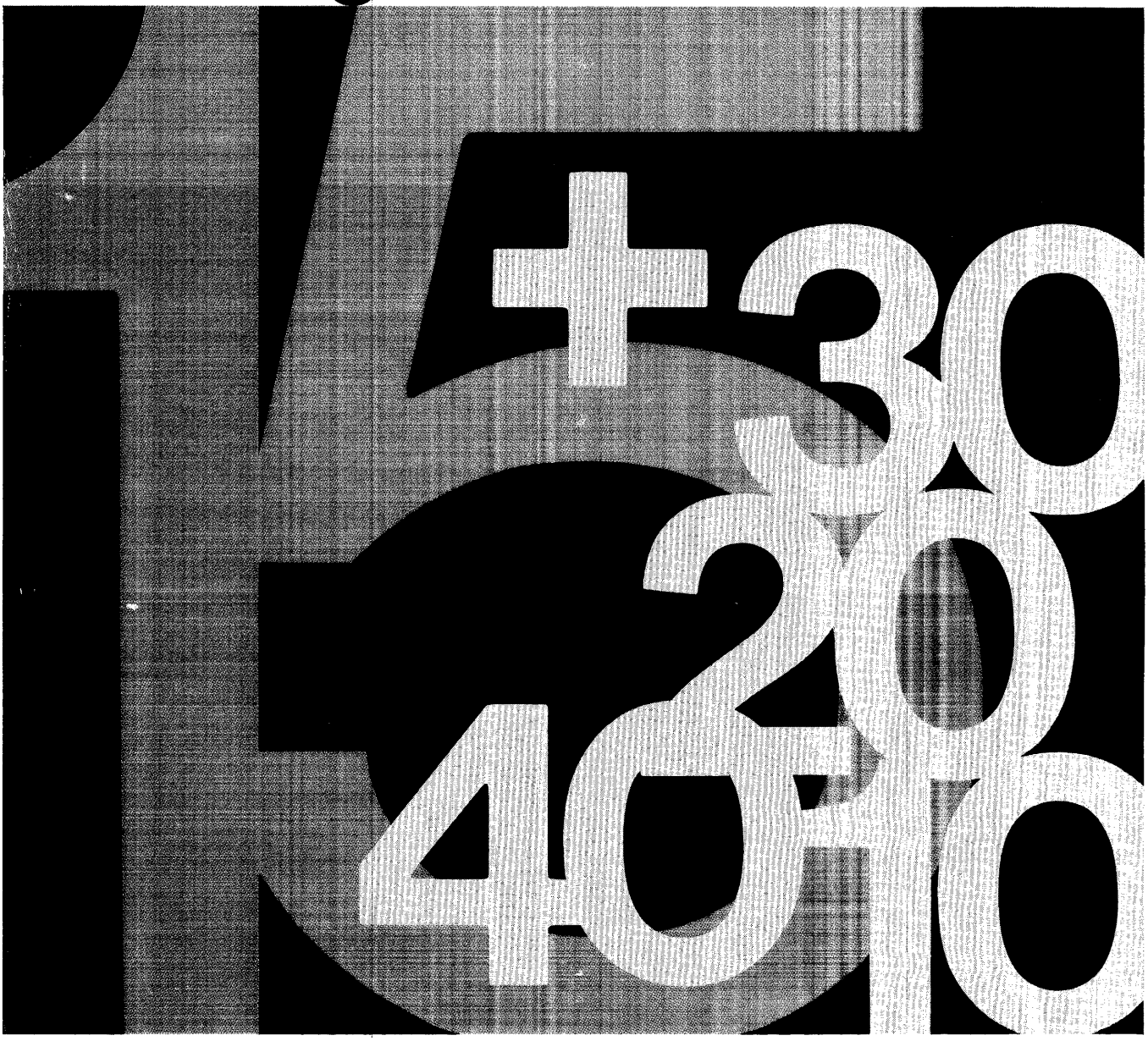
Digital Equipment Corporation
Maynard, Massachusetts

digital

PDP-15 Systems
Programmer's Reference Manual

FP15
Floating Point Processor



DEC-15-HQEA-D

94-003/084/36

PDP-15 SYSTEMS
FP15 FLOATING POINT PROCESSOR PROGRAMMER'S REFERENCE MANUAL

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Edition April 1971

CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The FP15 Floating-Point Processor (FPU) is a hardware option used with the PDP-15/20, /30, and /40 Central Processors; the FP15 enables the PDP-15 to perform arithmetic and logic operations using floating-point arithmetic. The prime advantage is increased speed without the necessity of writing complex floating-point software routines. The FP15 has single-precision and extended-integer capability, as well as single- and double-precision floating point. Prior to describing the FP15 Floating-Point Processor, several fundamentals of floating-point arithmetic are reviewed in this chapter.

1.2 FLOATING-POINT ARITHMETIC

Floating-point representation of a binary number consists of two parts, an exponent and a mantissa. The mantissa is a fraction with the binary point positioned between the sign bit and the most significant bit. If the mantissa is normalized, all leading 0s are eliminated from the binary representation; the most significant bit is thus a logical 1. Leading 0s are removed by shifting the mantissa left; however, each left shift of the mantissa must be followed by a decrement of the exponent value to maintain the true value of the number. The exponent value represents the power of 2, by which the mantissa is multiplied to obtain the value to be used. Figure 1-1 shows an unnormalized number in floating-point notation, and then the same number after it has been normalized. Note in the example that the mantissa is shifted eight places to the left, and the exponent has been decreased by eight to maintain the equivalent value.

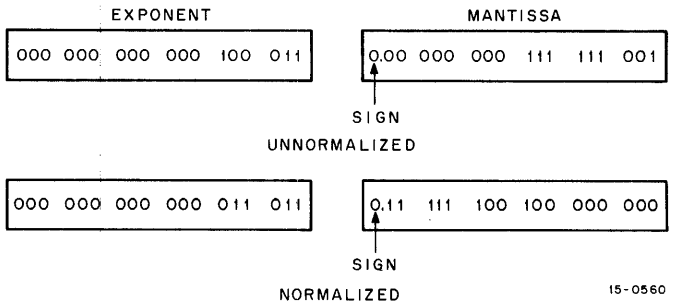


Figure 1-1 Floating-Point Representation

1.2.1 Floating-Point Addition and Subtraction

For floating-point addition and subtraction operations, the exponents must be aligned or equal; if they are not aligned, the mantissa with the smaller exponent is shifted right until they are. Each shift to the right is accompanied by an increment of the exponent value. When the exponents are aligned or equal, the mantissa can be added or subtracted, whichever the case may be. The exponent value indicates the number of places the binary point is to be moved to obtain the actual representation of the number.

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

Example:

$$0.10\overset{5}{\frown}100\overset{0}{\frown}000\quad 000\quad 000\quad 000\quad \times 2^{6} = 50_{8} = 40_{10}$$
$$0.11\overset{7}{\frown}100\quad 000\quad 000\quad 000\quad 000\quad \times 2^{3} = 7_{8} = 7_{10}$$

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

$$\begin{array}{llllllll} & 5 & 0 & & & & & \\ 0.10 & 100 & \overset{7}{} & 000 & 000 & 000 & 000 & \times 2^{6} = 50_{8} = 40_{10} \\ 0.00 & 011 & & 100 & 000 & 000 & 000 & \times 2^{6} = 7_{8} = 7_{10} \\ \hline 0.10 & 111 & & 100 & 000 & 000 & 000 & \times 2^{6} = 57_{8} = 47_{10} \end{array}$$

b. Move binary point six places to the right.

$$0.10\overset{5}{\frown}111\overset{7}{\frown}1.00\quad 000\quad 000\quad 000$$

1.2.2 Floating-Point Multiplication and Division

For floating-point multiplication, the mantissas are multiplied and the exponents are added. For floating-point division, one mantissa is divided by the other and the exponents are subtracted. There is no requirement to align the binary point in multiplication or division.

The following example shows the number 7_{10} multiplied by the number 5_{10}. A 9-bit register is assumed for simplicity.

			0.11	100	000	$\times 2^3 = 7_8 = 7_{10}$
			0.10	100	000	$\times 2^3 = 5_8 = 5_{10}$
	1	110	000	000	000	
			0			
	111	000	00			
.1	000	11.0	000	000	000	$= 43_8 = 35_{10}$

Move binary point six places to the right $= 35_{10} = 43_8$.

CHAPTER 2
FP15 FUNCTIONAL DESCRIPTION

2.1 INTRODUCTION

This chapter describes the simplified block diagram of the FP15, and its associated addresses and word formats.

2.2 FP15 SIMPLIFIED BLOCK DIAGRAM DISCUSSION

Figure 2-1 shows a simplified block diagram of the FP15 Floating-Point Processor. The FP15 is in parallel with the CPU on the memory bus, and monitors each instruction fetched by the CPU from core. If bits 00 through 05 of the instruction are equal to 71_8, it is recognized as a floating-point instruction; the CPU treats the instruction as an NOP. The FP15 takes control of memory, inhibits the CPU, and then simulates the CPU by completing the normal interface between CPU and memory. After the floating-point instruction has been executed, the CPU is enabled and both the CPU and FP15 are free to monitor the next instruction.

Functionally, the FP15 contains a memory buffer register and two operand registers. The memory buffer register provides temporary storage for all words transferred to the FP15. One operand register consists of an 18-bit exponent register (EPA), a 35-bit mantissa register (FMA), and a 1-bit sign register (A SIGN). This operand register is referred to as the floating-point accumulator. An additional 35-bit register designated the FMQ serves as an extension to the floating-point accumulator.

A second operand register consists of an 18-bit exponent register (EPB), a 35-bit mantissa register (FMB), and a 1-bit sign register (B SIGN). This second operand register, EPB/B SIGN/FMB, serves as a temporary accumulator to hold the argument fetched from core.

The exponent registers store the exponents associated with floating-point numbers and are not used during integer operations. Basically, if two numbers (integer or floating-point) are to be manipulated, one number is loaded in the floating-point accumulator by a Load type instruction. The second number is usually loaded in the temporary accumulator [EPB (B SIGN) FMB] by an instruction specifying an arithmetic operation. Both numbers are gated into a 36-bit adder, where the arithmetic operation is

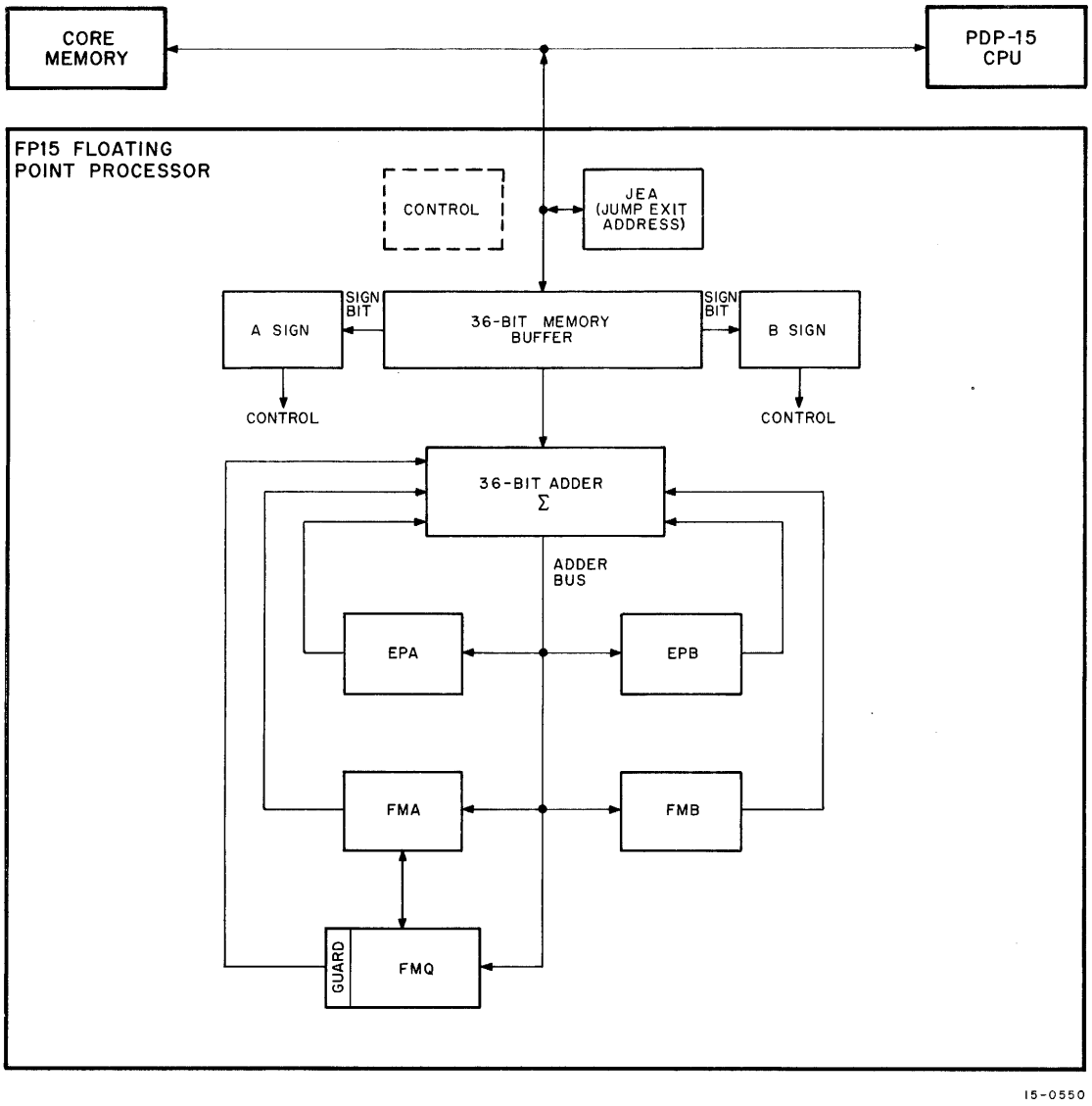


Figure 2-1 FP15 Simplified Diagram

performed. The result is then transferred to the floating-point accumulator. The major registers are described below:

Memory Buffer Register - A 36-bit register which provides the FP15/memory interface. All data transferred into the FP pass through this register.

Adder - A 36-bit arithmetic logic unit (ALU) which serves as the central point in the FP15 and performs all arithmetic and logic operations. The output of the adder is connected to all major registers via an adder bus.

A SIGN - A 1-bit register used to store the polarity of the associated operand (A mantissa).

EPA - An 18-bit register used to store the 2's complement of the exponent associated with the mantissa loaded in the FMA. The most significant bit of the EPA represents the sign of the exponent; in single-precision floating arithmetic, the most significant bit of the exponent is bit 09. It is, therefore, necessary to extend the value of this bit from bits 00 through 08. If bit 09 is a 1, bits 00 through 08 in the EPA are forced to 1s, and if bit 09 is a 0, bits 00 through 08 in the EPA are forced to 0s. The EPA and FMA serve as the floating-point accumulator.

FMA - A 35-bit register used to store the integer in integer arithmetic, or the mantissa in floating-point arithmetic. The binary point is located between bit 00 and bit 01 of the FMA.

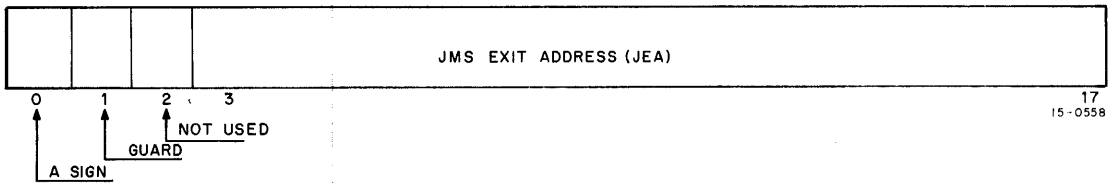
FMQ - A 35-bit extension of the FMA register used during multiplication and division operations.

B SIGN - A 1-bit register used to store the polarity of the associated operand (B mantissa).

EPB - An 18-bit register used to store the exponent associated with the mantissa in the FMB. The most significant bit of the EPB represents the sign of the exponent. In single-precision arithmetic, where the most significant bit in the EPB is bit 09, the value of this bit is extended to bits 00 through 08 (refer to EPA register). The EPB and FMB serve as a temporary accumulator to store the argument fetched from core. The EPB is a dynamic register, and is therefore not directly accessible by software.

FMB - A 35-bit register used to store the integer in integer arithmetic or the mantissa argument in floating-point arithmetic. The binary point is located between the most significant bit (bit 00) and bit 01 of the FMB. The FMB is a dynamic register and is therefore not directly accessible by software.

JEA (JMS Exit Address) - A 17-bit register used to store two status bits and a 15-bit base exit address for floating-point interrupts. When an interrupt condition (overflow, underflow, abnormal division, or memory protect violation) occurs in the FP15, the base exit address (a unique address for each type of interrupt) is returned. This indicates a service routine associated with the interrupt. The guard bit is used in rounding operations; for a more detailed description, refer to Paragraph 3.3 (Interrupt Handling).



2.3 INSTRUCTION AND ADDRESS FORMATS

Floating-point instructions consist of two 18-bit words: an instruction word with a 71 code (see Figure 2-2), followed by an address word (see Figure 2-3). The instruction word specifies type of operation, type of precision, and data format. The address word specifies direct or indirect addressing and contains the address of the memory operand, if direct, or the address of a word containing the address of the memory operand, if indirect. Each instruction received from memory is monitored by both the FP15 and

CPU. An instruction with an octal code of 71 in bits 00 through 05 is recognized as a floating-point instruction.

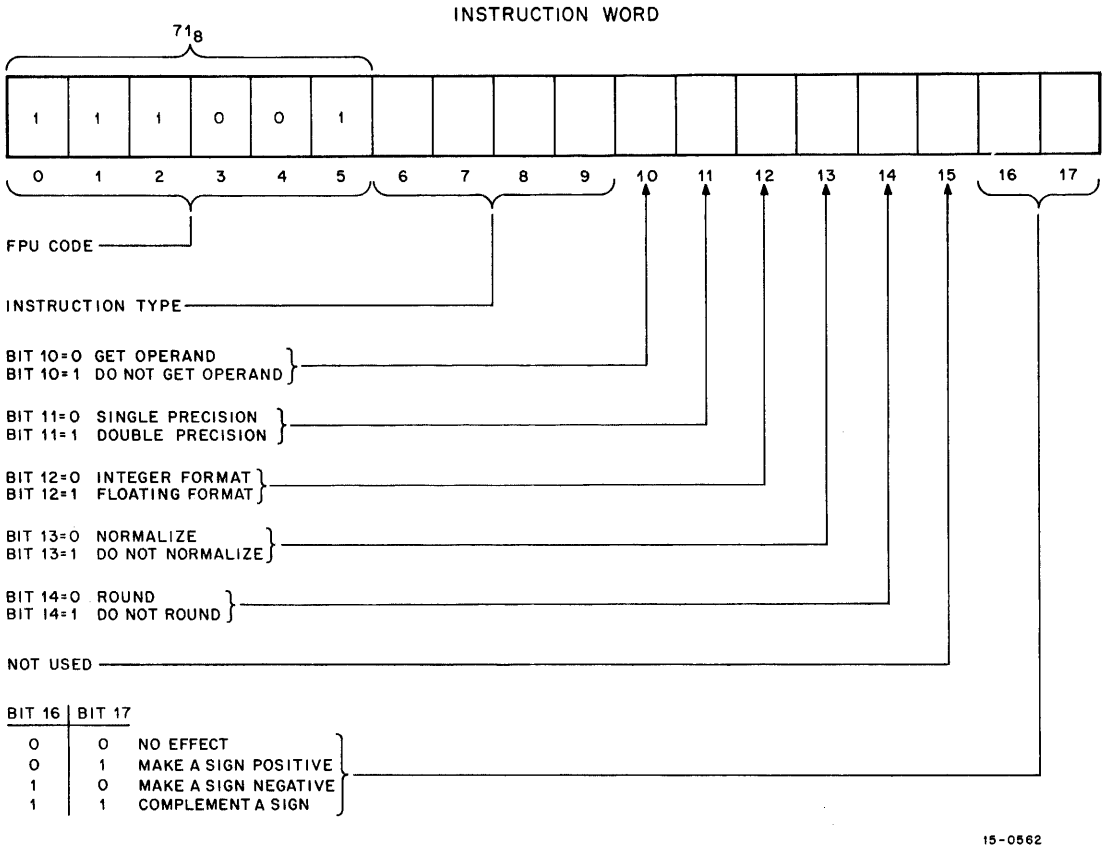


Figure 2-2 Floating-Point Instruction Format

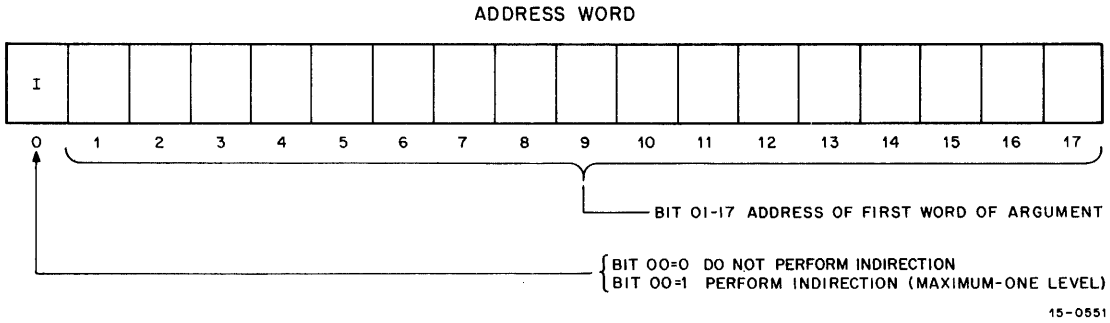


Figure 2-3 Floating-Point Address Format

2.4 DATA FORMATS

The single- and double-precision floating point and single-precision integer data formats are identical to those in the existing PDP-15 floating-point software. Extended (double-precision) integer format is not presently supported by the PDP-15 software. The above formats are shown in Figures 2-4, 2-5, 2-6, and 2-7.

2.5 DATA TRANSFER TO FP15 FROM MEMORY - INTEGER FORMAT

For single-precision integer words, the 18-bit 2's complement operand is loaded from memory into bits 18 through 35 of the FMA. The value of bit 18 (sign bit) is loaded into the remaining bit positions (bits 17 through 00) to extend the sign bit.

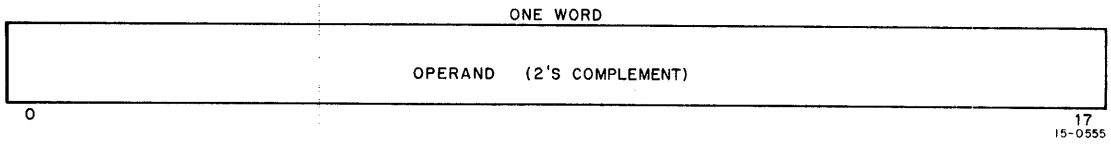


Figure 2-4 Single-Precision Integer Format

For extended integer words, the high-order operand from memory is loaded into bits 00 through 17 of the FMA, and the low-order operand is loaded into bits 18 through 35.

All integers loaded into the floating-point processor are converted to 36-bit sign and magnitude numbers.

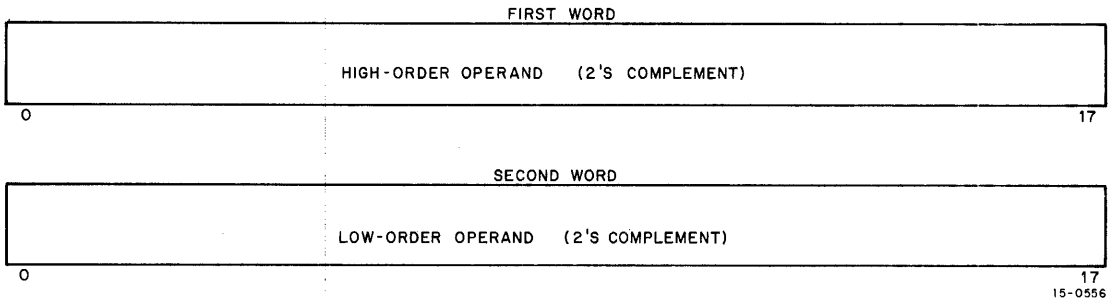


Figure 2-5 Extended Integer Format

2.6 DATA TRANSFER TO FP15 FROM MEMORY - FLOATING-POINT FORMAT

For single-precision floating-point words, the first word from memory consists of nine bits of low-order mantissa and nine bits of exponent. The nine bits of mantissa are loaded into bits 18 through 26 of the FMA, and bits 27 through 35 are zeroed. The nine bits of exponent in 2's complement form are loaded

into bits 09 through 17 of the EPA, with bit 09 representing the sign bit. The unloaded portion of the EPA register (bits 00 through 08) is loaded with the value of bit 09. If this bit is a 1, 1s are placed in bit positions 00 through 08, and if the bit is a 0, 0s are placed in bit positions 00 through 08. This extends the sign bit to bit positions 00 (the bit normally reserved for sign of the exponent value). The second word from memory is loaded into bits 00 through 17 of the FMA and represents the 18 bits of high-order mantissa. Figure 2-7 shows the loading of single-precision floating-point words from memory. The first word is 044022, and the second is 212346.

Note that the EPA and bits 18 through 26 of the FMA are loaded by the first word, and bits 00 through 17 of the FMA are loaded with the second word.

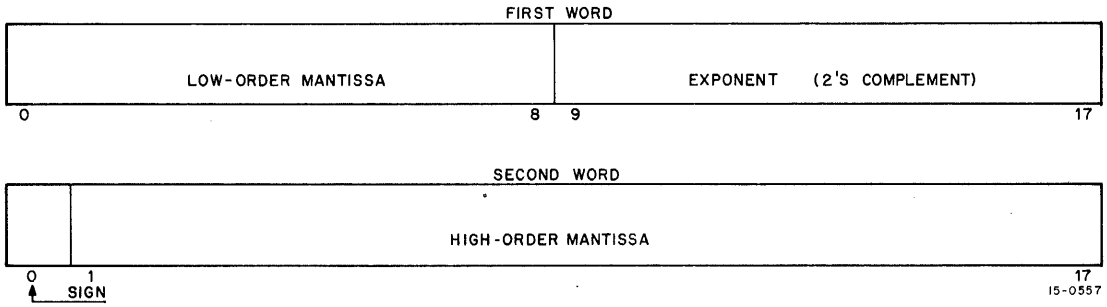


Figure 2-6 Single-Precision Floating-Point Format

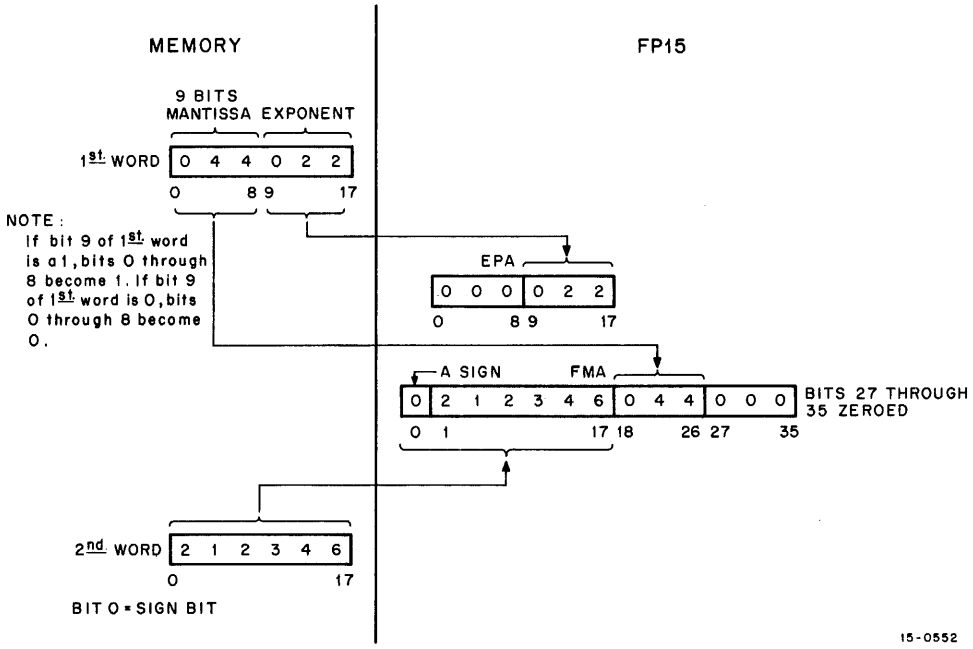


Figure 2-7 Loading of Single-Precision Floating Point

For double-precision floating-point words, the 18-bit 2's complement exponent is first loaded into the EPA, the 18-bit high-order mantissa is loaded into A SIGN and bits 01 through 17 of the FMA, and the low-order mantissa is loaded into bits 18 through 35 of the FMA. All 36 bits of the FMA are loaded at one time.

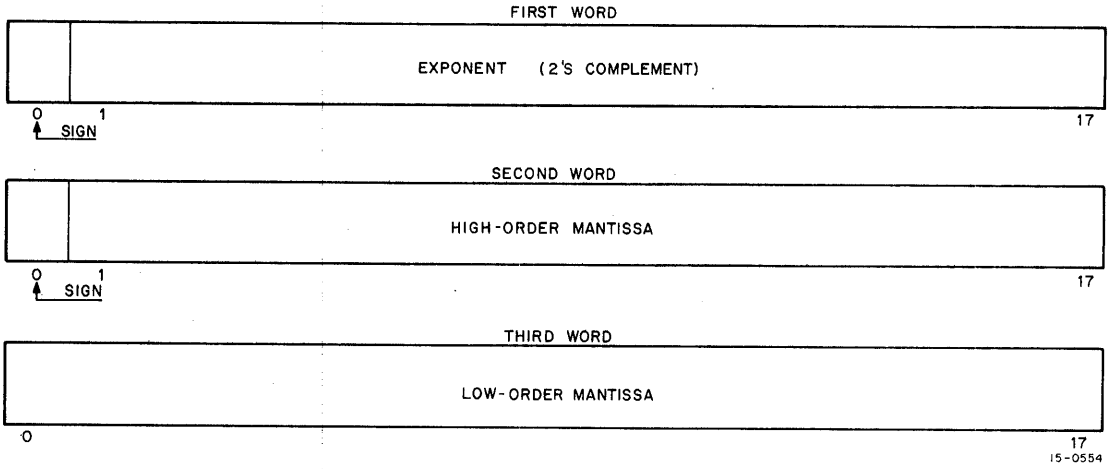


Figure 2-8 Double-Precision Floating-Point Format

CHAPTER 3
FP15 ARITHMETIC

3.1 INTRODUCTION

Negative integers are stored in memory as 2's complement numbers. Such operands are converted to sign and magnitude format when transferred to the FMA or FMB in the FP15. Load and reverse arithmetic instructions transfer operands to the FMA, while arithmetic instructions transfer operands to the FMB. Positive integers and floating-point numbers stored in memory require no conversion, as they are already in sign and magnitude format.

As an example of how negative integers are handled, consider the integer designated as negative 2. This number is stored in memory as 777776_8. When transferred to the FMA, for example, the number is converted to 000002_8 with a negative sign (1), as shown in Figure 3-1.

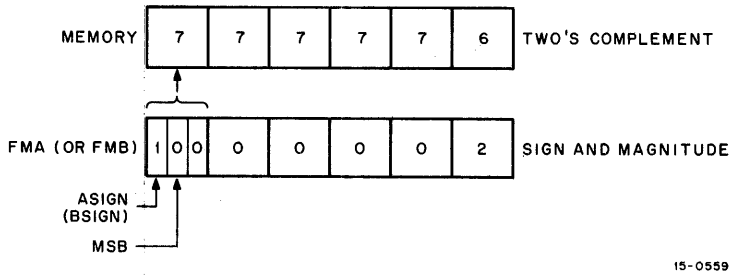


Figure 3-1 Handling of Negative Integers

Negative integers in sign and magnitude format in the FP15 are converted to two's complement format prior to being stored in memory by a STORE instruction.

3.2 GUARD BIT AND ROUNDING

The FP15 has an internal guard bit that is used under certain conditions to determine whether the FMA is to be rounded. The guard bit is set independent of any request for rounding. When set, and rounding is requested, it adds +1 to the least significant bit of the FMA. The guard bit is cleared at the beginning of all instructions except Floating-Point Test, Load JEA, Store JEA, and Branch.

During alignment of the mantissas in floating-point addition and subtraction, bits shifted out of the FMA or FMB are shifted into the FMQ. If rounding is requested, and FMQ 01 is a 1, the mantissa that is being aligned is rounded. Further, if the addition or subtraction produced a carry out of the most significant stage of the adder, the adder is right-shifted and the exponent is incremented. This returns the true number to the FMA (see Figure 3-2). The least significant bit shifted out of the FMA is not shifted into the FMQ, but is shifted into a guard bit. If rounding is requested, and the guard bit is set, +1 is added to the least significant bit of the FMA.

For floating-point multiplication and division operations, the guard bit is set if FMQ 01 is on a 1. If rounding is requested, and the guard bit is set, +1 is added to the least significant bit of the FMA.

For a Fix instruction, the bits in the FMA and FMQ are right-shifted. If, upon completion of the shifting process, FMQ 01 is on a 1, the guard bit is set. If rounding is requested, and the guard bit is set, +1 is added to the least significant bit of the FMA.

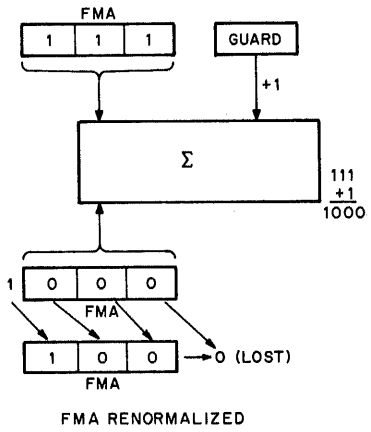


Figure 3-2 Handling of Guard Bit During Round Request

In single precision floating-point arithmetic, after numbers are loaded into the FP15 they are handled as double-precision numbers - 18-bits of exponent and 35-bits of mantissa. Due to this, +1 is added to bit 35 of the floating-point accumulator during arithmetic operations when rounding is performed. When rounding takes place in the single-precision floating STORE instruction, however, +1 is added to bit 26 of the FMA if bit 27 is a one. Bits 27-35 are then cleared.

3.3 INTERRUPT HANDLING

The FP15 can cause an interrupt under the following conditions:

Overflow - Occurs when the final magnitude of an arithmetic operation exceeds the maximum number that can be represented by the FP15. Overflow can occur with both integer and floating-point numbers.

Underflow - Occurs when the final magnitude of an arithmetic operation is less than the minimum number which can be represented by the FP15. Underflow applies to floating-point numbers only.

Abnormal Divide - Occurs when division by an unnormalized operand is attempted on either integer or floating-point numbers (0 represents a special case of the unnormalized operand).

Memory Protect Trap - Occurs when the system is in user mode and a memory protect violation or non-existent memory reference has been made by the FPU.

Prior to starting FP15 floating-point operation, the 15-bit JEA register is loaded with an address representing a core location to which the FP15 can exit when a particular error condition (overflow, underflow, abnormal divide, or memory protect trap) is detected. When one of these conditions is detected, the FP15 forces the CPU to execute a JMS to a location specified by the JEA plus a fixed constant, N. This location is the entry point to a specific routine associated with the error condition. If the interrupt exception is overflow, the CPU will execute a JMS to the JEA address; if the exception is underflow, the CPU will execute a JMS to the JEA address +2; if abnormal divide, the CPU will execute a JMS to the JEA +4; and if memory protect trap, the CPU will execute a JMS to the JEA address +6. The JEA is a 15-bit register which holds the exit address as follows:

```
EXIT ADDRESS    0
     +1         JMP OVR    /GO TO OVERFLOW
     +2         0
     +3         JMP UND    /GO TO UNDERFLOW
     +4         0
     +5         JMP DIV    /GO TO DIVIDE
     +6         0
     +7         JMP TRAP   /GO TO MEMORY VIOLATION
```

NOTE

To determine the data mode on an interrupt exception, it is necessary to examine the instruction that was being executed. The address which was stored, due to the JMS instruction, is equal to the location of the original instruction +3.

3.3.1 Memory Protect Trap

When a memory protect violation occurs during a floating-point instruction, the FP15 forces the CPU to execute a JMS to the location specified by JEA +6 (as previously described), no trap will occur, user mode will remain on, and no modification of core above or below the boundary will occur.

An example of this is shown below, where, upon occurrence of a memory protect violation, a JMS to location JEA +6 occurs and the PC points to A+3.

Example:

```
                1000 DAC    LOC (JEA+6)     1004
A               1001 DAD          +7        JMP MP Service
A+1             1002 400
A+2             1003
A+3             1004
```

An exception to the above occurs if the JEA points to an address above or below the protect boundaries and a floating-point memory violation occurs. In this case, the CPU will trap and service the attempted boundary violation, and the PC will point to A+3.

CHAPTER 4
INSTRUCTION SET

4.1 INTRODUCTION

Table 4-1 is a summary of all FP15 Floating-Point instructions by categories. Following this table is a description of the FP15 instruction set. The mnemonic, instruction type, execution time, and octal code are provided for each instruction, followed by a general description of its operation. The instructions which can cause interrupt exceptions (underflow, overflow, abnormal division, or memory trap) are specified. Section 4.3 discusses worst-case timing.

The XCT of any FP15 instruction is permissible, and the address associated with the FP15 instruction is contained in the location following the XCT. The EXEC switch will not execute a FP15 instruction; an NOP will occur. SING TIME, SING STEP, or SING INST switches will not stop the execution of a FP15 instruction.

The instruction modifiers, formats, and operations of the FP15 instruction set are designated by the following characters:

MODIFIERS

- UR - unrounded
- UN - unnormalized
- UU - unrounded and unnormalized

FORMATS

- I - single precision integer
- E - extended (double-precision) integer
- F - single-precision floating point
- D - double-precision floating point

OPERATIONS

- AD - Add
- SB - Subtract
- RS - Reverse Subtract
- MP - Multiply
- DV - Divide
- RD - Reverse Divide
- ST - Store
- LF - Load and Float
- LD - Load

OPERATIONS (Cont)

FL - Float
LX - Load and Fix
FX - Fix
LQ - Load FMQ
SWQ - Swap

Generally, the FP15 instructions are in the following format:

XX	X	XX
MODIFIER	FORMAT	OPERATION

For example, if an unrounded, unnormalized, double-precision floating point Add instruction is specified, the mnemonic is specified as UUDAD; where the UU is the modifier, D is the format, and AD is the operation. Modify FMA instructions, branch instructions, and diagnostic instructions do not follow this general pattern.

All the FP15 instructions (except Floating-Point Test, Branch, Load or Store JEA, and diagnostic instructions) can be microprogrammed with bits 16 and 17 of the instruction word as described below:

Bit 16	Bit 17		
0	0	No effect	
0	1	Make A SIGN positive	Not used in FP test, Load or Store JEA, Branch on condition, and diagnostic instructions.
1	0	Make A SIGN negative	
1	1	Complement A SIGN	

For example, the instruction 710540 specifies double-precision floating-point subtraction. If desired to make A SIGN negative, the instruction would be specified as 710542.

Table 4-1
FP15 Instruction Summary

Mnemonic	Instruction Type	Octal Code
FPT	Floating-Point Test	710314
ISB	Single Integer Subtract	710400
ESB	Extended Integer Subtract	710500
FSB	Single-Precision Float Subtract	710440
URFSB	Unrounded, Single-Precision Float Subtract	710450
UNFSB	Unnormalized, Single-Precision Float Subtract	710460
UUFSB	Unrounded, Unnormalized, Single-Precision Float Subtract	710470

Table 4-1 (Cont)
FP15 Instruction Summary

Mnemonic	Instruction Type	Octal Code
DSB	Double-Precision Float Subtract	710540
URDSB	Unrounded, Double-Precision, Float Subtract	710550
UNDSB	Unnormalized, Double-Precision Float Subtract	710560
UUDSB	Unrounded, Unnormalized, Double-Precision Float Subtract	710570
IRS	Single Integer Reverse Subtract	711000
ERS	Extended Integer Reverse Subtract	711100
FRS	Single-Precision Float Reverse Subtract	711040
URFRS	Unrounded, Single-Precision Float Reverse Subtract	711050
UNFRS	Unnormalized, Single-Precision Float Reverse Subtract	711060
UUFRS	Unrounded, Unnormalized, Single-Precision Float Reverse Subtract	711070
DRS	Double-Precision Float Reverse Subtract	711140
URDRS	Unrounded, Double-Precision Float Reverse Subtract	711150
UNDRS	Unnormalized, Double-Precision Float Reverse Subtract	711160
UUDRS	Unrounded, Unnormalized, Double-Precision Float Reverse Subtract	711170
IMP	Single Integer Multiply	711400
EMP	Extended Integer Multiply	711500
FMP	Single-Precision Float Multiply	711440
URFMP	Unrounded, Single-Precision Float Multiply	711450
UNFMP	Unnormalized, Single-Precision Float Multiply	711460
UUFMP	Unrounded, Unnormalized, Single-Precision Float Multiply	711470
DMP	Double-Precision Float Multiply	711540
URDMP	Unrounded, Double-Precision Float Multiply	711550
UNDMP	Unnormalized, Double-Precision Float Multiply	711560
UUDMP	Unrounded, Unnormalized, Double-Precision Float Multiply	711570
IDV	Single-Precision Integer Divide	712000
EDV	Extended Integer Divide	712100
FDV	Single-Precision Float Divide	712040
URFDV	Unrounded, Single-Precision Float Divide	712050
DDV	Double-Precision Float Divide	712140
URDDV	Unrounded, Double-Precision Float Divide	712150
IRD	Single-Precision Integer Reverse Divide	712400

Table 4-1 (Cont)
FP15 Instruction Summary

Mnemonic	Instruction Type	Octal Code
ERD	Extended Integer Reverse Divide	712500
FRD	Single-Precision Float Reverse Divide	712440
URFRD	Unrounded, Single-Precision Float Reverse Divide	712450
DRD	Double-Precision Float Reverse Divide	712540
URDRD	Unrounded, Double-Precision Float Reverse Divide	712550
ILD	Single-Precision Integer Load	713000
ELD	Extended Integer Load	713100
FLD	Single-Precision Float Load	713050
UNFLD	Unnormalized, Single-Precision Float Load	713070
DLD	Double-Precision Float Load	713150
UNDLD	Unnormalized, Double-Precision Float Load	713170
IST	Single-Precision Integer Store	713600
EST	Extended Integer Store	713700
FST	Single-Precision Float Store	713640
URFST	Unrounded, Single-Precision Float Store	713650
UNFST	Unnormalized, Single-Precision Float Store	713660
UUFST	Unrounded, Unnormalized, Single-Precision Float Store	713670
DST	Double-Precision Float Store	713750
UNDST	Unnormalized, Double-Precision Float Store	713770
ILF	Single-Precision Integer Load and Float	714010
UNILF	Unnormalized, Single-Precision Integer Load and Float	714030
ELF	Extended Integer Load and Float	714110
UNELF	Unnormalized, Extended Integer Load and Float	714130
FLA	Float FMA	714210
UNFLA	Unnormalized Float FMA	714230
FLX	Single-Precision Float Load and Fix	714460
URFLX	Unrounded, Single-Precision Float Load and Fix	714470
DLX	Double-Precision Float Load and Fix	714560
URDLX	Unrounded, Double-Precision Float Load and Fix	714570
FXA	Fix EPA, FMA	714660
URFXA	Unrounded, Fix EPA, FMA	714670
ILQ	Single-Precision Integer Load FMQ	715000

Table 4-1 (Cont)
FP15 Instruction Summary

Mnemonic	Instruction Type	Octal Code
ELQ	Extended Integer Load FMQ	715100
FLQ	Single-Precision Float Load FMQ	715050
UNFLQ	Unnormalized, Single-Precision Float FMQ	715070
DLQ	Double-Precision Float Load FMQ	715150
UNDLQ	Unnormalized, Double-Precision Float Load FMQ	715170
SWQ	Swap FMA and FMQ	715250
UNSWQ	Unnormalized, Swap FMA and FMQ	715270
LJE	Load JEA Register	715400
SJE	Store JEA Register	715600
IAD	Single-Precision Integer Add	716000
EAD	Extended Integer Add	716100
FAD	Single-Precision Float Add	716040
URFAD	Unrounded, Single-Precision Float Add	716050
UNFAD	Unnormalized, Single-Precision Float Add	716060
UUFAD	Unrounded, Unnormalized, Single-Precision Float Add	716070
DAD	Double-Precision Float Add	716140
URDAD	Unrounded, Double-Precision Float Add	716150
UNDAD	Unnormalized, Double-Precision Float Add	716160
UUDAD	Unrounded, Unnormalized, Double-Precision Float Add	716170
BZA	Branch on 0 FMA	716601
BMA	Branch on Minus FMA	716602
BLE	Branch if FMA ≤ 0	716603
BPA	Branch on positive FMA	716604
BRU	Branch Unconditional	716606
BNA	Branch on non-zero FMA	716610
BAC	Branch if GUARD bit is Set	716620
FZR	Zero EPA (A SIGN) FMA	711200
FAB	Make A SIGN positive (Absolute Value)	713271
FNG	Make A SIGN negative	713272
FCM	Complement A SIGN	713273
FNM	Normalize EPA (A SIGN) FMA	713250
DMF	Diagnostic Mode Off	717200

Table 4-1 (Cont)
FP15 Instruction Summary

Mnemonic	Instruction Type	Octal Code
DMN	Diagnostic Mode On	717300
DRR	Diagnostic Read Registers	710000
DSR	Diagnostic Step and Read Registers	710100+n
DBK	Debreak	703304

4.2 FPU INSTRUCTION SET

4.2.1 Integer Subtract

Mnemonic	Instruction Type	Time (μs)	Octal Code
ISB	Single Integer Subtract	6.2	710400
ESB	Extended Integer Subtract	7.3	710500

The argument is transferred from memory to the (B SIGN) FMB. The content of (B SIGN) FMB is subtracted from the content of (A SIGN) FMA, and the difference is placed in (A SIGN) FMA. If the difference is 0, EPA and A SIGN are zeroed. The FMQ is zeroed at the beginning of the instruction.

Interrupt Exception: Overflow - An overflow interrupt will occur if the subtraction generates a magnitude greater than $2^{35}-1$. The result left in the FMA is modulo 2^{35}. The A SIGN is the sign of the result, as if no overflow occurred.

Example (DBL Precision):	A SIGN (1) FMA =	3 7 7 7 7 7 7 7 7 7 7 7	
	B SIGN (0) FMB =	0 0 0 0 0 0 0 0 0 0 0 7	
RESULT	A SIGN (1)	4 0 0 0 0 0 0 0 0 0 0 6	
RESULT LEFT IN FMA		0 0 0 0 0 0 0 0 0 0 0 6	followed by overflow interrupt

4.2.2 Floating-Point Subtract

Mnemonic	Instruction Type	Time (μs)	Octal Code
FSB	Sng. Float Subtract	8.4	710440
URFSB	Unround, Sng. Float Subtract	8.4	710450
UNFSB	Unnorm., Sng. Float Subtract	8.3	710460
UUFSB	Unround, Unnorm., Sng. Float Subtract	8.3	710470
DSB	Dbl. Float Subtract	11.2	710540
URDSB	Unround, Dbl. Float Subtract	11.2	710550
UNDSB	Unnorm., Dbl. Float Subtract	11.2	710560
UUDSB	Unround, Unnorm., Dbl. Float Subtract	11.2	710570

The argument is transferred to EPB (B SIGN) FMB [exponent to EPB and mantissa to (B SIGN) FMB]. The mantissas in the FMA and FMB are aligned by finding the difference between the EPA and EPB, and right-shifting the mantissa with the smaller exponent until the number of shifts equals the exponent difference. Bits shifted out of the mantissa with the smaller exponent are shifted into the FMQ, which is cleared at the beginning of the instruction. The bits shifted into the FMQ are retained there. When the mantissas are aligned, the FMB mantissa (fraction) is subtracted from the FMA mantissa and the difference placed in (A SIGN) FMA. If a carry occurs out of the most significant bit of the FMA, the difference is shifted right one place and the exponent incremented by 1. The least significant bit (LSB) of the FMA is not shifted into the FMQ but into a guard bit to be saved for rounding (see Paragraph 3.2).

Rounding - Rounding can occur at two times: once after the align, and then after the subtract. After the align, if rounding is requested and FMQ 01 is a 1, +1 is added to the least significant bit of the FMA (bit 35). After the subtract, if rounding is requested and the guard bit is set, +1 is added to the least significant bit of the FMA (bit 35).

Normalizing - If the most significant bit of the FMA is not a 1 after the subtract and normalize is requested, the FMA is shifted left until the most significant bit (MSB) contains a 1 (up to a maximum of 35 shifts). The FMA is a 35-bit register and, if a number contained therein is shifted more than 35 times and is still not normalized, that number was equal to 0 and cannot be normalized (0s are shifted into the least significant positions of the FMA). For each left shift, the exponent is decremented by 1.

Interrupt Exception: Underflow - If the exponent of the result is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA, and an underflow interrupt exception occurs. The contents of the (A SIGN) FMA are correct. The correct exponent is -2^{18} +EPA.

Interrupt Exception: Overflow - If the exponent of the result is greater than 377777_8 ($2^{17}-1$), it cannot be represented correctly in the EPA and an overflow interrupt exception occurs. The contents of A SIGN (FMA) are correct; the correct exponent is 2^{18} +EPA.

4.2.3 Integer Reverse Subtract

Mnemonic	Instruction Type	Time (μs)	Octal Code
IRS	Sng. Integer Reverse Subtract	6.2	711000
ERS	Ext. Integer Reverse Subtract	7.3	711100

The argument is transferred from memory to the FMB. The contents of the FMA are subtracted from the argument in the FMB and the difference is placed in the FMA. If the difference is 0, EPA and A SIGN are zeroed. The contents of the FMQ are zeroed at the beginning of the instruction.

Interrupt Exception: Overflow - An overflow interrupt will occur if the subtraction generates a magnitude greater than $2^{35}-1$. The result left in the FMA is modulo 2^{35}. The A SIGN is the sign of the result, as if no overflow occurred.

4.2.4 Floating Point Reverse Subtract

Mnemonic	Instruction Type	Time (μs)	Octal Code
FRS	Sng. Float Reverse Subtract	8.6	711040
URFRS	Unround, Sng. Float. Reverse Subtract	8.6	711050
UNFRS	Unnorm., Sng. Float Reverse Subtract	8.5	711060
UUFRS	Unround, Unnorm., Sng. Float Reverse Subtract	8.5	711070
DRS	Dbl. Float Reverse Subtract	11.6	711140
URDRS	Unround, Dbl. Float Reverse Subtract	11.6	711150
UNDRS	Unnorm., Dbl. Float Reverse Subtract	11.2	711160
UUDRS	Unround, Unnorm., Dbl. Float Reverse Subtract	11.2	711170

The argument is transferred to EPB (B SIGN) FMB [exponent to EPB and mantissa to (B SIGN) FMB]. The mantissas in the FMA and FMB are aligned by finding the difference between the EPA and EPB and right-shifting the mantissa with the smaller exponent until the number of shifts equal the exponent difference. Bits shifted out of the mantissa with the smaller exponent are shifted into the FMQ, which is cleared at the beginning of the instruction. The bits shifted into the FMQ are retained. When the

mantissas are aligned, the FMA mantissa (fraction) is subtracted from the FMB mantissa and the difference placed in (A SIGN) FMA. If a carry occurs out of the most significant bit of the FMA, the difference is shifted right one place and the exponent incremented by 1. The LSB of the FMA is not shifted into the FMQ, but into the guard bit to be saved for rounding (see Paragraph 3.2).

Rounding - Rounding can occur at two times: once after the align, and then after the subtract. After the align, if rounding is requested and FMQ 01 is a 1, +1 is added to the least significant bit of the FMA (bit 35). After the subtract, if rounding is requested and the guard bit is set, +1 is added to the least significant bit of the FMA (bit 35).

Normalizing - If the most significant bit of the FMA is not a 1 after the subtract and normalize is requested, the FMA is shifted left (up to a maximum of 35 shifts) until the MSB contains a 1. Zeros are shifted into the least significant positions of the FMA. For each left shift, the exponent is decremented by 1.

Interrupt Exception: Underflow - If the exponent of the result is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA and an underflow interrupt exception occurs. The contents of the A SIGN (FMA) are correct; the correct exponent is -2^{18} +EPA.

Interrupt Exception: Overflow - If the exponent of the result is greater than 377777_8 ($2^{17}-1$), it cannot be correctly represented in the EPA, and an overflow interrupt exception occurs. The contents of A SIGN (FMA) are correct; the correct exponent is 2^{18} +EPA.

4.2.5 Integer Multiply

Mnemonic	Instruction Type	Time (μs)	Octal Code
IMP	Sng. Integer Multiply	14.1	711400
EMP	Ext. Integer Multiply	17.0	711500

The multiplicand argument is transferred to the (B SIGN) FMB. The multiplier is contained in the (A SIGN) FMA. The product is retained in the (A SIGN) FMA and FMQ with the low-order bits in the FMA (the former contents of the FMQ are lost). The FMQ can be accessed through the Load FMQ instruction.

Interrupt Exception: Overflow - An overflow interrupt exception occurs if the magnitude of the product is greater than $2^{35}-1$; that is, if any of the high-order 35-bits of the product are in 1s. The A SIGN is the sign of the result, as if no overflow occurred.

4.2.6 Floating Point Multiply

Mnemonic	Instruction Type	Time (μs)	Octal Code
FMP	Sng. Float Multiply	16.6	711440
URFMP	Unround, Sng. Float Multiply	16.6	711450
UNFMP	Unnorm., Sng. Float Multiply	16.2	711460
UUFMP	Unround, Unnorm., Sng. Float Multiply	16.2	711470
DMP	Dbl. Float Multiply	18.6	711540
URDMP	Unround, Dbl. Float Multiply	18.6	711550
UNDMP	Unnorm., Dbl. Float Multiply	18.2	711560
UUDMP	Unround, Unnorm., Dbl. Float Multiply	18.2	711570

The multiplicand is transferred to the EPB (B SIGN) FMB, and the multiplier is contained in the EPA (A SIGN) FMA. The product is retained in the EPA (A SIGN) FMA and FMQ; the former contents of the FMQ are lost. The FMA retains the high-order bits, and the FMQ retains the low-order bits. For multiplication, the EPA and EPB are added together, with the sum retained in the EPA.

Rounding - If rounding is requested and the most significant bit of the FMQ is a 1, the guard bit is set (see Paragraph 3.2) and +1 is added to the least significant bit of the FMA. If this addition produces a carry out of the most significant bit of the FMA, the FMA is right-shifted by 1 and the EPA is incremented by 1.

Normalizing - If the most significant bit of the FMA is not a 1 and normalize is requested, the FMA and FMQ are shifted left as one 70-bit register until the most significant bit of the FMA is a 1, not to exceed a maximum of 35 shifts. For each left shift, the EPA is decremented.

Interrupt Exception: Underflow - If the exponent of the result is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA and an underflow interrupt exception occurs. The contents of the A SIGN (FMA) are correct; the correct exponent is -2^{18} + EPA.

Interrupt Exception: Overflow - If the exponent of the result is greater than 377777_8 ($2^{17}-1$), it cannot be correctly represented in the EPA, and an overflow interrupt exception occurs. The contents of A SIGN (FMA) are correct; the correct exponent is 2^{18} + EPA.

4.2.7 Integer Divide

Mnemonic	Instruction Type	Time (μs)	Octal Code
IDV	Sng. Integer Divide	11.8	712000
EDV	Ext. Integer Divide	14.4	712100

The divisor argument is transferred to the (B SIGN) FMB, and the dividend is contained in the (A SIGN) FMA. The quotient is retained in the (A SIGN) FMA and the remainder is left in the FMQ, replacing the previous contents of the FMQ. Integer division is whole number division; if the dividend is less than the divisor, indicating a fractional number, the quotient is 0.

Interrupt Exception: Abnormal Divide - If the divisor is 0, an abnormal interrupt exception occurs because division by 0 is not possible. Execution of the Divide instruction is aborted immediately; the programmer cannot rely on the contents of the registers after the instruction is aborted.

4.2.8 Floating Point Divide

Mnemonic	Instruction Type	Time (μs)	Octal Code
FDV	Sng. Float Divide	15.6	712040
URFDV	Unround, Sng. Float Divide	15.6	712050
DDV	Dbl. Float Divide	18.3	712140
URDDV	Unround, Dbl. Float Divide	18.3	712150

The divisor argument is transferred to the EPB (B SIGN) FMB and is divided into the dividend in the EPA (A SIGN) FMA. The dividend is normalized prior to the actual divide. The 35-bit quotient is normalized and is retained in the FMA. The previous contents of the FMQ is lost and the remainder is retained in this register.

Normalize - For floating-point division, the dividend is normalized. The quotient is left in normalized form.

Rounding - If rounding is requested, and the most significant bit of the FMQ is a 1, the guard bit is set and +1 is added to FMA bit 35. If this addition produces a carry into the MSB of the FMA, the FMA is right-shifted one place and the EPA incremented by 1.

Interrupt Exception: Underflow - If the exponent of the result is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA and an underflow interrupt exception occurs. The contents of the A SIGN (FMA) are correct. The FMQ retains the remainder, and the correct exponent is -2^{18} + EPA.

Interrupt Exception: Abnormal Divide - If the divisor is unnormalized (or 0) an abnormal divide interrupt exception occurs. Execution of the Divide instruction is aborted immediately. The programmer cannot rely on the contents of the registers after the instruction is aborted.

Interrupt Exception: Overflow - If the exponent of the result is greater than 377777_8 ($2^{17}-1$), it cannot be correctly represented in the EPA and an overflow interrupt exception occurs. The contents of A SIGN (FMA) are correct. The FMQ retains the remainder; the correct exponent is 2^{18} + EPA.

4.2.9 Integer Reverse Divide

Mnemonic	Instruction Type	Time (μs)	Octal Code
IRD	Sng. Integer Reverse Divide	11.8	712400
ERD	Ext. Integer Reverse Divide	14.4	712500

The dividend argument is transferred to the (B SIGN) FMB and is divided by the contents of (A SIGN) FMA. The quotient is retained in the (A SIGN) FMA. The previous contents of the FMQ is lost and the remainder is left in this register. Integer division is whole number division; if the dividend is less than the divisor, indicating a fractional number, the quotient is 0.

Interrupt Exception: Abnormal Divide - If the divisor is 0, an abnormal divide interrupt exception occurs because division by 0 is not possible. Execution of the Divide instruction is aborted immediately; the programmer cannot rely on the contents of the registers after the instruction is aborted.

4.2.10 Floating Point Reverse Divide

Mnemonic	Instruction Type	Time (μs)	Octal Code
FRD	Sng. Float Reverse Divide	15.6	712440
URFRD	Unround, Sng. Float Reverse Divide	15.6	712450
DRD	Dbl. Float Reverse Divide	18.3	712540
URDRD	Unround, Dbl. Float Reverse Divide	18.3	712550

The dividend argument is transferred to the EPB (B SIGN) FMB and is divided by the divisor contained in EPA (A SIGN) FMA. The dividend is normalized prior to the actual divide. The 35-bit quotient is automatically normalized and is retained in the FMA. The previous contents of the FMQ are lost and the remainder is retained in this register. For floating-point reverse division, the dividend and divisor are normalized. If rounding is requested, and the most significant bit of the FMQ is a 1, the guard bit is set (see Paragraph 3.2) and +1 is added to FMA bit 35. If this addition produces a carry into the MSB of the FMA, the FMA is right-shifted one place and the EPA incremented by 1.

Interrupt Exception: Underflow - If the exponent of the result is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA and an underflow interrupt exception occurs. The contents of the A SIGN (FMA) are correct; the correct exponent is -2^{18} +EPA.

Interrupt Exception: Abnormal Divide - If the divisor is unnormalized (or 0), an abnormal divide interrupt exception occurs. Execution of the Divide instruction is aborted immediately. The programmer cannot rely on the contents of the registers after the instruction is aborted.

Interrupt Exception: Overflow - If the exponent of the result is greater than 377777_8 (2^{17} -1), it cannot be correctly represented in the EPA and an overflow interrupt exception occurs. The contents of A SIGN (FMA) are correct; the correct exponent is 2^{18}+EPA.

4.2.11 Integer Load

Mnemonic	Instruction Type	Time (μs)	Octal Code
ILD	Sng. Integer Load	6.6	713000
ELD	Ext. Integer Load	7.8	713100

The argument is transferred from memory to the (A SIGN) FMA. The contents of the FMQ remain unchanged.

4.2.12 Floating Point Load

Mnemonic	Instruction Type	Time (μs)	Octal Code
FLD	Sng. Float Load	8.3	713050
UNFLD	Unnorm., Sng. Float Load	7.9	713070
DLD	Dbl. Float Load	9.5	713150
UNDLD	Unnorm., Dbl. Float Load	9.3	713170

The argument is transferred from memory to the EPA (A SIGN) FMA. The contents of the FMQ remain unchanged.

Normalize - If the most significant bit of the FMA is not a 1 and normalize is requested, the FMA is shifted left (up to a maximum of 35 shifts) until the most significant bit is a 1. 0s are shifted into the least significant positions of the FMA. For each left shift, the EPA is decremented.

Interrupt Exception: Underflow - If the exponent of the result due to normalizing is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA and an underflow interrupt exception occurs. The contents of the A SIGN (FMA) are correct; the correct exponent is -2^{18} + EPA.

4.2.13 Integer Store

Mnemonic	Instruction Type	Time (μs)	Octal Code
IST	Sng. Integer Store	6.6	713600
EST	Ext. Integer Store	7.8	713700

The FMA is stored in the location specified by the argument address. For single-precision integer format, the A SIGN and bits 19 through 35 of the FMA are stored in 2's complement format at the argument address.

For extended-precision integer format, the first word consists of A SIGN and bits 01 through 17 of the FMA; the second word consists of bits 18 through 35 of the FMA. Both words are stored in 2's complement format, starting at the argument address. No interrupt exceptions occur during extended integer store; the contents of the FMQ remain unchanged.

Interrupt Exception: Overflow (Single Precision) - If the magnitude of the number in the FMA is greater than 377777_8 ($2^{17}-1$), an overflow interrupt exception occurs. The STORE instruction is aborted prior to the write into memory. The (A SIGN) FMA and contents of the FMQ remain unchanged.

4.2.14 Floating Point Store

Mnemonic	Instruction Type	Time (μs)	Octal Code
FST	Sng. Float Store	7.9	713640
URFST	Unround, Sng. Float Store	7.9	713650
UNFST	Unnorm., Sng. Float Store	7.7	713660

(Continued on next page)

Mnemonic	Instruction Type	Time (μs)	Octal Code
UUFST	Unround, Unnorm., Sng. Float Store	7.7	713670
DST	Dbl. Float Store	9.1	713750
UNDST	Unnorm., Dbl. Float Store	8.9	713770

For single-precision floating-point format, the first word is stored in 2's complement format at the argument address, and consists of bits 9 through 17 in the EPA register, and bits 18 through 26 in the FMA. The second word consists of A SIGN and bits 01 through 17 of the FMA, and is stored in the argument address plus one.

For double-precision floating-point format, the first word is stored in 2's complement format at the argument address, and consists of EPA bits 0 through 17. A SIGN and FMA bits 1 through 17 comprise the second word, which is stored in sign and magnitude format at the argument address plus one. FMA bits 18 through 35 comprise the third word, which is stored at the argument address plus two.

Normalize - If normalize is requested and the most significant bit of the FMA is not a 1, the FMA is shifted left (up to a maximum of 35 shifts) until the most significant bit is a 1 (0s are shifted into the least significant bits). For each left shift, the EPA is decremented by 1.

Rounding - If rounding is requested in single-precision store, and bit 27 is a 1, +1 is added to FMA bit 26. If bit 27 is a 0, rounding has no effect. Bits 27 through 35 are then zeroed. If a carry occurs out of the most significant bit of the FMA as a result of rounding, the FMA is shifted right one place and the EPA is incremented. Rounding is not done on double-precision floating-point store instructions since it occurs during the arithmetic operation.

Interrupt Exception: Overflow - If the EPA in a single-precision store is greater than $2^8 - 1$, an overflow will occur. The store instruction is aborted prior to the write into memory; the contents of the EPA/A SIGN/FMA are not changed.

Interrupt Exception: Underflow - If the exponent of the result due to normalization is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA, and an underflow interrupt exception occurs. The contents of the A SIGN (FMA) are correct; the correct exponent is -2^{18} + EPA.

Also, if the number in the EPA is less than -2^8 on a single-precision store, an underflow interrupt will occur. The store instruction is aborted prior to the write into memory; the contents of EPA (A SIGN) FMA are unchanged.

4.2.15 Load and Float

Mnemonic	Instruction Type	Time (μs)	Octal Code
ILF	Sng. Integer, Load and Float	11.2	714010
UNILF	Unnorm., Sng. Integer Load and Float	6.6	714030
ELF	Ext. Integer, Load and Float	11.0	714110
UNELF	Unnorm., Ext. Integer Load and Float	7.9	714130

The Load and Float instruction converts integer format to floating-point format. The integer argument is first transferred from memory to the (A SIGN) FMA. The EPA is loaded with 35_{10}, which effectively relocates the binary point from the right of the integer to a point between the sign bit and most significant bit of the FMA. The integer is consequently converted to a floating-point number; the contents of the FMQ remain unchanged.

Normalize - If the most significant bit of the FMA is not a 1 and normalize is requested, the FMA is shifted left (up to a maximum of 35 shifts) until the most significant bit is a 1 (0s are shifted into the least significant bits of the FMA). For every left-shift of the FMA, the EPA is decremented.

Interrupt Exception: None

4.2.16 Float (FMA)

Mnemonic	Instruction Type	Time (μs)	Octal Code
FLA	Float FMA	8.2	714210
UNFLA	Unnorm., Float FMA	5.3	714230

The Float FMA instruction converts integer format to floating-point format. The integer argument is already contained in (A SIGN) FMA; the second word (address) of this instruction is not used and can have any value.

The EPA is loaded with 35_{10}, which effectively relocates the binary point to the left of the number. The integer number is consequently converted to a floating-point number; the contents of the FMQ remain unchanged.

Normalize - If the most significant bit of the FMA is not a 1, and normalize is requested, the FMA is shifted left (up to a maximum of 35 shifts) until the most significant bit is a 1; 0s are shifted into the least significant bits of the FMA. For every left shift of the FMA, the EPA is decremented.

Interrupt Exception: None

4.2.17 Load and Fix

Mnemonic	Instruction Type	Time (μs)	Octal Code
FLX	Sng. Prec. Load and Fix	11.0	714460
URFLX	Unround, Sng. Prec. Load and Fix	11.0	714470
DLX	Dbl. Prec. Load and Fix	12.4	714560
URDLX	Unround, Dbl. Prec. Load and Fix	12.4	714570

The Fix instruction converts floating-point format to integer format. The argument is transferred from memory to the EPA (A SIGN) FMA in floating-point format. The FMA and FMQ are shifted right 35 minus EPA places. For example, if the EPA is 10, the FMA is shifted right 25 places. The least significant bits shifted out of the FMA are shifted into the most significant bit of the FMQ. The EPA retains its original contents; if the EPA is negative, the number in the FMA is fractional and cannot be converted to an integer. As a result, the A SIGN and FMA are zeroed. The original contents of the FMQ are lost and the FMQ retains the bits shifted in during the Fix instruction.

Rounding - If rounding is requested and the most significant bit of the FMQ is a 1, the guard bit is set (see Paragraph 3.2) and +1 is added to the least significant bit of the FMA (bit 35).

Interrupt Exception: Overflow - If the EPA is greater than 35_{10}, an overflow interrupt will occur, because the FMA does not have enough bits to represent an integer magnitude greater than 35 bits. The EPA (A SIGN) FMA remains unchanged.

4.2.18 Fix EPA (FMA)

Mnemonic	Instruction Type	Time (μs)	Octal Code
FXA	Fix EPA, FMA	8.3	714660
URFXA	Unround, Fix EPA, FMA	8.3	714670

The Fix EPA (FMA) instruction converts the EPA (FMA) from floating-point format to integer format. The second word (address) of this instruction is not used, and can have any value.

The FMA and FMQ are shifted right 35 minus EPA places. The least significant bits shifted out of the FMA are shifted into the most significant bit of the FMQ. The EPA retains its original contents. If the EPA is negative, the number in the FMA is fractional and cannot be converted to integer; as a

result, the A SIGN and FMA are zeroed. The original contents of the FMQ are lost, and the FMQ retains the bits shifted in during the Fix instruction.

Rounding - If rounding is requested and the most significant bit of the FMQ is a 1, the guard bit is set (see Paragraph 3.2), and +1 is added to the least significant bit of the FMA (bit 35).

Interrupt Exception: Overflow - If the EPA is greater than 35_{10}, an overflow interrupt will occur, because the FMA does not have enough bits to represent an integer magnitude greater than 35 bits. The EPA (A SIGN) FMA remain unchanged.

4.2.19 Load FMQ (Integer)

Mnemonic	Instruction Type	Time (μs)	Octal Code
ILQ	Sng. Integer Load FMQ	6.6	715000
ELQ	Ext. Integer Load FMQ	7.9	715100

The integer argument is transferred from memory to the (A SIGN) FMA. The contents of the FMA and FMQ are swapped; A SIGN remains unchanged.

Interrupt Exception: None

4.2.20 Load FMQ (Floating Point)

Mnemonic	Instruction Type	Time (μs)	Octal Code
FLQ	Sng. Float Load FMQ	14.0	715050
UNFLQ	Unnorm., Sng. Float Load FMQ	7.9	715070
DLQ	Dbl. Float Load FMQ	9.5	715150
UNDLQ	Unnorm., Dbl. Float Load FMQ	9.3	715170

The floating-point argument is transferred from memory to the EPA (A SIGN) FMA.

The contents of the FMA and FMQ are swapped; normalize, if specified, occurs after the swap.

Normalize - If normalize is requested and the most significant bit of the FMA is not a 1, the FMA and FMQ are shifted left (up to a maximum of 35 shifts) until the most significant bit is a 1. Zeros are shifted into the least significant positions of the FMQ; FMQ 01 is shifted into FMA 35. For every left shift of the FMA, the EPA is decremented.

Interrupt Exception: Underflow - If the exponent of the result is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA, and an underflow interrupt exception occurs. The contents of the A SIGN (FMA) are correct; the correct exponent is -2^{18} + EPA.

4.2.21 Swap FMA and FMQ

Mnemonic	Instruction Type	Time (μs)	Octal Code
SWQ	Swap FMA and FMQ	5.5	715250
UNSWQ	Unnorm., Swap FMA and FMQ	5.3	715270

No argument is transferred to the floating-point processor for a Swap instruction; the contents of the FMA are swapped with the contents of the FMQ. Normalize, if specified, occurs after the swap. The second word (address) of this instruction is not used and can have any value.

Normalize - If normalize is requested and the most significant bit of the FMA is not a 1, the FMA is shifted left (up to a maximum of 35 shifts) until the most significant bit is a 1. Zeros are shifted into the least significant positions of the FMQ; FMQ 01 is shifted into FMA 01. For every left shift of the FMA, the EPA is decremented.

Interrupt Exception: Underflow - If the exponent of the result is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA and an underflow interrupt exception occurs. The contents of the A SIGN (FMA) are correct. The correct exponent is -2^{18} + EPA.

4.2.22 Load JEA (Jump Exit Address)

Mnemonic	Instruction Type	Time (μs)	Octal Code
LJE	Load JEA Register	6.6	715400

The Load JEA instruction causes the 15-bit JEA register to be loaded from bits 3 through 17 of the argument in memory. The instruction is not protected, and any user can issue it (in user mode) without causing a memory protect trap. The guard bit is loaded from bit 1 of the argument, and the A SIGN will remain unchanged regardless of the contents of bit 0 of the argument.

Interrupt Exception: None

4.2.23 Store JEA (Jump Exit Address)

Mnemonic	Instruction Type	Time (μs)	Octal Code
SJE	Store JEA Register	6.6	715600

The contents of the 15-bit JEA register are stored as bits 03 through 17 in memory at the argument address; the contents of the FMQ remain unchanged. The A SIGN is stored as bit 00 and the guard bit as bit 01 in memory at the argument address.

Interrupt Exception: None

4.2.24 Integer Add

Mnemonic	Instruction Type	Time (μs)	Octal Code
IAD	Sng. Integer Add	6.6	716000
EAD	Ext. Integer Add	7.9	716100

The argument is transferred from memory to the (B SIGN) FMB. The contents of (B SIGN) FMB is added to the contents of (A SIGN) FMA, and the sum retained in (A SIGN) FMA. The contents of the FMQ are zeroed at the beginning of the instruction.

Interrupt Exception: Overflow - An overflow interrupt will occur if the addition generates a magnitude greater than $2^{35}-1$. The result left in the FMA is modulo 2^{35}. The A SIGN is the sign of the result as if no overflow occurred.

4.2.25 Floating Point Add

Mnemonic	Instruction Type	Time (μs)	Octal Code
FAD	Sng. Float. Add	8.2	716040
URFAD	Unround, Sng. Float. Add	8.2	716050
UNFAD	Unnorm., Sng. Float. Add	8.3	716060
UUFAD	Unround, Unnorm., Sng. Float. Add	8.3	716070
DAD	Dbl. Float. Add	9.3	716140
URDAD	Unround, Dbl. Float. Add	9.3	716150
UNDAD	Unnorm., Dbl. Float. Add	9.3	716160
UUDAD	Unround, Unnorm., Dbl. Float. Add	9.3	716170

The argument is transferred to EPB (B SIGN) FMB [exponent to EPB and mantissa to (B SIGN) FMB]. The mantissa in FMA and FMB are aligned by finding the difference between EPA and EPB, and right-shifting the mantissa with the smaller exponent until the number of shifts equals the exponent difference. Bits shifted out of the register containing the mantissa with the smaller exponent are shifted into the FMQ, which is cleared at the beginning of the instruction. These bits are retained in the FMQ. When the mantissas are aligned, the FMB mantissa is added to the FMA mantissa, and the sum placed in (A SIGN) FMA. If a carry occurs out of the most significant bit of the FMA, the difference is shifted right one place and the exponent incremented by 1. The LSB of the FMA is not shifted into the FMQ, but is shifted into a guard bit to be saved for rounding (see Paragraph 3.2).

Rounding - Rounding can occur at two times, once after the align, and again after the addition takes place. After the align, if rounding is requested and FMQ 01 is a 1, +1 is added to the least significant bit of the FMA (bit 35). After the addition, if rounding is requested and the guard bit is set, +1 is added to the least significant bit of the FMA (bit 35).

Normalize - If the most significant bit of the FMA is not a 1 after the addition, and normalize is requested, the FMA is shifted left until the MSB contains a 1 (up to a maximum of 35 shifts). Zeros are shifted into the least significant positions of the FMA. For each left shift, the exponent is decremented.

Interrupt Exception: Underflow - If the exponent of the result is less than 400000_8 (-2^{17}), it cannot be correctly represented in the EPA and an underflow interrupt exception occurs. The contents of the A SIGN (FMA) are correct; the correct exponent is -2^{18} + EPA.

Interrupt Exception: Overflow - If the exponent of the result is greater than 377777_8 ($2^{17}-1$), it cannot be correctly represented in the EPA and an overflow interrupt exception occurs. The contents of A SIGN (FMA) are correct; the correct exponent is 2^{18} + EPA.

4.2.26 Branch

Mnemonic	Instruction Type	Time (μs)	Octal Code
BZA	Branch if FMA zero	5.2	716601
BMA	Branch if FMA negative	5.2	716602
BLE	Branch if FMA ≤ 0	5.2	716603
BPA	Branch if FMA positive	5.2	716604
BRU	Branch unconditional	5.2	716606
BNA	Branch if FMA non-zero	5.2	716610
BAC	Branch if guard bit is set (see Paragraph 3.2)	5.2	716620

The Branch instruction provides conditional alteration of the sequence of program execution, and does not affect the FMQ. The instruction includes a Test Mask to test the status of the FP15. The Test Mask is contained in bits 13 through 17 of the first word of the FP15 instruction. These bits may be microprogrammed to test for more than one condition. Microprogramming produces an ORed condition of the bits that are set.

If any one of the tests is made and is successful, a program branch is made. For example, if the programmer sets bit 17 to a 1, and the FMA is 0, a branch is made. If bit 17 is not set, and the FMA is 0, no branch is made. The second word of the two-word FP instruction is the branching address (if direct) or is a pointer to the branching address (if indirect). However, the branching address for both indirect and direct addressing allows transfer within the current memory block of 32K only, because bits 01 and 02 of that branching address are ignored.

Example:

A mask of 04_8 (bit 15 = 1) tests for the FMA ≥ 0. If the test is successful, bits 03 through 17 of the branching address are placed in bits 03 through 17 of the program counter in the CPU. If the test is unsuccessful, the program continues sequentially.

Bits 16 and 17 of the Branch on Condition instruction do not modify A SIGN.

Program Interruption: If the branch address causes a memory trap, the CPU (not the FPU) is flagged. As in a memory trap on a CPU JMP instruction, the user cannot immediately determine the branching address.

4.2.27 Modify FMA

All FP15 floating-point instructions except Load or Store JEA, Branch, FT Test and diagnostic instructions can be microprogrammed to modify the FMA. The second word (address) of this class instructions is not used and can have any value. The contents of the FMQ remain unchanged.

Mnemonic	Instruction Type	Time (μs)	Octal Code
FZR	Zero EPA (A SIGN) FMA	5.2	711200
FAB	Make A SIGN positive (absolute value)	5.2	713271
FNG	Make A SIGN negative	5.2	713272
FCM	Complement A SIGN	5.2	713273
FNM	Normalize EPA (A SIGN) FMA	8.4	713250

Interrupt Exception: Underflow - The only possible interrupt exception for this class of instructions is an underflow interrupt as a result of normalize EPA (A SIGN) FMA. If the exponent of the result is less than 400000_8 (-2^{17}), an underflow interrupt occurs. The resultant exponent cannot be correctly represented in the EPA; the correct exponent is -2^{18} + EPA. The contents of (A SIGN) FMA are correct.

4.2.28 Floating Point Test

Mnemonic	Instruction Type	Time (μs)	Octal Code
FPT	Floating Point Test	5	710314

This instruction tests the presence of the FP15 Floating-Point Processor in the system. If the FP15 is installed, 710314 is an NOP for the FP15 and the PDP-15; the PDP-15 continues from PC +2. If the FP15 is not installed, a normal IORS is executed in the PDP-15 and the PDP-15 continues from PC + 1.

Interrupt Exception: None

4.3 WORST-CASE TIMING

The floating point execution times used throughout this manual are considered typical times, i.e., they are measured times using normalized numbers. They should be considered the average time to perform the instruction. The user should not encounter times greater than the worst case times listed below. These worst case times include indirection, normalized arithmetic on unnormalized numbers and memory relocate. Worst case times are: 24 μs for add and subtract; 26 μs for multiply; 27 μs for divide; 18 μs for load; and 17 μs for store.

CHAPTER 5
DIAGNOSTIC INSTRUCTIONS

5.1 INTRODUCTION

The FP15 instruction repertoire includes additional instructions used for diagnostic purposes in simulating actual floating-point instructions; these instructions are described below.

NOTE

Diagnostic instructions cannot be executed in User Mode.

5.2 DEBREAK

Mnemonic	Instruction Type	Time (μs)	Octal Code
DBK	Debreak		703304

The Debreak instruction in the PDP-15 is normally used in an active API routine to return the routine to its preassigned priority level, after the need for its temporary raising (by ISA or CAL) has been satisfied. The Debreak is used as a clear instruction in the FP15.

If an FP15 is connected to the memory bus and is in maintenance mode, when the DBK is issued, the FP15 will be forced out of this mode and all major cycle states will be zeroed. The contents of the FMQ remain unchanged.

Interrupt Exception: None

5.3 DIAGNOSTIC MODE ON, DIAGNOSTIC MODE OFF

Mnemonic	Instruction Type	Time (μs)	Octal Code
DMN	Diagnostic Mode On	5.3	717300
DMF	Diagnostic Mode Off	16.1	717200

On execution of an Diagnostic Mode On instruction (during non-user mode), the FP15 leaves the normal mode and enters a special maintenance mode. In this mode, the next floating-point instruction executed stops in Phase 3, Time State 3 of the FETCH cycle. Control is returned to the Central Processor; any non-floating point instruction may now be fetched and executed. The contents of the FMQ remain unchanged.

In the FP15, the only two modes of operation are the Normal Mode and the Diagnostic Mode. If the Central Processor is in User Mode, the FP15 is prevented from entering Diagnostic Mode because the Diagnostic Mode On instruction is handled as a no-operation. If it is desired to check out instructions in User Mode, this can be accomplished by first putting the FP15 in Diagnostic Mode during Non-user Mode, and then changing the CPU from Non-user Mode to User Mode.

The Diagnostic Mode Off instruction returns the FP15 from Diagnostic Mode to the Normal Mode (see Diagnostic).

Interrupt Exception: None

5.4 DIAGNOSTIC READ, STEP AND READ

Mnemonic	Instruction Type	Time (μs)	Octal Code
DRR	Diagnostic Read Registers	16.1	710000
DSR	Diagnostic Step and Read Registers	17.6	710100 + N where $0 < N < 77_8$

5.4.1 Diagnostic Read

After entering Diagnostic Mode, the next FP15 instruction executed stops in Phase 3, Time State 3 of the FETCH cycle. Control is returned to the Central Processor, leaving the FP15 instruction only partially completed. The next FP15 instruction is normally Diagnostic Read or Diagnostic Step and Read.

If it is desired to abort the partially completed instruction and return the FP15 to Normal Mode, a Debreak Clear (DBK) (703304) instruction should be issued.

The diagnostic Read instruction causes sixteen 18-bit words to be transferred from the FP15 to memory, starting at the argument address. The words are transferred in the following order:

1. BMB 00-17 (Buffered Memory Buffer)
2. BMB 18-35
3. SC 0-5 and IR 06-17 (Shift Counter and Instruction Register)
4. EPA 00-17
5. A SIGN and FMA 01-17
6. FMA 18-35
7. EPB 00-17
8. B SIGN and FMB 01-17
9. FMB 18-35
10. B SIGN and FMQ 01-17
11. FMQ 18-35
12. ADD 00-17 (Adder)
13. ADD 18-35
14. JEA 00-17 — 00 ASIGN, 01 GUARD, 02 Blank (Jump Exit Address)
15. STA 00-17 (See Note below)
16. AR 00-17 (Address Register)

NOTE

The STA 00-17 is a status word comprised of the following information:

STA 00	FP15 BUSY
STA 01	FETCH CYCLE
STA 02	OPAND CYCLE
STA 03	EXP CYCLE
STA 04	FUN CYCLE
STA 05	NOR CYCLE
STA 06	WRITE CYCLE
STA 07	INT1
STA 08	INT2
STA 09	TIME STATE 1
STA 10	TIME STATE 2
STA 11	TIME STATE 3
STA 12-17	DIR 12-17 (Diagnostic Instruction Register)

The Diagnostic Read instruction leaves the partially completed instruction unchanged; control is returned to the CPU after the sixteen 18-bit words have been transferred. The Diagnostic Read instruction may be executed indefinitely without affecting the partially completed instruction. Any non-floating point instruction or instructions may be fetched and executed when control is returned to the Central Processor.

5.4.2 Diagnostic Step and Read

The Diagnostic Step and Read instruction restarts the partially completed instruction and allows execution of the instruction to continue until N+1 steps are completed. At this point, execution ceases, the sixteen 18-bit words are transferred from the FP15 to memory, and control is then returned to the

Central Processor. The original instruction may or may not be completed, depending on the instruction and operand values, which will determine the number of steps to be executed. One step is counted at each of the following times:

FETCH * T3 * P3	
FETCH * T3 * P3	(if indirection)
OPAND * T3 * P3	(if operand fetch) } Depends on data format (1, 2, or 3 words)
OPAND * T3 * P3	
OPAND * T3 * P3	
EXP * T1 * P3	
EXP * T2 * P3	(FMA and FMB aligned - 1 step count for every align shift)
EXP * T3 * P3	
FUN * T1 * P3	
FUN * T2 * P3	(FMA and FMB are multiplied or divided here - 1 step count per shift. FMA also fixed here - 1 step count per every fix shift)
FUN * T3 * P3	
NOR * T1 * P3	(FMA normalized here - 1 step count per every normalize shift)
NOR * T2 * P3	
NOR * T3 * P3	
WRITE * T3 * P3	(if a store type)
WRITE * T3 * P3	(if a store type) } Depends on data format (1,2, or 3 words)
WRITE * T3 * P3	(if a store type)
INT2 * T3 * P3	(if BRANCH or INTERRUPT EXCEPTION)

The Diagnostic Step and Read instruction may be utilized to finish the partially completed instruction. The last step to be counted in the partially completed instruction is NOR * T3 * P3. Exceptions to this are the Store, Store JEA and Branch instructions.

The last step to be counted in the Store and Store JEA instruction is WRITE * T3 * P3, and the last step to be counted in the Branch instruction is INT2.

When the last step is counted (regardless of whether the diagnostic has sequenced through N+1 steps or not), the FP15 instruction stops, the sixteen 18-bit words are transferred from the FP15 to memory, and then control is returned to the CPU. The original instruction, however, is still not complete at this point; one more step is required to clear FP BUSY. The cycle and time states in the FP15 stop and the sixteen 18-bit words are transferred to memory. Control is then returned to the CPU. When FP BUSY is no longer true, the next FP15 instruction causes FP BUSY to be true and also causes the floating-point processor to stop in Phase 3, Time State 3 of the FETCH cycle. Control is again returned to the CPU. A new FP15 non-diagnostic instruction is recognized only if FP BUSY is not true. Thus, in order for Diagnostic Mode Off to be effective, FP BUSY must not be true. When Diagnostic Mode Off is recognized, the FP15 returns to Normal Mode. For both Diagnostic Read and Diagnostic Step and Read, the contents of the FMQ remain unchanged.

CHAPTER 6
FP15 PROGRAMMING EXAMPLES

6.1 INTRODUCTION

The following four examples provide an illustration of how the FP15 can be programmed to accomplish various integer and floating-point operations. An example for each data format is presented (single-precision integer, extended integer, single-precision floating-point, and double-precision floating-point). Each example also contains all arithmetic operations (add, subtract, multiply, and divide).

6.2 SINGLE-PRECISION INTEGER

This program performs the following arithmetic operation.

$$\frac{(A + B)\,C - D}{E}, \text{ where}$$

A = 000212
B = 000121
C = 000222
D = 700000
E = 000005
ILD = 713000
IAD = 716000
IMP = 711400
ISB = 710400
IDV = 712000
IST = 713600

NOTE

In the example shown, NUMD (700000) is a negative number and is loaded into the FP15 in 2's complement format, and is added to the quantity (A+B) C.

```
000200                      .LOC 200
000200    340217            TAD ARGA      /CPU INSTRUCTION
000201    040220            DAC TEMP      /CPU INSTRUCTION
000202    713000            ILD           /LOAD NUMA
000203    000221            NUMA          /ADDRESS OF NUMA
000204    716000            IAD           /ADD NUMB TO NUMA
```

```
000205    000222          NUMB    /ADDRESS OF NUMB
000206    711400          IMP     /MULTIPLY BY NUMC
000207    000223          NUMC    /ADDRESS OF NUMC
000210    710400          ISB     /SUBTRACT NUMD
000211    000224          NUMD    /ADDRESS OF NUMD
000212    712000          IDV     /DIVIDE BY NUME
000213    000225          NUME    /ADDRESS OF NUME
000214    713600          IST     /STORE RESULT IN
000215    000226          PERM    /000226
000216    740040          HLT
000217    222222  ARGA    222222  /STORAGE
000220    000000  TEMP    000000  /STORAGE
000221    000212  NUMA    000212  /NUMA
000222    000121  NUMB    000121  /NUMB
000223    000222  NUMC    000222  /NUMC
000224    700000  NUMD    700000  /NUMD
000225    000005  NUME    5       /NUME
000226    031224  PERM    031224  /STORAGE
```

6.3 DOUBLE-PRECISION INTEGER PROGRAMMING EXAMPLE

This program performs the following arithmetic operation:

$$\frac{(A+B)\,C - D}{E}, \text{ where}$$

A = 004444444444
B = 002222222222
C = 000000000011
D = 055555555554
E = 000002222222
ELD = 713100
EAD = 716100
EMP = 711500
ESB = 710500
EDV = 712100
EST = 713700

```
000600                    .LOC 600
000600    340617          TAD DECI    /CPU INSTRUCTION
000601    040620          DAC PERM    /CPU INSTRUCTION
000602    713100          ELD         /LOAD NUMA IN F.P. AC
000603    000621          NUMA        /ADDRESS OF NUMA
000604    716100          EAD         /ADD NUMB
000605    000623          NUMB        /ADDRESS OF NUMB
000606    711500          EMP         /MULTIPLY NUMC
000607    000625          NUMC        /ADDRESS OF NUMC
000610    710500          ESB         /SUBTRACT NUMD
000611    000627          NUMD        /ADDRESS OF NUMD
000612    712100          EDV         /DIVIDE BY NUME
000613    000631          NUME        /ADDRESS OF NUME
000614    713700          EST         /STORE RESULT IN
```

```
000615    000633          ANSW    /000633
000616    740040          HLT
000617    333333  DECI            /STORAGE
000620    000000  PERM    000000  /STORAGE
000621    004444  NUMA    004444  /HIGH ORDER OPERAND
000622    444444          444444  /LOW ORDER OPERAND
000623    002222  NUMB    002222  /HIGH ORDER OPERAND
000624    222222          222222  /LOW ORDER OPERAND
000625    000000  NUMC    000000  /HIGH ORDER OPERAND
000626    000011          000011  /LOW ORDER OPERAND
000627    055555  NUMD    055555  /HIGH ORDER OPERAND
000630    555554          555554  /LOW ORDER OPERAND
000631    000002  NUME    000002  /HIGH ORDER OPERAND
000632    222222          222222  /LOW ORDER OPERAND
000633    000000  ANSW    000000  /HIGH ORDER OPERAND
000634    007000          007000  /LOW ORDER OPERAND
```

6.4 SINGLE-PRECISION FLOATING POINT

This program performs the following arithmetic operations.

$$\frac{(A + B)\,C - D}{E} = \text{ where}$$

$A = 2^5 \times 000111_8$
$B = 2^5 \times 111000_8$
$C = 2^2 \times 333000_8$
$D = 2^7 \times 000222_8$
$E = 2^2 \times 222000_8$

FLD = 713050
FAD = 716040
FMP = 711440
FSB = 710440
FDV = 712040
FST = 713640

```
000101                    .LOC 101
000101    340120  TAD ARGA        /CPU INSTRUCTION
000102    040121  DAC TEMP        /CPU INSTRUCTION
000103    713050  FLD             /LOAD NUMA IN F.P. AC
000104    000122  NUMA            /ADDRESS OF NUMA
000105    716040  FAD             /ADD NUMB
000106    000124  NUMB            /ADDRESS OF NUMB
000107    711440  FMP             /MULTIPLY BY NUMC
000110    000126  NUMC            /ADDRESS OF NUMC
000111    710440  FSB             /SUBTRACT NUMD
000112    000130  NUMD            /ADDRESS OF NUMD
000113    712040  FDV             /DIVIDE BY NUME
000114    000132  NUME            /ADDRESS OF NUME
000115    713640  FST             /STORE RESULT
```

```
000116    000134          PERM    /ADDRESS OF RESULT
000117    740040          HLT     /CPU INSTRUCTION
000120    000000  ARGA    0       /STORAGE
000121    000000  TEMP    0       /STORAGE
000122    000005  } NUMA  5       /EXPONENT OF NUMA
000123    000111  }       000111  /MANTISSA OF NUMA
000124    000005  } NUMB  5       /EXPONENT OF NUMB
000125    111000  }       111000  /MANTISSA OF NUMB
000126    000002  } NUMC  2       /EXPONDENT OF NUMC
000127    333000  }       333000  /MANTISSA OF NUMC
000130    000007  } NUMD  7       /EXPONENT OF NUMD
000131    000222  }       000222  /MANTISSA OF NUMD
000132    000002  } NUME  2       /EXPONENT OF NUME
000133    222000  }       222000  /MANTISSA OF NUME
000134    000004  PERM    4       /RESULT - EXPONENT = 4
000135    332333          332333  /MANTISSA = .332333
```

Answer = $2^4 \times .332333000 = 15.51554_8$

6.5 DOUBLE-PRECISION FLOATING POINT

This program performs the following arithmetic operations.

$$\frac{(A + B)\,C - D}{E}, \text{ where}$$

A $= 2^0 \times 373737111111$

B $= 2^0 \times 000000303030$

C $= 2^2 \times 222222010101$

D $= 2^3 \times 020202101010$

E $= 2^2 \times 313131212121$

```
UUDLD  = 713170
UUDAD  = 716170
UUDMP  = 711570
URDSB  = 710550
URDDV  = 712150
UUDST  = 713770
```

```
000076                  .LOC 76
000076    020700        LAC TEMP1   /CPU INSTRUCTION
000077    040701        DAC TEMP2   /CPU INSTRUCTION
000100    713170        UUDLD       /LOAD NUMA IN F.P. AC
000101    000117        NUMA        /ADDRESS OF NUMA
000102    716170        UUDAD       /ADD NUMB
000103    000122        NUMB        /ADDRESS OF NUMB
000104    711570        UUDMP       /MULTIPLY NUMC
000105    000125        NUMC        /ADDRESS OF NUMC
```

```
000106   710550          URDSB    /SUBTRACT NUMD
000107   000130          NUMD     /ADDRESS OF NUMD
000110   712150          URDDV    /DIVIDE BY NUME
000111   000133          NUME     /ADDRESS OF NUME
000112   713770          UUDST    /STORE RESULT
000113   000136          PERM     /IN ADDRESS 136
000114   740040          HLT      /PROGRAM HLT
000115   777777  TEMP1   777777   /STORAGE
000116   000000  TEMP2   000000   /STORAGE
000117   000000  NUMA    0        /EXPONENT A
000120   373737          373737   /HIGH MANTISSA
000121   111111          111111   /LOW MANTISSA
000122   000000  NUMB    0        /EXPONENT B
000123   303030          303030   /HIGH MANTISSA
000124   101010          101010   /LOW MANTISSA
000125   000002  NUMC    2        /EXPONENT C
000126   222222          222222   /HIGH MANTISSA
000127   010101          010101   /LOW MANTISSA
000130   000003  NUMD    3        /EXPONENT D
000131   020202          020202   /HIGH MANTISSA
000132   101010          101010   /LOW MANTISSA
000133   000002  NUME    2        /EXPONENT E
000134   313131          313131   /HIGH MANTISSA
000135   212121          212121   /LOW MANTISSA
000136   000001  PERM    1        /STORE EXP. RESULT
000137   214335          214335   /STORE MANTISSA
000140   635572          635572   /HIGH AND LOW
```

Answer = $2^4 \times .214335635572 = 10.6156716675_8$

Digital Equipment Corporation
Maynard, Massachusetts

digital

printed in U.S.A.