EK-FP11F-TM-002

FP11-F
floating-point processor
technical manual

digital equipment corporation • maynard, massachusetts

1st Edition, August 1979
2nd Edition, November 1979

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.

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

FIGURES

FIGURES (Cont)

TABLES

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The FP11-F Floating-Point Processor is a hardware option that enables the PDP-11/44 central processor to execute floating-point arithmetic operations. The FP11-F 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-F 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-F features both single- and double-precision (32- or 64-bit) capability and floating-point modes.

The FP11-F 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-11/44 floating-point instruction set and the FP11-F.

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 I/O 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-F 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-F 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-F and the central processor's general registers or memory.

1.4 PHYSICAL DESCRIPTION

The FP11-F consists of a single hex board M7093. Figure 1-1 shows the basic signal paths between the central processor and the FP11-F. The bidirectional data bus transfers instructions and data between the processors. An expanded control store in the KD11-Z accommodates floating-point requirements.

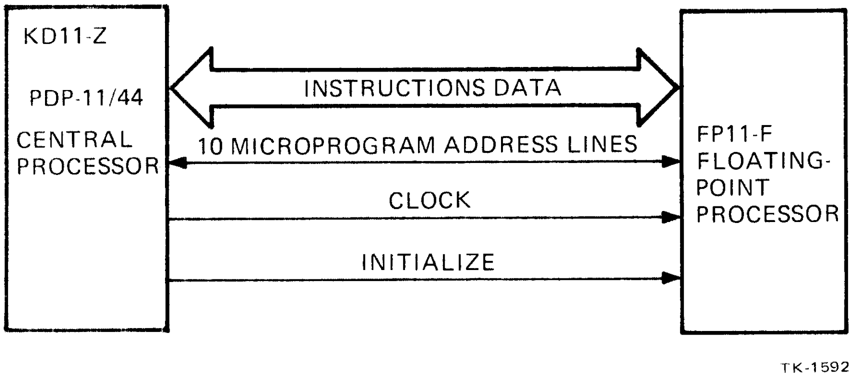


Figure 1-1 KD11-Z/FP11-F Signal Interface

1.5 RELATED DOCUMENTATION

The following documents supplement this manual on the FP11-F Floating-Point Processor.

Manual	Document Number
KD11-Z Processor Manual (PDP-11/44)	EK-KD11Z-TM
PDP-11 Peripherals Handbook	EB-05961
PDP-11/04,34A,44,60,70 Processor Handbook	EB-17716
KD11-Z Processor Manual	EK-KD11Z-MM

CHAPTER 2
DATA FORMATS

2.1 INTRODUCTION

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

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

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

2.2 FP11-F 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 (MSB) of the data word is the sign bit. Figure 2-1 shows the integers 5 and –5 in both I and L formats.

Figure 2-2 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-F are arranged and manipulated according to the type of instruction being executed.

2.3 FP11-F 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 2-2 shows that the MSB is the sign of the fraction (and the floating-point number being represented). The next eight bits contain the value of the exponent, expressed in excess 200 notation. 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.

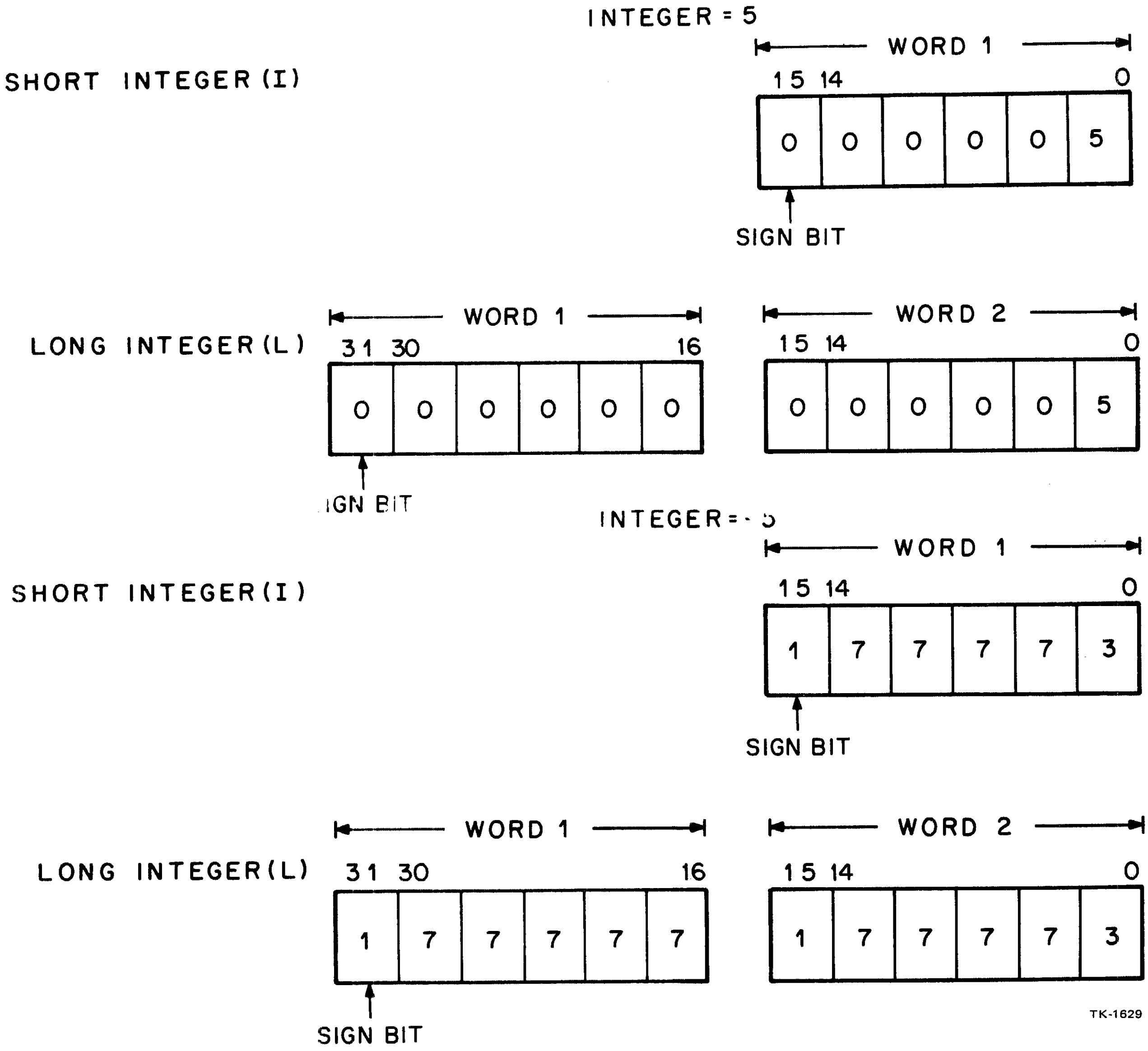
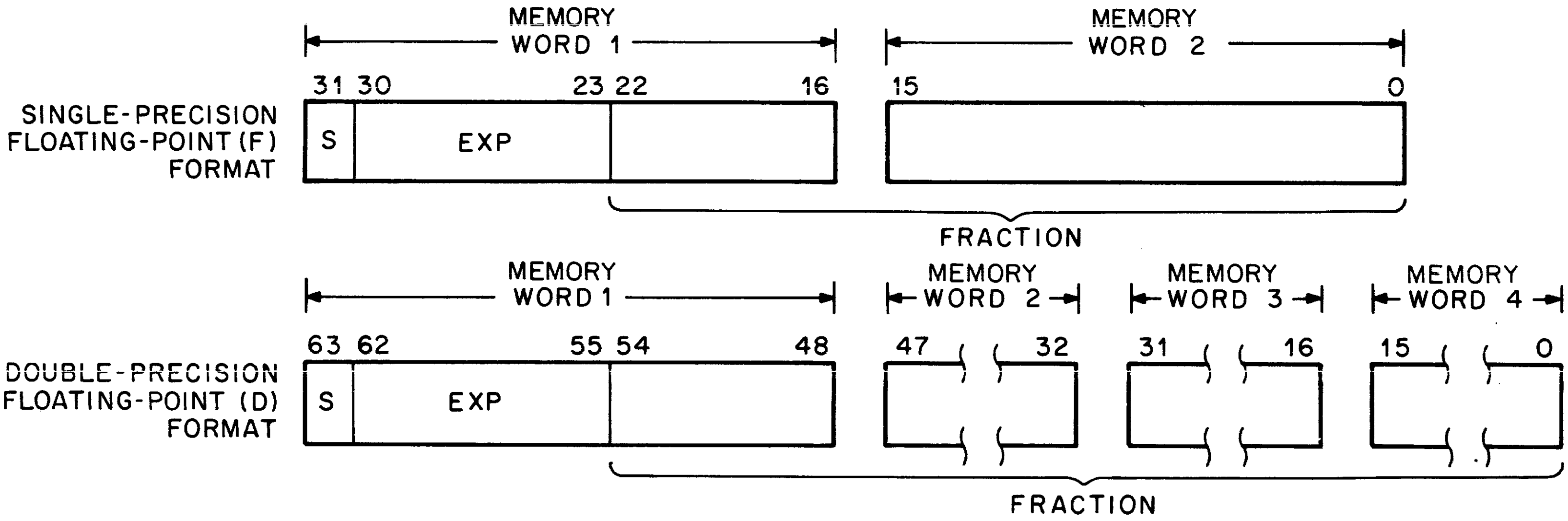


Figure 2-1 Integer Formats



S = Sign
EXP = Exponent in excess 200 notation
Fraction = 23 or 55 bit fraction in sign and magnitude format.

TK-1565

Figure 2-2 Floating-Point Data Formats

2.3.1 FP11-F Floating-Point Data Word

Figure 2-3 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-F are arranged as illustrated in Figure 2-4.

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

For purposes of discussion, the FP11-F 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.

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

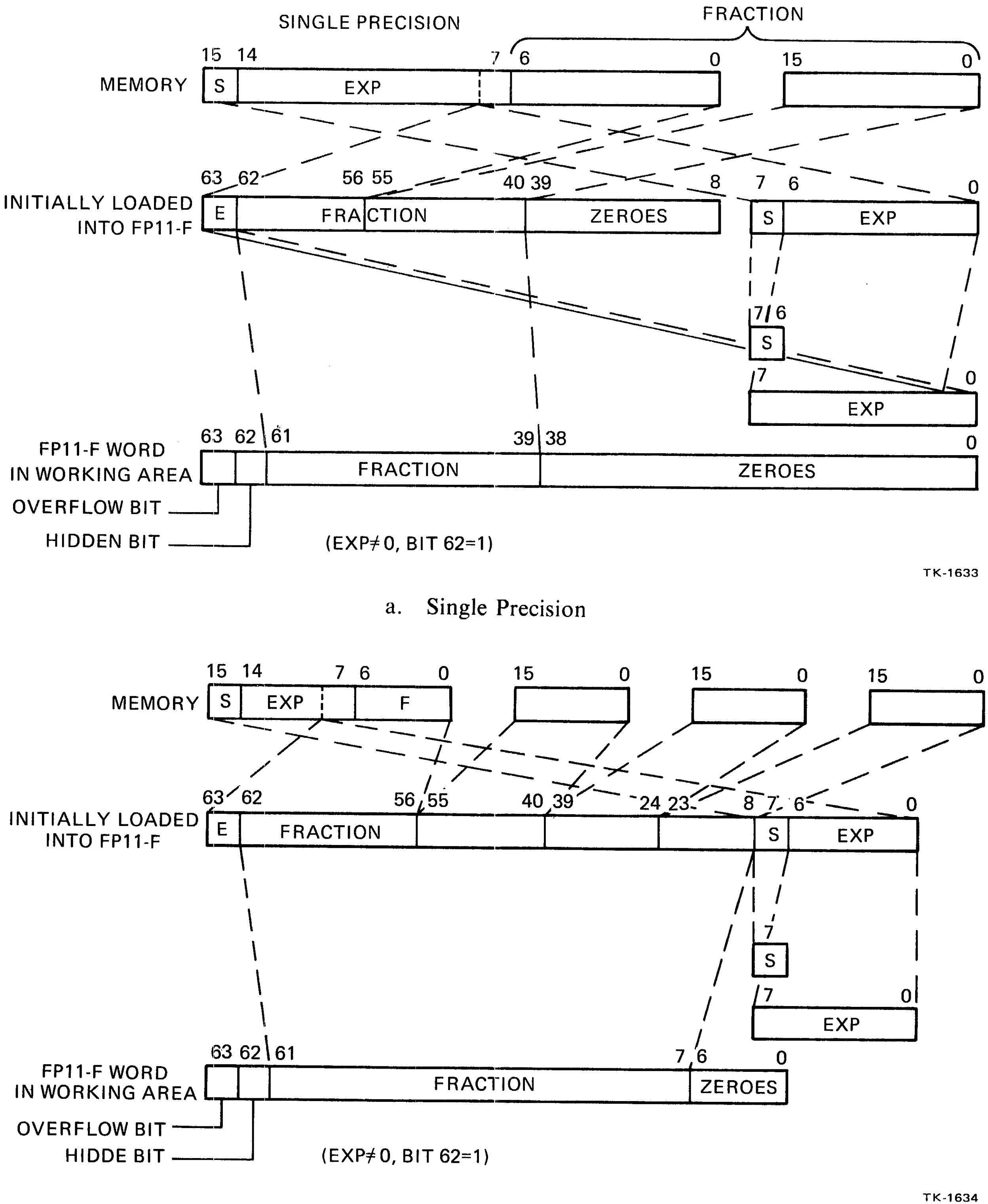
	2's Complement Notation	**Sign and Magnitude Notation**
+2	000010	$\underset{\text{Sign}}{0}\underbrace{00010}_{\text{Magnitude}}$
−2	111110	$\underset{\text{Sign}}{1}\underbrace{00010}_{\text{Magnitude}}$

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

Unnormalized floating-point fractions have a range from approximately 0 through 2 as shown in Figure 2-5. The FP11-F, 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.

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-F 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-F, 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.



a. Single Precision

b. Double Precision

Figure 2-3 Floating-Point Data Words

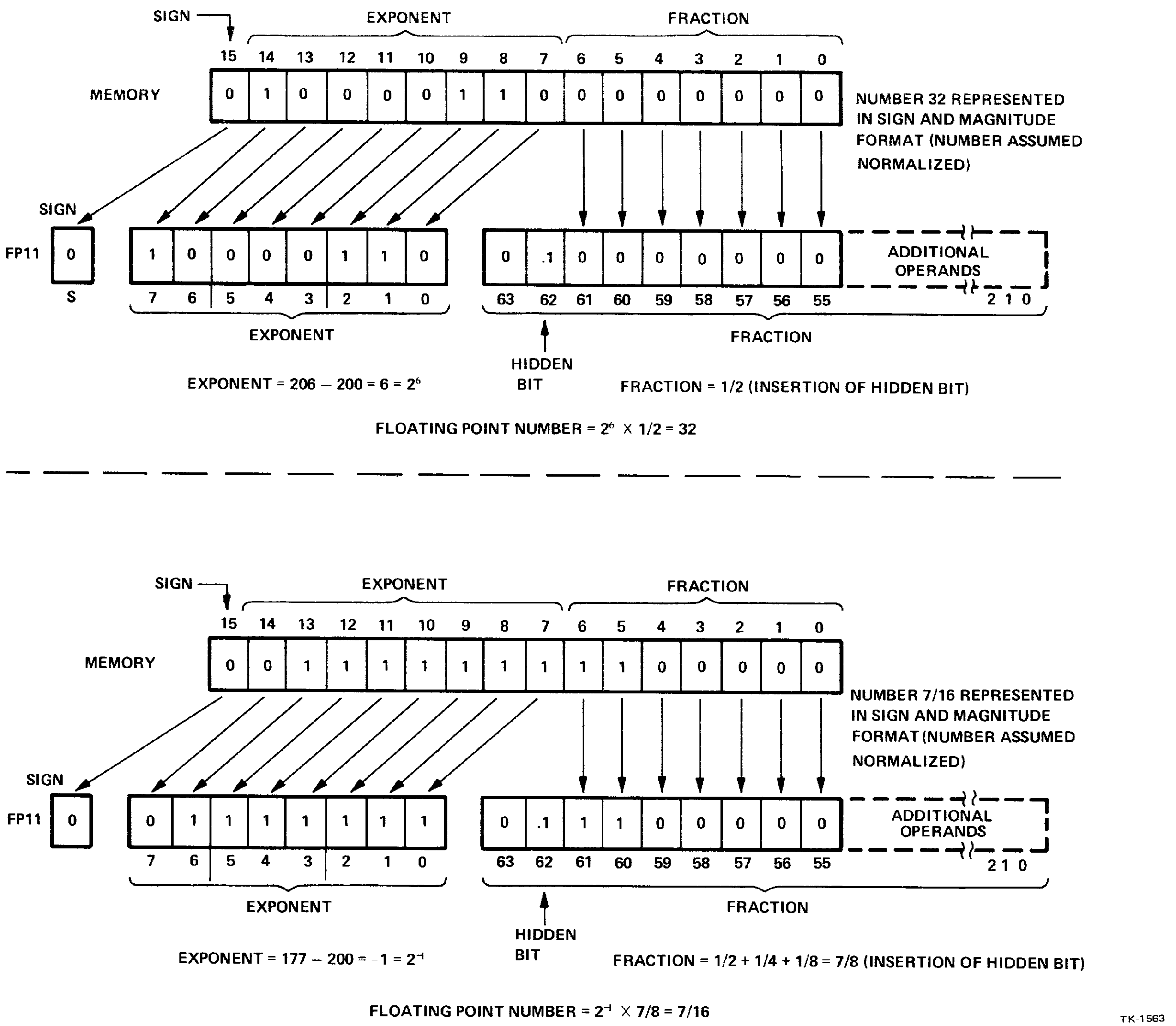


Figure 2-4 Interpretation of Floating-Point Numbers

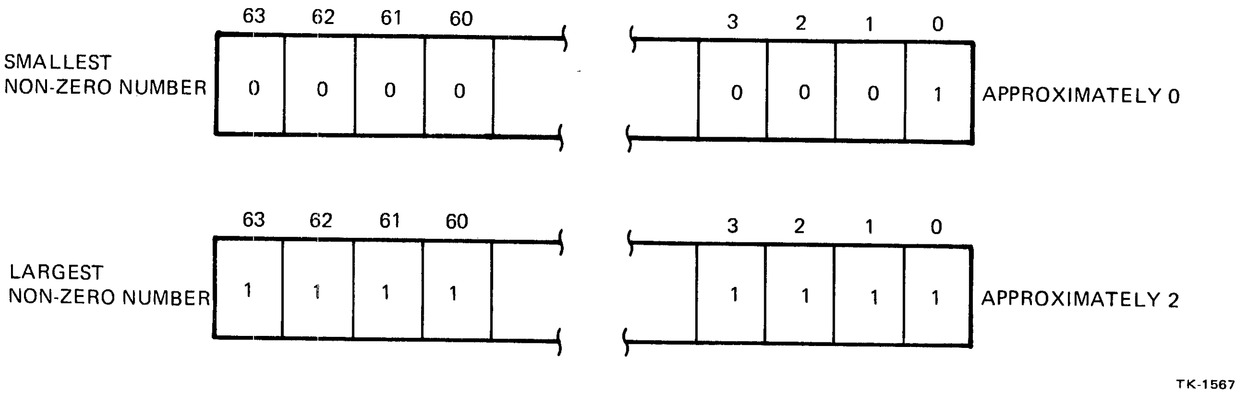


Figure 2-5 Unnormalized Floating-Point Fraction

2.3.1.2 Floating-Point Exponent – The 8-bit floating-point exponent is expressed in excess 200 notation. The following chart 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	Positive Exponents	377 Most positive exponent
	0 Least positive exponent		200 Least positive exponent
Negative Exponents	377 Least negative exponent	Negative Exponents	177 Least negative exponent
	200 Most 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-F 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 2-5 illustrates the range of floating-point numbers that can be handled by the FP11-F. For simplicity, a fraction length of only three bits is shown.

2.4 FP11-F PROGRAM STATUS REGISTER

The FP11-F 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 three mode bits and additional bits to enable various interrupt conditions. Figure 2-6 shows the layout of the program status register. Each bit shown in Figure 2-6 is described in Table 2-1.

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

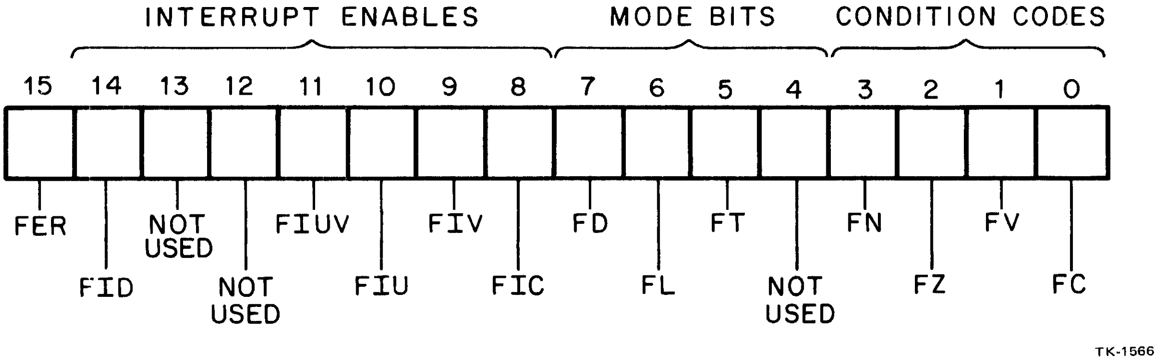


Figure 2-6 FP11-F Status Register Format

Table 2-1 FP11-F Status Register

Bit	Name	Function
15	FER	This bit indicates an error condition of the FP11-F.
14	FID	Floating Interrupt Disable – All interrupts by the FP11-F 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.

Table 2-1 FP11-F Status Register (Cont)

Bit	Name	Function
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.
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 N, Z, V, and C condition codes of the CPU, respectively. This is accomplished by the copy floating condition codes (CFCC) instruction.

2.5 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-F 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 described briefly in Table 2-2.

Table 2-2 FP11-F Exception Codes

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

Refer to the *PDP-11/04, 34A, 44, 60, 70 Processor Handbook* for further details concerning FP11-F 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.

CHAPTER 3
INTERFACING

3.1 GENERAL

The CPU loads floating-point instructions and operands into the FP11-F and then reads operands from the FP11-F and stores them in memory. Figure 3-1 illustrates the CPU/FP11-F interface; Table 3-1 describes the interface signals.

NOTE

The FP11-F does not directly interface with the Unibus; it connects to the CPU via a bus separate from the Unibus and uses the internal CPU control facilities for data transfers to and from memory.

When the FP11-F is installed in a system, it asserts an FP11-F ATTACHED signal that informs the CPU that it is interfaced with an FP11-F module.

3.2 INITIAL OPERATION

Initially, the CPU fetches an instruction from memory and decodes it. If the four high-order bits (12–17) are set, the instruction will have an operation code of 17XXXX. Thus, the instruction fetched is a floating-point instruction that requires use of the FP11-F. The CPU next writes 50 on the FP11-F MPC 0–10 microprogram counter. This MPC format is required to set up the FP11-F at the start of every new floating-point instruction. The 50 is decoded by control store PROMs into a 104-bit field that controls the microflow to the FP11-F microprocessor and CPU. The control word is clocked via EXT CLK A L, which is received from the CPU.

Depending on the instruction, the CPU next sends the FP11-F AMUX 0–15 H. This is applied to an instruction register that defines what type of floating-point instruction is to be used. The instruction register is clocked by B PROC CLK L and LOAD IR L from the CPU.

3.3 MICROCODE GENERATION

The next microinstruction from the CPU informs the FP11-F how the second operand (data) received is to be processed. When the second operand is received, a series of microcodes will be generated in the FP11-F to control microprocessor operation on the second operand. After the second operand is processed, the FP11-F will inform the CPU, via a microcode it asserts on the MPC 0–10 lines, that data can now be read from the AMUX 0–15 lines. The FP11-F will also send the CPU a TRI STATE AMUX L signal, which enables it to read data from the AMUX lines. The CPU then stores the data (operand) and continues operation.

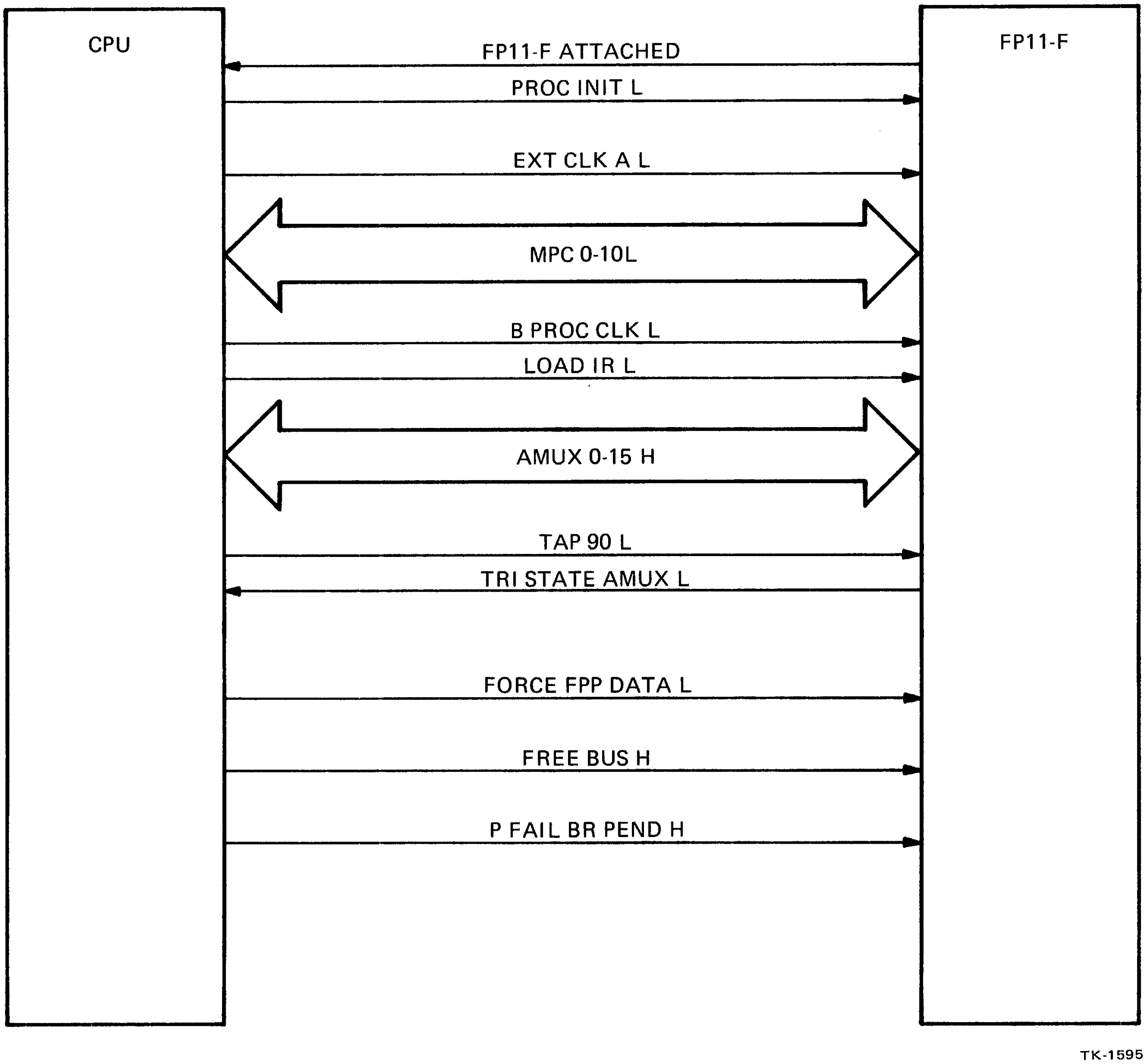


Figure 3-1 FP11-F/CPU Interface

Table 3-1 FP11-F/CPU Signal Interface

Signal(s)	Direction	Function
MPC<00:10>L	Bidirectional	Microprogram address lines. Used to sequence the CPU and FP11-F through the microprogram. Derived from CPU microcode, but can be altered by either CPU or FP11-F.
AMUX<00:15>L	Bidirectional	Data lines that are used to transfer instructions, operands, and FP11-F status information between CPU and FP11-F.
TBUS<00:15>L	Bidirectional	Buffered AMUX lines used internally to the FP11-F.
PROC CLK L	CPU to FP11-F	CPU clock. Used to generate FP11-F clock to load FP11-F registers and RAM.
PROC INIT L	CPU to FP11-F	CPU initialize. Used to initialize FP11-F status registers in FP11-F.
LOAD IR L	CPU to FP11-F	Causes FP11-F to load its instruction register from AMUX lines.
TRI-STATE AMUX L	FP11-F to CPU	Causes CPU to remove data from AMUX lines. Turns on drivers that allow the FP11-F to drive the AMUX lines.
PFAIL BR PEND H	CPU to FP11-F	When high, indicates that an interrupt needs servicing. Used by FP11-F to abort long instructions to maintain interrupt latency of less than 20 μs.
TAP 90 L	CPU to FP11-F	CPU delay line generated clock used to skew FP11-F microprocessor output data.
FORCE FPP DATA L	CPU to FP11-F	Console-generated signal to monitor FP11-F status or T BUS data.
FREE BUS H	CPU to FP11-F	Console-generated signal to cause FP11-F to release AMUX lines.
EXT CLK A L	CPU to FP11-F	CPU clock that clocks control word through FP11-F control logic.
FP11-F ATTACHED L	FP11-F to CPU	Indicates to CPU that it is interfacing with an FP11-F module.

CHAPTER 4
ARITHMETIC ALGORITHMS

4.1 INTRODUCTION

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

4.2 FLOATING-POINT ADDITION AND SUBTRACTION

Floating-point addition and subtraction are performed in the ALUs of the AM2901s. The operands are designated source and destination. The following chart lists the register associated with the exponent, fraction, and sign of each operand.

Operand	**Exponent**	**Fraction**	**Sign**
Destination	EAC(X13)	(X11)	SD (AC:Bit 7)
Source	EFRSC(X14)	(X12)	SS (X10:Bit 8)
Result	EAC(X13)	(X12)	SD (AC:Bit 7)

For example, the exponent of the result of an addition or subtraction is found in the EAC, the fraction is found in X12, and the sign is found in X10.

The source operand is located in an AC if mode 0 is specified; it is located in memory if mode 0 is not specified. In the latter case, the operand in memory is transferred to the FP11-F and temporarily stored in X10 and X12 (shift left one place).

4.2.1 Description of Sign Processing

To understand how the hardware implements sign calculations for floating-point addition and subtraction, refer to Table 4-1. The following text attempts to educate the reader in the use of this table. Normally, SS (sign of source) represents the sign associated with the source operand (ACS) and SD (sign of destination) represents the sign of the destination operand (ACD). The sign of the result is stored in SD.

Table 4-1 Add and Subtract Implementations

<table>
<tr><th rowspan="2">Combination</th><th rowspan="2">SS</th><th rowspan="2">SD</th><th rowspan="2">Instruction</th><th rowspan="2">Hardware Performs</th><th colspan="2">Sign of Result</th></tr>
<tr><th>Positive Parentheses</th><th>Negative Parentheses</th></tr>
<tr><td></td><td></td><td></td><td>Add Instruction</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>ACD ← +(|ACD|+|ACS|)</td><td>Add</td><td>SD ← SD</td><td></td></tr>
<tr><td>2</td><td>0</td><td>1</td><td>ACD ← -(|ACD|-|ACS|)</td><td>Subtract</td><td>SD ← SD</td><td>SD ← SS</td></tr>
<tr><td>3</td><td>1</td><td>0</td><td>ACD ← +(|ACD|-|ACS|)</td><td>Subtract</td><td>SD ← SD</td><td>SD ← SS</td></tr>
<tr><td>4</td><td>1</td><td>1</td><td>ACD ← -(|ACD|+|ACS|)</td><td>Add</td><td>SD ← SD</td><td></td></tr>
<tr><td></td><td></td><td></td><td>Subtract Instruction</td><td></td><td></td><td></td></tr>
<tr><td>5</td><td>0</td><td>0</td><td>ACD ← +(|ACD|-|ACS|)</td><td>Subtract</td><td>SD ← SD</td><td>SD ← ~SS</td></tr>
<tr><td>6</td><td>0</td><td>1</td><td>ACD ← -(|ACD|+|ACS|)</td><td>Add</td><td>SD ← SD</td><td></td></tr>
<tr><td>7</td><td>1</td><td>0</td><td>ACD ← +(|ACD|+|ACS|)</td><td>Add</td><td>SD ← SD</td><td></td></tr>
<tr><td>8</td><td>1</td><td>1</td><td>ACD ← -(|ACD|-|ACS|)</td><td>Subtract</td><td>SD ← SD</td><td>SD ← ~SS</td></tr>
</table>

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

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

Example 1

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

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

Example 2

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

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

Example 3

$$\begin{array}{r} +8 \\ -7 \\ \hline +1 \end{array} \qquad \begin{array}{r} -8 \\ +7 \\ \hline -1 \end{array} \qquad \begin{array}{r} +7 \\ -8 \\ \hline -1 \end{array} \qquad \begin{array}{r} -7 \\ +8 \\ \hline +1 \end{array}$$

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

Example 4

$$\begin{array}{c} +8 \\ -(+7) \\ \hline +1 \end{array} \qquad \begin{array}{c} -8 \\ -(-7) \\ \hline -1 \end{array} \qquad \begin{array}{c} +7 \\ -(+8) \\ \hline -1 \end{array} \qquad \begin{array}{c} -7 \\ -(-8) \\ \hline +1 \end{array}$$

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

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

Example 1
Assume an add instruction is specified.

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

$$ACD \leftarrow + (|ACD| - |ACS|) = +(3 - 5) = -2$$

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

Example 2
Assume a subtract instruction is specified.

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

$$ACD \leftarrow -(|ACD| - |ACS|) = -(5 - 3) = -2$$

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

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

4.2.2 Relative Magnitude

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

4.2.3 Testing for Normalization

All floating-point numbers must be normalized. In order to normalize a number, bit 59 must be a 0 and bit 58 must be a 1. The result of any arithmetic operation must be normalized. In addition, the fraction of the result is always positive; therefore, the hardware will simply normalize the number. In subtraction, the fraction may be negative or 0, neither of which can be normalized. After a subtraction operation has been performed in which X12 was not aligned, the result in X12 is tested to ensure that it can be normalized. If the number in X12 is negative, it indicates that the number cannot be 0 and cannot be normalized. If the number in X12 is positive, it may be 0. Consequently, 1 is subtracted from the X12 and if the result is negative (change of signs), the number in X12 is known to be 0, which cannot be normalized. If there is no sign change in the subtraction, X12 contains a positive number, which can be normalized.

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

4.2.4 Floating-Point Addition

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

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

Addition with Like Signs

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

Subtraction with Unlike Signs

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

In all examples, the two quantities are actually added. Paragraph 4.2.5 describes floating-point subtraction, which consists of the addition of two numbers with unlike signs or the subtraction of two numbers with like signs. Several examples demonstrate this point.

Addition with Unlike Signs

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

Subtraction with Like Signs

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

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

> **The AM2901 ALU is performing A minus B for subtraction; therefore, the result must be examined for the possible cases of 0 or negative results that require special treatment by the FP11-F hardware.**

4.2.4.1 Hardware Implementation of Addition – The difference between the two exponents is initially stored in X13 and represents the destination exponent minus the source exponent. The destination fraction is loaded in X11 and the source fraction is loaded in X12.

4.2.4.2 Align – Certain operations, such as addition and subtraction, require that the exponent of a number be aligned. For example, if two numbers are to be added, the exponent of the smaller number, if different from the exponent of the larger number, must be aligned. That is, the exponent of the smaller number must be increased until it equals that of the larger number; each increase of the exponent must be accompanied by a right-shift of the smaller number's fraction.

4.2.4.3 Normalize – A nonzero floating-point number is normalized by shifting the fraction to the left until nonsignificant leading zeros are eliminated; each shift is accompanied by a subtraction from the exponent. The number is normalized when the first two bits are different (i.e., 0.1 or 1.0) or when only the first two bits of the fraction are 1s (i.e., the number is 6000).

4.2.4.4 Truncate or Rounding – If the FP11-F is in truncate mode and the shift-within-range flag is asserted, the result is shifted the required number of shifts, routed through ALU, and stored in X12. If the FP11-F is in round mode, a 1 is inserted in bit 34 (single-precision) or bit 02 (double-precision). The result is now a normalized, rounded fraction. However, if the fraction contained all 1s, adding the round bit to it causes an arithmetic overflow (bit 59=1). This condition is detected by the normalization shift logic which now left-shifts the result by 1, causing bit 59 to go to 0 and bit 58 to go to 1 as the fraction is stored in the destination accumulator.

4.2.4.5 Adjusting Exponent During Normalization – When the result of the addition is being normalized, it is necessary to keep track of the number of shifts required to normalize so that the exponent of the result may be adjusted properly. The EAC contains the larger of the two exponents, i.e., the exponent of the answer. This exponent is updated during normalization by adding the number of right-shifts (or conversely, subtracting the number of left-shifts) directly.

Example

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

4.2.5 Floating-Point Subtraction

In floating-point subtraction, the source operand is subtracted from the destination operand. The source operand is loaded into X11 and the destination operation is loaded in X12, which means that the ALU will perform X12 - X11.

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

4.2.5.1 Negative Exponent Difference – If the exponent difference is negative, it means that the source operand in X12 is greater than the destination operand in X11. Since X11 is the smaller number, it is right-shifted to align the fractions. When the fractions are aligned and then subtracted, the difference will be a 2's complement negative number. This number is 2's complemented to make it a positive number and the sign is adjusted to be the sign of the source operand. A simple example to demonstrate this point follows.

Example

	25_{10}	11001	
Subtract	31_{10}	11111	Take 2's complement and add
	-6_{10}		

	11001	
2's complement of 11111	+00001	
	11010	= 26_{10} which is not the correct result and which represents a 2's complement negative number.

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

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

4.2.5.2 Determining Exponent Difference – During addition of unlike signs or subtraction of like signs, the ALU performs a subtract. To determine if the exponent difference is 0, the FP11-F logic clocks the branch condition codes and checks BZ. If BZ is asserted, this indicates an exponent difference of 0. In this case, neither X12 nor X11 were shifted and the subtraction of the fractions could result in a zero difference, a negative difference, or a positive difference. If bit 59=1, the resultant fraction is a 2's complement negative number and must be converted to a positive sign and magnitude number.

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

4.2.5.3 Positive Exponent Difference – If the exponent difference is positive (X11 – X12), it indicates that the destination operand is larger than the source operand. Since X12 is the source operand and is smaller than the destination operand, it means that X12 is right-shifted to align the fractions. The fractions are then subtracted. This subtraction must result in a positive number in X12; therefore, the FP11-F does not have to test it for 0 or negative, but instead normalizes it immediately.

4.3 FLOATING-POINT MULTIPLICATION

4.3.1 Basic Concepts

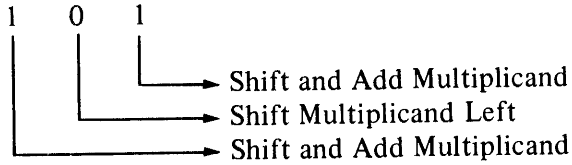
The FP11-F uses a straightforward multiplication scheme. In effect, the method employs a series of shifts and additions to generate the final product.

The pencil and paper procedure for the multiplication of two binary numbers generally illustrates the method:

```
   110101     =  65      =  53
      101     =   5      =   5
  -------       ---        ---
   110101                  265
  1101010
 --------
100001001     = 411      = 265
```

The multiplier is examined on a bit-by-bit basis. If the bit is a 0, the multiplicand is shifted left one place. If the bit is a 1, the multiplicand is added to the partial product and shifted left one place.

Example



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

4.3.2 Hardware Implementation of Multiplication

Multiplication begins with the calculation of the exponent. The exponent of the source in X14 is added to the exponent of the accumulator in X13. The constant 200 is then subtracted to get the exponent into proper form; the hidden and guard bits are inserted, the multiplier is routed to the Q-register, and the sign is routed into X10. After clearing X12, which is used for the accumulation of the partial product, the multiplication loop is entered. During each cycle the LSB of the multiplier is tested. If the bit is a 1, the multiplicand is added to the partial product to generate a new partial product. The partial product is then shifted one place toward the least significant bit (LSB) (shifted right) and the multiplier is also shifted one place toward the LSB. The old LSB of the multiplier is discarded and the cycle repeats until the shift count (56 or 24) is reduced to 0.

The multiplier is in the Q-register of the AM2901, the partial product is in X12 of the AM2901 RAM, and the multiplicand is in X11 of the RAM.

4.4 FLOATING-POINT DIVISION

4.4.1 Basic Concepts

Floating-point division in the FP11-F is accomplished by a normalizing nonrestoring division method. The nonrestoring method is a repeated subtraction-addition method. The initial remainder is the dividend itself.

The quotient is formed in the Q-register with the dividend in X11 and the divisor in X12.

The exponent is computed by subtracting X13 (EFSRC) from X14 (EAC). The constant 200 is then added to X13 to complete the exponent calculation; hidden and guard bits are inserted, the sign is routed to X10, and the divide loop is then entered.

4.4.2 Nonrestoring Division (Hardware Method)

In the nonrestoring method, the divisor is either added to or subtracted from the partial remainder, depending on the sign of the divisor and that of the partial remainder. If the two signs agree, a subtraction is performed and the quotient bit is a 1. If the signs do not agree, an addition is performed and the quotient bit is 0. In both cases, a new partial remainder is next formed by a proper shift and the process continues until the remainder is 0 or the desired number of quotient digits is obtained.

Example

$$\begin{array}{lll}
X = \text{dividend} & = 0.1000 & (8/16) \\
Y = \text{divisor} & = 0.1010 & (10/16)
\end{array}$$

$$\begin{array}{ll}
0.1000 = X & \\
0.1010 = Y & q = 1 \\
\hline
1.0000 = 2r & \\
1.0110 = -Y & \\
\hline
0.0110 = r & q = 1 \\
0.1100 = 2r & \\
1.0110 = -Y & \\
\hline
0.0010 = r & q = 1 \\
0.0100 = 2r & \\
1.0110 = -Y & \\
\hline
1.1010 = r & q = 0 \\
1.0100 = 2r & \\
0.1010 = +Y & \\
\hline
1.1110 = r &
\end{array}$$

$$\begin{array}{llll}
 & 1.110 & =\text{quotient digits} & (12/16) \\
 & +1.0001 & =\text{correction} & \\
\cline{2-2}
Q = & 0.1101 & = 13/16 = 0.8125 & \\
 & & & \\
R = 2\,r & & = 1.11111110 & \\
X & & & \\
Y & = 0.8000 & &
\end{array}$$

The algorithm as implemented in the FP11-F does not require the correction of $-1 + 2^n$ to the indicated quotient Q. In this example, the 12/16 answer requires more quotient digits to converge on the actual answer of 0.80.

CHAPTER 5
THEORY OF OPERATION

5.1 GENERAL

The major circuit in the FP11-F (Figure 5-1) is the Data Manipulation Logic (DML) that processes operands received as T BUS 0–15 H. It is controlled via signals and instructions generated in the control store logic.

Floating-point instructions received from the CPU as MPC 0–10 L are converted by the control store logic into microcode instructions and control signals. These determine how the DML operates on T BUS 0–15 H operands (received from the CPU as AMUX 0–15 L). The data buffering and storage logic performs a bidirectional interface between the CPU and the DML. It can also store data when the CPU cannot be interrupted to accept it. The instruction decoding logic monitors operation of the DML and generates commands used in the control store logic.

5.2 DATA MANIPULATION LOGIC (FIGURE 5-2)

All floating-point operands that the CPU accesses from memory are processed in the FP11-F DML (Figure 5-1). Operands are loaded into registers in the DML and, after several microstates have occurred, the result is output on the T BUS 0–15 H or stored in an accumulator. During these microstates, data manipulation consists of adding, subtracting, shifting, and status operations required to produce the required result. The DML is controlled by instructions and control signals from the 56-bit control store logic and consists of 16 microprocessors and carry look-ahead logic.

5.2.1 Microprocessor

5.2.1.1 Data Sectors – Functionally, the 64-bit microprocessor consists of four 16-bit sectors (Figure 5-2), with each sector comprising two data bytes. Sector loading with T BUS 0–15 H data is determined by a 4-bit code (SECTOR CLOCK 0–3 L) received from the control store logic. Which byte is to be loaded in a particular sector is determined by BYTE 0–7 ENABLE L. The operation of each byte is controlled by ALU instructions I_{0-8} and RAM A/B PORT SELECT 0–3 L received from the control store logic. Each byte in the four sectors is comprised of two 4-bit microprocessors.

Sector Content – Of the four sectors, sector 3 (Figure 5-2) contains the exponent, sign, and most significant fraction bits; sectors 2, 1, and 0 contain progressively less significant fraction bits. The microprocessor logic, controlled by the control store logic, ensures that transferred data is loaded into the correct sector and that any unused sectors are cleared. During a single precision (32-bit) calculation, only sectors 3 and 2 contain data and sectors 0 and 1 (the 32 low-order bits of the DML) are cleared. When the 32 low-order bits are being cleared or data is being transferred from the CPU, the two or three low-order sectors are loaded with zeros simultaneously. If the data buffering logic has received and loaded the required operand, the interface transfers are finished for that particular instruction. However, if the instruction is one of the Store class (send data to the CPU), the interface logic must then select and transfer the data out of the FP11-F.

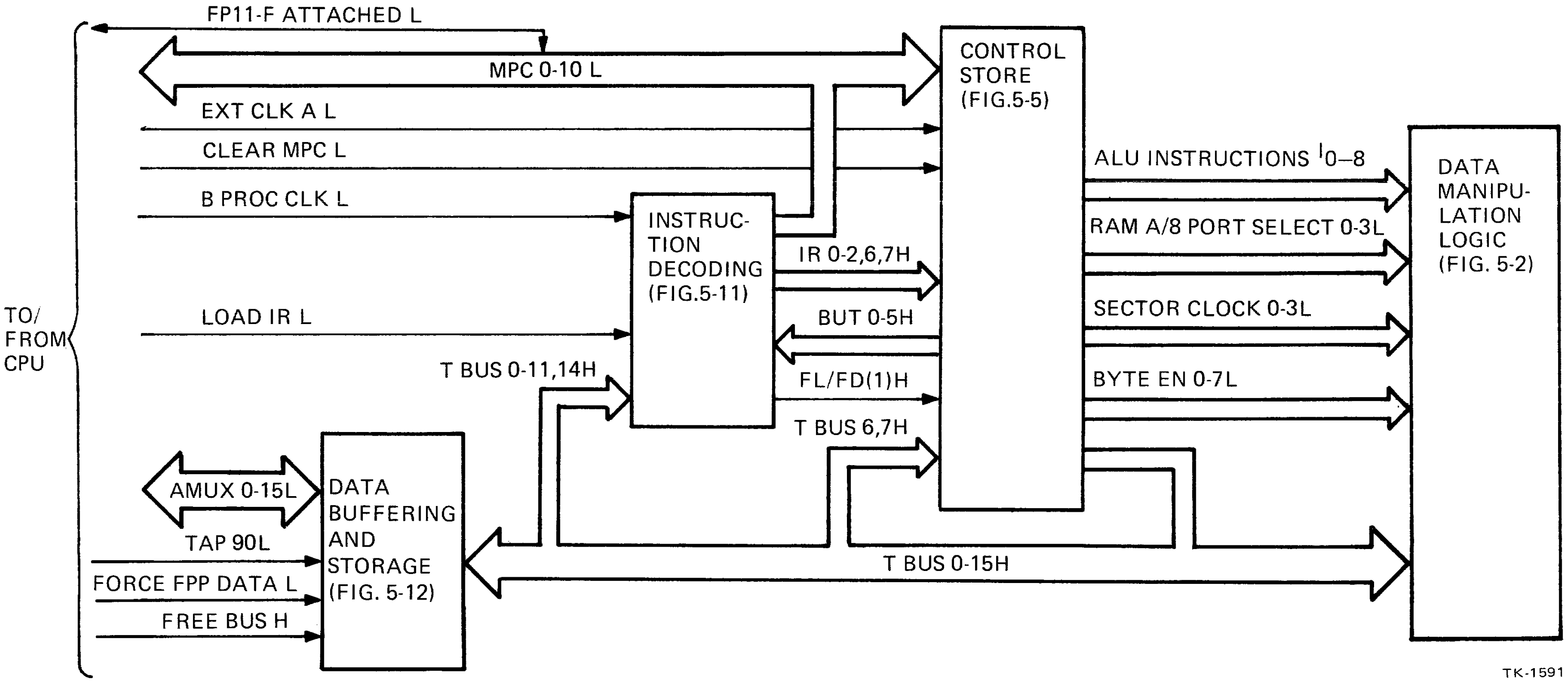


Figure 5-1 FP11-F Block Diagram

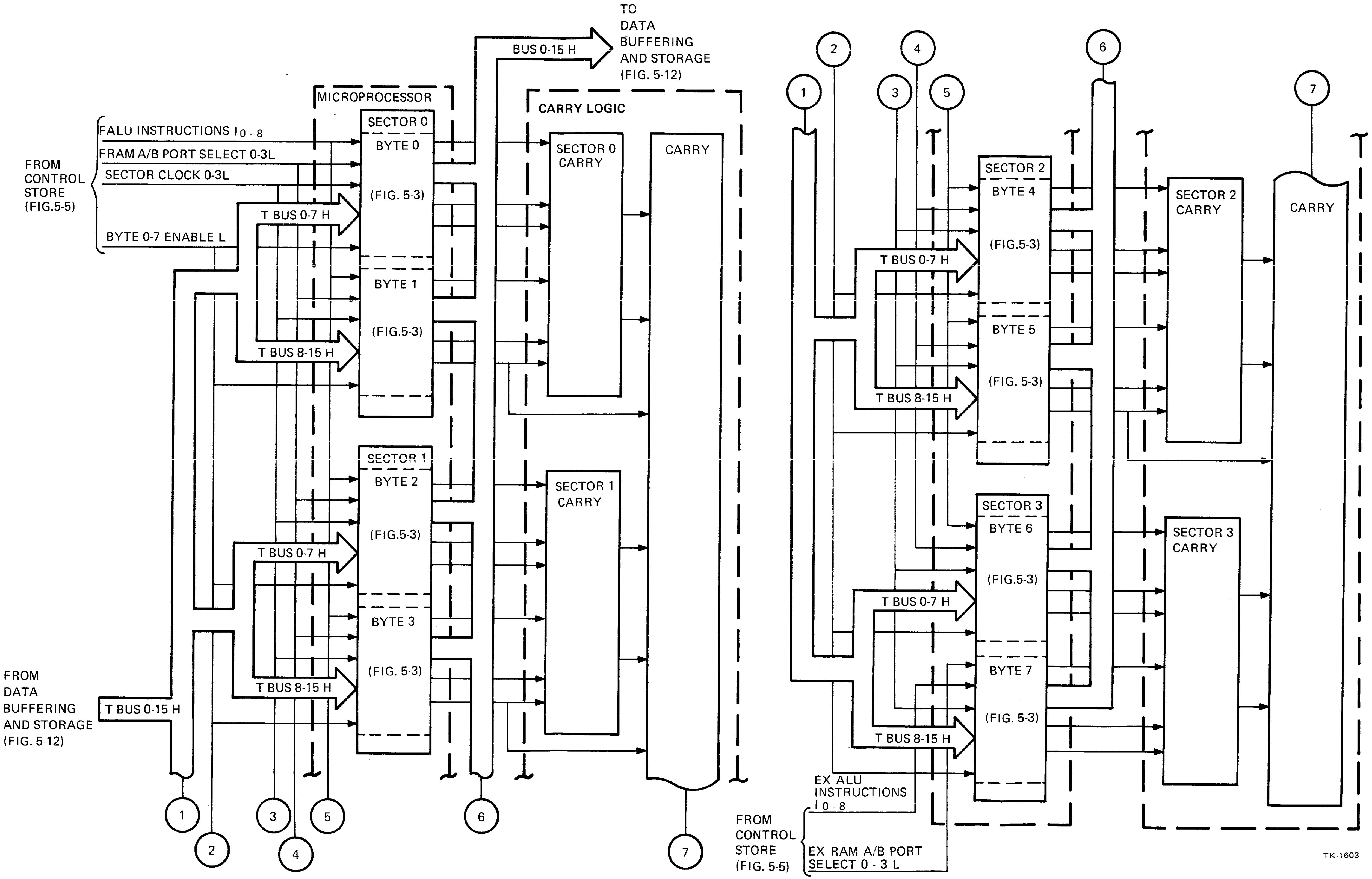


Figure 5-2 Data Manipulation Logic

Sector Loading – One, two, or four 16-bit transfers are required to load the FP11-F microprocessor sectors with memory data. Integer data in integer format (16 bits) requires one transfer and is loaded into sector 2 of X12 floating-point accumulators. Integer data in long format (32 bits) requires two transfers. The first transfer is used to load the most significant half into sector 2 of X12; the second transfer loads the least significant half into sector 1 of X12. Floating-point data is loaded into X10 in two (float precision, 32 bits) or four (double precision, 64 bits) transfers. In both cases, the sign exponent and 23 MSBs are loaded into sector 3 during the first transfer. During subsequent transfers (1 or 3 per type of precision), progressively less significant fraction bits are loaded into sectors 2, 1, and 0.

Four data transfers are required to load floating-point data (32- or 64-bit format) from the microprocessor sectors into memory. One or two data transfers are required to convert and store floating-point data as integers. One or two data transfers are required to transfer control and status information via a Store class instruction.

The DML normally operates on the whole 64-bit wide data path and unused sectors are cleared before any computations begin. During integer transfers, all of X12 is initially cleared before the integer is loaded into it. During all floating-point transfers, unused sectors are cleared after the data has been loaded from memory.

5.2.1.2 Data Bytes – Each of the four sectors in the DML consists of two bytes. Each byte consists of two 4-bit 2901A microprocessors. The 2901A (Figure 5-3) consists of a 16-word × 4-bit two-port RAM, an ALU, a Q-register, and control circuitry.

RAM – The RAM register is illustrated in Figure 5-4. It is the scratchpad area where the results of arithmetic and logical operations can be stored temporarily. The contents of the RAM are read into the ALU per control signals (microcode) received from the control store logic. It consists of 16 64-bit words [each of the 16 microprocessors (2901A) contains a 16 × 4-bit RAM].

Six of the 64-bit registers are allocated as accumulators and are accessible to the programmer. RAM registers 6 and 7 are unused, while registers 10–17 are reserved for special functions. Registers 10–17 are accessed only by the control store logic. Registers 10–14 constitute a working storage area for the microcode contained in the control store logic PROM. Other functions included are the floating-point status register, condition codes, and exception codes.

Data in any of the 16 words of RAM can be read from the A-port, via the 4-bit RAM A PORT 0–3 H input. The address applied to both A- and B-address lines will cause identical data to appear at each RAM output port. The RAM A- and B-outputs are applied to latches. When write enabled, new data is written into the RAM word selected by RAM B PORT 0–3 H. The RAM input data is received from a 3-input multiplexer. The multiplexer inputs are from the ALU output and permit the ALU result to be loaded into the RAM directly, or left- or right-shifted one place.

Before floating-point data is transferred to the CPU, it is initially loaded into RAM address 11_8 in the DML microprocessor. The control store logic BYTE 0–7 ENB L output signal then selects and output enables the particular byte or bytes of $X11_8$ to be transmitted to the CPU during any one transfer. The byte(s) content is then loaded onto the T-bus, buffered, and then received by the CPU via its AMUX.

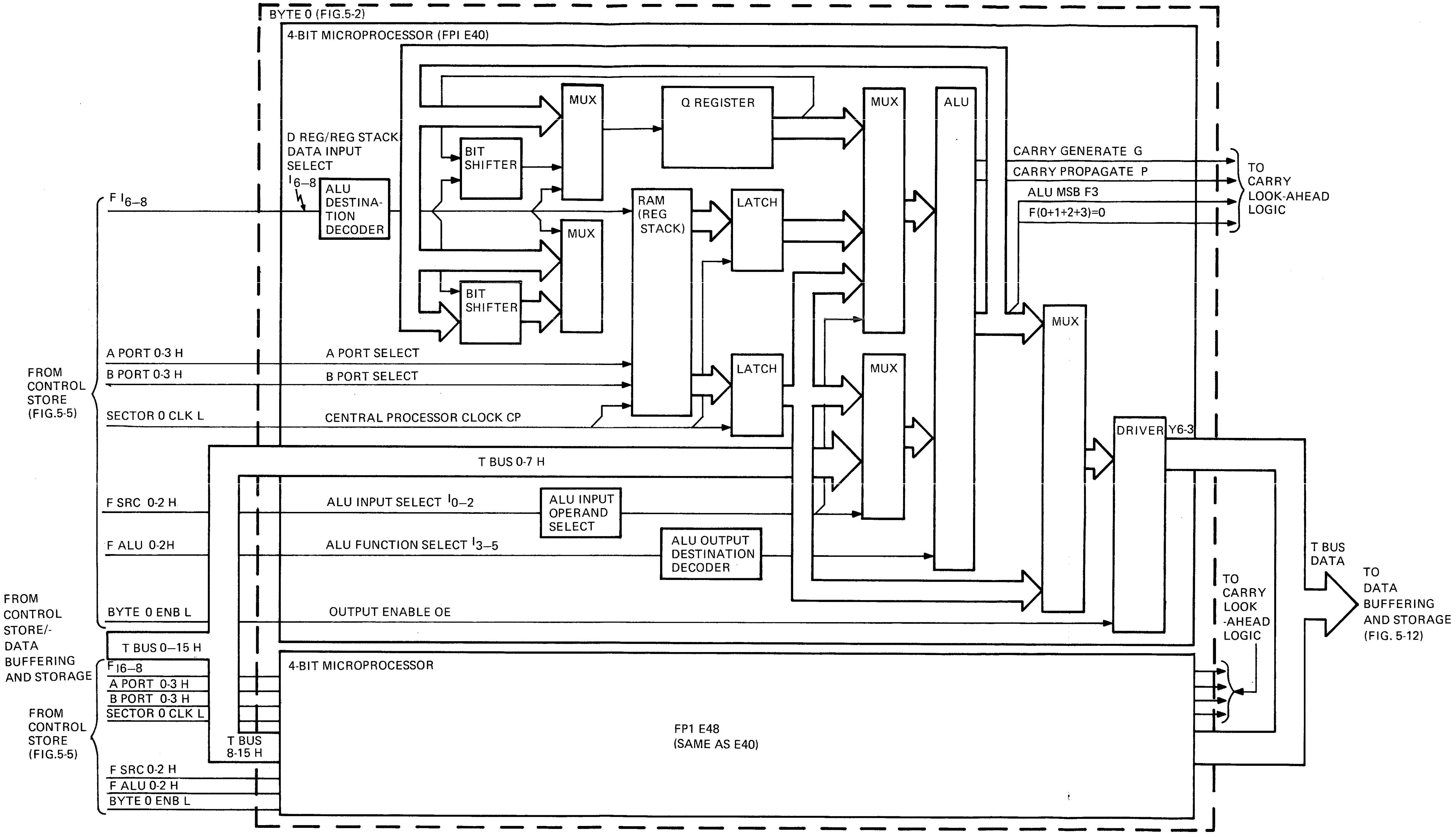


Figure 5-3 Sector/Byte Logic

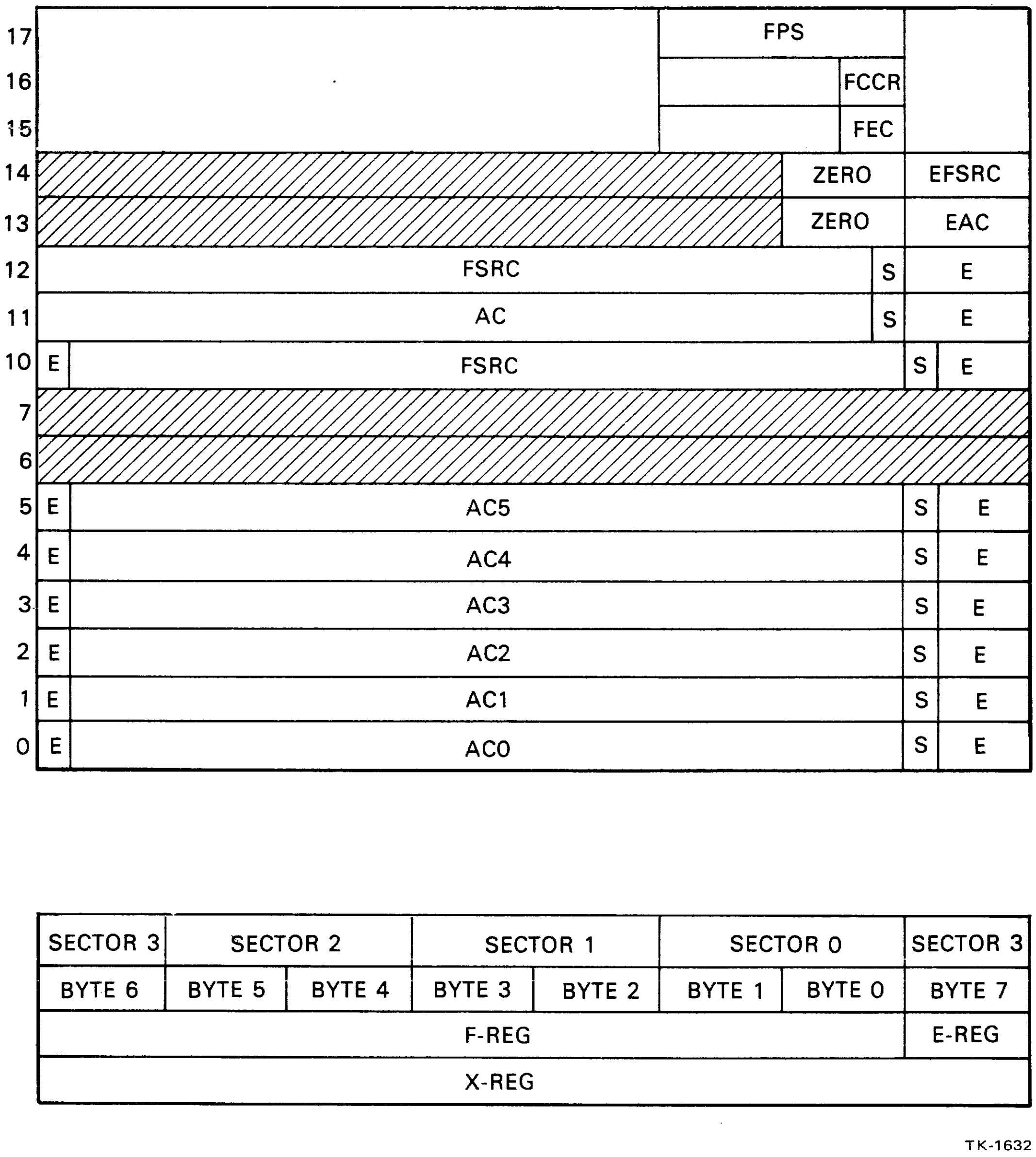


Figure 5-4 RAM Register Usage

Arithmetic Logic Unit (ALU) – The ALU is the data path component that actually performs the arithmetic/logical operation under command of the microcode (control word) (Table 5-1) contained in the control store logic PROM. R-inputs are fed in via a 2-input multiplexer whose inputs are the direct data inputs (T BUS 0–15 H) 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 5-1 Source Operand and ALU Function Matrix

	I_{210} Octal	0	1	2	3	4	5	6	7
I_{543} Octal	ALU Source ALU Function	**A,Q**	**A,B**	**O,Q**	**O,B**	**O,A**	**D,A**	**D,Q**	**D,O**
0	C_n=L R Plus S	A+Q	A+B	Q	B	A	D+A	D+Q	D
	C_n=H	A+Q+1	A+B+1	Q+1	B+1	A+1	D+A+1	D+Q+1	D+1
1	C_n=L S Minus R	Q–A–1	B–A–1	Q–1	B 1	A 1	A–D–1	Q–D–1	-D–1
	C_n=H	Q–A	B–A	Q	B	A	A–D	Q–D	-D
2	C_n=L R Minus S	A–Q–1	A–B–1	-Q–1	-B–1	–A–1	D–A–1	D–Q–1	D–1
	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	$\overline{D}\forall A$	$\overline{D}\forall Q$	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_{0-3} (T BUS 0–15 H). 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 ALU instructions i_{0-8} from the control store logic.

The ALU can perform three binary arithmetic and five logic operations on the two input words received via the R- and S-inputs. The R-input is driven by a 2-input multiplexer and the S-input from a 3-input multiplexer. As Figure 5-3 illustrates, the R-input multiplexer can be used to select either RAM A-port data or a direct data input as T BUS 0–15 H. The S-input multiplexer is from either the Q-register or the RAM output (port A or B). Both multiplexers have an inhibit output capability that produces a zero source operand.

ALU 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 (ALU INPUT SELECT I_{0-2}, ALU FUNCTION SELECT I_{3-5}) are described in Table 5-2. The carry input (C_n) has no effect in logic mode but does affect operations in arithmetic mode (Table 5-3). Both carry-in HIGH ($C_n = 1$) are defined.

Table 5-2 ALU Logic Mode Functions

Octal I_{543}, I_{210}	Group	Function	Octal I_{543}, I_{210}	Group	Function
40		$A \wedge Q$	74	Invert	$\overline{A}$
41		$A \wedge B$	77		$\overline{D}$
45	AND	$D \wedge A$			
46		$D \wedge Q$	62		Q
			63		B
30		$A \vee Q$	64	Pass	A
31		$A \vee B$	67		D
35	OR	$D \vee A$			
36		$D \vee Q$	32		Q
			33		B
60		$A \forall Q$	34	Pass	A
61		$A \forall B$	37		D
65	EX OR	$D \forall A$			
66		$D \forall Q$	42		0
			43		0
70		$\overline{A \vee Q}$	44	"Zero"	0
71		$\overline{A \vee B}$	47		0
75	EX NOR	$\overline{D \vee A}$			
76		$\overline{D \vee Q}$	50		$\overline{A} \wedge Q$
			51		$\overline{A} \wedge B$
72		$\overline{Q}$	55	Mask	$\overline{D} \wedge A$
73		$\overline{B}$	56		$\overline{D} \wedge Q$

Table 5-3 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

ALU Logical Functions for G, P, C_{n+4}, and OVR – Signals G, P, C_{n+4}, and OVR indicate carry and overflow conditions when the microprocessor is in the add or subtract mode. Table 5-4 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 5-4.

Q-Register – The Q-register is a file loaded from the ALU that is used to accumulate the quotient during division routines. It also functions as a temporary storage register. The Q-register output can be loaded back into itself, shifted right or left, as during fraction and multiplication and division operations.

Table 5-4 Logic Equations for ALU Functions

Definitions (+ =OR)

$P_0 = R_0 + S_0$	$G_0 = R_0 S_0$
$P_1 = R_1 + S_1$	$G_1 = R_1 S_1$
$P_2 = R_2 + S_2$	$G_2 = R_2 S_2$
$P_3 = R_3 + S_3$	$G_3 = R_3 S_3$

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

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

5.2.1.3 Data Manipulation – After data has been parallel loaded into the DML microprocessor, both the Q-register and any RAM address can be rotated or shifted either left or right. During a rotate, the bit transferred out one end is transferred in on the far end. During a shift operation, the bit transferred out is lost and a new bit must be generated and transferred in at the far end. To accomplish these shifts and rotations, the MSB of each 4-bit microprocessor is connected to the LSB of the adjacent more significant 4-bit microprocessor via a bidirectional transfer line. To complete the wraparound required to rotate data, the MSB of the complete data path is connected to the LSB via a bidirectional transfer line. To accomplish the shifts, the bidirectional transfer line between the MSB and LSB answer bit is left-shifted into bit 4 of byte 7. After 56 left-shifts, the complete answer is in bits 61 to 4 of bytes 6 to 0 and 7. During a double-precision divide, the quotient bits are left-shifted and can be interrupted by control logic. The same logic also generates the new bit for insertion into the LSB or MSB of the data path.

Division – During a divide operation, the divide quotient is generated one bit at a time and then left-shifted into the Q-register in the microprocessor. For a single-precision (float) quotient, the answer bits are left-shifted into bit 36 (byte 3) and, after 24 shifts, the complete fraction quotient is in bits 63 through 36 (bytes 6 through 3).

For a double-precision quotient, the answer bit is left-shifted into bit 4 (byte 7) and, after 56 left-shifts, the complete answer is in bits 61 to 4 (bytes 6 through 0 and 7). During a double-precision divide operation, the quotient bits are left-shifted from bit 35 to 36.

Multiplication – During a multiply operation, the multiplier is parallel loaded into the microprocessor Q-register in single- (float) or double-precision format. As the multiply operation is executed, the multiplier bits are right-shifted out, one bit at a time, always from the LSB position. For a single- (float) precision format, bit 40 is the LSB; for double-precision format, bit 8. The LSB (bit 40 or 8) is then used to determine if the multiply step should be an add and shift (LSB = 1) or a shift (LSB = 0).

Because the LSB of byte 7 is in the LSB of the data path and the MSB of byte 6 is in the MSB of the data path, special logic controls the transfer of data between these two bits. During a rotate right or left, the LSB and MSB are connected by bidirectional transfer lines. During a left-shift, the LSB is always zeroed; during a right-shift, a one or a zero can be loaded into the MSB.

5.2.1.4 Status Bits – Each DML 4-bit microprocessor generates two status bits: F=0 and F=3.

The F=0 status bit provides zero detection by indicating when exponent or fraction data equals zero. It goes high when all four ALU output bits are zero (low). It is an open-collector output that is combined in both fraction (bytes 6 through 0) and exponent (byte 7) portions of the data.

The F=3 output is used to monitor the MSB of each 4-bit microprocessor and also the sign of both exponent and fraction portions of the data path. Bit 7 is the exponent sign. It is the MSB of byte 7 and is monitored via the F3 output of that particular 4-bit microprocessor. Bit 63 is the fraction sign and is monitored via the F3 output of the byte 6 microprocessors. The F3 outputs of all other 4-bit microprocessors are not used.

The F=3 status bit is the MSB and can be monitored without enabling the output driver in the 4-bit microprocessor. The MSB output can be used as a sign bit during floating-point operations.

5.2.2 Carry Look-Ahead Logic

The DML contains full look-ahead carry logic that speeds instruction execution. Each of the sixteen 4-bit microprocessors generates both a carry generate (G) and a carry propagate (P) output. The four pairs of G and P signals from a sector are combined in a single look-ahead carry generator; the outputs of the four sector look-ahead carry generators are combined in a single generator, providing a second level of look-ahead carry generation (Figure 5-2). This arrangement allows the FP11-F data path to function with full 64-bit look-ahead carry generation.

5.3 CONTROL STORE (FIGURE 5-5)

During floating-point calculations, a sequence of microinstructions is applied (as control signals) to the microprocessor in the DML (Figure 5-2). As the microprocessor performs the instructions, it continually receives a new, revised set of instructions until an answer is sent to memory. These instructions are stored in PROM in the control store logic (Figure 5-5). They are read out of the PROM via MPC 0–10 H addressing and control read/write of microprocessor sectors/bytes; cause constants to be added as required; initiate branch testing; address RAM in the microprocessor; and also control fraction/exponent calculations.

The FP11-F control store logic instructions are used jointly with those of the CPU. The FP11-F control store logic PROM (twelve 1K × 4 bit) and the CPU control store ROM (seven 1K × 8 bit) comprise a 96-bit word that controls CPU/FP11-F operation (Figure 5-6).

5.3.1 Floating-Point Instruction Starting Code

At the start of a floating-point instruction, the CPU sends the FP11-F a 050 operation code (via MPC 0–10 L lines). This code is a PROM address that causes a floating-point instruction latch to generate FPX H. The FPX H signal is an enable for all the other latches in the control store logic. When nonfloating-point instructions are applied to the FP11-F, the floating-point instruction latch output is FPX L and, therefore, inhibits reading instructions out of PROM into the other latches.

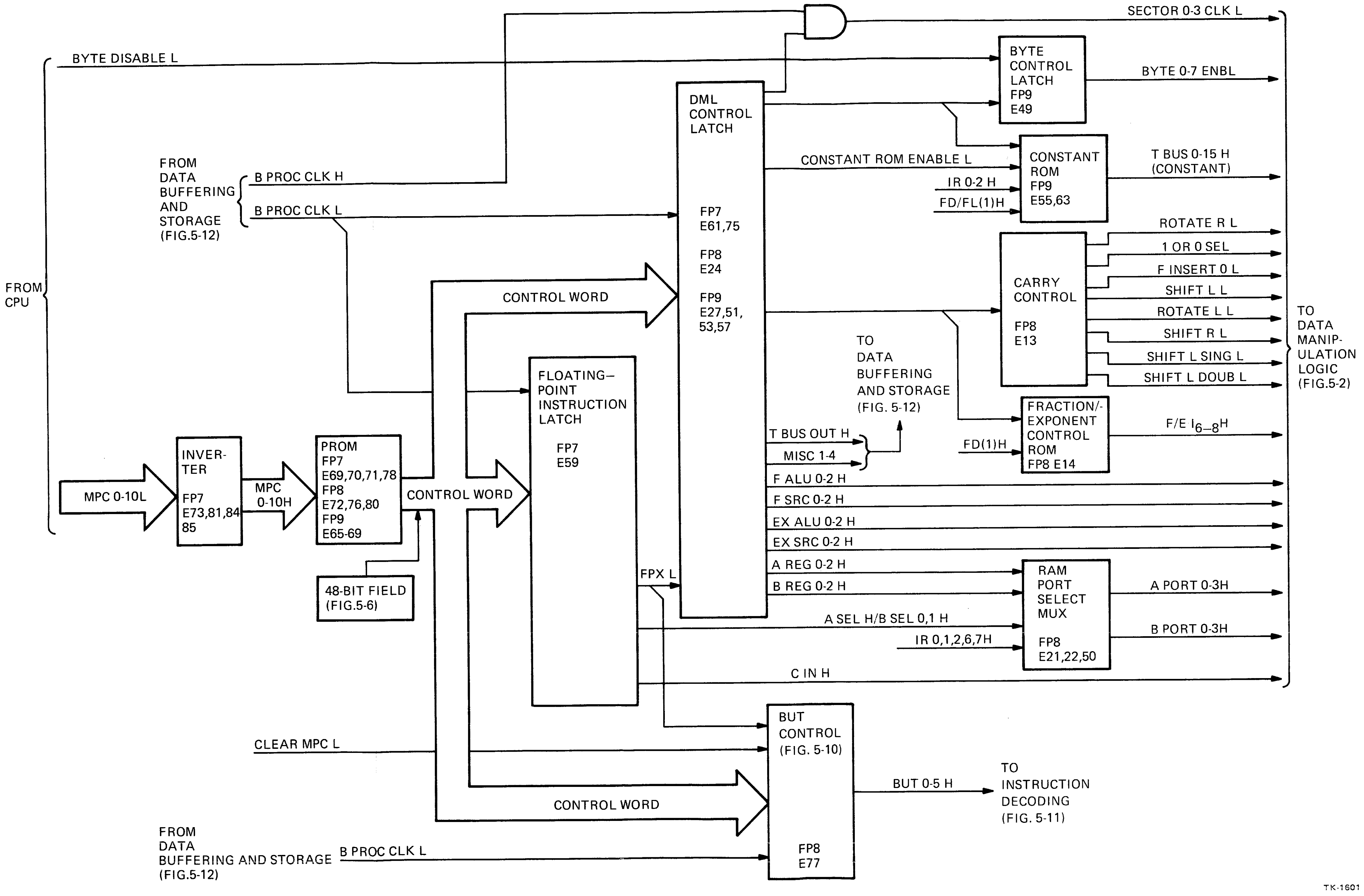
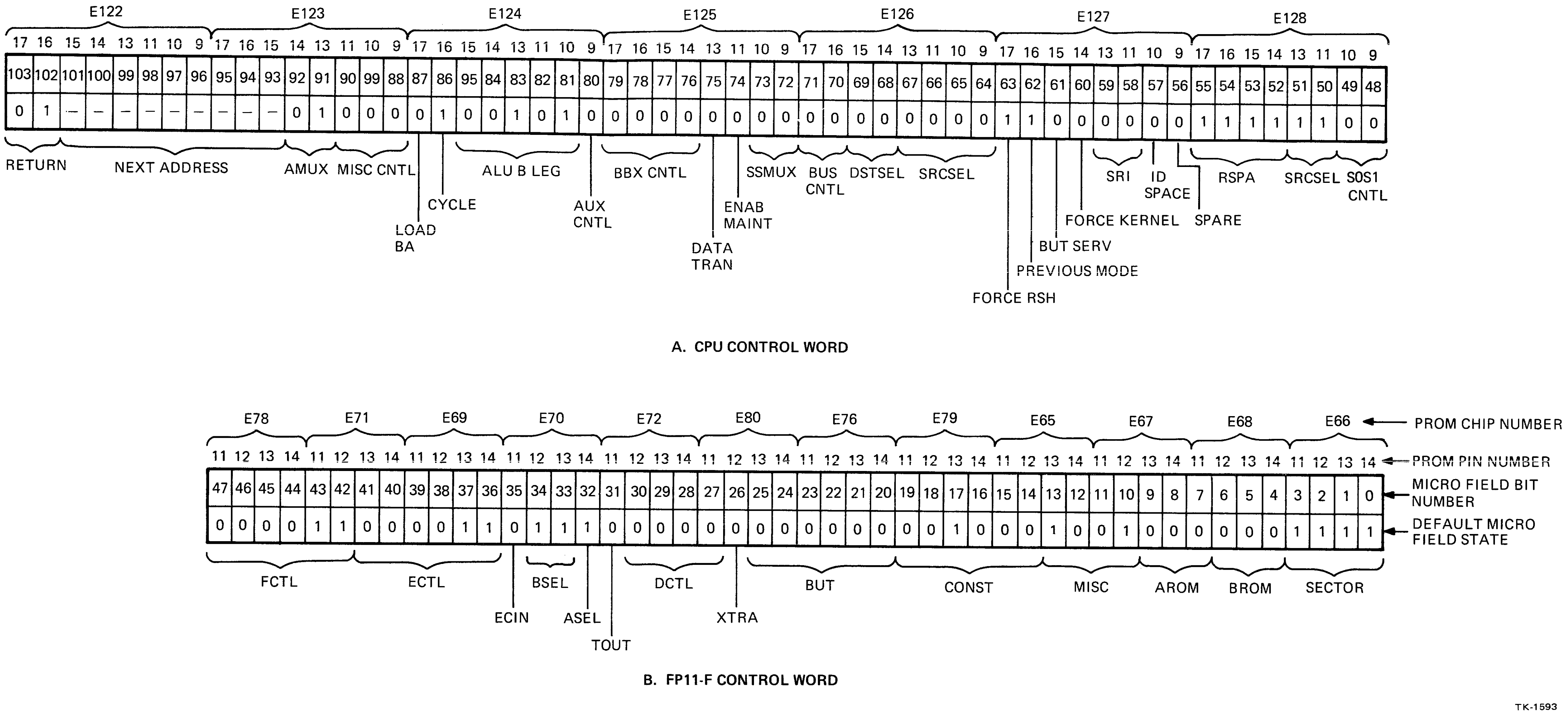


Figure 5-5 Control Store Logic



E122	17	103	0
	16	102	1
	15	101	–
	14	100	–
	13	99	–
	11	98	–
	10	97	–
	9	96	–
E123	17	95	–
	16	94	–
	15	93	–
	14	92	0
	13	91	1
	11	90	0
	10	99	0
	9	88	0
E124	17	87	0
	16	86	1
	15	95	0
	14	84	0
	13	83	1
	11	82	0
	10	81	1
	9	80	0
E125	17	79	0
	16	78	0
	15	77	0
	14	76	0
	13	75	0
	11	74	0
	10	73	0
	9	72	0
E126	17	71	0
	16	70	0
	15	69	0
	14	68	0
	13	67	0
	11	66	0
	10	65	0
	9	64	0
E127	17	63	1
	16	62	1
	15	61	0
	14	60	0
	13	59	0
	11	58	0
	10	57	0
	9	56	0
E128	17	55	1
	16	54	1
	15	53	1
	14	52	1
	13	51	1
	11	50	1
	10	49	0
	9	48	0

A. CPU CONTROL WORD

PROM CHIP NUMBER	PROM PIN NUMBER	MICRO FIELD BIT NUMBER	DEFAULT MICRO FIELD STATE
E78	11	47	0
	12	46	0
	13	45	0
	14	44	0
E71	11	43	1
	12	42	1
	13	41	0
	14	40	0
E69	11	39	0
	12	38	0
	13	37	1
	14	36	1
E70	11	35	0
	12	34	1
	13	33	1
	14	32	1
E72	11	31	0
	12	30	0
	13	29	0
	14	28	0
E80	11	27	0
	12	26	0
	13	25	0
	14	24	0
E76	11	23	0
	12	22	0
	13	21	0
	14	20	0
E79	11	19	0
	12	18	0
	13	17	1
	14	16	0
E65	11	15	0
	12	14	0
	13	13	1
	14	12	0
E67	11	11	0
	12	10	1
	13	9	0
	14	8	0
E68	11	7	0
	12	6	0
	13	5	0
	14	4	0
E66	11	3	1
	12	2	1
	13	1	1
	14	0	1

B. FP11-F CONTROL WORD

Figure 5-6 CPU/FP11-F Control Words

5.3.2 Sector Control

The sector control logic in the control store generates SECTOR 0,1,2 or 3 CLK L that selects one of four sectors in the DML microprocessor. It generates SECTOR 0,1,2, or 3 CLK L in accordance with a 4-bit sector field in the control word (Figure 5-6) that is read out of PROM. Within the DML each sector clock causes data to be loaded into a RAM or a Q-register in the 2901A. The sector control logic (Figure 5-7) consists of four NANDs and a DML control latch, synchronized via a buffered processor clock.

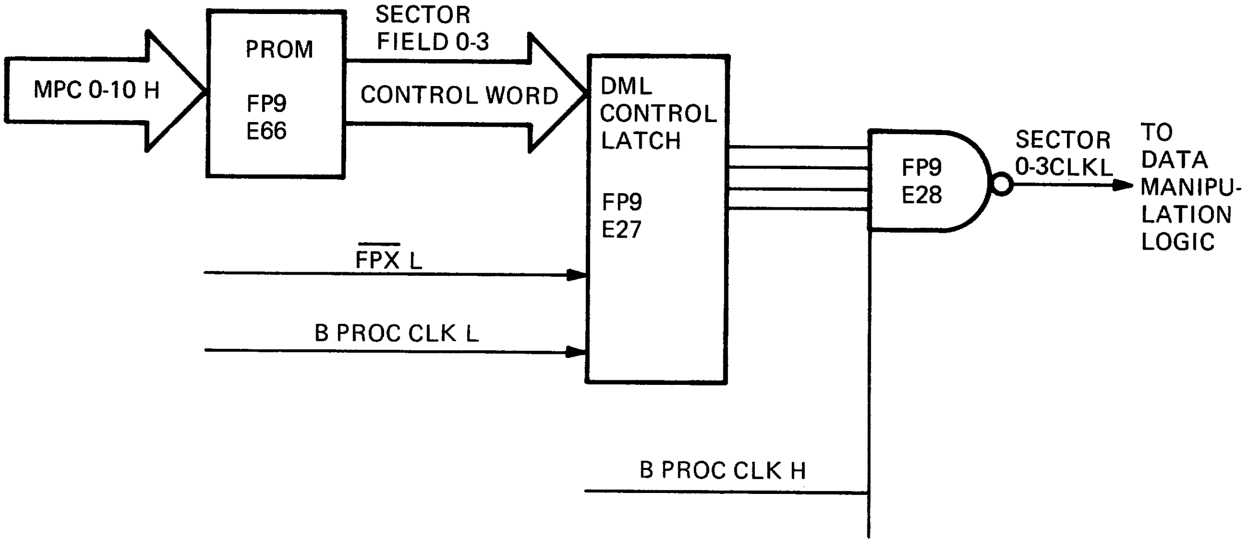


Figure 5-7 Sector Control Logic

An MPC 0–10 H address applied to PROM E66 selects one of four control words (SECTOR FIELD 0–3) that is loaded into latch E27. The B PROC CLK L clocked output of the latch is NANDed with B PROC CLK H as sector 0/1/2/3 CLK L. In the DML (Figure 5-2), this input is applied as CLOCK CP to RAM, two latches at the RAM output ports, and the Q-register.

Table 5-5 explains the octal coding of SECTOR FIELD 0–3 required to select a particular sector in the DML microprocessor. Each bit corresponds to a sector clock; a logical 1 in the bit position enables a clock for that sector and a 0 indicates no clock.

The sector clocks are independently controlled. The control word output coding of PROM can select any combination of sector clocks.

Table 5-5 Sector Enable

Sector	Octal Code	Definition
S1	1	Sector(s) 0 clocked
S2	2	Sector(s) 1 clocked
S3	3	Sector(s) 10 clocked
S4	4	Sector(s) 2 clocked
S7	7	Sector(s) 210 clocked
S10	10	Sector(s) 3 clocked
S14	14	Sector(s) 32 clocked
S17	17	Sector(s) 3210 clocked

NOTE
A zero octal code corresponds to no sectors being clocked.

5.3.3 Shift and Destination Control

The Q-register and register stack (RAM) in the DML (Figure 5-3) receive an input that is selected via I_{6-8} generated in the shift and destination control logic. This logic consists of two ROMs that are addressed via a latch loaded with destination control (DCTL) word field bits 27–30 (Figure 5-5).

Table 5-6 lists ROM outputs versus PROM control word output (DCTL). The field mnemonics in Table 5-2 are followed by the octal code for the mnemonic and a generalized data transfer operation. For example, SLALUOQ has an octal value of 15 in the control word field. It affects both the fraction and the exponent. The operation B → SLO (ALU)F → Y;Q → SLC63(Q) is interpreted as the B-port of the DML microprocessor ROM and gets the value of the ALU shifted left one position with a 0 inserted into the LSB. The ALU output is selected to Y, and the Q-register, loaded with the Q-register output shifted left one position with the carry-out bit, is inserted into the LSB of the Q-register.

Table 5-7 gives the field mnemonic (SLALUOQ) followed by its octal value (15), followed by the ROM output values. The FDST value (6) is the value applied to F I_{6-8} fraction data path of the microprocessor ALU destination decoder; EX DST value (6) is applied to E I_{6-8} of the exponent data path, causing both sets of 4-bit microprocessors to perform a shift-left (up) and load the RAM and Q-registers.

At the shift and destination control logic, output highs on signals ROTATE R L, SHIFT T L, ROTATE L L, and F INSERT 0 L cause shift gates in the DML to go to the high impedance state. The low on SHIFT L L causes a 0 to be inserted into the LSB of the RAM shift path. The values of SHIFT L DOUB L and SHIFT L SING L depend on the FD status bit value. One of these two signals would be low, enabling the carry-out to be inserted into the Q-register shift path.

Table 5-6 Shift and Destination Control ROM Outputs – Data Transfer Operations

Function	Octal Code	Fraction/ Exponent	Data Transfer
NOP	0	Both	- - -
LDBT	1	Both	B-ALU T→Y
LDBF	2	Both	B-ALU F→Y
LDQF	3	Both	Q-ALU F→Y
ROTL	4	Both	B-ROTL(ALU) F→Y
ROTR	5	Both	B-ROTR(ALU) F→Y
SLALU0	6	Both	B-SL0(ALU) F→Y
SR0ALU	7	Both	B-SR0(ALU) F→Y
SR1ALU	10	Both	B-SR1(ALU) F→Y
SLALU0.LDBF	11	Fraction	B-SL1(ALU) F→Y; EXP B-ALU F→Y
SR1ALU.LDBF	12	Fraction	B-SR1(ALU) F→Y; EXP B-ALU F→Y
SR0ALUQ	14	Both	B-SR0(ALU) F→Y; Q-SR0 (Q)
SLALU0Q	15	Both	B-SL0(ALU) F→Y; Q-SLC63 (Q)

Table 5-7 Shift and Destination Control ROM Output Values

Shift Dest Field Function	F DST (3-Bit Field)	EX DST (3-Bit Field)	↑ SHIFT L DOUB L	↑ SHIFT L SING L	↑ SHIFT R L	↑ ROTATE LL	↑ SHIFT LL	↑ F INSERT 0 L	↑ 1 OR 0 SEL L	↑ ROTATE R L
NOP (0)	1	1	H	H	H	H	H	H	H	H
LDBT (1)	2	2	H	H	H	H	H	H	H	H
LDBF (2)	3	3	H	H	H	H	H	H	H	H
LDQF (3)	0	0	H	H	H	H	H	H	H	H
ROTL (4)	7	7	H	H	H	L	H	H	H	H
ROTR (5)	5	5	H	H	H	H	H	H	H	L
SLALU 0 (6)	7	7	H	H	H	H	L	H	H	H
SR0ALU (7)	5	5	H	H	L	H	H	H	L	H
SR1ALU (10)	5	5	H	H	L	H	H	H	H	H
SLALU0.LDBF (11)	7	3	H	H	H	H	H	L	H	H
SR1ALU.LDBF (12)	5	3	H	H	L	H	H	H	H	H
SR0ALUQ (14)	4	4	H	H	L	H	H	H	L	H
SLALUQQ (15)	6	6	*	†	H	H	L	H	H	H

* L if FD(1) = 1; H if FD(1) = 0
† L if FD(1) = 0; H if FD(1) = 1

5.3.4 RAM A/B Port Control

RAM A/B port control logic (Figure 5-8) generates A PORT 0–3 H, B PORT 0–3 H that control RAM data flow in the DML (Figure 5-2). It consists of two multiplexers.

Multiplexers E50 and E21,22 receive an IR input from the instruction decoding logic and an A REG 0,1,2 H/B REG 0,1,2 H input from an A/B port select latch (E27, 51). The latch input is read out of PROM as control word A ROM (7, 8, 9) and B ROM (4, 5). Control word bits A SEL (32) and B SEL (33, 34) are clocked (via B PROC CLK L) out of an FP instruction latch (E59) to become A SEL H or B SEL 0/1 H. These signals select one of the two multiplexers to apply A PORT 0–3 H or B PORT 0–3 H to the DML.

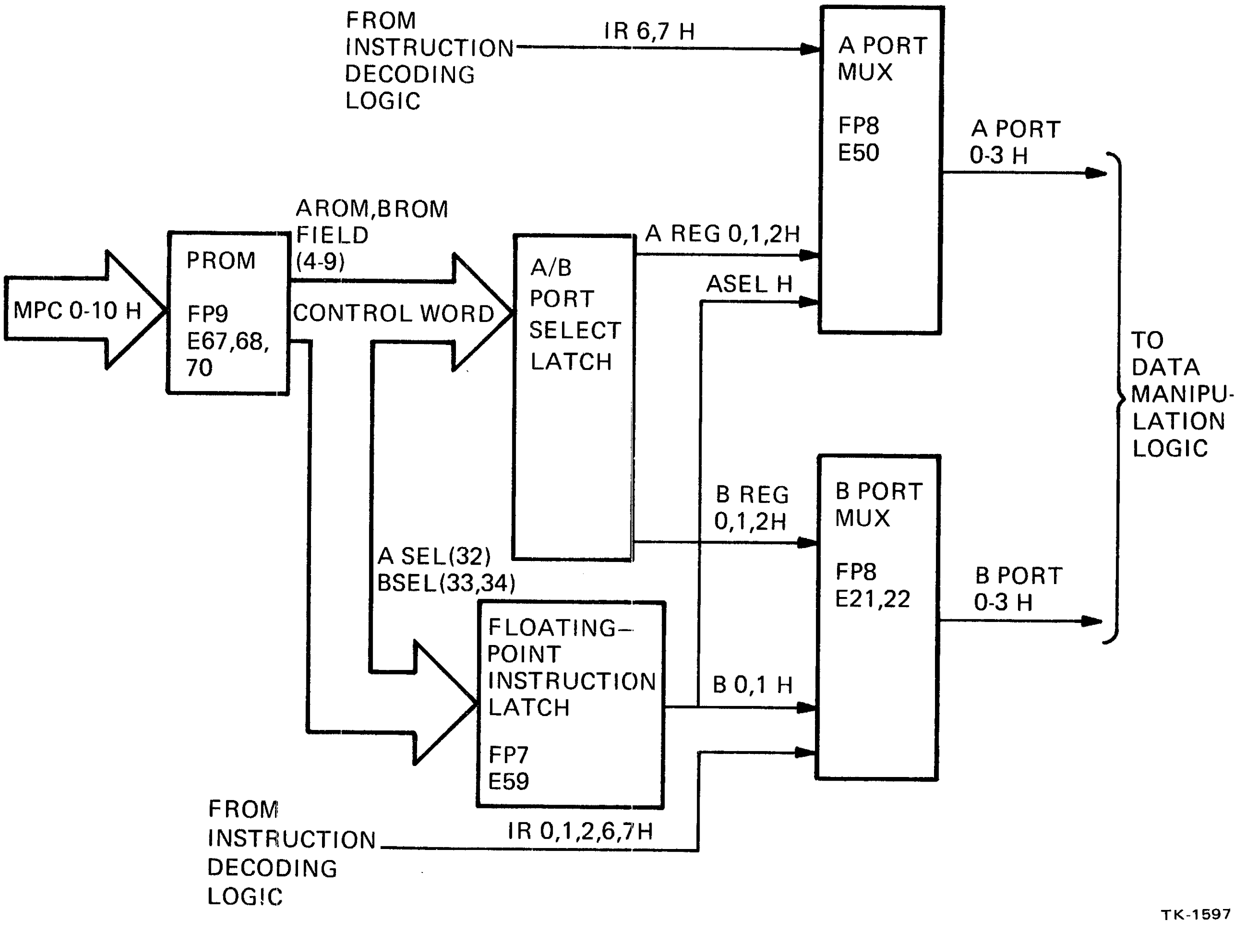


Figure 5-8 RAM A/B Port Control Logic

Table 5-8 explains octal coding of the A ROM control word output (bits 7–9) of PROM and Table 5-9 explains coding of the B ROM control word (bits 4–6).

Table 5-8 A-Address ROM

Address	Octal Code	Description
AR10	0	
AR11	1	
AR12	2	Used to specify E, F, or X registers (e.g., X12 = AR12) on the A-port of the RAM.
AR13	3	
AR14	4	
FEC	5	Corresponds to register 15 in the RAM
FCCR	6	Corresponds to register 16 in the RAM
FPS	7	Corresponds to register 17 in the RAM

Table 5-9 B-Address ROM

Address	Octal Code	Description
BR10	0	
BR11	1	
BR12	2	Used to specify as E, F, or X register on the B-port of the RAM.
BR13	3	
BR14	4	
FEC	5	Corresponds to register 15 of the RAM
FCCR	6	Corresponds to register 16 of the RAM
FPS	7	Corresponds to register 17 of the RAM

5.3.5 Fraction, Exponent Control

The fraction and exponent fields in the control word (Figure 5-6) are used to generate a 6-bit word that controls the input and functions of the ALU in the DML microprocessor. It provides this control via F/EX SRC 0–2 H and F/EX ALU 0–2 H. Within the DML microprocessor, these signals function as ALU INPUT SELECT I_{0-2} and ALU FUNCTION SELECT I_{3-5}, respectively.

The control word (Figure 5-6) in the control store PROM (E69, 71, 78) contains fraction control FCTL bits 42–47 and exponent control EXCTL bits 36–41. These bits are applied as 6-bit addresses to ROM. The ROMs are clocked by B PROC CLK L to apply either F ALU 0–2 H, F SRC 0–2 H, or EX ALU 0–2 H, EX SRC 0–2 H to the 4-bit microprocessors in the DML. Here, the signals are ALU INPUT SELECT I_{0-2} (applied to an ALU operand select circuit) and ALU FUNCTION SELECT I_{3-5} (applied to an ALU output destination decoder).

Because the fraction and exponent portions of the microprocessor T BUS 0–15 H input lines are independently controlled, the fraction and exponent can be manipulated individually during a single microstate. However, the complete data input (all eight bytes) of the microprocessor can be used as a unit by placing the same fraction and exponent codes on both sets of control lines.

Table 5-10 shows the octal value of the control store logic control word (Figure 5-6) and the octal value for control of the source operand ALU function in the microprocessor.

Table 5-10 Fraction and Exponent Control Fields

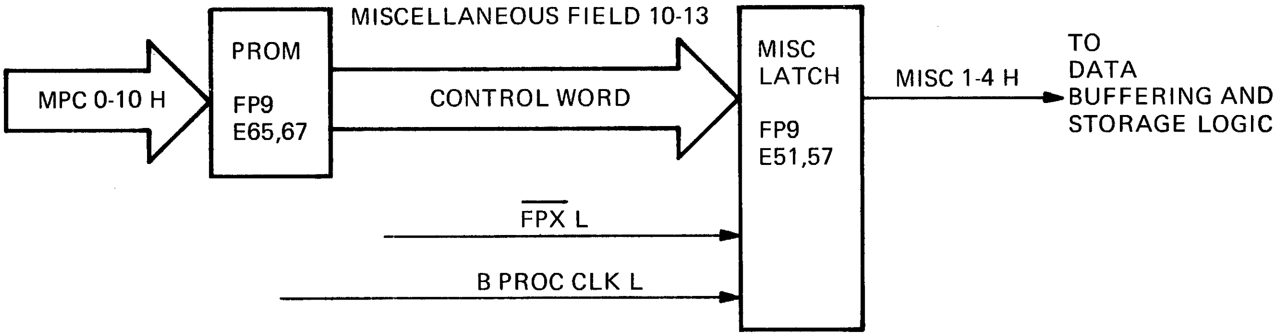
Source Operand ALU Function	Octal Code for Field	Source Operand ALU Functions	Octal Code for Field
A AND Q	40	DBAR	77
A AND B	41		
D AND A	45	QPASS	62
D AND Q	46	BPASS	63
		APASS	64
A OR Q	30	DPASS	67
A OR B	31		
D OR A	35	ZERO	42
D OR Q	36		
		ABAR AND Q	50
A XOR Q	60	ABAR AND B	51
A XOR B	61	ABAR AND A	55
D XOR A	65	ABAR AND Q	56
D XOR Q	66		
		A PLUS Q	0
A XNOR Q	70	A PLUS B	1
A XNOR B	71	D PLUS A	5
D XNOR A	75	D PLUS Q	6
D XNOR Q	76		
		Q PLUS	2
QBAR	72	B PLUS	3
BBAR	73	A PLUS	4
ABAR	74	D PLUS	7

Table 5-10 Fraction and Exponent Control Fields (Cont)

Source Operand ALU Function	Octal Code for Field	Source Operand ALU Functions	Octal Code for Field
Q-1	12	Q-A-1	10
B-1	13	B-A-1	11
A-1	14	A-D-1	15
D-1	27	Q-D-1	16
		A-Q-1	20
-Q-1	22	A-B-1	21
-B-1	23	D-A-1	25
-A-1	24	D-Q-1	26
-D-1	17		

5.3.6 Miscellaneous Control

A 4-bit miscellaneous field read out of the control store logic PROM controls data direction flow in the data buffering logic and also provides byte control in the DML. Miscellaneous control is performed by two latches (Figure 5-9).



TK-1594

Figure 5-9 Miscellaneous Control Logic

MPC 0–10 H addressing causes PROM (E64, 67) to apply miscellaneous field bits 10–13 of the control word (Figure 5-6) to latches E51 and 57. Signal MISC 1, 2, 3, or 4 H is clocked as an address into a bus control ROM (E20) in the data buffering and storage logic (Figure 5-12). E20 causes an AMUX 0–15 L data transfer into the FP11-F data buffering logic, loads the data buffers, and then loads the buffered data onto the T BUS as T BUS 0–15 H.

5.3.7 Constants Generation

Two ROMs in the control store logic contain various fixed-value numbers (constants) required during floating-point calculations. These constants include excess 200 bias, iteration counts, rounding bytes, correlation factors, and index numbers. The control word loaded into the DML control latch is applied to the constant ROMs along with status bits [FD (1)/FL (1) H], and IR 0–2 H. These inputs are combined into a ROM address that selects a certain constant output to be applied as T BUS 0–15 H to the DML. The constant output is a correction word for calculations being performed in the DML microprocessor.

Two ROMs contain the constants required for floating-point calculations. One ROM (low-byte constant) drives T-bus bits 0–7, the other ROM (high byte) drives bits 8–15. Many of the constants accessed are relatively small numbers; as a result, only 28 out of 256 locations in the high-byte ROM are nonzero. When they are enabled, the ROMs output a 16-bit constant on the T-bus.

The 6-bit constant field PROM output generates most of the ROM pair addresses, with the FD, FL, and PC address mode bits completing the address. Of the three inputs applied to the ROMs, the FD/FL bits indicate which data format is being used. FD indicates either single- or double-precision format. FL indicates either integer (16-bit) (FL=0) or long (32-bit) (FL=1) format for integer numbers. Constant generation also defines the floatingpoint instruction. Thus determining single, immediate, or double mode operation.

5.3.8 Byte Control

The byte enable ROM BYTE 0–7 ENB L output enables one of eight individual data bytes to be loaded from the DML onto the T-bus. The byte enable ROM (E49) and the constant ROM (E55, 63) are both controlled by the 6-bit constant field read out of PROM. The MSB of the constant field (bit 19) is the enable for these ROMs.

5.3.9 Branch Under Test Control

The control store logic generates a BUT 0–5 H output that is used in the instruction decoding logic (Figure 5-11) to modify the base MPC 0–10 L it generates. In the instruction decoding logic, BUT 0–5 H may be combined with IR 0–11 H to select a particular address in one or two of four branching ROMs. The address accessed in the ROMs contains a mask that selects the conditions to be monitored and combined with MPC 0–10 L.

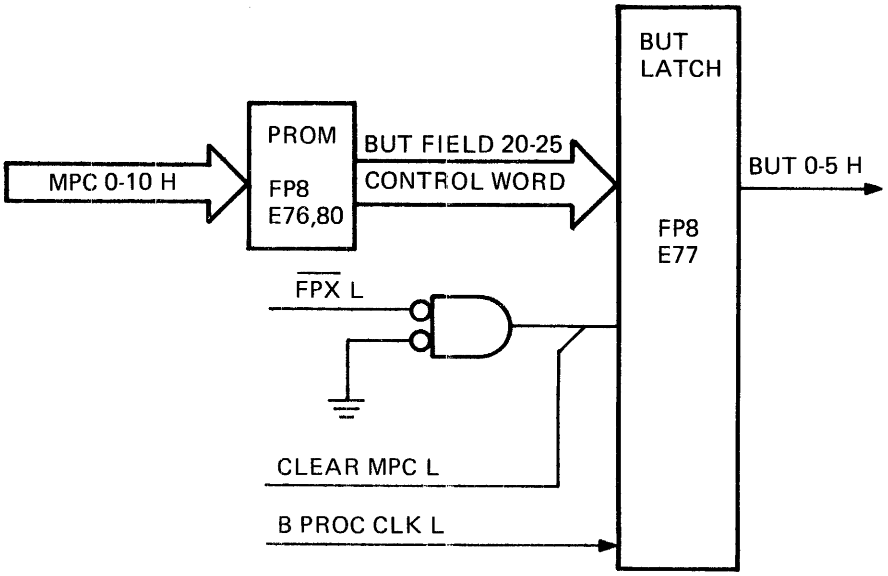


Figure 5-10 BUT Logic

The BUT logic (Figure 5-10) consists of latch E77 that is loaded from PROM E76, 80 with BUT bits 20–25. It is clocked by B PROC CLK L. The CLR MPC L input is a factory test only signal input.

Table 5-11 lists the mnemonics and octal values of control word BUT bits 20–25, followed by the definition of the bits (or bits under test). For example, EZBT.Y8.Y9 corresponds to a field value of 62 and indicates that branching is on three possible values of the exponent zero bit, T-bus 0 bit (corresponding to Y8) and the T-bus 1 bit (corresponding to Y9). The X.Y nomenclature does not mean that a branch on the ANDed value is occurring, but that a simultaneous branch is occurring on the two values. Thus, a branch to one of four locations is possible with X.Y.

Table 5-11 BUT Control (Branch on Test Enables)

Function	Octal Code	Definition
NOP	0	
ENBT	54	Exponent Sign Bit
EZBT	52	Exponent Zero Detect Bit
BUSRQ	55	Bus Request-Grant Pending
FNBT	56	Fraction Sign Bit
XZBT	57	Combined Zero Detect Bit
Q8.Q40	60	LSB of Multiplier
COUT63	61	Fraction Carry-Out
EZBT.OP1A	13	EZBT.OP1A
FID	41	Floating Interrupt Disable
EZBT.Y8.Y9	62	EZBT, TBUS0, TBUS1
EZBT.Y8	63	EZBT, TBUS0
ENBT.Y8	64	ENBT, TBUS1
ENBT.EZBT.FNBT	65	See Above
FNBT.XZBT	66	See Above
FIV	42	Interrupt on Over Flow
Y62	43	TBUS 6
FIU	45	Interrupt on Under Flow Bit
FIUV	46	Interrupt on Undefined Variable
FIC	47	Interrupt on Integer Conversion Error Bit
FT	44	Truncate Bit
ENBT.EZBT	67	See Above
Y8	50	TBUS 0
Y61	44	TBUS 5
FD.FIV	2	Floating Double Bit, Interrupt on Overflow
Y62.OP1A	3	TBUS 6
BREAKOUT	31	Initial Instruction Decode
FD	32	Floating Double Mode Bit
OP1B	34	Instruction Decode
OP1C	35	
OP1D	36	
OP1E	37	
OP2A	77	

Table 5-11 BUT Control (Branch on Test Enables) (Cont)

Function	Octal Code	Definition
FDST	71	Decodes Floating Destination
FSRC	71	Decodes Floating Source
DST	72	Decodes Destination
SRC	72	Decodes Source
GR7	73	General Register 7
GR7.FLBAR	74	General Register 7, FL Bit = 0
FL	75	FL Bit = 1
FLBAR	76	FL Bit = 0

Some of the branches are branches on T-bus bits but are indicated by other mnemonics. A T-bus bit value depends on what device is enabled onto the T-bus; the mnemonic thus reflects what is being enabled onto the T-bus at that time. Table 5-12 may be referenced for the possible bits that are branched on by enabling onto the T-bus. For example, a branch on T BUS 5 is actually occurring on a branch of the FT bit or Y61.

Table 5-12 BUT Control Functions

(X = Don't Care)

(Z = Bits to be decoded)

(A = corresponds to address lines of ROM)

BUT Field						Functions
5	4	3	2	1	0	T-Bus BUT ROM not decoding; CC BUT ROM decodes bits 0,1,2 and generates proper BUT.
A5	A4	A3	A2	A1	A0	
X	0	1	Z	Z	Z	
			0	0	0	Y8/T-Bus 0
			0	0	1	Unused
			0	1	0	EZ Bit
			0	1	1	Unused
			1	0	0	EN Bit
			1	0	1	PFAIL BR PEND (BUS RQST)
			1	1	0	FN Bit
			1	1	1	FZ Bit
X	0	0	Z	Z	Z	CC BUT ROM not decoding; T-Bus BUT ROM decodes bits 0,1,2 and generates proper BUT.
			0	0	0	
			0	0	1	T-Bus 14 (FID)
			0	1	0	T-Bus 9 (FIV)

Table 5-12 BUT Control Functions (Cont)

BUT Field						Functions
			0	1	1	T-Bus 6 (Y62)
			1	0	0	T-Bus 5 (Y61/FT)
			1	0	1	T-Bus 10 (FIU)
			1	1	0	T-Bus 11 (FIUV)
			1	1	1	T-Bus 8 (FIC)
5	4	3	2	1	0	Both CC and T-Bus ROMS to decode bits 0,1,2 and generate proper BUT.
A5	A4	A3	A2	A1	A0	
X	1	0	Z	Z	Z	
			0	0	0	Q8.Q40
			0	0	1	COUT 63
			0	1	0	EZ Bit, T-Bus 0 (Y8), T-Bus 1 (Y9)
			0	1	1	EZ Bit, T-Bus 0 (Y8)
			1	0	0	EN Bit, T-Bus 0 (Y8)
			1	0	1	EN Bit, EZ Bit, FN Bit
			1	1	0	EN Bit, XZ Bit
			1	1	1	EN Bit, EZ Bit
X	1	1	X	X	X	Neither CC BUT ROM or T-Bus BUT ROM Enabled
			A9	A8	A7	OP1 ROM (Decodes Upper 6 Bits of IR Register)
0	X	X	Z	Z	Z	OP1 ROM Enabled
			2	1	0	
			0	0	0	NOP
			0	0	1	BUT Breakout
			0	1	0	BUT FD
			0	1	1	OP1A
			1	0	0	OP1B
			1	0	1	OP1C
			1	1	0	OP1D
			1	1	1	OP1E
						OP2 ROM (Decodes Lower 6 Bits of IR Register)
1	1	1	Z	Z	Z	OP2 ROM Enabled
			0	0	0	NOP
			0	0	1	FDST/FSRC
			0	1	0	DST/SRC
			0	1	1	GR7
			1	0	0	GR7.FL BAR
			1	0	1	FL
			1	1	0	FL BAR
			1	1	1	OP2A

5.4 INSTRUCTION DECODING (FIGURE 5-11)

The instruction decoding logic generates operation codes as MPC 0–10 L. These codes are applied to the control store logic which, in turn, generates control signals and instructions for the DML. The instruction decoding logic also generates IR 0–2,6,7 H that are used in the addressing of constant ROMs in the control store logic. The instruction decoding logic consists of an instruction register and four ROMs.

Instructions received as T BUS 0–15 H from the CPU or the DML are decoded in the instruction register (E58, 62) as IR 0–11 H. This output is applied to the control store logic constant ROMs. It is also applied to two (OP1, OP2) operation code ROMs.

The OP1 and OP2 ROMs are used for branching based on IR 0–1 H coding. The OP1 ROM is addressed via IR 6–1 H and is used when branch decisions must be based on operation code and accumulator content. The OP2 ROM is addressed via IR 0–5 and branches on address mode and the special no-argument instruction.

The condition code ROM (E83) and the T-bus branch ROM (E85) are used for signal monitoring. The condition code ROM monitors eight hardware status bits that can be viewed directly. The T-bus branch ROM monitors signals stored or generated in the DML. Both ROMs are addressed via BUT 0–5. They control the MPC bits in two different ways. The condition code and T-bus ROMs each contain 32 8-bit masks. These masks either enable or prevent a particular signal from affecting MPC 0–8. The ROMs control only eight signals each and each signal can affect only one MPC bit.

5.5 DATA BUFFERING AND STORAGE (FIGURE 5-12)

The data buffering and storage logic provides bidirectional buffering of data transfers to and from the CPU and also temporary storage of data to be transferred. It consists of a bus transceiver, latch, and T-bus control.

5.5.1 Data Buffering

AMUX 0–15 L data received from the CPU is buffered by the transceiver (E17,25). Conversely, the T BUS 0–15 H output of the DML microprocessor is buffered and then applied to the CPU as AMUX 0–15 L. The bus transceiver is enabled by T BUS OUT L, which is generated in the T-bus control.

In the T-bus control (E9, 20), TAP 90 L from the CPU clocks a flip-flop (E9), the low output of which is ANDed with FREE BUS H.

NOTE

TAP 90 L is generated via a delay line in the CPU and provides signal skew. Skew provides enough settling time in the 2901As when loading data.

The resultant signal output is ORed with FORCE FPP DATA L to produce T BUS OUT L. T BUS OUT L enables the bus transceiver.

5.5.2 Data Storage

A latch provides temporary T-bus data storage when the CPU cannot be interrupted to receive data from the DML microprocessor. The latch consists of four 4-bit data registers (E30, 54, 56, 64). They are loaded from the T-bus during B PROC CLK L and are enabled by IN H/L ENB L from the T-bus control. Data in the latches is gated to the T-bus by OUT ENB L. Signal IN L ENB L is used to load the buffer with condition code status information.

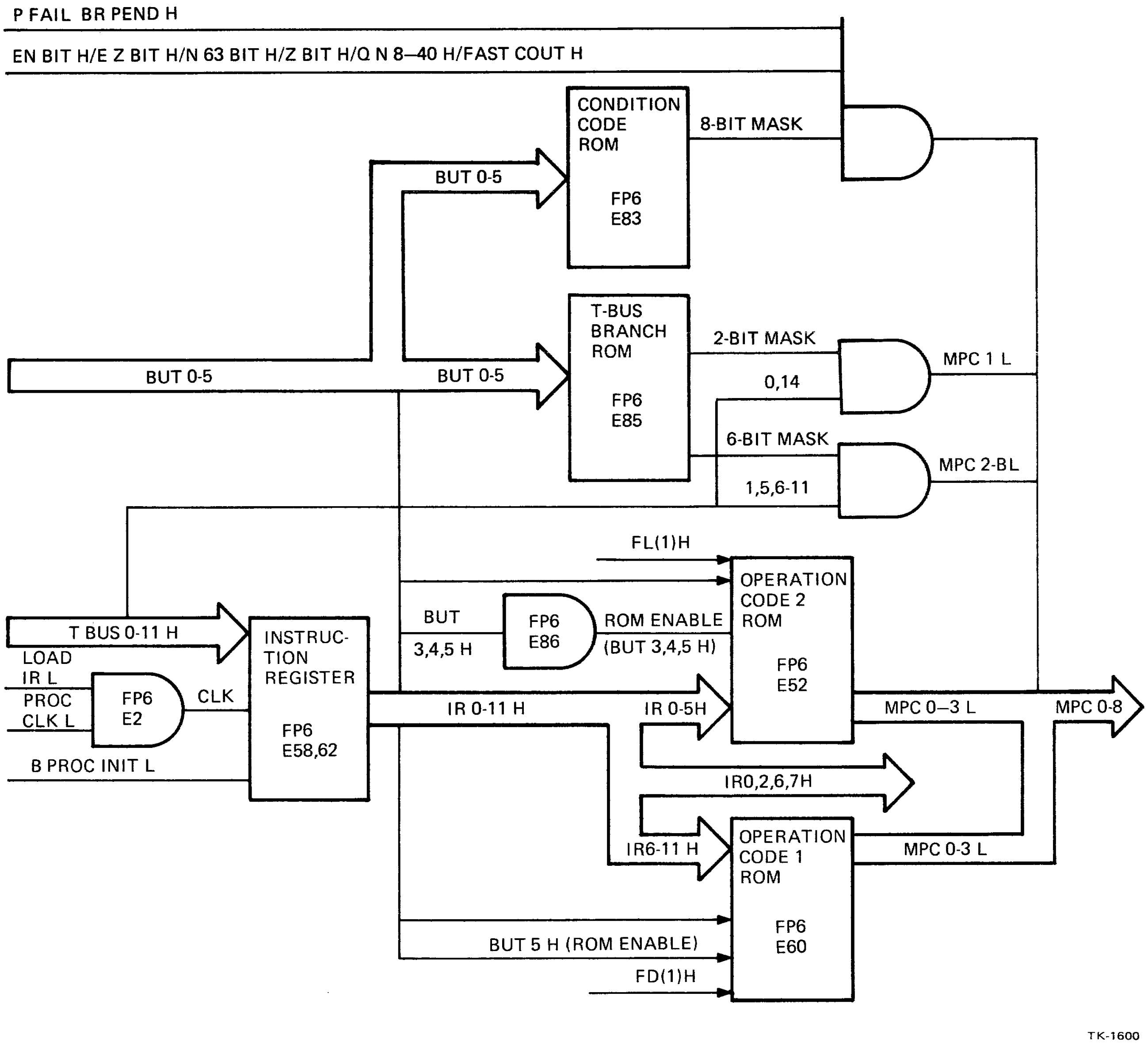


Figure 5-11 Instruction Decoding Logic

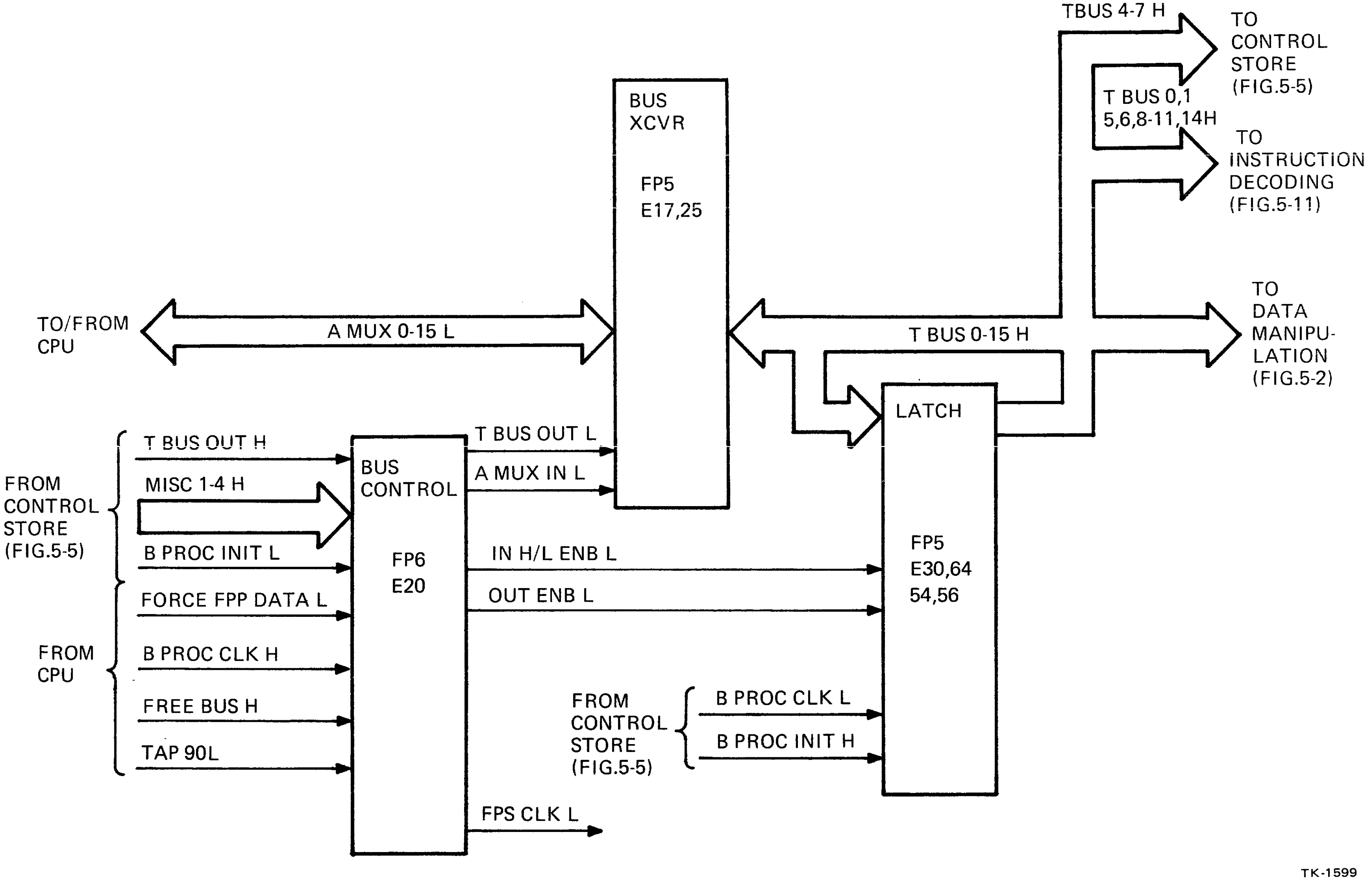


Figure 5-12 Data Buffering and Storage Logic

CHAPTER 6
FLOATING-POINT INSTRUCTIONS

6.1 FLOATING-POINT ACCUMULATORS

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

Each accumulator is interpreted as being either 32 or 64 bits long, depending on the instruction and the FP11-F status . If an accumulator is interpreted as being 64 bits long, 64 bits of data occupy the entire accumulator. If an accumulator is interpreted as being 32 bits long, 32 bits of data occupy only the leftmost 32 bits of an accumulator as shown in Figure 6-1.

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

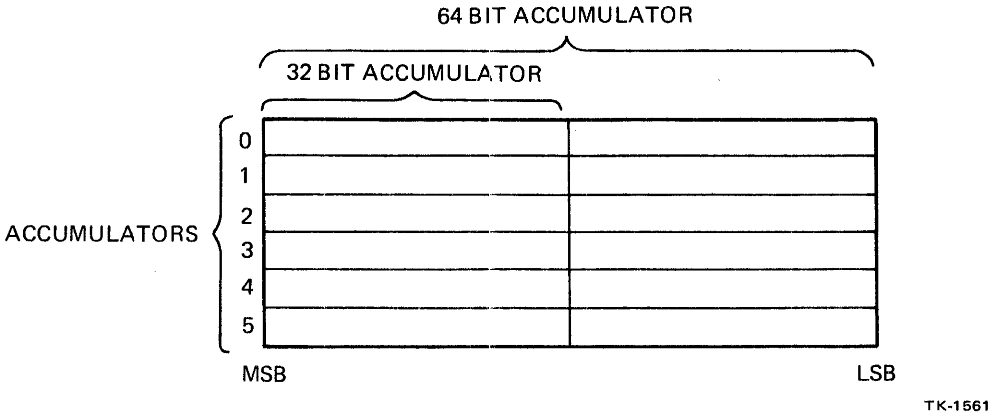


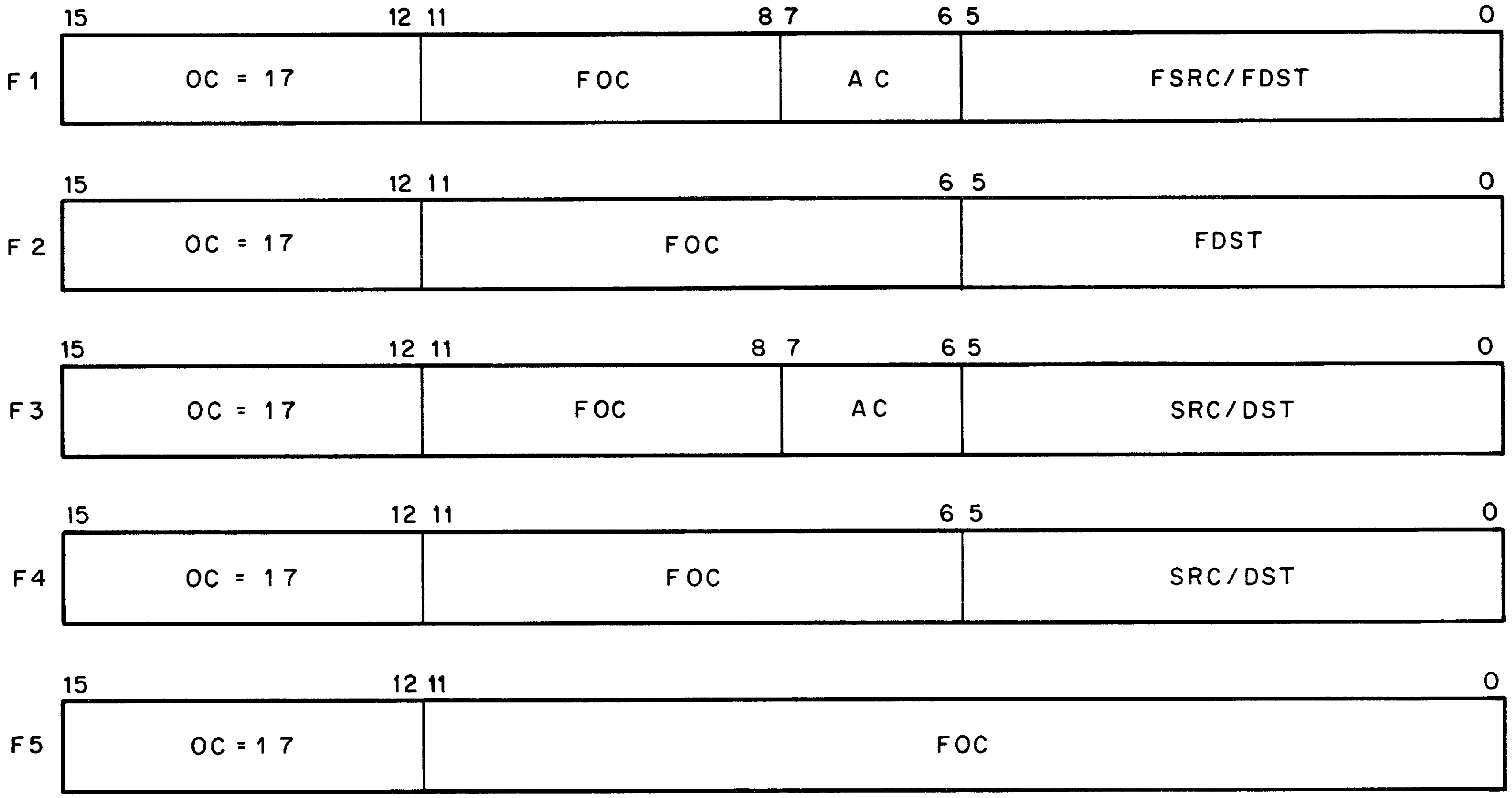
Figure 6-1 Floating-Point Accumulators

6.2 INSTRUCTION FORMATS

An FP11-F instruction must be in one of five formats. These formats are summarized in Figure 6-2.

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-F traps if floating-point interrupts are enabled (FID = 0). The FEC will indicate an illegal op code error (exception code 2).



TK-1628

Figure 6-2 Instruction Formats

The fields of the various instruction formats (as summarized in Table 6-1) 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-F scratchpad accumulators 0–3.

Table 6-1 Format of FP11-F 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

Table 6-1 Format of FP11-F Instructions (Cont)

Instruction Format	Instruction	Mnemonic
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

6.3 INSTRUCTION SET

Table 6-2 contains the instruction set of the FP11-F. 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 on 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 6-2 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 6-2 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 6-2.

Table 6-2 FP11-F 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

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

Table 6-2 FP11-F Instruction (Cont)

Mnemonic	Instruction Description	Octal Code

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 6-2 FP11-F 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-F'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 $\lvert(AC)*(FSRC)\rvert \geqslant$ LOLIM or FIU = 1; otherwise AC ← 0
	FC ← 0
	FV ← 1 if $\lvert AC \rvert \geqslant$ UPLIM; otherwise FV ← 0
	FZ ← 1 if (AC) = 0; otherwise FZ ← 0
	FN ← 1 if (AC) < 0; otherwise FN ← 0
		171400+AC*100+FSRC
		F1 Format

Table 6-2 FP11-F Instruction (Cont)

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

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 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 6-2 FP11-F 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| ≥ 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.	

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

Table 6-2 FP11-F Instruction (Cont)

Mnemonic	Instruction Description	Octal Code
STFPS DST	Store FP11-F's Program Status Word DST ← (FPS)	170200+DST F4 Format
STST DST	Store FP11-F'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 $\lvert (AC) - (FSRC) \rvert$ ⩾ 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

6.3.1 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-F 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-F, and there is no transfer of the result.

6.3.2 Floating-Modulo Instruction

The Floating-Modulo (MOD) instruction causes the FP11-F 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-F.

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-F.

6.3.3 Load Instruction

The load instruction causes the FP11-F 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.

6.3.4 Store Instruction

The store instruction causes the FP11-F 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.

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

The Load Convert Double-to-Floating (LDCDF) instruction causes the FP11-F to assume that the source specifies a double-precision floating-point number. The FP11-F 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 6-3). If the MSB of the double-precision segment is 0, the single-precision word remains unchanged after rounding.

The Load Convert Floating-to-Double (LDCFD) instruction causes the FP11-F to assume that the source specifies a single-precision number. The FP11-F 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.

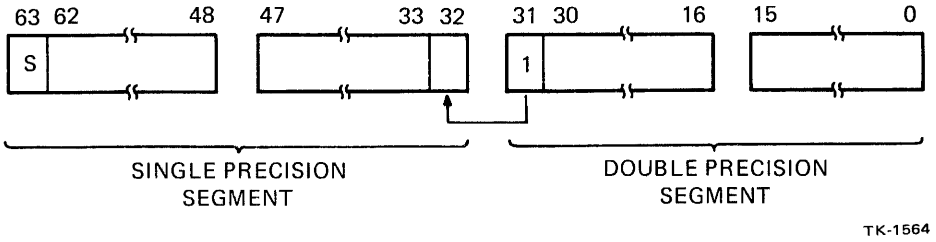


Figure 6-3 Double-to-Single Precision Rounding

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

The Store Convert Double-to-Floating (STCDF) instruction causes the FP11-F to convert a double-precision number located in the source accumulator to a single-precision number. The FP11-F 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 6-3); otherwise, the single-precision segment remains unchanged.

The Store Convert Floating-to-Double (STCFD) instruction causes the FP11-F to convert a single-precision number located in the source accumulator to a double-precision number. The FP11-F 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 6-4).

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.

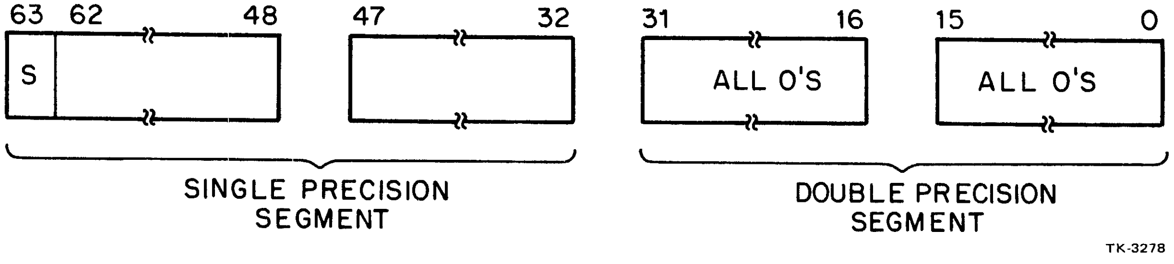


Figure 6-4 Single-to-Double Precision Appending

6.3.7 Clear Instruction

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

6.3.8 Test Instruction

The test instruction causes the FP11-F to test the sign and exponent of a floating-point number and update the FP11-F 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.

6.3.9 Absolute Instruction

The absolute instruction causes the FP11-F 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 nonzero, 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: if it is 0, the entire data word in memory is zeroed. If the exponent is nonzero, the integer exponent is restored to memory.

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

6.3.10 Negate Instruction

The negate instruction causes the CPU (or the FP11-F, 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; if it is 0, zeros are written into the accumulator. If the exponent is nonzero, 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 nonzero, the original fraction and exponent are restored to memory.

6.3.11 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 in the following text.

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-F 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 nine high-order bits are all 0s. Similarly, all legal negative exponents from memory have their nine high-order bits equal to 1. Therefore, to detect a legal exponent, only the nine high-order bits need be examined for all 1s or all 0s.

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

$$\begin{array}{lr|l} \text{Exponent of 34} & 00000000 & 00011100 \\ \quad 200 & + \qquad & 10000000 \\ \hline & & \underbrace{10}_{2}\underbrace{011}_{3}\underbrace{100}_{4} \end{array}$$

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

Example 2: LDEXP 201

$$\begin{array}{lr|l} \text{Exponent of 201} & 00000000 & \overbrace{10}^{2}\overbrace{000}^{0}\overbrace{001}^{1} \\ \quad 200 & + \qquad 0 & 10000000 \\ \hline & \nearrow \boxed{1} & 00000001 \\ & \text{Overflow} \qquad & \end{array}$$

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

Example 3: LDEXP 100200

```
                                      |  2      0   0
Exponent of 100200           10000000 | 10000000
 200                     +            | 10000000
                         -------------+----------
                            ->  [ 1 ] | 00000000
                     Underflow
```

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

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

```
Exponent of 177600           11111111 | 10000000
                         +          0 | 10000000
                         -------------+----------
                             00000000 | 00000000
```

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

6.3.12 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 6-5).

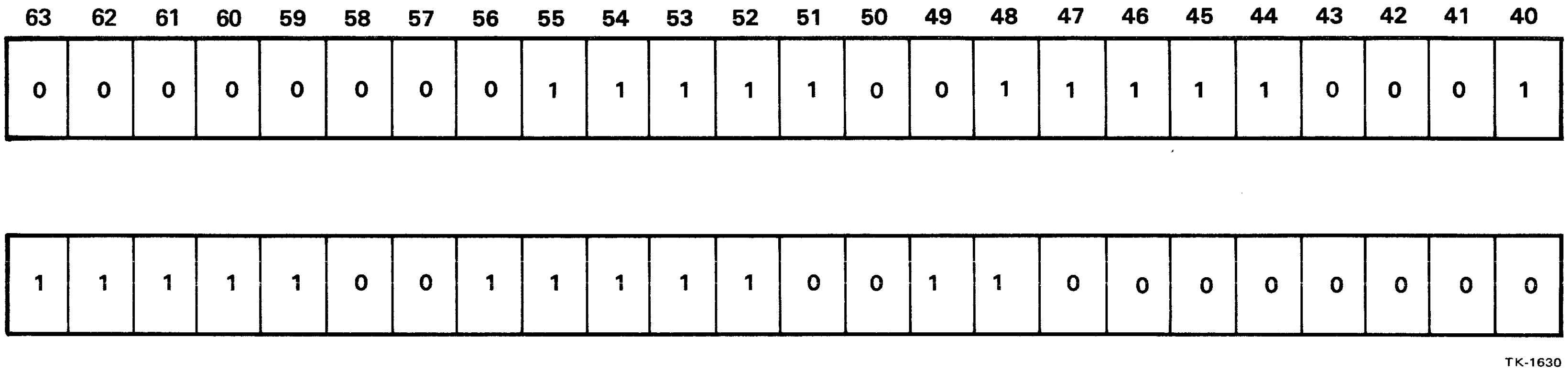
The integer is then assigned an exponent of 217_8 short integer. This is the result of adding 200_8 (since the exponent is expressed in excess 200 notation) to 17_8, which represents 15_{10} shifts. This number of shifts is the maximum number required to normalize a number. If long-integer mode is specified, 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 6-6).

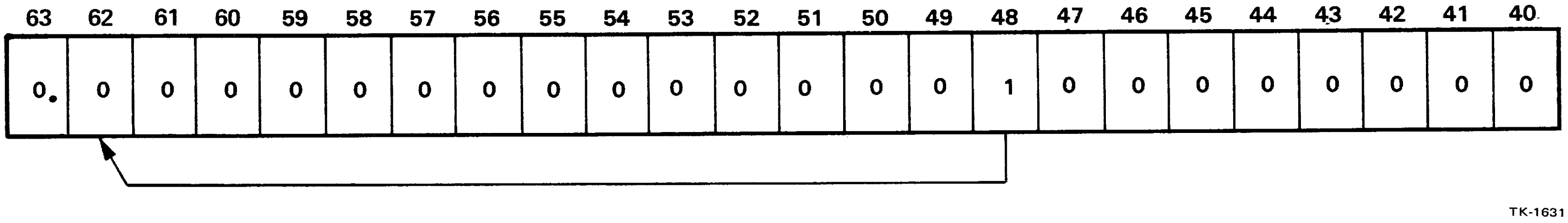
EXP = 217_8	Shift integer 15 places to the left to normalize.
-17_8	Bit 59 = 0, bit 58 = 1
200_8	Decrease exponent by 15_{10}, which is 17_8.

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	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

Figure 6-5 Integer Left-Shift Example



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 6-6 Normalized Integer Example

6.3.13 Store Exponent Instruction

The Store Exponent (STEXP) instruction causes the CPU to access a floating-point number in the FP11-F, 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 7) extended through the eight 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 6-7 and 6-8).

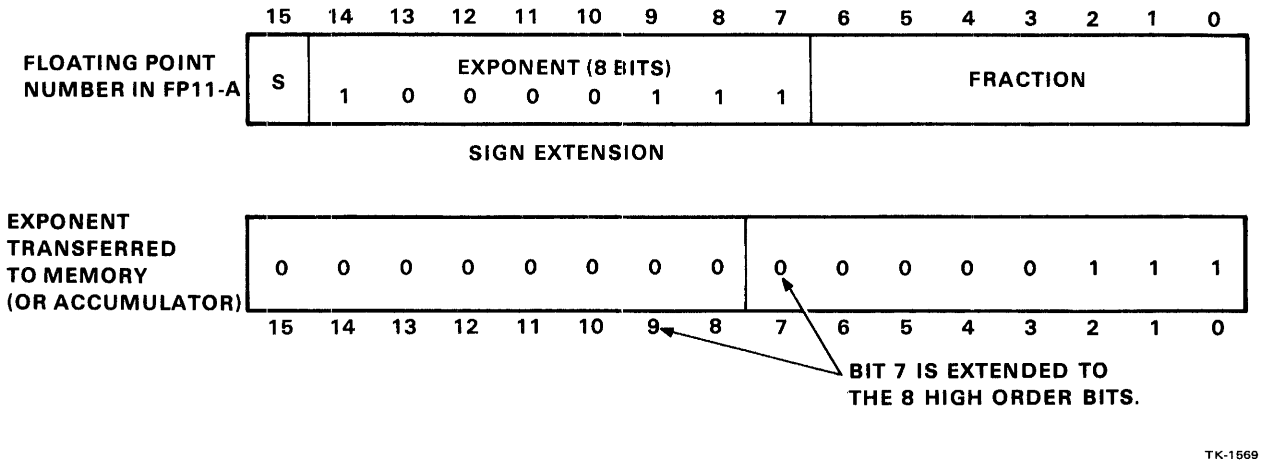


Figure 6-7 Store Exponent Example No. 1

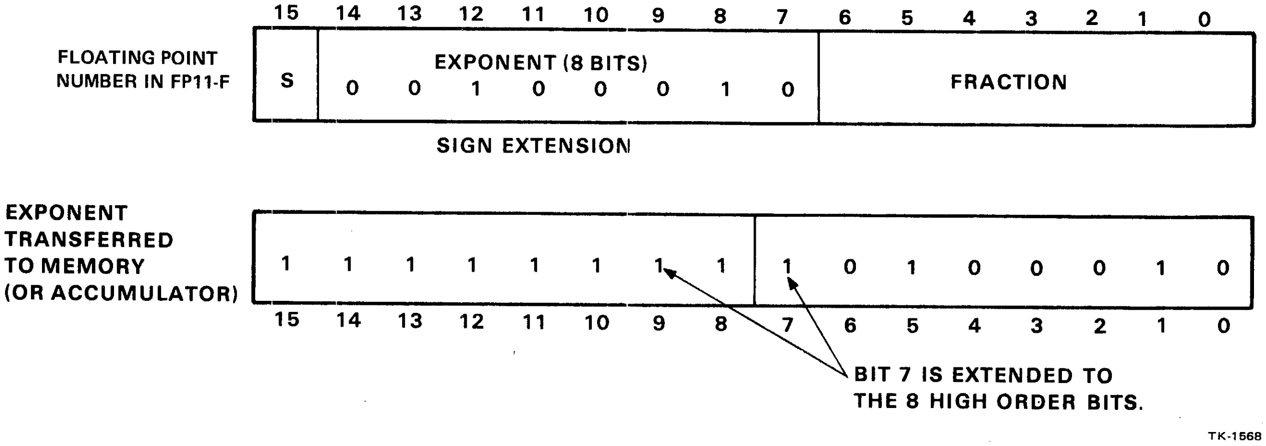


Figure 6-8 Store Exponent Example No. 2

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

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

Sign Bit

Example 2: Exponent = 42

$$\begin{array}{lr} \text{Exponent of 42} & 00100010 \\ \text{2's Complement of 200} & +10000000 \\ \hline \text{Result} = -42 & \rightarrow 1\underbrace{010}_{4}00\underbrace{010}_{2} \end{array}$$

Sign Bit

6.3.14 Store Convert Floating-to-Integer Instruction

The Store Convert Floating-to-Integer (STCFI) 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 nonzero integer (Figure 6-9).

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.

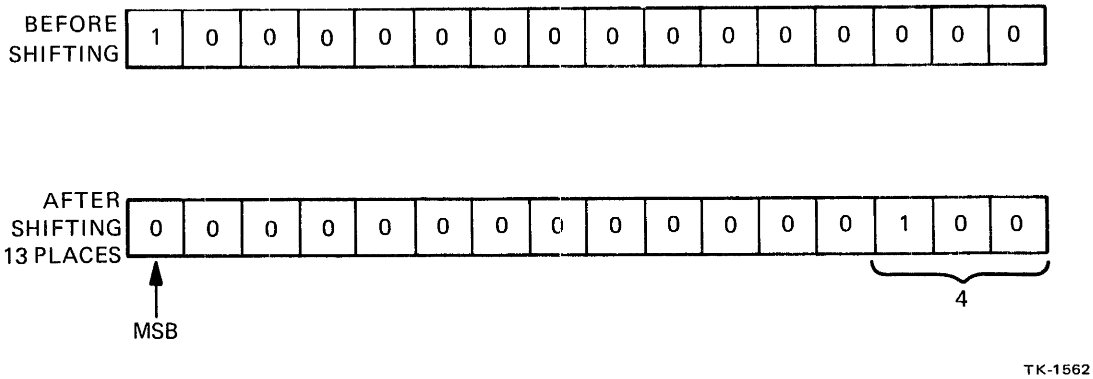


Figure 6-9 Store Convert Integer Example

	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 conversion 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 the preceding table). If, in this case, the number is not the most negative integer, it will be detected by a third conversion error test (following) 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 × 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 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?

$$\begin{array}{ll} \text{Exponent:} & 240_8 \\ & \underline{-201} \\ \text{No} & 37_8 \end{array}$$

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.)

$$\begin{array}{ll} \text{Result of Test 1:} & 37 \\ & \underline{-40} \\ & {-1} \end{array}$$

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 MSB 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.

6.3.15 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-F in order to enable and disable interrupts, set and clear mode bits, and set condition codes.

6.3.16 Store FP11's Program Status

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

6.3.17 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 four 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.

6.3.18 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).

6.3.19 Set Floating Mode

The Set Floating Mode (SETF) instruction causes the CPU to clear the FD bit (bit 7 of the FPS register) and indicate single-precision operation.

6.3.20 Set Double Mode

The Set Double Mode (SETD) instruction causes the FPP to set the FD bit (bit 7 of the FPS register) and indicate double-precision operation.

6.3.21 Set Integer Mode

The Set Integer Mode (SETI) instruction causes the FPP to clear the IL bit (bit 6 of the FPS) and indicate that short-integer mode (16 bits) is specified.

6.3.22 Set Long-Integer Mode

The Set Long-Integer Mode (SETL) instruction causes the FPP to set the IL bit (bit 6 of the FPS) and indicate that long-integer mode (32 bits) is specified.

6.4 FP11-F PROGRAMMING EXAMPLES

This paragraph contains two programming examples using the FP11-F 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,ACO      ;LOAD ACO FROM A
ADDF     B,ACO      ;ACO HAS (A + B)
LDF      C,AC1      ;LOAD AC1 FROM C
SUBF     D,AC1      ;AC1 HAS (C - D)
MULF     AC1,ACO    ;ACO HAS (A + D)*(C - D)
DIVF     X,ACO      ;ACO HAS (A + D)*(C - D)/X
STF      ACO,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 7
INSTALLATION AND CHECKOUT

7.1 INSTALLATION

Determine that +5 Vdc current is adequate prior to installation. If current is adequate, install FP11-F in backplane slot 3.

NOTE

Current calculation must include use of BA11-A box as a host to the PDP-11/44 backplane.

7.2 CHECKOUT

Checkout consists of turning on power and running the diagnostics in the following order.

PDP-11/44 CPU Test	
PDP-11/44 Traps Test (At least Rev. C)	
PDP-11/44 EIS Test	DFKAC
PDP-11/44 FPP Diagnostic, Part 1	CKFPAA
PDP-11/44 FPP Diagnostic, Part 2	CKFPBA
PDP-11/44 FPP Diagnostic, Part 3	CKFPCA

CHAPTER 8
MAINTENANCE

8.1 INTRODUCTION

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

8.2 FP11-F DIAGNOSTICS

Three diagnostics are available to validate and diagnose the FP11-F. However, since the KD11-Z 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-Z failure may cause only floating-point instructions to fail. The three FP11-F 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 microstep and where the problem is located may result.

8.2.1 MAINDEC CKFPAA

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)

8.2.2 MAINDEC CKFPBA

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

8.2.3 MAINDEC CKFPCA

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
I and D Space Tests
Auto Increment/Decrement Check - SR1

8.3 ASCII PROGRAMMER'S CONSOLE

Normal console and maintenance features provided by the programmer's console to debug and diagnose the KD11-Z processor are directly extendable in use to the FP11-F 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, floating-point data and machine dependent registers.

The console displays MPC 0–10 L if the proper command is selected at the programmer's console. Thus, single microstepping the machine through floating-point microcode is possible.

A change in the KD11-Z processor from the KD11-E processor enables the AMUX lines onto the Unibus data lines.

NOTE

Refer to the 11/44 Serial Console Specification for full use of the console.

8.4 FP11-F FLOW DIAGRAMS

Each microstep in the FP11-F 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 8-1.

For a detailed description in microflow symbology, refer to Sheet 2 of FP11-F flows (FD-FP11-F-2).

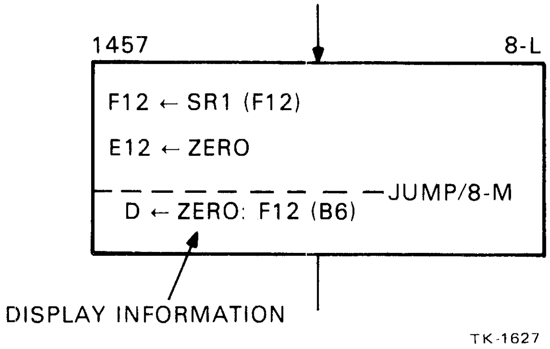


Figure 8-1 Display Information

FP11-F FLOATING-POINT PROCESSOR
TECHNICAL MANUAL
EK-FP11F-TM-002

Reader's Comments

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

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

____

____

____

What features are most useful? ________________________________

____

____

____

What faults 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-FP11F-TM-002

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

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

digital

No Postage
Necessary
if Mailed in the
United States

Digital Equipment Corporation
Educational Services Development and Publishing
1925 Andover Street
Tewksbury, Massachusetts 01876