.08
DP11-K	(DP11-D)	Internal DP11 clock		0.18				0.02
DQ11-D								0.62
DQ11-D	1 SU	Full/half duplex sync interface	7010117	6.0	0.07		0.04	0.62

*+20 V not used in this configuration.

†When installing a DB11-A bus repeater in a BA11-K 10.5 Inch Mounting Box, the AC LO and DC LO wires must be removed from the harnesses of all the options (located in the same box) after the DB11-A.

Table A-2 PDP-11 Family Options Power Requirements (Cont)

Option	Mounting Code	Description	Power Harness	Current Needed (Amperes)*				AC line Current (Amperes)
				+5 V	-15 V	-5 V	+15 V	
DQ11-E	1 SU	Full/half duplex sync interface	7010117	6.0	0.07		0.04	0.62
DFC11-A	(DU/DP CLOCK)	Level converter clock recovery		0.4	0.02		0.02	0.05
DQ11-K	(DQ11-D/A)	Crystal clock			0.05			0.012
DR11-B	SPC	General purpose DMA	7009562	3.3				0.32
DR11-C	1 SU	General purpose digital interface		1.5				0.15
DR11-K	SPC	Digital I/O		N/A	0.15			0.6
DU11-D	SPC	Full/half duplex		2.2	2.5		0.05	0.27
DU11-EA	SPC	Sync prog. interface		2.6	0.20		0.07	0.33
DV11	DBL SU	Sync MUX		13.5	.083		0.435	0.5
KG11-A	SPC	Comm. arith unit		1.2				0.12
KW11-L	(CPU)	Line clock		0.8				0.08
KW11-P	SPC	Prog. line clock		1.0				0.1
LC11-A	SPC	LA30 control		1.5				0.15
LP11-R	SPC	1200 LPM printer		1.0				0.1
LP11-S	SPC	900 LPM printer		1.0				0.1
LP11-W	SPC	240 LPM printer		1.5				0.15
LP11-V	SPC	300 LPM printer		1.5				0.15
LS11-A	SPC	60 LPM printer		1.5				0.15
LV11-B	SPC	Electrostatic printer, 500 LPM		1.5				0.15
MR11-DB	2 SPC	Bootstrap						

*+20 V not used in this configuration.

Table A-2 PDP-11 Family Options Power Requirements (Cont)

			Power	Current Needed (Amperes)*				AC Line Current
Option	Mounting Code	Description	Harness	+5 V	-15 V	-5 V	+15 V	(Amperes)
PC11	SPC	Papertape		1.5				0.15
PR11	SPC	Papertape (reader)						
RH11	DBL SU			1.9				0.19
RK11-D	SU	Disk and control	7010115	8.0				0.8
TA11-A	SPC	Dual cassette interface						
VT11	SU	Graphic processor		6.5	100.			0.8
VR11-A	SPC	Pushbutton box		4.				0.4

*+20 V not used in this configuration.

FP11-A FLOATING-POINT PROCESSOR
USER'S GUIDE
EK-FP11A-UG-001

Reader's Comments

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

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

What features are most useful? ______________________________

What faults 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 on 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: Communications Services (NR2/M15)
Customer Services Section

Order No. EK-FP11A-UG-001

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

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