SPERRY UNIVAC

V70 Series Floating Point Processor

Operation and Service Manual

Mini-Computer Operations

2722 Michelson Drive
P.O. Box C-19504
Irvine, California 92713

UP-8642

SPERRY UNIVAC®

V70 SERIES
FLOATING POINT PROCESSOR
OPERATION AND SERVICE MANUAL

UP-8642

98A 9906 113

FEBRUARY 1978

The statements in this publication are not intended to create any warranty, express or implied. Equipment specifications and performance characteristics stated herein may be changed at any time without notice. Address comments regarding this document to Sperry Univac, Mini-Computer Operations, Publications Department, 2722 Michelson Drive, P.O. Box C-19504, Irvine, California, 92713.

Sperry Univac is a division of Sperry Rand Corporation

Printed in U.S.A.

CHANGE RECORD

Page Number	Issue Date	Change Description
Various	11-77	Deleted all references to Varian.

Change Procedure:

When changes occur to this manual, updated pages are issued to replace the obsolete pages. On each updated page, a vertical line is drawn in the margin to flag each change and a letter is added to the page number. When the manual is revised and completely reprinted, the vertical line and page-number letter are removed.

LIST OF EFFECTIVE PAGES

Page Number	Change in Effect
All	Complete revision.

96A0731-0008

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (continued)

LIST OF TABLES

LIST OF TABLES (continued)

SECTION 1
GENERAL DESCRIPTION

The SPERRY UNIVAC V70 Series Floating Point Processor Manual describes the Floating Point Processor (FFP) and its interface with other 70 series system components.

The manual is divided into the following seven sections:

- o General Description (Introduction to the FPP, related publications, and specifications)
- o Installation
- o Operation
- o Theory of operation
- o Maintenance
- o Mnemonics list
- o Test programs

There is also a system documentation package which is assembled when the hardware is shipped and reflects the configuration of a specific system. It contains engineering documents such as logic and installation drawings.

The following list contains the part numbers of other manuals pertinent to the SPERRY UNIVAC 70 series computers (the x at the end of each document part number is the revision number and can be any digit 0 through 9):

V70 Series Architecture Reference Manual	98A 9906 00x
Processor Manual	98A 9906 02x
Option Board Manual	98A 9906 05x
Microprogramming Guide	98A 9906 07x
Writable Control Store Manual	98A 9906 08x
Memory Map Manual	98A 9906 10x
V76 System Reference Manual	98A 9906 23x
V77-600 System Reference Manual	98A 9906 40x
MAINTAIN III Manual	98A 9952 07x

The FPP is a high-speed, special-purpose processor which performs the arithmetic operations on single and double precision real numbers represented in V70 series floating point format. It also performs conversions between floating point and integer formats.

When the FPP is installed in a V70 series system, its 56-bit floating point accumulator and floating point instruction set are fully integrated into the computer architecture both at the machine language programming level and at the FORTRAN level. This minimizes the overhead associated with passing control between the FPP and the central processor. Pipelining of instructions is implemented in order to increase throughput. Direct memory access is used to obtain operands and store results.

The basic FPP clock period is 165 nanoseconds; however, fast shift operations occur during 82 nanosecond clock periods. This provides faster average execution times by minimizing the time spent in shifting the contents of the accumulator in connection with alignment or normalization.

Specifications for the FPP are listed in table 1-1.

Table 1-1. **Floating Point Processor Specifications**

Parameter	Specification
Arithmetic operations	Addition, subtraction, multiplication, or division of single or double precision floating point representations of real numbers. (Mixed precision operations such as addition of single precision operand to double precision operand can be performed.)
Format conversions	Conversions between floating point format and 16-bit twos complement integer formats. (Operands in integer format must be converted to floating point format prior to performing arithmetic. Results can be converted to integer format before storage in memory.
Number range (magnitude ranges are same for positive and negative numbers)	Largest magnitude: Single precision: $2^{127}(1 - 2^{-22})$ Double precision: $2^{127}(1 - 2^{-45})$ Smallest non-zero magnitude: 2^{-129} Zero is also a valid magnitude.
Normalization	All result values except zero are normalized before storage. All operand values except zero are normalized.
Round off	Results are rounded off before storage.
Out of range indication	The FPP initiates an interrupt when an out of range result occurs. (The interrupt is also initiated if a string of FPP instructions does not end with a store instruction within 500 micro-seconds).

Table 1-1. **Floating Point Processor Specifications (continued)**

Parameter	Specification
Interrupt inhibit	The FPP inhibits all interrupts when it is executing a sequence of instructions. (A sequence is defined to end when an instruction which stores a result is executed.)
Software operating system	Normally VORTEX I or VORTEX II. Will operate in any environment which provides required instruction, address, and operand formats.
Interrupt address	076 (octal).
Priority assignment	Between real-time clock and highest priority PIM (for VORTEX I or VORTEX II environment).
Basic cycle time	165 Nanoseconds.
Fast shift cycle time	82.5 Nanoseconds.
Dimensions	Contained on a 15.6 by 19 inch (39.6 by 48.3 cm) wire-wrap board.
Installation	Plugs into V70 series mainframe chassis using three module slots.
Input power	+5V dc at 16 amperes.
Operational environment	0 to 50 degrees C, 0 to 90 percent relative humidity without condensation.

SECTION 2
INSTALLATION

2.1 INSPECTION

The FPP has been packed and inspected to ensure its arrival in good working order. To prevent damage, take care during unpacking and handling. Check the shipping list to ensure that all equipment has been received. Immediately after unpacking, inspect the equipment for shipping damage. Ascertain that wires are neither loose nor broken and that hardware is secure. If damage exists:

a. Notify the transportation company.

b. Notify Sperry Univac.

c. Save all packing material.

2.2 PHYSICAL DESCRIPTION

The FPP circuits are on a 15.6 by 19 inch wire-wrap circuit board (p/n 44P0723). Figure 2-1 shows dimensions and connectors of the FPP board.

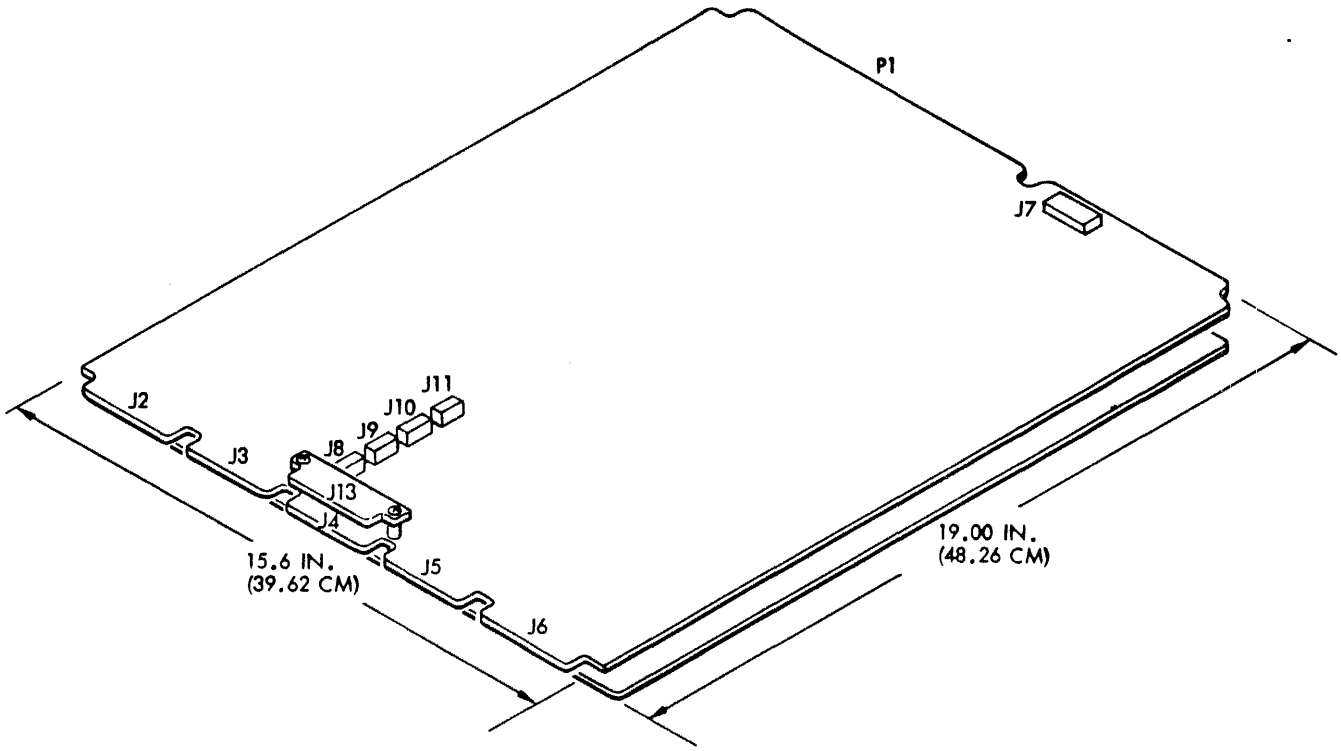
2.3 DISCRETIONARY WIRING

Some of the FPP board wiring is dependent upon the system configuration in which the floating point processor is used. The wiring required for the particular application is normally installed at the factory. If a change in the system configuration is required, refer to FPP option drawing 01A1682 (in the system documentation package) for information concerning optional wiring.

2.4 INTERCONNECTION

The FPP board plugs into the V70 series mainframe chassis using three module slots. The FPP board is essentially connected in parallel with the central processor board. The only exception to this involves three I/O control signals which pass directly between the central processor and the option board when an FPP board is not used but which are modified by the FPP when an FPP board is used.

The FPP board operates from a +5-volt power source and draws 16 amperes.



VTI1-3205

Figure 2-1. FPP Board

For details of signal and power interconnection for any system configuration refer to FPP option drawing 01A1682 (in the system documentation package).

SECTION 3
OPERATION

The FPP contains no operating controls or indicators. The FPP is normally operated in a VORTEX or VORTEX II environment using FORTRAN. However, the FPP can operate in any environment which supplies instructions, addresses, and operands in appropriate formats.

3.1 FPP INSTRUCTIONS

When so directed by the user, the FORTRAN compiler provides the FPP instructions listed in table 3-1. The user also has the option of coding FPP instructions directly in assembly language using the octal codes specified by table 3-1 or defining macros and using the table 3-1 mnemonics.

Table 3-1 gives minimum and maximum execution times for both semiconductor memory and core memory. The maximum times do not include unusual cases such as overflow.

Figure 3-1 illustrates the format of the FPP instruction word. The address format is illustrated in figure 3-2. There is no specific limitation on the number of levels of indirect addressing. However, the time allotted for the execution of any string of FPP instructions, including all required memory accesses, is 500 microseconds.

A string of FPP instructions is defined as any sequence during which operands are obtained and processed and a result is stored in memory. Thus, an FST, FSTD, or FIX instruction is always the last instruction of an FPP string.

3.2 NUMBER FORMATS

3.2.1 Single Precision Floating Number Format

As illustrated in figure 3-3, each single precision real floating point number is stored in memory in two consecutive word locations. The first word contains the sign bit (S), the exponent (in excess-128 format), and the seven high order bits of the fraction. The second word contains the 15 low order bits of the fraction (bit 15 is always zero).

Table 3-1. FPP Instructions

Mnemonic	Octal Code	Description	Execution Times (Nanoseconds)	
			Semiconductor Memory	Core Memory
		Memory Reference		
FLD	105420	Load floating point accumulator with single precision number.	1815	2723
FLDD	105522	Load floating point accumulator with double precision number.	2475	4043
FST	105600	Store floating point accumulator in memory in single precision format.	1980	2888
FSTD	105710	Store floating point accumulator in memory in double precision format.	2640	4208
FLT	105425	Reformat single precision integer and load into floating point accumulator.	1485-2475	2063-2723
FIX	105621	Reformat floating point accumulator and store integer in memory.	1980-3960	2558-4538
		Arithmetic Instructions		
FAD	105410	Add single precision memory to floating point accumulator.	1815-3300	2723-3878

Table 3-1. FPP Instructions (continued)

Mnemonic	Octal Code	Description	Execution Times (Nanoseconds)	
			Semiconductor Memory	Core Memory
FADD	105503	Add double precision memory to floating point accumulator.	2475-5940	4043-7178
FSB	105450	Single precision floating point subtraction.	1815-3300	2723-3878
FSBD	105543	Double precision floating point subtraction.	2475-5940	4043-7178
FMU	105416	Single precision floating point multiply.	2970-3300	3383-3713
FMUD	105506	Double precision floating point multiply.	5445-5775	6518-6848
FDV	105401	Single precision floating point divide.	3465-5280	4043-5693
FDVD	105535	Double precision floating point divide.	6105-9735	7343-10973

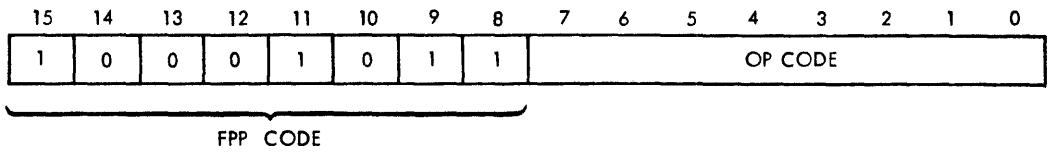


Figure 3-1. FPP Instruction Format

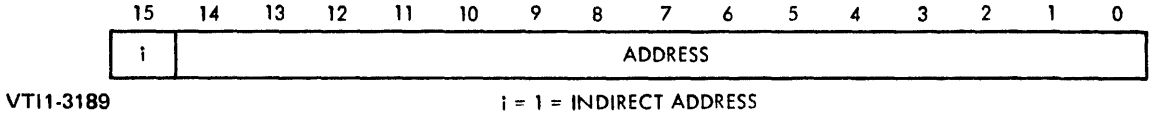


Figure 3-2. Operand Address Format

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1ST WORD (SA)	S	EXPONENT + 128								FRACTION (HIGH)						
2ND WORD (SB)	0	FRACTION (LOW)														

A. FIELDS

S	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64	1/128
0	$\frac{1}{256}$	$\frac{1}{512}$	$\frac{1}{1024}$	$\frac{1}{2048}$	$\frac{1}{4096}$	$\frac{1}{8192}$	$\frac{1}{16384}$	$\frac{1}{32768}$	$\frac{1}{65536}$	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}

B. DECIMAL WEIGHTS

$\frac{1}{4,194,304}$

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

EXPONENT + 128 = 128 + 8
EXPONENT = 8
FRACTION = 1/2 + 1/4 + 1/64 + 1/256
VALUE OF NO. = 2^E X (FRACTION)
= 2^8 (1/2 + 1/4 + 1/64 + 1/256)
= 256 (1/2 + 1/4 + 1/64 + 1/256)
= 128 + 64 + 4 + 1 = 213

C. EXAMPLE OF POSITIVE NUMBER (213)

WORD CONTAINING SIGN IS IN 1'S COMPLEMENT FORM →																
	1	0	1	1	1	0	1	1	1	0	0	1	1	1	0	1
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NOTE: THIS IS -213. SAME AS C EXCEPT NEGATIVE

D. EXAMPLE OF NEGATIVE NUMBER (-213)

VTI2-443

Figure 3-3. Single Precision Number Format

The value of a floating point number is 2^E X (FRACTION), where E = exponent. The 8-bit exponent code provides an exponent range of +127 (1111 1111) through -128 (0000 0000). Unless the value of a number is zero, the fraction must be normalized; that is, the most significant 1 of the fraction must be in the 2^{-1} bit position. The range of values that can be expressed by the fraction is thus 2^{-1} (represented by a 1 in the 2^{-1} bit position followed by all zeros) through $(1 - 2^{-22})$ (represented by all ones). Combining the exponent and fraction ranges, the largest magnitude that can be represented is 2^{127} X $(1 - 2^{-22})$ and the smallest non-zero magnitude is 2^{-128} X $2^{-1} = 2^{-129}$. A negative number is represented in the same manner as the corresponding positive number except that the entire word containing the sign bit is in one's complement form.

Figure 3-4 illustrates the representation of various numbers within the allowable range. Notice that the zero which always appears in bit position 15 of the second operand word is not shown in the binary representation since it is not a component of the fraction. It is, however, a component of the second operand word format, shown in octal format.

3.2.2 Double Precision Floating Point Number Format

As illustrated in figure 3-5, each double precision real floating point number is stored in memory in four consecutive word locations. The first word contains the exponent in excess-128 code. The second word contains the sign bit and the 15 most significant bits of the fraction. The range of values that can be represented by the double-precision fraction is 2^{-1} through $(1 - 2^{-45})$. The double precision words thus provide a non-zero magnitude range from 2^{-128} X $2^{-1} = 2^{-129}$ through 2^{127} X $(1 - 2^{-45})$. For a negative number, the entire second operand word is in one's complement form (since this is the word containing the sign bit).

3.2.3 Integer Format

As illustrated in figure 3-6, the integer format employs a single 16-bit word. Positive numbers appear in absolute form and negative numbers appear in two's complement form.

3.3 PROGRAM INTERRUPTS

The FPP normally inhibits program interrupts from the time that the first instruction of an FPP string is received until the final instruction of the string is executed, regardless of intervening instructions. However, the FPP can only inhibit interrupts for a maximum interval of 500 microseconds after which interrupts are again enabled and the FPP generates a fault interrupt. The FPP also generates a fault interrupt

	VALUE OF NUMBER	S	EXPONENT	HIGH FRACTION	LOW FRACTION	1ST OPERAND WORD (OCTAL)	2ND OPERAND WORD (OCTAL)
LARGEST MAGNITUDE	$2^{127} \times (1 - 2^{-22})$	0	11111111	1111111	11111111111111	077777	077777
	• • •						
	$2^{127} \times 2^{-1} = 2^{126}$	0	11111111	1000000	00000000000000	077700	000000
POSITIVE RANGE	• • •						
	$2^{1} \times 2^{-1} = 1$	0	10000001	1000000	00000000000000	040300	000000
	• • •						
SMALLEST MAGNITUDE	$2^{-128} \times 2^{-1} = 2^{-129}$	0	00000000	1000000	00000000000000	000100	000000
ZERO	$2^{-128} \times 0 = 0$	0	00000000	0000000	00000000000000	000000	000000
SMALLEST MAGNITUDE	$-2^{-128} \times 2^{-1} = -2^{-129}$	1	11111111	0111111	00000000000000	177677	000000
	• • •						
	$-2^{1} \times 2^{-1} = -1$	1	01111110	0111111	00000000000000	137477	000000
NEGATIVE RANGE	• • •						
	$-2^{127} \times 2^{-1} = -2^{126}$	1	00000000	0111111	00000000000000	100077	000000
	• • •						
LARGEST MAGNITUDE	$-2^{127} \times (1 - 2^{-22})$	1	00000000	0000000	11111111111111	100000	077777

VTI1-3212

Figure 3-4. Examples of Single Precision Numbers

following the completion of any FPP string during which a fault condition has been sensed.

3.4 FAULT CONDITIONS

3.4.1 Exponent Overflow

Exponent overflow can occur during the execution of any arithmetic instruction. It can also occur, in connection with round-off, during the execution of the memory reference instruction which stores the result. When an overflow is detected, the magnitude of the result is set to the largest possible in-range value and the sign of the number is not changed.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1ST WORD (DA)	0	ZEROS							EXPONENT + 128							
2ND WORD (DB)	S	FRACTION (HIGH)														
3RD WORD (DC)	0	FRACTION (MID)														
4TH WORD (DD)	0	FRACTION (LOW)														

A. FIELDS

0	0	0	0	0	0	0	0	128	64	32	16	8	4	2	1
S	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$	$\frac{1}{512}$	$\frac{1}{1024}$	$\frac{1}{2048}$	$\frac{1}{4096}$	$\frac{1}{8192}$	$\frac{1}{16384}$	$\frac{1}{32768}$
0	$\frac{1}{65536}$	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}	2^{-24}	2^{-25}	2^{-26}	2^{-27}	2^{-28}	2^{-29}	2^{-30}
0	2^{-31}	2^{-32}	2^{-33}	2^{-34}	2^{-35}	2^{-36}	2^{-37}	2^{-38}	2^{-39}	2^{-40}	2^{-41}	2^{-42}	2^{-43}	2^{-44}	2^{-45}

B. DECIMAL WEIGHTS

$$\frac{1}{35,184,372,088,832}$$

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

$+4,398,046,511,104.25 = 2^{E} \times (\text{FRACTION})$

$= 2^{43} \times (2^{-1} + 2^{-45}) = 2^{42} + 2^{-2}$

C. EXAMPLE OF POSITIVE NUMBER $(2^{42} + 2^{-2})$

WORD CONTAINING SIGN IS IN COMPLEMENT FORM →

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

NOTE: SAME AS C EXCEPT NEGATIVE

D. EXAMPLE OF NEGATIVE NUMBER

VTI2-445

Figure 3-5. Double Precision Number Format

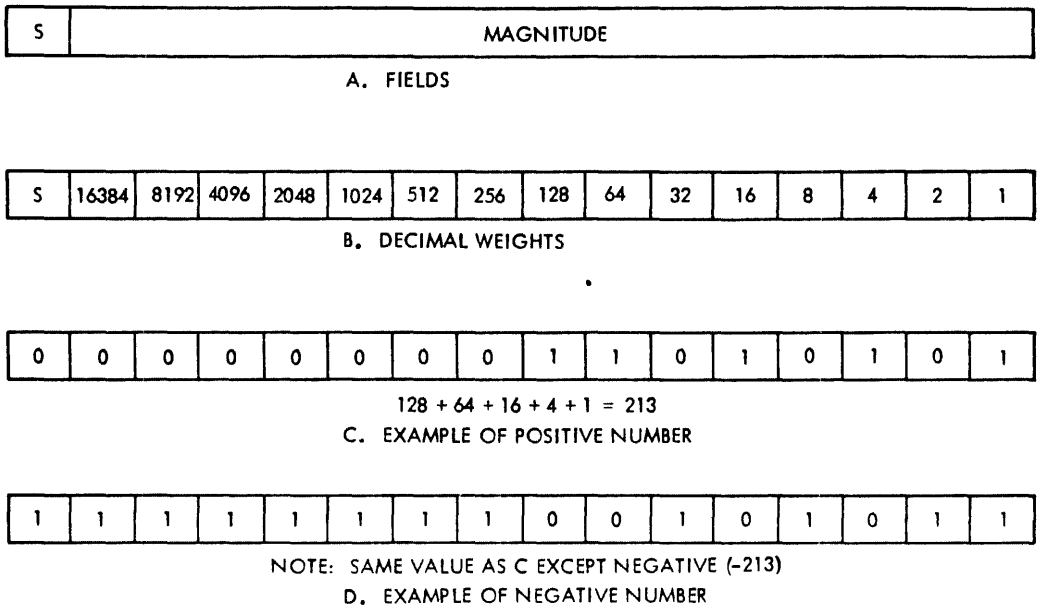


Figure 3-6. Integer Format

3.4.2 Exponent Underflow

Exponent underflow can occur during the execution of any arithmetic instruction. When an underflow is detected, the result is set to zero.

3.4.3 Integer Overflow

Integer overflow can occur during the FIX instruction. When a positive integer overflows, the integer is set to the largest positive in-range value. When a negative integer overflows, the integer is set to the most negative in-range value.

3.4.4 Divide by Zero

When an attempt to divide by zero is sensed, the result is set to the largest possible in-range value and the sign of the result is not changed.

3.4.5 Time-out

If the execution of an FPP instruction string is not completed within 500 microseconds, time-out is sensed. The result is then set to an illegal value to flag the type of fault that has led to the fault interrupt. (The fraction is set to the non-zero, non-normalized value, 0.01000...0. The exponent field is set to all zeros. The sign is not changed.)

SECTION 4
THEORY OF OPERATION

4.1 GENERAL

This section begins with a description of the major FPP functions and of the manner in which the FPP interfaces with the central processor and other components of the computer system. This is followed by a description of the microprogram facilities of the FPP. This includes information about the microinstruction word format as well as descriptions of the various FPP instruction microprogram routines. Flow charts are provided for each microprogram routine. The remainder of the section provides a more detailed description of the functional circuits which implement the various operations required to interface with the other system components and to execute the steps in the microprogram routines.

4.2 OVERALL FPP OPERATION AND SYSTEM INTERFACE

Figure 4-1 illustrates the FPP in terms of seven major functions.

4.2.1 Data Buffer, Address Output, and I/O Selection

This function provides a buffer register which can hold one instruction, address, or operand word received from memory. It provides an address counter. It provides multiplexing required to distribute operand words to the arithmetic section and to select result words from the arithmetic section for transfer to memory.

4.2.2 Instruction Double Buffer and Decoder

This function provides storage for two FPP instruction OP codes. It also provides a decoder which recognizes the FPP code. Every instruction fetched by the central processor is loaded into the FPP data buffer at the same time that it is loaded into the central processor instruction buffer. When an FPP instruction is fetched, the next instruction fetch brings the starting operand address into the central processor instruction buffer. Just before this occurs, the decoder recognizes the FPP code of the instruction held in the FPP data buffer. This recognition causes the OP code of the instruction to be transferred to the instruction double buffer at the time that the starting operand address is loaded into the data buffer. A double buffer is provided so that a second OP code can be loaded while the previously received instruction is still being executed.

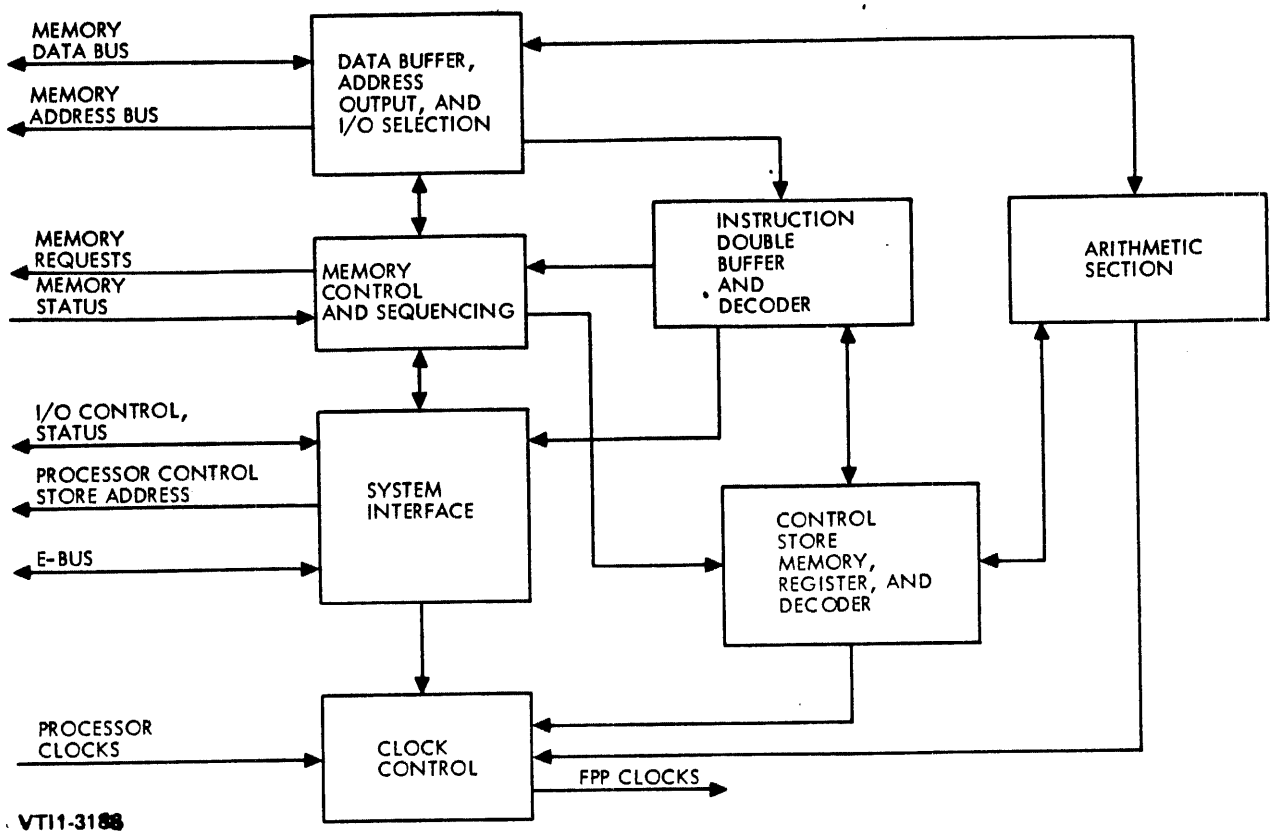


Figure 4-1. FPP Simplified Functional Block Diagram

4.2.3 System Interface

This section provides the facilities for passing program control back and forth between the central processor and the FPP. It inhibits interrupts during the execution of an FPP instruction string. It transmits an interrupt after a fault condition has been recognized. It resolves contention for memory access. The system interface passes program control to the FPP by forcing the central processor microprogram to a control store address containing a no-operation (NOP) microinstruction. The central processor microprogram is held at this address until the portion of the FPP microprogram which requires memory access has been completed. The system interface then forces the central processor microprogram to a location containing a microinstruction which leads to another instruction fetch. This allows the central processor to fill its instruction pipeline while the FPP is completing the execution of an FPP instruction microprogram routine. If the next instruction is an FPP instruction, the central processor microprogram is again forced to the location containing a NOP at the time that it fetches the starting operand address. Thus,

during the execution of a string of FPP instructions, the central processor activity is limited to the fetching of instructions and associated operand starting addresses. From the time that the first instruction of the string is recognized by the FPP decoder until control is passed back to the central processor during the execution of the final instruction of the string, the system interface inhibits interrupts. An instruction string ends when an instruction which stores a result in memory is executed. During FPP memory accesses, the system interface controls various interface lines so as to prevent other system facilities from accessing memory. At the end of an FPP memory access cycle or if an FPP memory access is not currently in progress, the system interface yields memory access priority if a direct memory access (DMA) or priority memory access (PMA) request is pending. If the FPP requires memory access and does not have memory access priority, it suspends the activity that requires the memory access until memory access priority is again available.

4.2.4 Memory Control and Sequencing

This section initiates memory requests, recognizes the completion of memory cycles, and provides the signal which loads the data from the memory data bus into the data buffer. During central processor instruction fetches, the memory control merely steers a central processor control signal to the data buffer as required to load it. In this way, first an FPP instruction and then the associated operand address are loaded into the data buffer. If the operand address is an indirect address, the memory control is now responsible for initiating as many memory access requests as may be necessary to obtain a direct address. For an instruction during which an operand must be obtained from memory, the memory control is also responsible for initiating the memory access required to obtain the first operand word. These memory accesses can be accomplished without intervention of the FPP microprogram. Th s, they can occur while a previous instruction microprogram routine is still being executed.

4.2.5 Arithmetic Section

This section provides the registers and arithmetic/logic units (ALUs) required to perform floating point arithmetic as well as associated control circuits.

4.2.6 Control Store Memory, Register, and Decoder

This function provides the control store read-only memory in which the FPP microprogram is stored. It provides an address counter which determines the order in which microinstructions are

executed, a register which holds the instruction currently being executed, and circuits which decode certain microinstruction fields. The OP code of each FPP instruction points to the starting address of the associated microinstruction routine. Within the routine, microinstructions are executed in sequential order except when a jump alters the contents of the address counter. The microprogram initiates the memory accesses required to obtain operand words (with the exception of the first operand word), controls the transfer of operand words into the arithmetic section, and controls the sequence of operations within the arithmetic section. During the execution of an instruction which stores a result in memory, the microprogram initiates the memory accesses required to store each of the operand words.

4.2.7 Clock Control

This function provides the clock signals which time parallel data transfers, serial shifts of data, transitions of control signals, and advancement of the microprogram. All of the FPP clock rates are derived from clock rates received from the central processor. The clock control section exercises control over the timing of microprogram execution by inhibiting the clock which times the microprogram advancement. When the microprogram initiates a memory access, the clock is inhibited until the required memory cycle has been completed. The clock is also inhibited during certain operations of the arithmetic section until the required number of iterations of the operation have been completed. When the final microinstruction of a routine reaches the control store register, the clock is inhibited until the next valid FPP instruction OP code is available in the instruction buffer. Another function of clock control is to select the fast shift clock rate. The fast shift clock period is 82.5 nanoseconds as compared to the basic clock rate of 165 nanoseconds.

4.3 FPP DETAILED BLOCK DIAGRAM

Figure 4-2 is a multi-sheet block diagram which illustrates the FPP in terms of 24 functional blocks. Section 1 illustrates the operand, result, memory address, and instruction data parallel paths. Section 2 illustrates the interface between the control store function and the other functional blocks. Section 3 illustrates the control signals, other than control store signals, which pass between the various functional blocks. Section 4 illustrates the distribution of clock signals and shows clock control inputs other than those from the control store function. Section 5 illustrates the control interface with the central processor and option board.

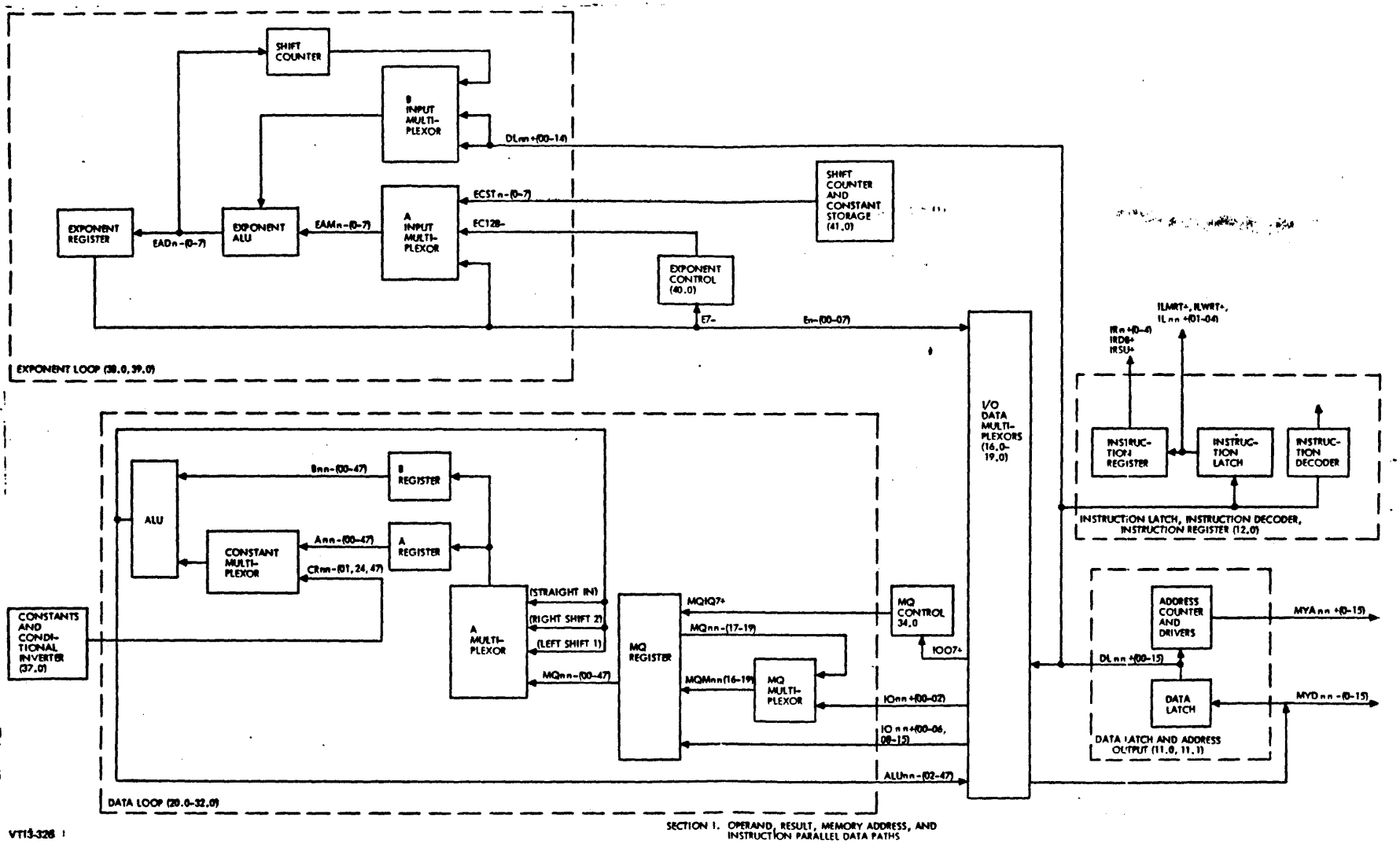


Figure 4-2. FPP Detailed Block Diagram (Sheet 1 of 4)

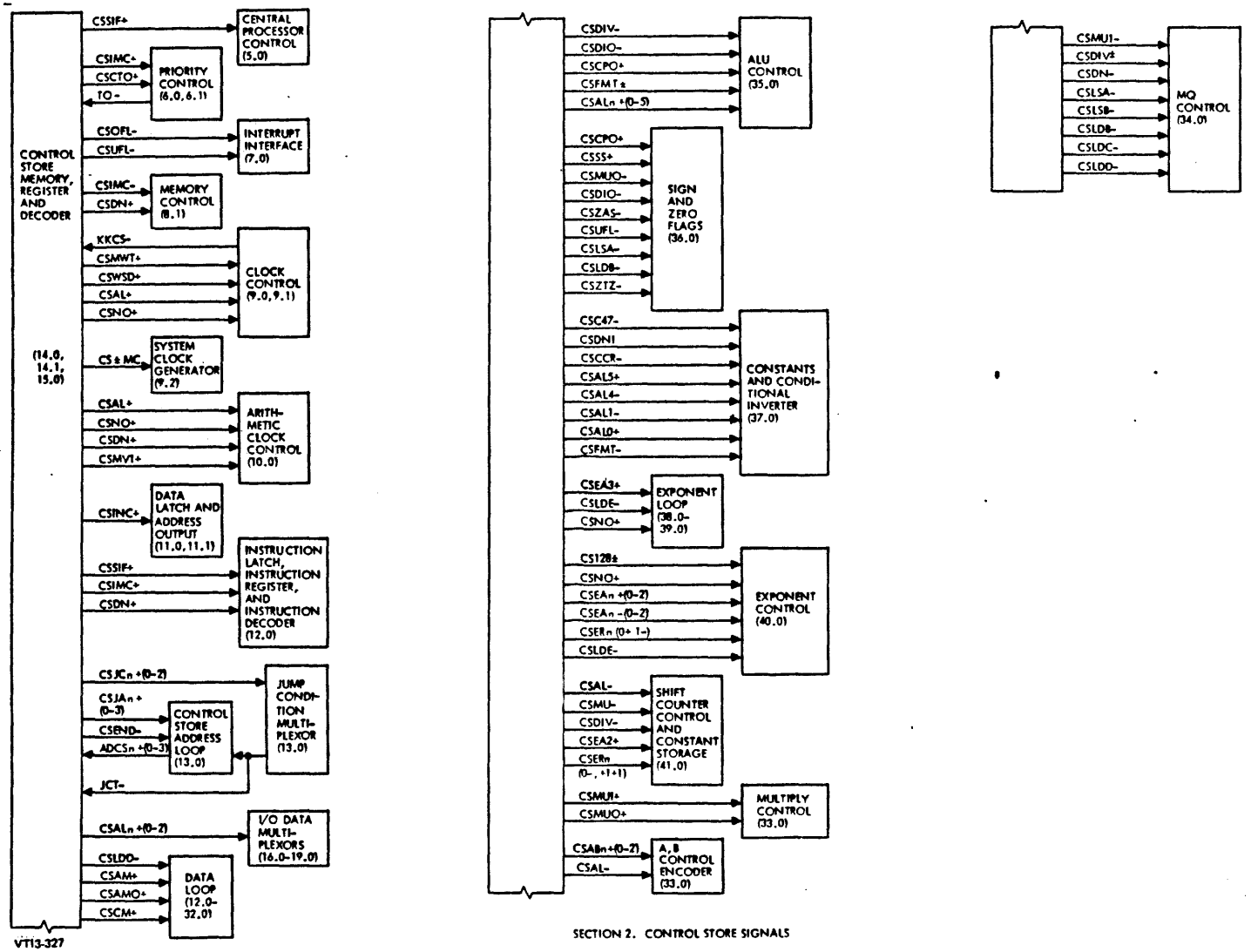


Figure 4-2. FPP Detailed Block Diagram (Sheet 2 of 4)

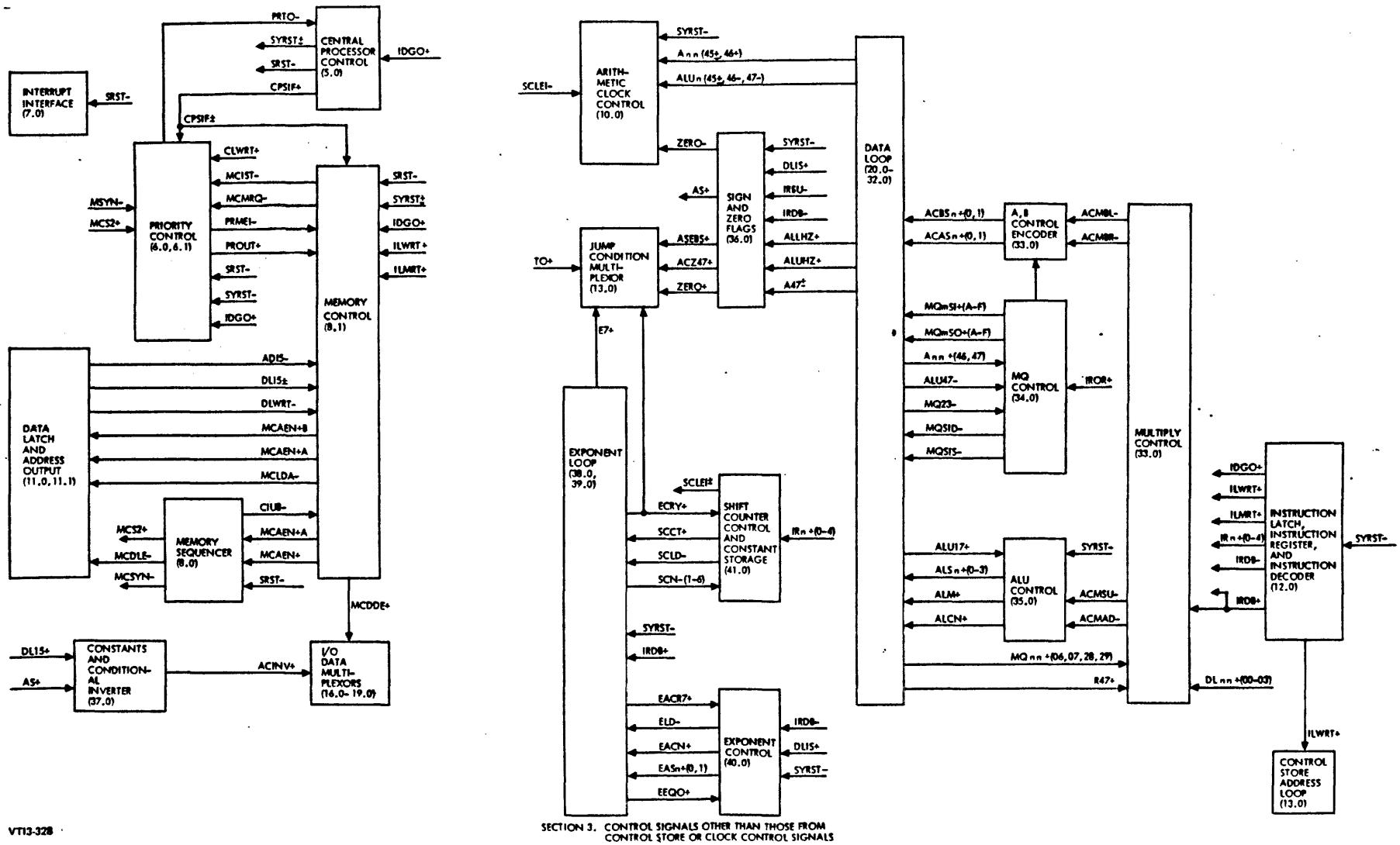
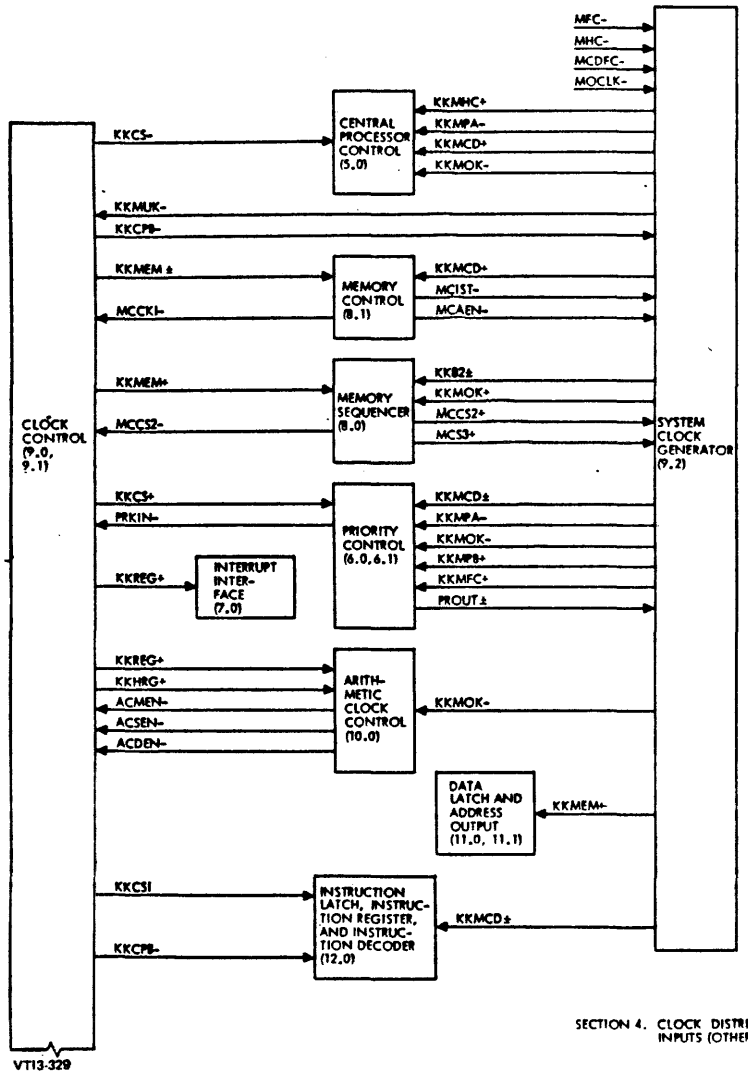
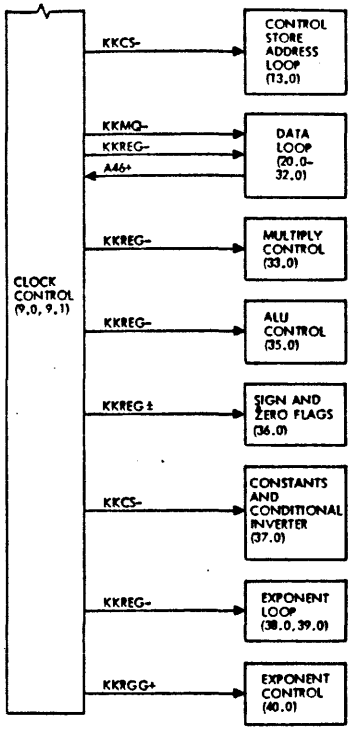
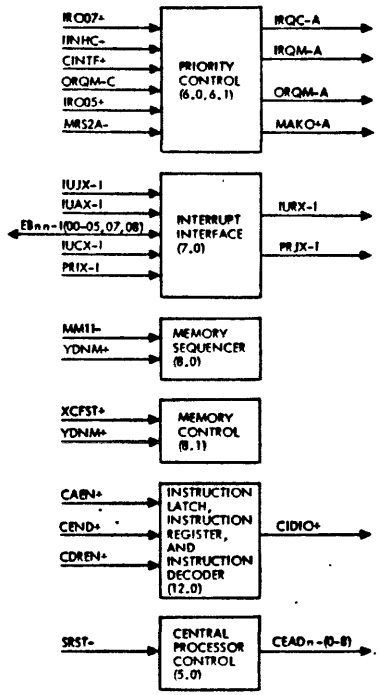


Figure 4-2. FPP Detailed Block Diagram (Sheet 3 of 4)



SECTION 4. CLOCK DISTRIBUTION AND CLOCK CONTROL INPUTS (OTHER THAN FROM CONTROL STORE).





SECTION 5. CONTROL INTERFACE WITH CENTRAL PROCESSOR AND OPTION BOARD

Figure 4-2. FPP Detailed Block Diagram (Sheet 4 of 4)

Functions shown in figure 4-2 correspond in most cases with functions which are separately identified in FPP logic diagram 91C0499. The instruction latch, instruction register, and instruction decoder block shown on figure 4-2 corresponds to the instruction double buffer and decoder block shown in figure 4-1. Other blocks shown in figure 4-2 are subsidiary to blocks shown in figure 4-1. Table 4-1 summarizes the correlation between nomenclature used on figure 4-1 and 4-2 and nomenclature used on the FPP logic diagram.

Numbers within parenthesis in figure 4-2 refer to sheets of FPP logic diagram 91C0499.

4.4 MICROINSTRUCTION FORMAT

Figure 4-3 illustrates the two formats used in microinstruction words. The mnemonic assigned to each of the 32 bits of the microinstruction word are shown at the left of the figure. If CSFMT+ is low, then format 0 is designated. If CSFMT+ is high, then format 1 is designated. The only difference between the two formats is in the interpretation of the status of the CSALn+ (0-5) bits. In format 0, bits CSALn+ (3-5) constitute the IN field while bits CSALn+ (0-2) constitute the IO field. In format 1, the six CSALn+ (0-5) bits constitute the ALU field.

Table 4-2 lists the various active codes used in each microinstruction field. Each active code associated with a field is assigned a mnemonic which is listed in the FLOW CHART MNEMONIC column of the table. This mnemonic is either used directly on the microprogram flow charts or else is used on supporting documents to indicate the field codes associated with particular operations.

Where a field code is represented by an explicit decoding signal, the decoding signal mnemonic is listed. (In many cases, the field code bits are supplied to function selection inputs of ALUs or registers or to address inputs of multiplexors so that separate decoding is not required.)

4.5 MICROPROGRAM ROUTINES

The paragraphs which follow provide descriptions of each of the routines contained in the FPP microprogram. Each description is supported by a flow chart. Figure 4-4 illustrates the microinstruction block format that is used on the flow charts and table 4-3 lists special notations that are used on the flow charts.

Table 4-1. Nomenclature Correlation

Figure 4-1	Figure 4-2	FPP Logic Diagram
Instruction double buffer and decoder	Instruction latch, instruction register, and instruction decoder	Instruction register (sheet 12.0)
System interface	Central processor control	Central processor control (sheet 5.0)
	Priority control	Priority control (sheet 6.0, 6.1)
	Interrupt interface	Interrupt interface (sheet 7.0)
Memory control and sequencing	Memory sequencer	Memory sequencer (sheet 8.0)
	Memory control	Memory control (sheet 8.1)
Clock control	Clock control	Clock control (sheets 9.0, 9.1)
	System clock generator	System clock generator (sheet 9.2)
Data buffer, address output, and I/O selection	Data latch and address output	Data latch and address output (sheets 11.0, 11.1)
	I/O data multiplexors	Input/output data (sheets 16.0 through 19.0)
Control store memory, register, and decoder	Control store memory, register, and decoder	Control store register, control store decoder (sheet 14.0, 14.1, 15.0)
	Control store address loop	Control store address loop (sheet 13.0)
	Jump condition multiplexor	

Table 4-1. Nomenclature Correlation (continued)

Figure 4-1	Figure 4-2	FPP Logic Diagram
Arithmetic section	Data loop	Data loop (sheets 20.0 through 32.0)
	Arithmetic clock control	Arithmetic control clock logic (sheet 10.0)
	Multiply control	Arithmetic control (A, B and multiply) (sheet 33.0)
	A, B control encoder	
	MQ control	Arithmetic control (MQ) (sheet 34.0)
	ALU control	Arithmetic control (ALU) (sheet 35.0)
	Sign and zero flags	Arithmetic control (Sign and Zero Flags) (sheet 36.0)
	Constants and conditional inverter	Arithmetic control (constants and conditional invert) (sheet 37.0)
	Exponent loop	Exponent loop (sheets 38.0, 39.0)
	Exponent control	Exponent control (sheet 40.0)
	Shift counter control and constant storage	Exponent control, shift counter and constant (sheet 41.0)

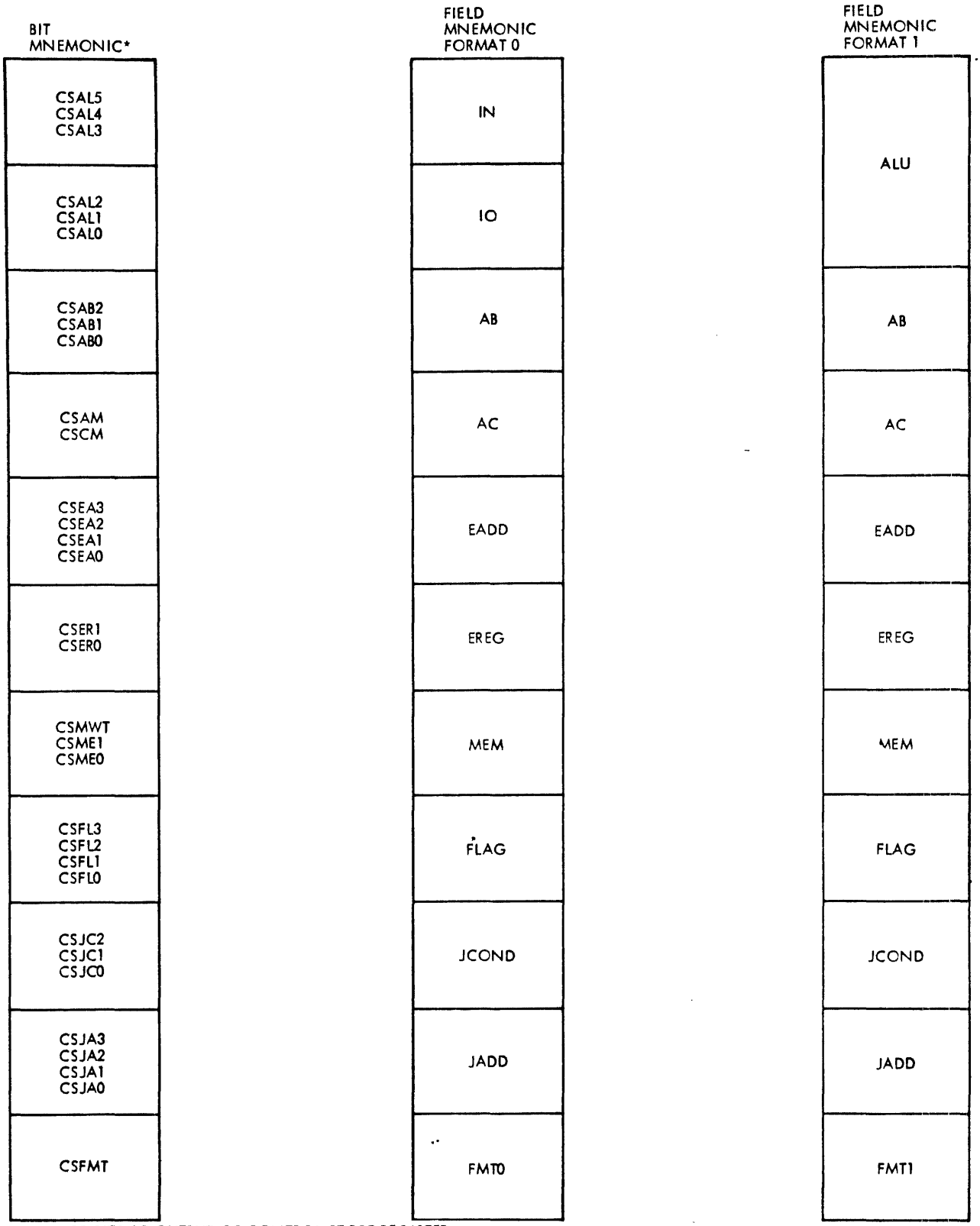


Figure 4-3. Microinstruction Word Formats

Table 4-2. Microinstruction Field Formats

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
IN	CSALn+ 5 4 3			
	L L L			No operation
	L L H	LSAQ	CSLSA-	Loads single precision high fraction into MQ register
	L H L	LSBQ	CSLSB-	Loads single precision low fraction into MQ register
	H L H	LDBQ	CSLDB-	Loads double precision high fraction into MQ register
	H H L	LDCQ	CSLDC-	Loads double precision middle fraction into MQ register
	H H H	LDDQ	CSLDD-	Loads double precision low fraction into MQ register
IO	CSALn+ 2 1 0			
	L L L	ID		Connects data latch outputs to 10nn+(00-15) bus in format appropriate for transfer of any operand field (except single precision high fraction) to MQ register

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
IO (Cont'd)	CSALn+ 2 1 0			
	L L H	OSA		Connects first single precision result word to IOnn+(00-15) bus
	L H L	OSB		Connects second single precision result word to IOnn+(00-15) bus
	L H H	IS		Connects data latch outputs to IOnn+(00-15) bus in format appropriate for transfer of single precision high fraction to MQ register
	H L L	ODA		Connects first word of double precision result to IOnn+(00-15) bus
	H L H	ODB		Connects second word of double precision result to IOnn(00-15) bus
	H H L	ODC		Connects third word of double precision result to IOnn+(00-15) bus
	H H H	ODD		Connects fourth word of double precision result to IOnn+(00-15) bus

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
ALU	CSALn+ 5 4 3 2 1 0			
	L L L L H L	CMPA		ALU = $\overline{A}$
	L L H H H L	ALONE		ALU = 1
	L H L H H L	BNOT		ALU = $\overline{B}$
	L H H L L H	AMINB		ALU = A MINUS B
	H L L H L L	APLUSB		ALU = A PLUS B
	H L H L H L	B		ALU = B
	H H L L L L	APLUSA		ALU = 2A
	H H L L H L	ALZERO		ALU = 0
	H H H H H L	A		ALU = A
	L L H H L L	MINUS1		ALU = -1
ALU (For CSWSD+=H)	CSALn+ 5 4 3 2 1 0			
	L L L L L H	AL	CSAL±	Alignment micro-instruction during which fraction of smaller operand is shifted to right to align it properly with fraction of larger operand
	L L L L H L	NO	CSNØ±	Normalize micro-instruction during which fraction is shifted to left until most significant 1 is in bit 46 position

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
AB	CSABn+ 2 1 0			
	L L L			No operation
	L L H	SRA		Shifts A register to right
	L H L	SLA		Shifts A register to left
	L H H	LDA		Loads A register
	H L L	LDB		Loads B register
	H L H	SRAB		Shifts A and B registers to right
	H H L	SLAB		Shifts A and B registers to left
	H H H	LDAB		Loads A and B registers
AC	CSAM+ CSCM+			
	L L	MQA		Connects MQ register outputs to AMnn-(00-47) bus; Connects A register to A input of ALU
	L H	MQC		Connects MQ register outputs to AMnn-(00-47) bus; connects constants register (CR) to A input of ALU

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
AC (Cont'd)	CSAM+ CSCM+			
	H L	ALA		Connects ALU outputs to AMnn- (00-47) bus; Connects A register to A input of ALU
	H H	ALC		Connects ALU outputs to AMnn- (00-47) bus; Connects CR register to A input of ALU
EADD	CSEAn+ 3 2 1 2			
	L L L L	EAZERO		Exponent ALU = 0
	L L L H	EMX		Exponent ALU = EMX, where EMX is data selected for connection to A input of exponent ALU
	L L H L	MINEMX		Exponent ALU = EMX
	L L H H	EA255		Exponent ALU = decimal 255
	L H L L	C		Exponent ALU = C_e, where C_e is value from constant store
	L H L H	CPLSMX		Exponent ALU = C_e + EMX

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
EADD (Cont'd)	CSEAn+ 3 2 1 0			
	L H H L	CMINMX		Exponent ALU = C_e - EMX
	L H H H	CMIN1		Exponent ALU = C_e - 1
	H L L L	E		Exponent ALU = E: where E is output of exponent register
	H L L H	EPLSMX		Exponent ALU = E + EMX
	H L H L	EMINMX		Exponent ALU = E - EMX
	H L H H	EMIN1		Exponent ALU = E - 1
	H H L L	EPLUS1		Exponent ALU = E + 1
EREG	CSERn+ 1 0			
	L L			No operation
	L H	LDE		Load exponent register
	H L	LDSC1		Load shift counter if ECRY- = H
	H H	LDSC		Load shift counter

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
MEM	CSMWT+ CSMEn+ 1 0			
	L L L			No operation
	L H H	WSDN	CSWSD±	Inhibits control store clock until SC = 0; where SC is shift counter. Also enables special AL and NO code recognition (for ALU field)
	L H H	END	CSEND-	If jump condition (specified by JCOND field) is satisfied, JADD field is loaded into four MSBs of control store counter. If jump condition is not satisfied, CSDN+ is placed at high level. This enables transfer from instruction latch to instruction register and inhibits control store clock if new FPP instruction is not available in instruction latch.
	H L L	IMC	CSMST-, CSIMC+	Leads to initiation of memory cycle (unless time-out flag, TØ+ = H)

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
MEM (Cont'd)	CSMWT+ CSMEn+ 1 0			
	H L H	INC	CSINC-, CSIMC+	Leads to initiation of memory cycle (unless time-out flag, TØ = H) and increments memory address counter
	H H H	SIF	CSSIF	Enables starting of central processor instruction fetch at next control store clock time
FLAG	CSFLn+ 3 2 1 0			
	L L L L			No operation
	L L L H	CPO	CSCPO+	Reverses status of AS+ and sets SUB+ and ADD- high if ALU37+ is high. Sets SUB+ and ADD- low if ALU47+ is low
	L L H L	MUO	CSMUO±	Conditions arithmetic control for multiply-initialize microinstruction. Also, forces CSMU1+ to set high at control store clock time so as to establish main-multiply functions during next microinstruction period
	L L H H	ZTZ	CSZTZ±	Inhibits updating of status of ZERO flag

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
FLAG (Cont'd)	CSFLn+ 3 2 1 0			
	L H L L	ZAS	CSZAS-	Resets AS+ flag to low level
	L H L H	SS	CSSS±	Swaps states of AS and BS flags
	L H H L	OFL	CSOFL-	Sets interrupt flag
	L H H H	DIO	CSDIO-	Conditions arithmetic control for divide-initialize microinstruction
	H L L L	E128	CS128±	Conditions exponent control to invert MSB of E as required to maintain excess-128 code during addition or subtraction of exponents
	H L L H	EXC	CSEXC-	Forces CSLDE- to set low at control clock time as required to load exponent register from data latch during next microinstruction period
	H L H L	CCR	CSCCR-	Clears constants register (CR)
	H L H H	C47	CS47±	Loads 1 into bit 47 of CR register
	H H L L	DIV	CSDIV±	Conditions arithmetic control for main divide microinstruction

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
FLAG (Cont'd)	CSFLn+ 3 2 1 0			
	H H L H	UFL	CSUFL-	Sets interrupt flag; sets AS+ low
	H H H L	CTO	CSTØ+	Resets time-out flag (TØ+) to low level
JCOND	CSJn+ 2 1 0			
	L L L			Inhibits jump
	L L H	JMP		Unconditional jump
	L H L	JC		Jump if ECRY+ = H
	L H H	JZ		Jump if ZERO+ = H
	H L L	JSGN		Jump if A47+ = H or if ZERO+ = H
	H L H	JSAGN		Jump if AS+ = BS+
	H H L	JESGN		Jump if E7+ = H
	H H H	JTO		Jump if TØ+ = H
JADD				These bits are used to alter control store address counter if jump condition is satisfied. If CSEND+ = H, they are loaded into four MSBs of counter while if CSEND+ = L, they are loaded into four LSBs of counter.

Table 4-2. Microinstruction Field Formats (continued)

Field Mnemonic	Field Bit Levels	Flow Chart Mnemonic	Decoding Signal Mnemonic	Function
FMT	CSFMT+ = L			Microinstruction format 0
	CSFMT+ = H			Microinstruction format 1

Table 4-3. Special Flowchart Notations

Notation	Description
(DA), (DB), (DC), (DD)	Four words of double-precision operand
(SA), (SB)	Two words of single-precision operand
NC	Normalization count – indicates the number of shifts required to normalize fraction
CUFL	Conditional underflow – indicates exponent intermediate value which will produce underflow if fraction has to be shifted
COFL	Conditional overflow – indicates intermediate exponent value which may produce overflow
SC	Shift counter
E	Exponent register
EA	Exponent adder
EA8	Carry (borrow) from exponent adder
A	A register
B	B register
MQ	MQ register
UFL	Used on right side of microinstruction block to indicate the sensing of underflow condition. Used on left side of block to indicate FLAG field mnemonic
OFL	Used on right side of microinstruction block to indicate sensing of overflow. Used on left side of block to indicate FLAG field mnemonic.
CR	Constant register

As an aid in understanding the more complex routines, numerical examples are provided. Each numerical example is presented in the form of a figure which shows the microinstruction path that is followed for the particular operand values being assumed. The operations performed by each microinstruction and the status of significant registers and flags that results from these operations is shown.

4.5.1 FLD Routine

The purpose of the FLD routine (figure 4-5) is to obtain a single-precision operand from memory, place the exponent field of the operand in the exponent register, place the operand fraction in both the A and B registers, and place the sign of the operand in the AS flag.

When the routine is started, the memory access required to obtain the first operand word has already been initiated. The INC code in the MEM field of microinstruction LDSO causes the execution of the microinstruction to be delayed until the completion of the memory access cycle during which the first operand word is transferred from the memory to the FPP data latch. When the microinstruction is executed, the INC code causes the memory address count to be incremented to point to the memory location containing the second operand word and causes the initiation of the memory request required to obtain this second operand word. The EXC code in the FLAG field of the microinstruction prepares for the transfer of the exponent field of the first operand word from the data latch through the exponent loop ALU to the exponent register during the next microinstruction.

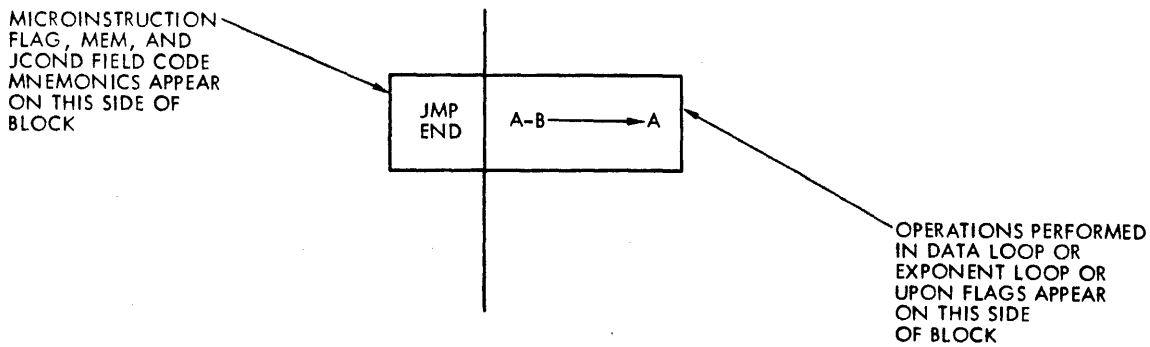


Figure 4-4. Flow Chart Microinstruction Block Format

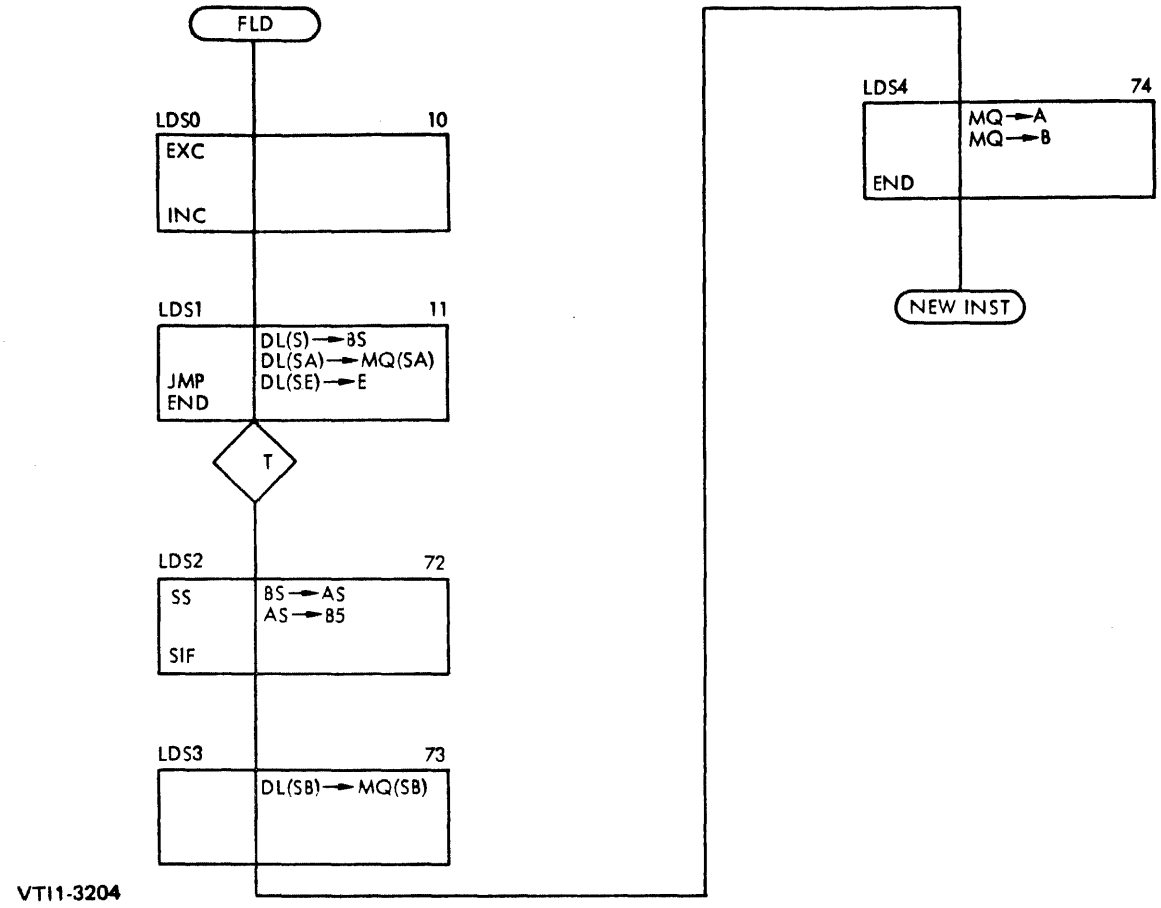


Figure 4-5. FLD Routine Flowchart

Microinstruction LDS1 transfers the sign bit (S) of the first operand word from the data latch (DL) to the BS flag, transfers the high fraction field (SA) of the first operand word from the data latch to the MQ register, and transfers the exponent field (SE) of the first operand word from the data latch to the exponent register (E).

The SIF field of microinstruction LDS2 causes the execution of this microinstruction to be suspended until the completion of the memory cycle during which the second operand word is transferred from memory to the FPP data latch. When the microinstruction is executed, the SIF code causes control to be returned to the central processor so that the next instruction fetch can be executed. The SS code in the FLAG field of the microinstruction

causes the swapping of the sign bits held in the AS and BS flags as required to enter the sign bit of the new operand into the AS flag.

Microinstruction LDS3 transfers the low fraction field of the second operand word from the data latch to the MQ register. Microinstruction LDS4 transfers the assembled fraction from the MQ register into the A and B registers.

4.5.2 FLDD Routine

The FLDD routine (figure 4-6) is similar to the FLD routine except that it obtains a double precision operand. It is thus required to initiate the three memory accesses needed to obtain the second through fourth operand words and to assemble the three fraction fields in the MQ register to form the 45-bit fraction. The INC code in the MEM field of microinstructions LDD0, LDD2, and LDD4 causes the necessary incrementing of the memory address

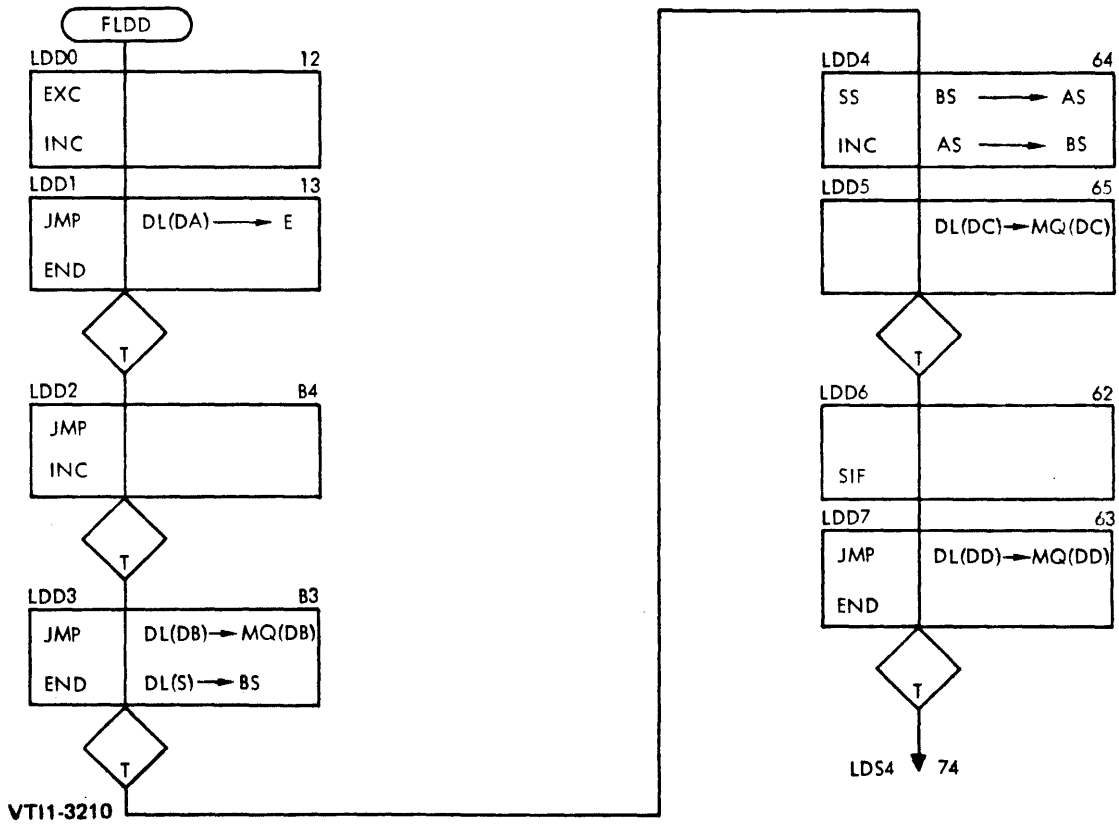


Figure 4-6. FLDD Routine Flowchart

to point to each successive operand word and causes the initiation of the required memory accesses. Microinstruction LDD1 transfers the exponent field of the first operand word from the data latch to the exponent registers. Microinstructions LDD3, LDD5, and LDD7 transfer the three fraction fields of the second, third, and fourth operand words respectively from the data latch to the MQ register. Microinstruction LDD3 transfers the sign bit (S) of the second operand word from the data latch to the BS flag and microinstruction LDD4 swaps the AS and BS flags as required to enter the new sign bit into the AS flag. LDD6 prepares for the return of control to the central processor so that the next instruction fetch can be performed. The routine jumps to microinstruction LDS4 of the FLD routine which transfers the assembled fraction from the MQ register to the A and B registers.

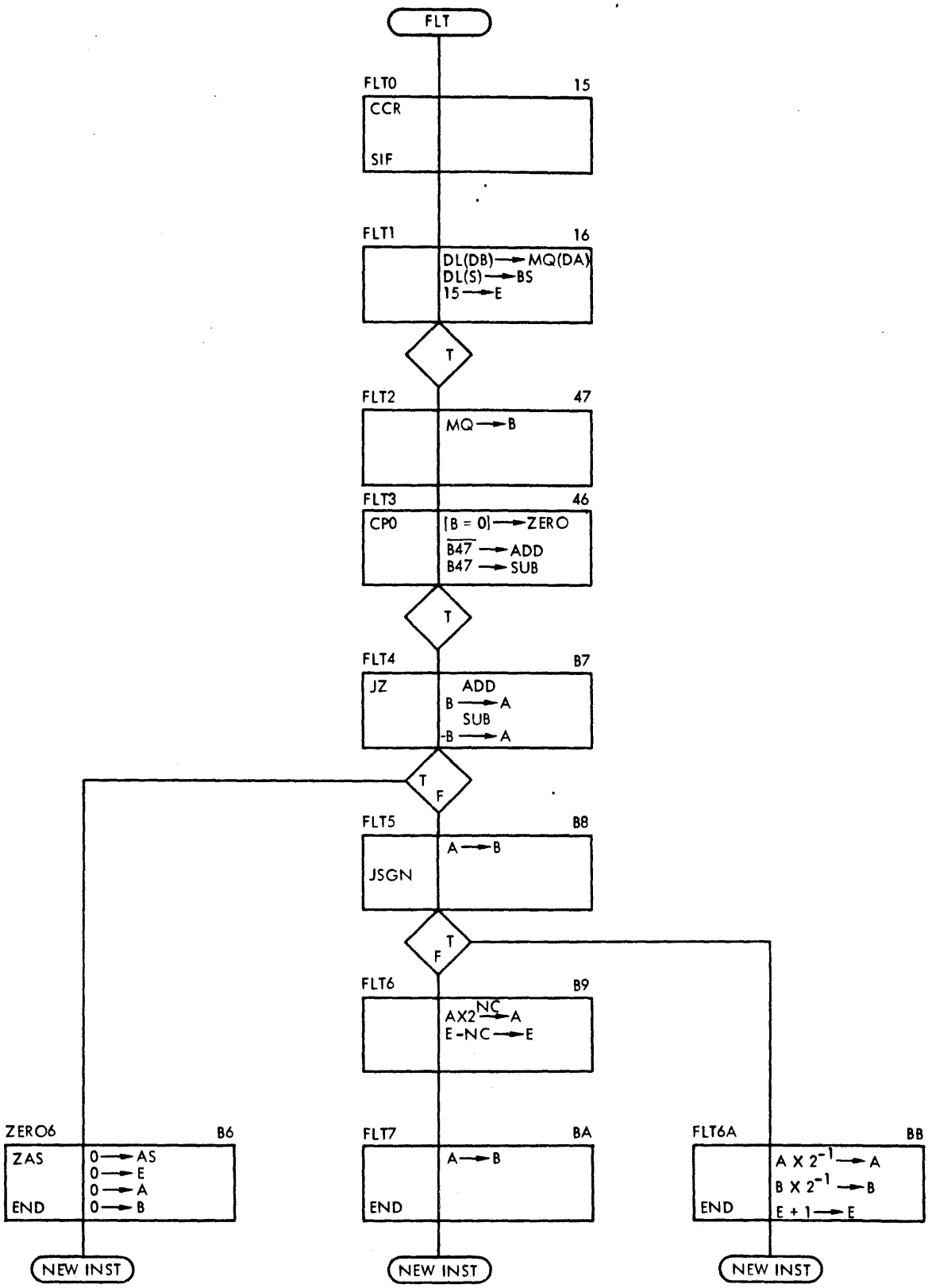
4.5.3 FLT Routine

The purpose of this routine is to obtain a single-precision integer from memory and convert it to the floating point format. At the completion of the routine, the exponent register contains the exponent of the converted operand, the A and B registers both contain copies of the operand fraction, and the AS flag contains the operand sign bit.

The integer format provides 15 bits to the left of the decimal point. In the floating point format, these bits must appear to the right of the decimal point. Thus, to maintain the same value, the fraction must be multiplied by 2^{15}. In order to meet this requirement, the exponent is provisionally set to 15. If the number being converted is the maximum negative number that can be represented by the integer format, there is a magnitude overflow into the sign bit position. In this case, the fraction must be shifted to the right and the exponent must be incremented by 1. Alternatively, the fraction may have to be shifted to the left to normalize it. In this case, the exponent must be decremented by 1 for each left shift.

A negative integer is represented in two's complement form. Thus, before the normalization status of the fraction is tested, it must be determined whether the integer is positive or negative. If the integer is negative, it must be complemented so as to convert it to absolute form. The integer must also be tested to determine whether it is zero since this requires a special floating point format in which all fields are set to zero.

At the time that the routine (figure 4-7) is started, the memory access required to obtain the integer word from memory has already been initiated. The SIF code in the MEM field of microinstruction FLTO suspends the execution of the instruction



VTI2-439

Figure 4-7. FLT Routine Flowchart

until the memory cycle during which the integer is transferred from memory to the FPP data latch has been completed. When the microinstruction is executed, the SIF code causes control to be returned to the central processor so that the next instruction fetch can be executed. The CCR code in the FLAG field causes the constant register (CR) to be cleared.

Microinstruction FLT1 transfers the integer from the data latch to the MQ register. The transfer is executed in response to the LDBQ and ID codes in the IN and IO fields of the instruction respectively. During a double precision operation, this combination of codes would cause a ones complementation of the data being transferred if that data were negative. However, in this case, the data is not converted because the double-precision bit of the FLT instruction OP code held in the instruction register is a zero. Thus, if the integer is negative, the binary one sign bit is loaded into bit position 47 of the MQ register. This is used subsequently in testing the sign of the integer. In addition to transferring the integer to the MQ register, the microinstruction also transfers the sign bit into the BS flag. It also loads 143 (the excess-128 code for 15) into the exponent register.

Microinstruction FLT2 transfers the integer from the MQ register to the B register. The CPO code in the FLAG field of microinstruction FLT3 sets the ADD flag if the sign bit of the integer in the B register is a zero (indicating a positive number) or sets the SUB flag if the sign bit is a one (indicating a negative number). (The function, ALU = B is performed in the data loop ALU and it is actually the status of bit 47 of the ALU that is evaluated rather than the status of bit 47 of the B register itself. Also, if the integer is zero, then the all-zero output of the ALU causes the ZERO+ flag to set high.)

Microinstruction FLT4 places either B or the twos complement of B in the A register depending upon whether the ADD or SUB flag was set in the previous microinstruction. This provides a twos complementation of the integer if it is initially negative. Thus, the contents of the A register after this microinstruction is executed is the absolute value of the integer. The explicit function performed by the ALU is either ALU = (CR + B) or ALU = (CR - B). However, since the CR register was reset to zero in microinstruction FLTO, the operations reduce to ALU = B and ALU = -B.

The JZ code in the JCOND field of microinstruction FLT4 causes a pump to microinstruction ZERO6 if ZERO+ was set high in FLT3; that is, if the integer is zero. The ZERO6 microinstruction sets all result fields to zero as required to provide the floating point zero format.

If the integer is not zero, the routine advances from FLT4 to FLT5. FLT5 transfers the operand value from the A register to the B register. The SS code in the FLAG field of the microinstruction causes the sign bits in the AS and BS flags to be swapped. This places the sign of the integer in the AS flag. The JSGN code in the JCOND field causes a jump to microinstruction FLT6A if A47 is high. This takes care of the case where the operand value is the maximum negative value that can be represented in the integer format. This value produces a magnitude overflow into the sign bit position when the two complementation is performed. (The two's complement of 1000 0000 0000 0000 is 1000 0000 0000 0000.) Microinstruction FLT6A shifts this fraction one position to the right to normalize it and increments the exponent by 1.

If a magnitude overflow in bit position 47 does not occur, the routine advances from microinstruction FLT5 to microinstruction FLT6. If left shifting of the fraction and decrementing of the exponent is required to normalize the fraction it is performed by this microinstruction. (If bit position 46 does not contain a binary one, then the fraction is left shifted until a binary one reaches this bit position. For each bit position shift of the fraction, the exponent is decremented by 1.) The normalization of the fraction is implemented in the A register. Microinstruction FLT7 copies the normalized fraction into the B register.

A numerical example of the FLT routine is provided in figure 4-8.

4.5.4 FAD/FSB Routine

As illustrated in the flowchart of figure 4-9, the same routine is used to execute either the FAD or the FSB instruction. For the subtraction instruction, the operand that is received at the start of the routine is the minuend. The sign of this number is reversed at the time that it is transferred from the data latch to the BS flag. This leads to the subtraction rather than the addition of this number and is the only difference between the execution of the FSB and FAD instructions. References to the sign of the new operand in the discussion which follows apply to the sign after it is loaded into the BS flag; that is, in the case of the subtraction instruction, they refer to the sign of the minuend after it has been reversed.

Unless the exponents of the two operands are equal, an alignment of the smaller operand is required; that is, the fraction must be shifted to the right the number of bit positions corresponding to the difference between the exponent values. If this difference is equal to or larger than 23, then the aligned fraction will

Microinstruction	Operation	Results	Comments
FLT0	Transfer from memory to data latch	1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 DL	Integer received is -54 in two's complement form
FLT1	DL(S) → BS DL(DB) → MQ(A) 15 → E	BS+ [H] MQ (47) [1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0...] E [1 0 0 0 1 1 1 1]	Negative sign is saved in BS Number is transferred to MQ where it is loaded into 16MSB positions E is set to 143 = 128 + 15
FLT2	MQ → B	B [1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0...]	Number is transferred from MQ to B
FLT3	$\overline{B47}$ → ADD B47 → SUB [B = 0] → ZERO	SUB+ [H] ZERO+ [L]	Subtraction function is selected because number is negative as indicated by high level in sign bit position B47 of B register ZERO flag remains reset because number is non-zero
FLT4	B $\xrightarrow{ADD}$ A B $\xrightarrow{SUB}$ A JZ	A [0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 ...]	With subtraction selected, number is converted to absolute form Go to FLT5 because ZERO+ is low.
FLT5	A → B SS JSGN	B [0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0...] AS+ [H] BS+ [?]	Number is copied into B Sign of number is transferred to AS Go to FLT6 because A47 = 0 (Magnitude overflow into sign bit position has not occurred.)
FLT6	A X 2^{NC} → A E - NC → E	A [0.1 1 0 1 1 0 0 0 .....0] E [1 0 0 0 0 1 1 0]	A is shifted left until most significant 1 reaches bit position A46 as required to provide normalized fraction E is decremented by 1 for each bit position shift of A Final value is E = 134 = 128 + 6 Final result is $-2^6 (2^{-1} + 2^{-2} + 2^{-4} + 2^{-5}) = -(2^5 + 2^4 + 2^2 + 2^1) = -54$
FLT7	A → B	A,B [0.1 1 0 1 1 0 0 .....]	Final fraction is copied into B

Figure 4-8. FLT Example (-54)

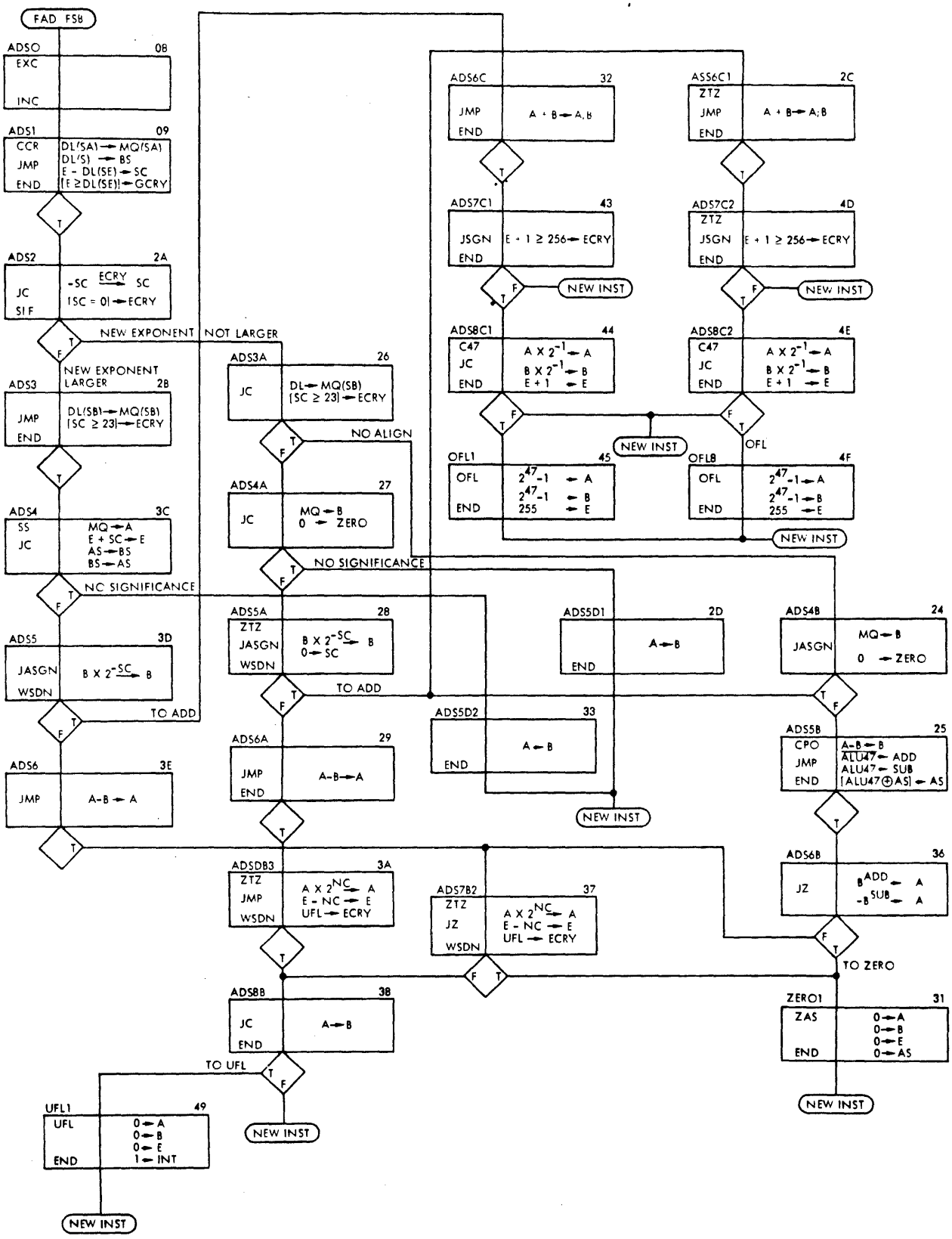


Figure 4-9. FAD/FSB Routine, Flowchart

have no significance. Thus, in this case, the routine is terminated without actually performing the alignment or a subsequent addition or subtraction. The result, in this case, is merely the larger operand. If the exponent difference is less than 23, the alignment is performed. The sum (A + B) is then formed if the signs of the two operands are the same or else the difference (A - B) is formed if the signs of the two operands are different. Because the smaller fraction is always located in the B register, the difference is always positive so that the result is always in absolute form. This simplifies subsequent handling of this result. The special case where the exponents of the two operands are equal is handled by a separate branch because in this case, if difference (A - B) must be formed, then a test must be made to determine whether the resultant difference is a negative number in complement form. If this is the case, then the number must be complemented to place it in absolute form before further processing can occur.

Once the difference (A - B) is known to be in absolute form, it can be normalized by shifting it to the left until a binary one appears in bit position 46. For each shift to the left, the result exponent, which is the exponent of the larger operand, must be decremented by 1. If an underflow occurs during normalization; that is, if the excess-128 exponent value is decremented from 0000 0000 to 1111 1111, then the routine enters an underflow branch in which it sets the result to zero and sets an interrupt flag.

In the case where the two operands are of the same sign, the sum (A + B) can overflow into the sign bit position. In this case, the result fraction must be shifted one place to the right and the result exponent must be incremented. This introduces the possibility of an exponent overflow. The routine checks for this possibility by evaluating whether the sum (E + 1) is equal to or larger than 256 (that is larger than allowable). If this is found to be the case and if the result fraction is found to have overflowed into the sign bit position, then the routine jumps into an overflow branch. This branch sets the result magnitude at the maximum value and sets the interrupt flag.

If the signs of the two operands are the same, then the sign bit initially residing in the AS flag can be used as the sign of the result so that no change in the status of AS is required.

If the signs of the two operands are different, the sign of the result is the sign of the larger operand. If the new operand is smaller, then the sign of the result is the sign that resides in the AS flag at the start of the routine. If the new operand is found to have a larger exponent than the operand resulting from the previous instruction, then the new operand fraction is placed

in the A register and the sign bits held in the BS and AS flags are swapped. Thus, after this swap, the AS flag holds the sign of the result. In the special case where the exponents of the two operands are equal, the initial status of AS must be reversed if a subtraction produces a negative difference (for in this case, the new operand has been found to be larger).

A description of the various branches of the FAD/FSB routine is presented in the paragraphs which follow. A numerical example is provided in figure 4-10.

Paragraph titles identify various branches in terms of the starting and ending microinstructions of the branch. Certain microinstructions are not part of any branch identified by a paragraph title. These microinstructions and the paragraphs in which they are discussed are listed below:

Microinstruction	Reference paragraph(s)
ADS5D2	4.5.4.2
ADS7B2	4.5.4.2, 4.5.4.5
ADS7B3	4.5.4.4
ADS8B	4.5.4.2, 4.5.4.4, 4.5.4.5
OFL1	4.5.4.3
UFL1	4.5.4.5
ZERO 1	4.5.4.5

4.5.4.1 Microinstructions ADSO through ADS2

Microinstruction ADSO performs the same functions as starting microinstruction LDSO of the FLD routine. (Refer to paragraph 4.5.1 for a description of LDSO.) Microinstruction ADS1 transfers the high fraction field (SA) of the first operand word from the data latch into the MQ register and transfers either the sign bit (S) or, if FSB is being executed, the inverse of the sign bit into the BS flag. It subtracts the new operand excess-128 code (obtained from the data latch) from the excess-128 code of the result exponent held in the E register and puts the difference in the shift counter. If this difference is positive or zero, it sets ECRY+ high to indicate that the new exponent is not larger than the result exponent. The CCR code in the FLAG field of the microinstruction causes the constant register (CR) to be cleared.

Microinstruction	Operation	Results	Comments
	Result of previous routine	E [1 0 0 0 0 0 1 1] A,B [0.1 1 1 0 0 .....0] AS+ [L]	Excess-128 code for exponent = +3 Fraction = 1/2 + 1/4 + 1/8 + 0 + ... + 0 = 7/8 Low AS+ indicates positive result Number is $+2^3$ X 7/8 = +7
ADS0	Transfer from memory to data latch	S Exponent Fraction (high) [0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0]	Sign bit (S) = 0. Therefore, number is positive. Excess-128 exponent code is 129. Exponent = +1. High fraction = 1/2 + 0 + ... + 0 Number = $+2^1$ X 1/2 = +1
ADS1	DL(SA)→MQ(SA) DL(S)→BS E - DL(SE)→SC E ≥ DL(SE)→ECRY	MQ [0.1 0 0 0 .....0] BS+ [L] SC [0 0 0 0 0 0 1 0] ECRY+ [H]	High fraction transferred to MQ Sign transferred to BS Difference between exponents put in SC ECRY set because difference is positive (New exponent is not larger)
ADS2	[SC = 0]→ECRY JC	ECRY+ [L]	ECRY reset because difference is not zero (that is, alignment is required) Go to DIS3A because ECRY set high in ADS1
ADS3A	DL→MQ(SA) [SC ≥ 24]→ECRY JC	MQ [0.1 0 0 0 ..... 0] ECRY+ [L]	Low fraction (in this case all zeros) transferred to section of MQ. Since all zeros are transferred to section, no change occurs. ECRY remains reset because difference is less than 23 Go to ADS4A because ECRY reset in ADS2
ADS4A	MQ→B 0→ZERO JC	B [0.1 0 0 0 ..... 0] ZERO+ [L]	New fraction transferred from MQ to B ZERO flag is reset because ALU = 1 function is selected Go to ADS5A because ECRY+ remained reset at end of ADS3A

Figure 4-10. FAD Example, +7 +1 = +8 (Sheet 1 of 2)

Microinstruction	Operation	Results	Comments
ADS5A	B X 2^{-SC}	SC: 0 0 0 0 0 0 0 1 — B: 0.0 1 0 ... 0 0 0 0 0 0 0 0 0 — 0.0 0 1 0	Contents of B register is shifted right each time shift count is decremented until shift count reaches 0. (This provides appropriate alignment of smaller operand fraction.) In this case, since SC is initially 2, two shifts occur.
	ZTZ	ZERO+ L	Change in status of ZERO flag is inhibited
	JASGN		Go to ADS6C1 because AS = BS
ADS6C1	A + B⟶A,B	A,B 1.0 0 0 0	0.1 1 1 0 0 ..... 0 +0.0 0 1 0 0 .....0 1.0 0 0 0 0 0
	ZTZ	ZERO+ L	Change in status of ZERO flag is inhibited
ADS7C2	[E + 1 ≥ 256]⟶ECRY	ECRY+ L	ECRY remains reset because (E + 1) = 132 < 256
	ZTZ	ZERO+ L	Change in status of ZERO flag is inhibited
	JSGN		Go to ADS8C2 because A47+ is high. (Fraction has overflowed into sign bit position.) Notice that jump is independent of any zero indication because ZERO flag was reset in ADS4A and has since been inhibited from changing status
ADS8C2	A X 2^{-1}⟶A B X 2^{-1}⟶B	A, B 0.1 0 0 0 0	Result fraction is normalized by shifting A and B one bit position to right
	E + 1⟶E	E 1 0 0 0 0 1 0 0	E is incremented when fraction is shifted to right as required to maintain same total value of result
			RESULT = 2^4 X 2^{-1} = 2^3 = 8
	JC, END		End because ECRY+ is low

Figure 4-10. FAD Example, +7 +1 = +8 (Sheet 2 of 2)

Microinstruction ADS2 performs twos complementation of the difference formed in the preceding microinstruction if that difference is a negative number in complement form (that is, if ECRY+ is low). The explicit function that is performed is the formation of the difference (0 - SC). This difference is loaded into SC only if ECRY+ was not set high during the preceding step. However, the subtraction operation is always performed because it is also used to determine whether SC is zero. If SC is zero no borrow occurs and ECRY+ is remains high. If SC is not zero, a borrow occurs and ECRY+ is reset to the low level. (This occurs as the microinstruction terminates and does not affect the conditional jump out of this microinstruction.)

The SIF code in the MEM field of microinstruction ADS2 suspends the termination of the microinstruction until the memory cycle during which the second operand word is received has been completed. When the microinstruction is terminated, the SIF code causes control to be returned to the central processor so that the next instruction fetch can be initiated. The JC code in the JCOND field causes a jump to microinstruction ADS3A if ECRY+ was set high during microinstruction ADS1; that is if the new exponent is not larger than the old exponent.

4.5.4.2 Microinstructions ADS3 through ADS6

This branch is entered when the new exponent is larger than the old exponent. Microinstruction ADS3 loads the high fraction field of the second operand word from the data latch into the MQ register and sets ECRY+ high if the exponent difference held in the shift counter is 23 or more. The specific operation that is used to test the shift counter magnitude is to add the twos complement of 23 to the contents of the shift counter. If the exponent difference is at least 23, this operation generates a carry which sets ECRY+ high.

Microinstruction ADS4 transfers the fraction of the new operand into the A register. (Since the result of the previous instruction is smaller than the new operand, the copy of the fraction portion of this result that resides in the B register is retained.) The SS code in the FLAG field causes the sign bits held in the BS and AS flags to be swapped. This puts the sign bit associated with the larger number (the new operand) in the AS flag as required. The exponent difference from the shift counter is added to the excess-128 exponent code held in the exponent register and the sum is placed in the exponent register. This effectively replaces the smaller exponent of the previous result with the larger exponent of the new operand. The JC code in the JCOND field causes a jump out of the branch to microinstruction ADS5D2 if ECRY+ was set high during the preceding microinstruction. ADS5D2 terminates the routine after copying A into B.

This path is followed when at least 23 right shifts of the smaller operand fraction would be required to align it with the larger operand. In this case, there could be no significant ones in the aligned smaller operand and thus addition or subtraction need not be performed.

Microinstruction ADS5 performs the right-shifting of the smaller operand fraction in the B register. The WSDN code in the MEM field inhibits the control store clock until the shift count reaches zero. The shift counter is counted down for each shift right. Thus, the duration of ADS5 is the number of clock periods required to complete the alignment. The JASGN code in the JCOND field produces a jump if the sign bits of the two operands are equal. This causes a branch to microinstruction ADS6C1 (where addition is performed) if the signs are the same or a continuation to microinstruction ADS6 (where subtraction is performed) if the signs are different.

The difference (A - B) that is formed and stored in A during microinstruction ADS6 is always a positive number in absolute form and is either normalized or can be normalized by left shifting. The required left-shifting (if any) is performed in microinstruction ADS7B2 and then the routine advances to microinstruction ADS8B during which the aligned fraction result is copied into the B register. If an underflow is detected during microinstruction ADS7B2, then ECRY+ is set high as microinstruction ADS8B is entered. In this case, the JC code in the JCOND field of microinstruction ADS8B causes a jump to microinstruction UFL1 which sets the fraction and exponent result values to zero and sets the interrupt flag. If an underflow does not occur, then ADS8B is the final microinstruction of the routine.

4.5.4.3 Microinstructions ADS6C through ADS8C1

This branch is entered from microinstruction ADS5 after the new operand fraction has been aligned in the B register and when the signs of the two operands are the same. Microinstruction ADS6C forms sum (A + B) and places it in the A and B registers.

Microinstruction ADS7C1 is mainly a decision instruction which is the end of the routine if the sum formed during the preceding microinstruction is normalized. However, the JSGN code in the JCOND field causes a jump to microinstruction ADS8C1 if the sum contains an overflow into the sign bit position. In this case the A and B registers must be shifted to the right one bit position and the exponent must be incremented by 1. In order to determine whether this will cause an exponent overflow, the summation (E + 1) is performed in the exponent adder during microinstruction ADS7C1. If the exponent is already at the maximum allowable value, this summation produces a carry which sets ECRY+ to the high level, as the jump to ADS8C1 occurs.

Microinstruction ADS8C1 performs the required right shift of the A and B registers and increments the exponent by 1. The C47 code in the FLAG field causes a one to be loaded into bit position 47 of the constants register (CR) in preparation for the conditional branch to microinstruction OFL1. If ECRY+ was not set high during ADS7C1, then the execution of ADS8C1 produces an in-range result and the routine terminates with this instruction. However, the JC code in the JCOND field causes a jump to OFL1 if ECRY+ was set high during ADS7C1.

Microinstruction OFL1 sets the maximum values (all ones) into the A and B registers and into the E register and sets the interrupt flag. The all-ones value is formed for entry into the A and B registers by forming the difference (CR - 1), where CR contains a one bit in position 47 and zero bits in all other positions.

4.5.4.4 Microinstructions ADS3A through ADS6A

This branch, which is entered from microinstruction ADS2 if the new exponent is not larger is very similar to the ADS3 through ADS6 branch which is entered if the new exponent is larger. Thus, only the differences are described. The JC code in the JCOND field of instruction ADS3A causes a jump out of the branch to microinstruction ADS4B if the exponents of the two operands are equal. (If this is the case, ECRY+ is high when the branch is entered as required to cause the jump.)

Microinstruction ADS4A transfers the new operand fraction from the MQ register to the B register. In this case, the new operand is smaller and thus its fraction is placed in the B register for alignment. Instruction ADS4A also sets the ZERO+ flag low. Since the ZERO+ flag responds to the output of the ALU, this is accomplished by selecting the ALU function, ALU = 1.

The ZTZ code in the FLAG field of microinstruction ADS5A holds the ZERO+ flag low. This is in preparation for a possible jump to the branch starting with microinstruction ADS6C1 which requires that ZERO+ be low in order to obtain a correct branch decision in microinstruction ADS7C2. The jump from ADS5A to ADS6C1 occurs if the signs of the two operands are the same. If the path through ADS6A is followed, then the normalization shifting (if required) occurs during microinstruction ADS7B3. From this microinstruction, the routine advances to ADS8B.

4.5.4.5 Microinstructions ADS4B through ADS6B

This branch is entered when the two operand exponents are equal. In this case no alignment is required. Microinstruction ADS4B transfers the new operand fraction from the MQ register to the

B register. It also resets the ZERO+ flag to the low level. This prepares for a possible jump to the branch beginning with microinstruction ADS6C1. (The status of the ZERO+ flag is not allowed to change once this branch is entered in order to avoid making the wrong branch decision in step ADS7C2 in the case where both operands are zero. The JSGN code which controls the jump decision causes a jump on either A47+ high or the ZERO+ flag high.) The JASGN code in the JCOND field of ADS4B causes a jump out of the branch to microinstruction ADS6C1 if the signs of the two operands are the same. If the signs are different, microinstruction ADS5B is performed. This microinstruction forms the difference (A - B) and stores it in B. It sets the ADD flag if the difference is positive or zero or sets the SUB flag if the difference is a negative number in twos complement form. (The sign of the difference is determined by evaluating the status of sign bit 47 from the ALU.) The CPO code in the FLAG field enables the setting of ADD or SUB and also causes the status of result sign flag AS to be reversed if (A - B) is negative. (The difference was formed because the signs of the two operands were different. The fact that the difference is negative indicates that the operand in B is the larger operand and therefore the sign of the result is the same as the sign of B).

During microinstruction ADS6B, the difference held in the B register is complemented and stored in A if SUB is high (that is, if the difference is negative). If ADD is high, the difference from the B register is supplied to the A register without being complemented. Thus, at the end of the microinstruction, the difference, in absolute form resides in the A register. The explicit ALU function that is performed is either (CR + B) or (CR - B). However, since the constant register (CR) was cleared in ADS1, the effective operation is ALU = B or ALU = - B.

If the difference computed in microinstruction ADS5B is zero, the ZERO+ flag is set high as the advance from ADS5B to ADS6B occurs. In this case, the JZ code in the JCOND field of microinstruction DS6B causes a jump to microinstruction ZERO 1. This microinstruction sets the fraction (in A and B) and the exponent (in E) to zero and also sets AS+ to the low level as required to produce the correct zero format. If the difference is not zero, the routine advances from ADS6B to ADS7B2. This microinstruction performs left shifting (if required) to normalize the fraction. The routine then terminates with microinstruction ADS8B if the result is in range or jumps to UFL1 if an underflow occurs in microinstruction ADS7B2.

4.5.4.6 Microinstructions ADS6C1 through ADS8C2

This branch is entered from either one of two other branches in the event that the signs of the two operands are the same. In

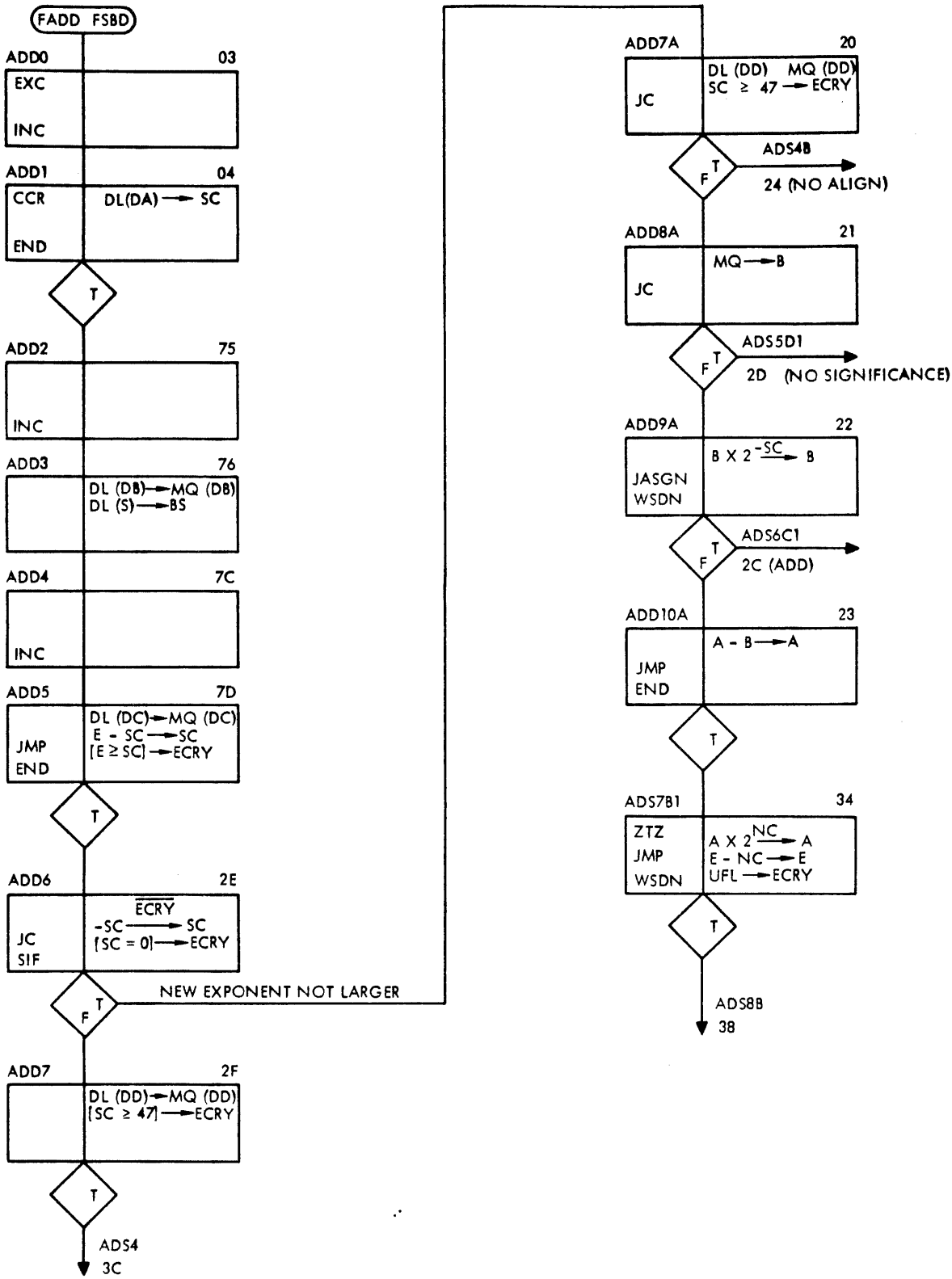
the case where the exponents of the two operands are equal, entry is from microinstruction ADS4B. In the case where the new exponent is smaller than the exponent of the previous result, entry is from microinstruction ADS5A. The branch is similar to branch ADS6C through ADS8C1 which is traversed when the signs of the operands are the same and the new exponent is larger. The only difference is that in microinstructions ADS6C1 and ADS7C2 of this branch, the ZTZ code in the FLAG field inhibits the setting of the ZERO+ flag to the high level. This is necessary in order to inhibit the setting of the ZERO+ flag in the case where both operands are zero. (A high ZERO+ signal level would cause a jump from ADS7C2 in response to the JSGN code.)

4.5.5 FADD/FSBD Routine

As illustrated by the flowchart of figure 4-11, the FADD/FSBD routine performs certain functions and then jumps into an appropriate entry point of the FAD/FSB routine. The most basic difference between double precision and single precision operations is that four operand words must be obtained from memory and that three fraction fields must be assembled in the MQ register. Another operation that is handled separately for the double precision operation is the shift counter test to determine whether the smaller operand is too small to have any possible significance. In the double precision operation, this is the case if the difference between the exponent values is 47 or more.

Microinstruction ADD0 is identical to the first instruction of the FAD/FSB routine. Its purpose is to initiate the memory access request required to obtain the second operand word and to prepare for the transfer of the exponent field from the data latch to the shift counter during the next microinstruction. Microinstruction ADD1 performs the exponent transfer and resets the constant register (CR). Microinstruction ADD2 initiates the memory access request required to obtain the third operand word. Microinstruction ADD3 transfers the high fraction field of the second operand word from the data latch to the MQ register and transfers the sign bit (S) from the data latch to the BS flag. As in the case of the FAD/FSB routine, the sign bit is inverted if the subtraction instruction (in this case FSBD) is being executed. Microinstruction ADD4 initiates the memory access required to obtain the fourth operand word.

Microinstruction ADD5 transfers the middle fraction field of the third operand word from the data latch to the MQ register. It also forms the difference between the new and old exponents by subtracting the contents of the shift counter (new exponent) from the contents of the exponent register. If this difference is positive or zero, the ECRY+ flag is set high.



VTI2-444

Figure 4-11. FADD/FSBD Routine Flowchart

Microinstruction ADD6 complements the exponent difference if ECRY+ was not set high during the previous microinstruction; that is, if the difference is a negative number in twos complement format. The SIF code in ADD6 suspends the termination of the microinstruction until the memory cycle during which the fourth operand word is received is completed. When the microinstruction is terminated, the SIF code causes control to be returned to the processor so that the next instruction fetch can be initiated. The JC code causes a jump to ADD7A if ECRY+ was set high; that is if the new exponent is not larger. As the routine exists from the step, ECRY+ remains high if the difference is zero or is reset if the difference is non-zero.

If the new exponent is larger, the routine advances from ADD6 into ADD7. In this microinstruction, the low fraction field of the fourth operand word is transferred from the data latch to the MQ register and ECRY+ is set high if the exponent difference in the shift counter is equal to or larger than 47. A jump into microinstruction ADS4 of the FAD/FSB routine is then executed.

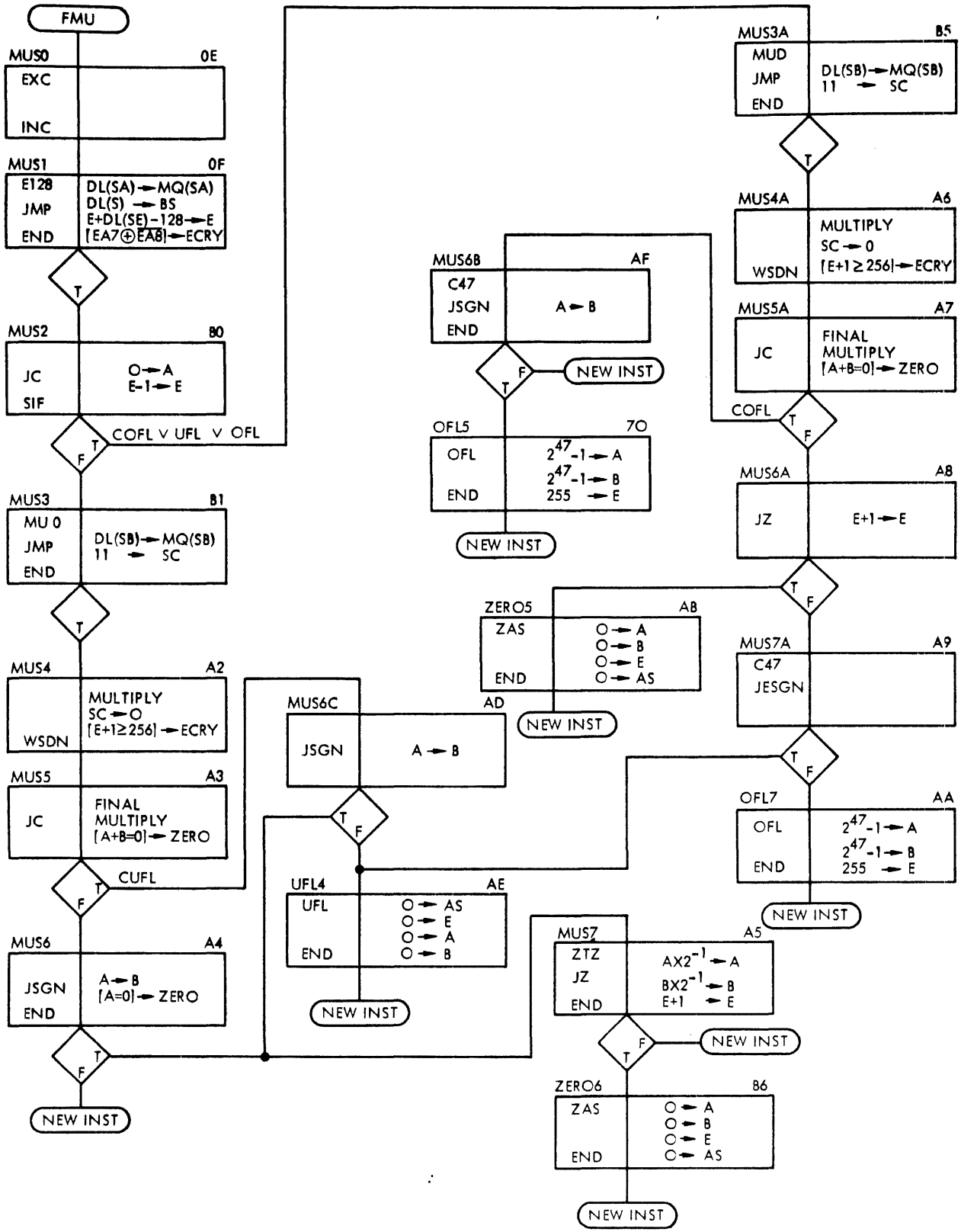
Microinstruction ADD7A, which is entered if the new exponent is not larger, performs the same functions as ADD7 and, in addition, provides a jump to microinstruction ADS4B of the FAD/FSB routine in the event that the exponent difference is zero.

If the new exponent is smaller, the routine advances from ADD7A to ADD8A. This microinstruction transfers the new fraction from the MQ register to the B register and provides a jump to microinstruction ADS5D1 if ECRY+ was set in ADD7A; that is, if the smaller operand is too small to have any possible significance.

If ECRY+ was not set high in ADD7A, the routine advances to microinstruction ADD9A which aligns the smaller (new) operand fraction in the B register. The JASGN code causes a jump into microinstruction ADS6C1 of the FAD/FSB routine if the signs of the two operands are the same. If they are different, this routine continues through microinstructions ADD10A and ADS7B1. ADD10A forms the fraction difference (A - B) and stores this in the A register. ADS7B1 performs normalization shifting of this difference (if and as required). If an exponent underflow occurs during the normalization, ECRY+ is set high. A jump into microinstruction ADS8B of the FAD/FSB routine is then executed.

4.5.6 FMU Routine

Multiplication (figure 4-12) involves the processing of the signs, exponents, and fractions of the two operands. Processing of signs involves setting AS+ low (to represent a positive result) if the signs of the two operands are the same or setting of AS+ high if the signs of the two operands are different. A provisional result exponent is computed by adding the operand exponents and subtracting 128.



VTI2-448

Figure 4-12. FMU Routine Flowchart

(The subtraction of 128 preserves the excess-128 code.) Fractions must be multiplied and the result normalized. After the multiplication of the fractions is complete, the result fraction is either in the normalized position or else has its most significant non-zero bit in the sign position (in which case it must be shifted to the right one position).

The provisional result exponent is decremented to compensate for an inherent left shift of the fraction which occurs during the multiplication process. For example, the multiplication, 1/2 X 1/2 (0.1000.. X 0.1000...) which should produce the result, 1/4 (0.0100...), actually produces the result, 1/2 (0.1000...). While the need for decrementing of the provisional result exponent is conditional upon the status of the result fraction, the routine invariably decrements it (in step MUS1). If it is subsequently found that the result fraction must be shifted right, the routine then increments the provisional result exponent at that time so as to cancel the earlier decrementing operation.

A provisional result exponent value of 128 represents a conditional overflow value (COFL). While 128 itself is outside of the allowable range, the value of 127 which will be the final exponent value if the result fraction does not require a shift to the right, is within range. Any provisional result exponent value greater than 128 is definitely an overflow value (OFL). A value of -129 or more negative is definitely an underlow value (UFL). In summary, any provisional result exponent value which is outside of the normal range (+127 through -128) indicates the possibility of an overflow or underflow. There is also one in-range provisional result exponent value which represents a conditional underflow value (CUFL). This is the value -128 which will produce the out-of-range final result exponent value -129 in the event that the result fraction does not require shifting to the right.

Descriptions of the various branches of the FMU routine are presented in the paragraphs which follow. A numerical example is provided in figures 4-13 and 4-14.

4.5.6.1 Microinstructions MUSO through MUS2

Microinstruction MUSO performs the same functions as starting microinstruction LDSO of the FLD routine. (Refer to paragraph 4.5.1 for a description of LDSO.)

Microinstruction MUS1 transfers the high fraction field of the first operand word (SA) from the data latch to the MQ register and transfers the sign bit (S) from the data latch into the BS flag. It also adds the exponent value residing in the exponent register (E) to the exponent value obtained from the data latch

Microinstruction	Operation	Results	Comments
	Results of previous operation	A,B [0 . 1 0 0 .... 0] E 1 0 0 0 0 0 0 0 AS+ [L]	1/2 + 0 + 0 + ... + 0 = 1/2 128 + 0 + ... + 0 = 128; Exponent = 0 Sign is positive Number is 2^{E-128} X FRACTION = 2^{0} X 1/2 = 1/2
MUS0	Transfer from memory to data latch	S Exponent Fraction (High) 1 [1 0 1 1 1 1 1 1 0 0 1 0 1 1 1 1] DL	Sign bit = 1. Therefore, number is negative. Thus, exponent and high fraction fields are in one's complement form Exponent = +1; High Fraction = 1/2 + 1/8 = 5/8
MUS1	DL(SA)⟶MQ(SA) DL(S)⟶BS E + DL(SE) - 128⟶E [EA ⊕ $\overline{EA8}$]⟶ECRY	MQ [0 . 1 0 1 0 0 .... 0] BS+ [H] E [1 0 0 0 0 0 0 1] ECRY+ [L]	Sign and high fraction are transferred to MQ. One's complementation performed in response to negative sign bit in DL15 converts data to absolute form. Sign bit is saved in BS. 128 + 129 - 128 = 129 Exponent is in range so ECRY remains reset.
MUS2	0⟶A Transfer from memory to data latch E - 1⟶E JC	A [0 . 0 0 0 ..... 0] DL [0 0 0 .....0] E [1 0 0 0 0 0 0 0]	A is cleared in preparation for product accumulation. 2nd operand word (in this case all 0s) is received by FPP E is decremented by 1 Go to MUS3 because ECRY+ is low.
MUS 3	DL(SB)⟶MQ(SB) 11⟶SC	MQ [0 . 1 0 1 0 0 . . . . 0] SC [0 0 0 0 1 1 0 1]	Low fraction transferred from DL to MQ. (Since low fraction is zero in this example, there is no change in status of MQ.) Shift counter is set to 11 to prepare for MUS4

Figure 4-13. FMU Example (+1/2)(-1 1/4) = 5/8 (Sheet 1 of 2)

Microinstruction	Operation	Results	Comments
MUS3 (Continued)	MU0	AS+ [H] (See figure 4-14.)	AS+ is set high to indicate negative result because AS ≠ BS First pair of multiplier bits is evaluated. (See figure 4-14.)
MUS4	[(E + 1) 256]⟶ECRY Multiply SC⟶0	ECRY+ [L] (See figure 4-14.)	ECRY remains reset because E + 1 = 129 < 256 Eleven iterations are performed. For each iteration; SUB, ADD, and CARRY flag status is updated in accordance with multiplier bit pair and CARRY flag status. Contents of B is added to or subtracted from previous partial product if ADD or SUB flag respectively is in set status and partial product is shifted right two positions. Also, B register is shifted left or right, if appropriate, to provide 2M or M (where M = multiplicand)
MUS5	Final Multiply [A + B = 0]⟶ZERO JC	A [0 . 1 0 1 0 ..... 0] ZERO+ [L]	(See figure 4-14.) Product fraction is non-zero. Go to MUS6 because ECRY+ is low. (E + 1 would be within range.)
MUS6	A⟶B JSGN END	A,B [0 . 1 0 1 0 .... 0]	Fraction result is copied into B Routine ends because result fraction is non-zero and has not overflowed into sign bit position.

Note: Final exponent value is zero (from MUS2); final fraction value is 1/2 + 0 + 1/8 +..+0 = 5/8 (from MUS5). Final sign value is minus (from MUS3).

Number is: -2^0 X 5/8 = -5/8

Figure 4-13. FMU Example (+1/2)(-1 1/4) = 5/8 (Sheet 2 of 2)

MULTIPLIER: 0.1 01 00 00 00 00 00 00 00 00 00 00

MULTIPLICAND: 0.1000 0

MICROINST	SHIFT COUNT (DEC)	B REGISTER	MULTIPLIER BITS EVALUATED	SUB	ADD	CARRY	A REGISTER
		47 46 45 44					47 46 45 44 43 . . . .
MUS3	11	0 1 0 0	00	0	0	0	0 0 0 0 0 . . . .
MUS4	10	1 0 0 0	00	0	0	0	0 0 0 0 0 . . . .
	9	1 0 0 0	00	0	0	0	0 0 0 0 0 . . . .
	8	1 0 0 0	00	0	0	0	0 0 0 0 0 . . . .
	7	1 0 0 0	00	0	0	0	0 0 0 0 0 . . . .
	6	1 0 0 0	00	0	0	0	0 0 0 0 0 . . . .
	5	1 0 0 0	00	0	0	0	0 0 0 0 0 . . . .
	4	1 0 0 0	00	0	0	0	0 0 0 0 0 . . . .
	3	1 0 0 0	00	0	0	0	0 0 0 0 0 . . . .
	2	1 0 0 0	00	0	0	0	0 0 0 0 0 . . . .
	1	0 1 0 0	01	0	1	0	0 0 0 0 0 . . . .
	0	0 1 0 0	01	0	1	0	0 0 0 1 0 . . . .
MUS5							0 1 0 1 0 . . . .

VTI1-3203

Figure 4-14. Example of Fraction Multiplication Procedure (1/2 X 5/8)

(SE) and places this provisional result exponent in the exponent register. The E128 code in the FLAG field of the microinstruction effectively introduces a component of -128 into the exponent summation. This is necessary in order to maintain the excess-128 code. If an out of range exponent sum (larger than +127 or more negative than -128) is obtained, the ECRY+ flag is set high. This prepares for a jump from the next microinstruction.

Microinstruction MUS2 clears the A register in preparation for using it to accumulate the produce of the operand fractions and decrements the provisional exponent result computed in the preceding microinstruction. This decremented exponent value becomes the final exponent result value unless a subsequent right shift of the fraction result is required or the fraction result is zero. The SIF code in the MEM field suspends the advance to the next microinstruction until the end of the memory cycle during which the second operand word is received from memory and loaded into the FPP data latch. As the advance to the next microinstruction occurs, SIF causes control to be returned to the central processor so that the next instruction fetch can be initiated. The JC code in the JCOND field causes a jump to microinstruction MUS3 if ECRY+ was set high during MUS1. Thus, this path is followed if an out-of-range provisional exponent

result value was obtained. If the provisional exponent result value was within range, the routine advances to microinstruction MUS3.

4.5.6.2 Microinstructions MUS3 through MUS7 and ZERO6

This branch is executed when the provisional exponent result obtained during microinstruction MUS1 is within range. Microinstruction MUS3 sets the AS+ flag to the low level if the signs of the two operands are the same or sets the AS+ flag to the high level if the signs are different. This action occurs in response to the MUO code in the FLAG field of the microinstruction.

Microinstructions MUS3 through MUS5 implement the fraction multiplication procedure. This is essentially an add and shift routine in which multiplier bits are processed in pairs. Each multiplier pair has a value of 0, 1, 2, or 3 times 2^n, where the value of 2^n is determined by the position of the pair in the total multiplier fraction. The basic scheme is to add a value of 0, 1, 2, or 3 times the multiplicand to the previous partial product in accordance with the value of the particular multiplier bit pair that is being processed. After each step, the intermediate product is shifted two bit positions to the right. This imparts the appropriate weight (2^n) to each successive component that is added to the previous partial product. In actual implementation, the basic scheme is modified so that only two explicit component values are required, M and 2M, where M = multiplicand. Using these two values, 0, 1, or 2 times M can be added to the partial product directly. To add 3M, M is subtracted during the current step and a carry is propagated into the next step. Since the significance of each succeeding bit pair value is $2^2 = 4$ times that of the preceding bit pair value, a carry into the next step is equivalent to the addition of 4M during the current step. Thus, the addition of 3M is implemented by adding 4M - M. When a carry from the preceding step is allowed, the number of possible values is increased to include the additional value, 4. However, the value 4 can be added simply by generating a carry into the next order.

The required values of M and 2M are obtained by controlling the position of the multiplicand in the B register. When 2M is required, the contents of the B register are shifted to the left one position. When M is again required, the contents of the B register are shifted to the right.

Each step consists of two sub-steps; namely the evaluation of the two multiplier bits and the carry bit and the addition or subtraction of the appropriate component to the previous partial product. Each sub step requires one clock period. However, once the process has been started, the addition or subtraction associated with the current step can be performed during the same clock period as the evaluation associated with the next step. Thus, the 12 pairs of multiplier bits can be processed in 13

clock periods. During the first of these clock periods, which is provided by microinstruction MUS3, only an evaluation occurs. During the next 11 clock periods, which are provided by microinstruction MUS4, both an evaluation and the addition or subtraction of a component occur. During the final clock period, which is provided by microinstruction MUS5, only an addition occurs.

The MUO code in the FLAG field of microinstruction MUS3 conditions the arithmetic control for the evaluation of the first pair of multiplier bits. This microinstruction moves the second word of the multiplier (which contains the first two bits to be evaluated) from the data latch to the MQ register. Thus, these bits must be monitored, for purposes of the first evaluation, as they reside in the data latch. At the same time, a second pair of multiplier bits is loaded from the data latch into a multiplier bit pair buffer. This buffer provides the source of bits during the evaluations of microinstruction MUS4. At the end of each clock period during microinstruction MUS4, the contents of the buffer is updated from the MQ register. During each clock period of MUS4 (except the first), the contents of the MQ register is shifted two bit positions to the right. The first bit of the first bit pair is zero. The second bit of the first bit pair is obtained from bit position 00 of the data latch. The second bit pair, which is loaded into the multiplier bit pair buffer during microinstruction MUS4, is obtained from bit positions 01 and 02 of the data latch. Thereafter, each successive bit pair is loaded into the multiplier bit pair buffer from bit positions 28 and 29 of the MQ register. (The multiplier bit pair which initially resides in bit positions 28 and 29 after the parallel data transfer from the data latch is loaded into the buffer at the end of the first clock period of MUS4. By the end of the second clock period, the bits initially residing in positions 30 and 31 have been shifted into positions 28 and 29 and are now loaded into the buffer. The shifting of the contents of the MQ register and the loading of the buffer continues in this manner for the remainder of MUS4.

Another function performed by microinstruction MUS3 is the setting of the shift counter (SC) to 11. This determines the number of clock periods in MUS4. At each clock time, SC is counted down and an advance to MUS5 occurs as SC reaches the count of 0. The MUO code in the FLAG field of microinstruction MUS3 causes the MU1 flag (not shown in the flow chart) to set as microinstruction MUS4 is entered. It is this flag which conditions the arithmetic control to perform evaluations, additions or subtractions, right shifting of each partial product, and right shifting of the MQ register. The WSDN code in the MEM field of MUS4 inhibits the control store clock until the shift count reaches 1 as required to continue the microinstruction for 11 clock periods.

A subsidiary function performed by MUS4 is the formation, in the exponent loop, of the sum (E + 1). If the excess-128 exponent code is 255, then this summation produces a carry out of the highest order of the ALU which sets the ECRY+ flag to the high level. This indicates the conditional underflow (CUFL) value. (In this case, the initial provisional exponent result was 000, representing an exponent value of -128. This was decremented in MUS2 to become 255 so that the incrementation provided in this step produces an overflow.)

In microinstruction MUS5, the final component is added to the partial product to form the final product. The JC code in the JCOND field causes a jump to microinstruction MUS6C for the conditional underflow case. If the final product formed in MUS5 is zero, the ZERO+ flag is set high. If the conditional underflow case is not sensed, the routine advances from MUS5 to MUS6. Microinstruction MUS6 copies the final product from the A register into the B register. If this final product is normalized, this is the final microinstruction of the routine. The JSGN code in the JCOND field causes a jump to microinstruction MUS7 if the final product has overflowed into the sign bit position or if the ZERO+ flag is high.

Microinstruction MUS7 shifts the contents of the A and B registers one position to the right and increments the exponent as required to normalize the result. Unless the ZERO+ flag is high, this is the final microinstruction of the routine. The JZ code in the JCOND field causes a jump to microinstruction ZERO6 if ZERO+ is high. Microinstruction ZERO6 sets the fraction fields in the A and B registers and the exponent field in the E register to zero. It also sets the sign bit in the AS flag to zero. (This last action occurs in response to the ZAS code in the FLAG field of the microinstruction.) This is the floating point number zero format. This microinstruction, when performed, is the final microinstruction of the routine.

4.5.6.3 Microinstructions MUS3A through MUS7A and OFL7; MUS6B and OFL5; and ZEROS

This branch is entered from MUS2 when the provisional exponent result is out of range. MUS3A through MUS5A are identical to MUS3 through MUS5, described in paragraph 4.5.6.2. However, they lead to different terminations as appropriate to the out-of-range exponent condition. Whereas the setting of ECRY+ to the high level in MUS4 indicates a conditional underflow condition, the setting of ECRY+ in MUS4A indicates a conditional overflow (COFL) condition. (In this case, the provisional exponent excess-128 code of 0000 0000 represents the overflow exponent value of +128. This causes a jump from MUS5A to MUS6B in response to the JC code in the JCOND field of MUS5A. MUS6B copies the result fraction from the A register to the B register. If the result

fraction is in the normalized condition, this is the final microinstruction of the routine. (Although the provisional exponent result value was out of range, this was decremented in MUS2 so that the exponent value in the E register now represents the in-range value of +127. If the fraction result does not require a right shift and is not zero, this becomes the final exponent value.)

The JSGN code in the JCOND field of MUS6B causes a jump to OFL5 if the result fraction requires right shifting. Microinstruction OFL5 establishes the overflow format. The A and B registers and the exponent register are set to all ones. This represents the largest possible magnitude. The sign bit is not affected.

If the COFL status is not sensed, the routine advances from MUS5A to MUS6A. The exponent is now known to be either an underflow or an overflow value. Thus, the final result must be an underflow or overflow unless the fraction result is zero. Microinstruction MUS6A forms the sun (E + 1) and stores this in the E register as required to return the exponent result to the initial value computed in MUS1. The JZ code in the JCOND field of this microinstruction causes a jump to microinstruction ZERO5 if the fraction result is zero. ZERO5 establishes the floating point zero format. (This is identical to microinstruction ZERO6.)

If the fraction result is not zero, the routine continues from MUS6A to MUS7A. The purpose of MUS7A is to distinguish between an overflow and an underflow. This determination is controlled by the status of the most-significant bit of the excess-128 exponent code. If this bit is a one, then the exponent code represents a negative out-of-range (underflow) value of -129 or more negative. If this bit is zero, then the exponent code represents a positive out-of-range (overflow) value of greater than +128. The JESGN code in the JCOND field of MUS7A causes a jump to UFL4 if the bit is a one or to OFL7 if the bit is a zero. UFL4 establishes the underflow format of zero in all fields including the sign bit field. OFL7 establishes the overflow format of the largest possible magnitude and does not change the sign.

4.5.6.4 Conditional Underflow Microinstruction MUS6C and Microinstruction UFL4

This microinstruction is entered from microinstruction MUS5 when the conditional underflow value of -128 is sensed in microinstruction MUS5. This value was decremented in MUS2 to produce the underflow value -129. However, if the fraction result has overflowed into the sign bit position, then the exponent will be incremented and so will be returned to the in-range value. The result will also be in range if the fraction result is zero. The JSGN code in the JCOND field of MUS6C causes a jump to MUS7 if

the fraction result has overflowed into the sign bit position or if the fraction result is zero. This microinstruction is described in paragraph 4.5.6.3. If the result fraction is found to be in the normalized status, the routine advances to microinstruction UFL4 which establishes the underflow format.

4.5.7 FMUD Routine

As illustrated in figure 4-15, the FMUD routine performs functions associated with the transfer of a double precision number and then, after performing a multiplication set-up instruction which sets the shift counter to 23, jumps into an appropriate microinstruction of the FMU routine. In the case of the double-precision number, there are four operand words to be obtained from memory rather than two. The routine must initiate the memory cycle requests required to obtain three of these words and it must assemble the fraction fields from three operand words in the MQ register to form the multiplier fraction.

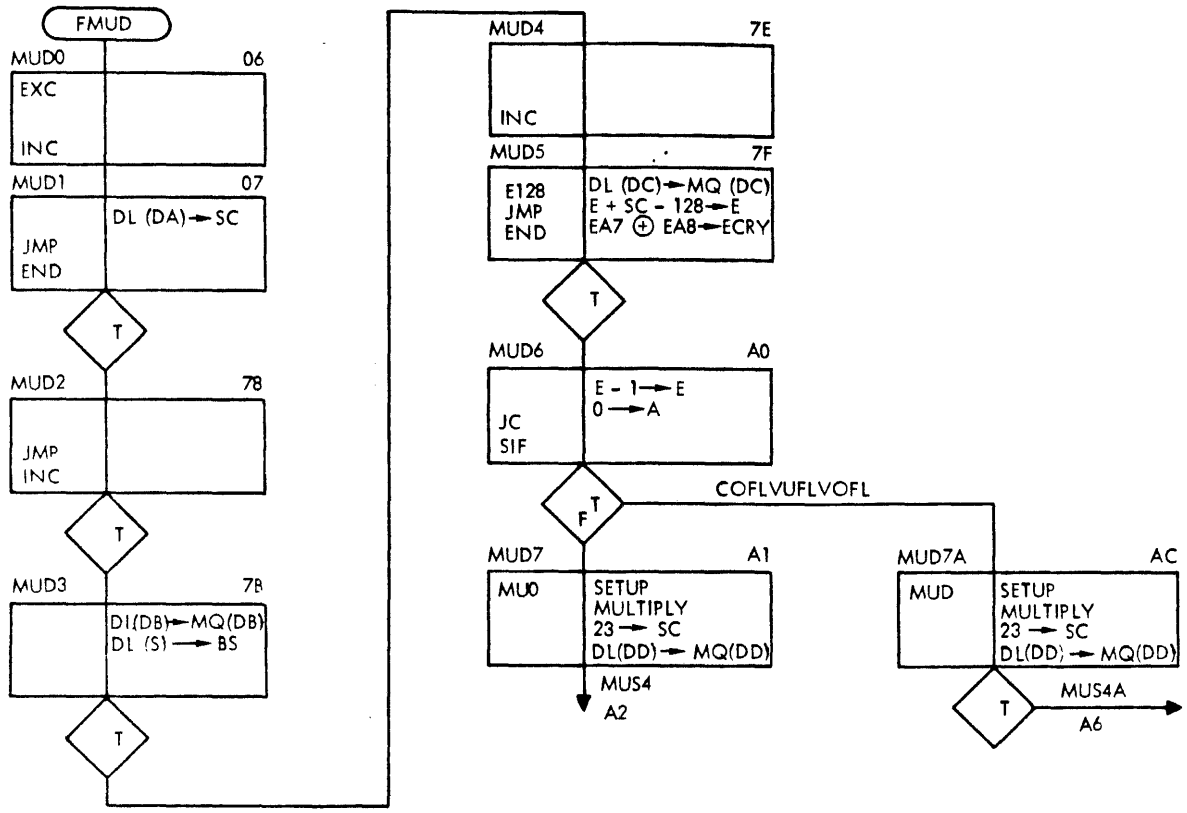
Microinstruction MUDO is identical to MUSO. Microinstruction MUD1 transfers the exponent field from the data latch to the shift counter where it is held until microinstruction MUD5 adds it to the exponent code held in the exponent register.

Microinstruction MUD2 waits for the completion of the memory cycle during which the second operand word is transferred from memory to the FPP data latch. It then increments the memory address to point to the location containing the third operand word and initiates the memory request required to obtain this word.

Microinstruction MUD3 transfers the high fraction field (DB) of the second operand word from the data latch to the MQ register and transfers the sign bit field of the second operand word from the data latch to the BS flag.

Microinstruction MUD4 waits for the completion of the memory cycle during which the third operand word is transferred from the memory to the FPP data latch. It then increments the memory address to point to the location containing the fourth operand word.

Microinstruction MUD5 transfers the middle fraction field (DC) of the third operand word from the data latch to the MQ register. It also adds the exponents of the two operands and sets ECRY+ high if an out-of-range provisional result exponent is obtained. (These are the same exponent-loop functions that are performed in microinstruction MUS1 of the FMU procedure except that the new exponent code now resides in the shift counter rather than the data latch.)

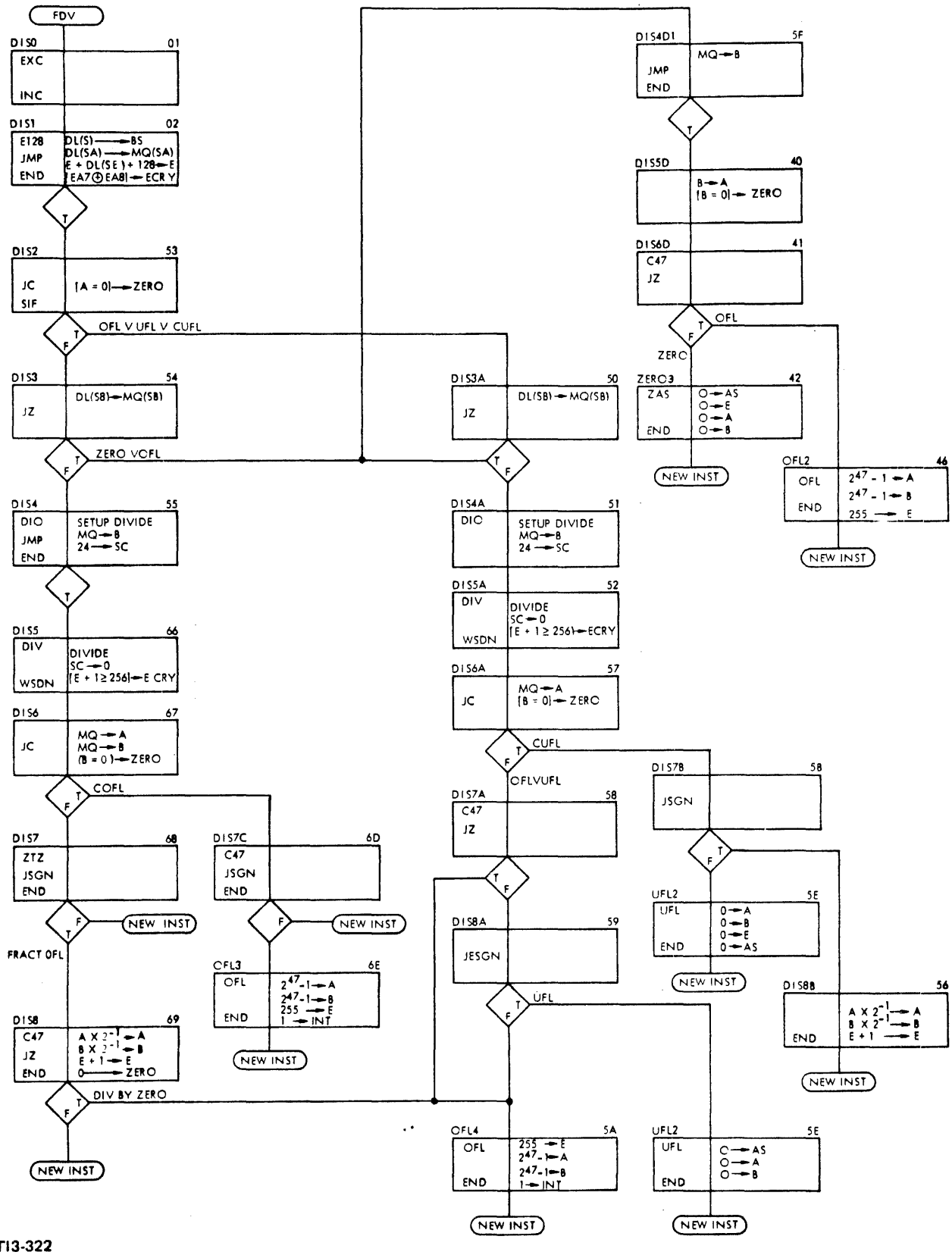


VTI1-3187

Figure 4-15. FMUD Routine Flowchart

Microinstruction MUD6 performs the same function as microinstruction MUS2 of the FMU routine. Microinstructions MUD7 and MUD7A are similar to microinstructions MUS3 and MUS3A. The differences are that the MUD7 and MUD7A microinstructions transfer the low fraction field of the fourth operand word from the data latch to the MQ register and that these two microinstructions set the shift counter to 23 rather than to 11. The higher shift count is required because in the case of a double-precision multiplier there are 24 pairs of multiplier bits to be processed rather than 12.

From MUD7 or MUD7A, the routine jumps into MUS4 or MUS4A respectively of the FMU routine. Double precision multiplication of functions in steps MUS4 and MUS4A is identical to single precision multiplication except for the number of iterations as noted above and the source of the bit pairs. In double precision multiplication, the first bit pair is obtained from bit positions 00 and 01 of the data latch. The second bit pair is obtained from bit positions 02 and 03 of the data latch. The third and remaining bit pairs are obtained from bit positions 06 and 07 of the MQ register.



VTI3-322

Figure 4-16. FDV Routine Flowchart

4.5.8 FDV Routine

Division (figure 4-16) involves the processing of signs, fractions, and exponents of the two operands. Processing of signs involves setting AS+ low (to represent a positive result) if the signs of the two operands are the same or setting AS+ high if the signs of the two operands are different. The dividend fraction must be divided by the divisor fraction and the result must be normalized. After the fraction division procedure has been completed, the result fraction is either in the normalized position or else has its most significant non-zero bit in the sign position (in which case it must be shifted to the right one position). The exponent of the divisor must be subtracted from the exponent of the dividend to produce a provisional result exponent which must be incremented if the result fraction requires shifting to the right.

A provisional result exponent value of -129 represents a conditional underflow value (CUFL). While -129 itself is outside of the allowable range, the value of -128 which will be the final exponent value if the result fraction requires shifting to the right, is within range. Any provisional result exponent value more negative than -129 is definitely an underflow value (UFL). A value of +128 or larger is definitely an overflow value (OFL). In summary, any provisional result exponent value which is outside of the normal range (+127 through -128) indicates the possibility of an overflow or underflow. There is also one in-range provisional result exponent value which represents a conditional overflow value (COFL). This is the value +127 which will produce the out-of-range final result exponent value +128 in the event that the result fraction requires shifting to the right.

An attempt to divide by zero is also evaluated as an overflow condition. The divide-by-zero attempt is recognized by sensing the zero status of the divisor fraction.

At the start of the floating point divide routine, the operand in the floating point accumulator is the dividend and the new operand from memory is the divisor. The divisor fraction is loaded into the B register and the dividend fraction remains in the A register.

To divide fractions, a comparison algorithm with fast shift over zeros is used. Twenty-four iterations of the fraction divide step are performed for single precision division. At each iteration, the partial remainder (or, in the case of the first iteration, the divident fraction) in A is compared with the divisor fraction in B and a quotient bit is shifted left into the MQ register. If A is greater than or equal to B, then the quotient bit is one and the difference, A minus B, is shifted left into A. If A is less than B, then the quotient bit is zero and the contents of the A register are shifted left. If bit 46 or bit 47 of A is a one, then the comparison and the subtraction (if required)

are performed by the arithmetic and logic unit (ALU). If bit 46 and bit 47 are both zero, then A is less than B, the quotient bit is zero, subtraction is not required, and A is shifted left. In the latter case, iterations are performed at twice the normal clock frequency.

A description of the various branches of the FDV routine is presented in the paragraphs which follow. A numerical example is provided by figures 4-17 and 4-18.

4.5.8.1 Microinstructions DISO through DIS2

Microinstruction DISO performs the same functions as starting microinstruction LDSO of the FLD routine. (Refer to paragraph 4.5.1 for a description of LDSO.)

Microinstruction DIS1 transfers the sign bit (S) field of the first operand word from the data latch to the BS flag and transfers the high fraction field of the first operand word (SA) from the data latch to the MQ register. It also subtracts the exponent field of the first operand word, obtained from the data latch, from the divident exponent code residing in the exponent register (E) and places the difference in the exponent register. A component of +128 is added to the difference as it is formed in order to preserve the excess-128 representation. If an out-of-range exponent difference (larger than +127 or more negative than -128) occurs, the ECRY+ flag is set high. This prepares for a jump from the next microinstruction. The E128 code in the FLAG field of the microinstruction effectively introduces a component of +128 into the exponent computation as required to maintain the excess-128 code.

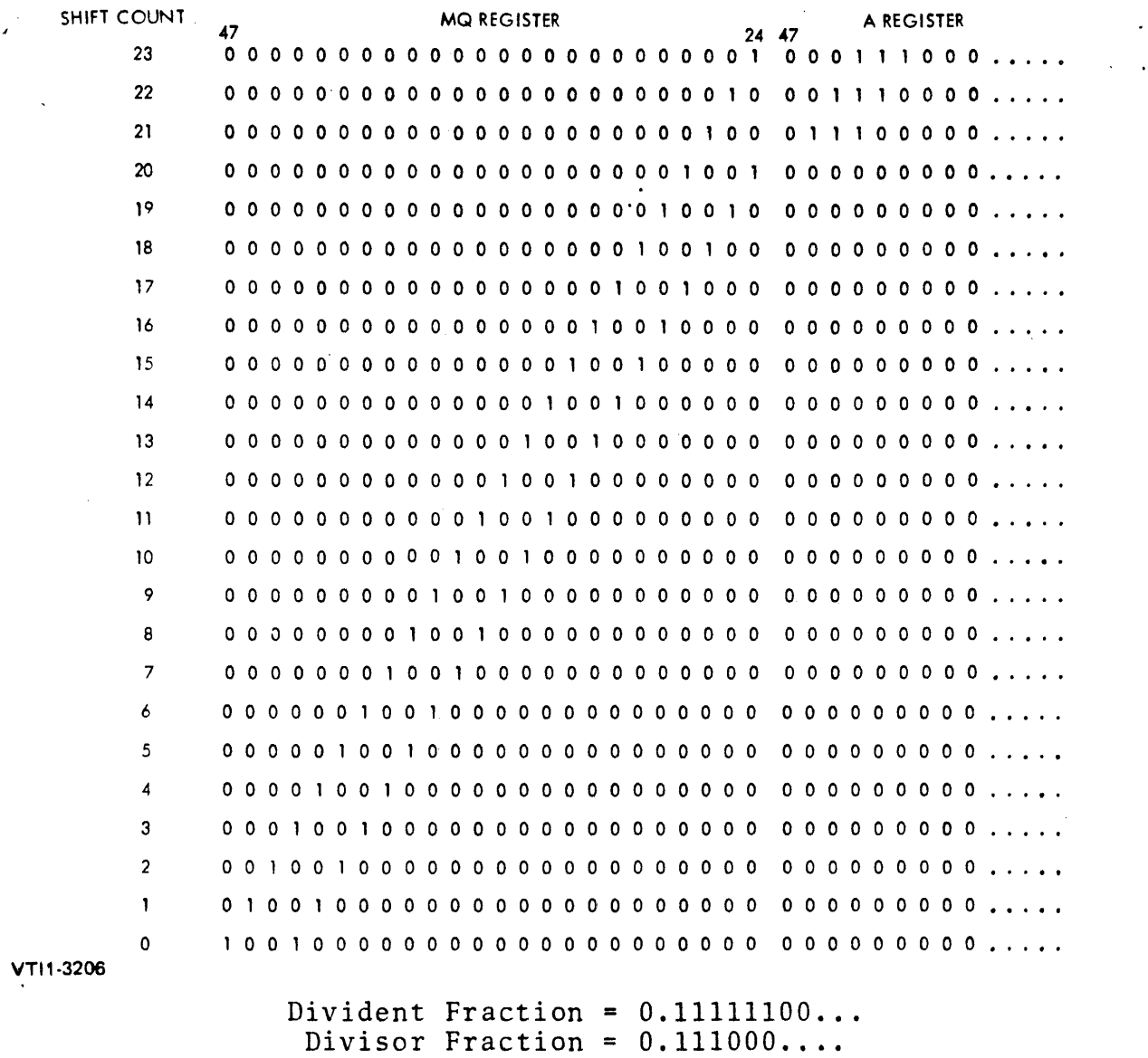
Microinstruction DIS2 sets the ZERO+ flag to the high level if the dividend fraction, held in the A register, is zero. Since the output of the data-loop ALU controls the switching of the ZERO+ flag, the function ALU = A must be selected in order to allow the status of the A register to determine the status of the ZERO+ flag. The SIF code in the MEM field of microinstruction DIS2 suspends the advance to the next microinstruction until the end of the memory cycle during which the second operand word is received from memory and loaded into the FPP data latch. As the advance to the next microinstruction occurs, the SIF code causes control to be returned to the central processor so that the next instruction fetch can be initiated. The JC code in the JCOND field causes a jump to microinstruction DIS3A if ECRY+ was set high during DIS1. Thus, this path is followed if an out-of-range provisional exponent result value is obtained. If the provisional exponent result value is within range, the routine advances to microinstruction DIS3.

Microinstruction	Operation	Results	Comments
	Dividend (Result of previous routine)	E [1 0 0 0 0 0 1 1] AS+ [L] A,B [0.1 1 1 1 1 1 0 0 0 .... 0]	$+2^3 \times (2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} + 0 + \ldots + 0)$ $= +2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 0 + .. + 0$ $= +7.875$
DIS0	Transfer from memory to data latch	S Exponent Fraction (High) 1 [0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0] DL	Sign bit (DL15) = 0. Therefore, number is positive. Exponent = +1. High fraction = $2^{-1} + 2^{-2} + 2^{-3} + 0 + \ldots + 0$ Number is $2^1 (2^{-1} + 2^{-2} + 2^{-3}) = 2^0 + 2^{-1} + 2^{-2}$ $= +1.75$
DIS1	DL(S)⟶BS DL(SA)⟶MQ(SA) E - DL(SE) + 128⟶E [EA7 ⊕ EA8]⟶ECRY	BS+ [L] MQ [0.1 1 1 0 0 0 .....0] E [1 0 0 0 0 0 1 0] ECRY+ [L]	Positive sign saved in BS. Sign and high fraction transferred to MQ 131 - 129 + 128 = 130 Exponent is within range so ECRY remains reset
DIS2	[A = 0]⟶ZERO Transfer from memory to data latch JC	ZERO+ [L] DL [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	Dividend fraction is non-zero so ZERO remains reset. 2nd operand word is received by FPP Go to DIS3 because ECRY+ is low.
DIS3	DL(SB)⟶MQ(SB) JZ	MQ [0.1 1 1 0 0 0 .....0]	Low fraction (in this case all zeros) is transferred from DL to MQ. Go to DIS4 because ZERO+ is low.
DIS4	MQ⟶B 24⟶SC DIO	B [0.1 1 1 0 0 0 .....0] SC [0 0 0 1 1 0 0 0] AS+ [L] SUB+ [H]	Divisor fraction is transferred from MQ to B. Shift counter is set to 24 to prepare for DIS5. AS+ remains low because AS = BS. SUB+ is set in preparation for subtractions in DIS5

Figure 4-17. FDV Example, +7.875 divided by +1.75 = +4.5
(Sheet 1 of 2)

Microinstruction	Operation	Results	Comments
DIS5	[E + 1 ≥ 256]⟶ECRY Divide SC⟶0	ECRY+ [L] First subtraction A = 0.1 1 1 1 1 1 0 0 0 ... B = 0.1 1 1 0 0 0 0 0 0 .... ALU = 0.0 0 0 1 1 1 0 0 0 .... Second subtraction A = 0.1 1 1 0 0 0 0 0 0 .... B = 0.1 1 1 0 0 0 0 0 0 .... ALU = 0.0 0 0 0 0 0 0 0 0 .... (See also figure 4-18.) MQ [1.0 0 1 0 0 0 0 ..... 0]	ECRY remains reset because (E + 1) = 131 < 256 (First subtraction occurs during first clock period. Result is shifted left one position as it enters A register so as to provide value shown for SC = 23 in figure 4-18.) Remainder or, if difference is positive or zero, remainder minus divisor from ALU is shifted left one bit position into A register at each clock time. A one is shifted into MQ each time that positive or zero difference is sensed. A zero is shifted into MQ each time negative difference is sensed.
DIS6	MQ⟶A,B [B = 0]⟶ZERO	A,B [1.0 0 1 0 0 0 0 .....0] ZERO+ [L]	Quotient is transferred from MQ to A,B ZERO flag remains reset because divisor fraction is non zero. Go to DIS7 because ECRY+ is low.
DIS7	ZTZ JSGN	ZERO+ [L]	ZERO flag status is inhibited from changing Go to DIS8 because A47 = 1. (Fraction result has overflowed into sign bit position.)
DIS8	A X 2^{-1}⟶A B X 2^{-1}⟶B E + 1⟶E JZ, END	A,B [0.1 0 0 1 0 0 0 ..... 0] E [1 0 0 0 0 0 1 1]	Final fraction value is 2^{-1} + 0 + 0 + 2^{-4} + 0 +..+ 0 Final exponent value is +3 Result is 2^{3} (2^{-1} + 2^{-4}) = 2^{2} + 2^{-1} = +4.5 Routine ends because ZERO+ is low.

Figure 4-17. FDV Example, +7.875 divided by +1.75 = +4.5
(Sheet 2 of 2)



Divident Fraction = 0.11111100...
Divisor Fraction = 0.111000....

Figure 4-18. Fraction Division Procedure Example

4.5.8.2 Microinstructions DIS3 through DIS8 and OFL2

This branch is entered when the provisional exponent result obtained during microinstruction DIS1 is within range. Microinstruction DIS3 loads the second operand word from the data latch into the MQ register. The JZ code in the JCOND field of this microinstruction causes a jump to microinstruction DIS4D1 if the ZERO+ flag is at the high level; that is, if the dividend fraction is zero. If ZERO+ is low, the routine advances from DIS3 to DIS4.

Microinstruction DIS4 sets the AS+ flag to the low level if the signs of the two operands are the same or sets the AS+ flag to the high level if the signs are different. This action occurs in response to the DIO code in the FLAG field of the microinstruction. This code also causes the SUB+ flag to be set high as required to select the subtraction function for the data loop ALU. The subtraction function remains selected throughout the fraction division procedure of the next microinstruction. This provides the means of determining whether the partial remainder is larger than or equal to the divisor fraction. The divisor is permanently subtracted from the partial remainder when the difference output from the ALU is actually loaded into the A register and this occurs only when the difference is positive or zero. The DIS4 microinstruction also sets the shift counter to 24. This determines the number of iterations of the fraction divide step during the next microinstruction. The constant 24 is loaded into the shift counter via the exponent loop ALU. It originates in a location within the constant PROM which is addressed by the FDV OP code in the FPP instruction register.

Microinstruction DIS5 performs the 24 steps of the fraction division procedure. The DIV code in the FLAG field conditions the arithmetic control for the continued selection of the subtraction function, for the shifting of and conditional subtractions from the partial remainder, and for the generation of the quotient bits and shifting of these bits into the MQ register. The WSDN code in the MEM field suspends the termination of the microinstruction until the clock time at which the shift count reaches zero.

A subsidiary function performed by DIS5 is the formation of (E+1) and the setting of ECRY+ to the high level if this summation generates a carry out of the highest order bit position of the exponent-loop ALU. This indicates a conditional overflow condition; that is, the provisional result exponent in excess-128 code is 255 so that an overflow will occur if the fraction result has to be shifted to the right.

Microinstruction DIS6 copies the quotient fraction from the MQ register into the A and B registers. It also sets the ZERO+ flag to the high level if the divisor fraction initially in the B register is zero. In order to provide the control of the ZERO+ flag, the function ALU = B must be selected so that the divisor fraction value residing in the B register appears at the output of the ALU. The JC code in the JCOND field of microinstruction DIS6 causes a jump to microinstruction DIS7C if ECRY+ was set high in DIS5; that is, if the provisional exponent result value is +127. If ECRY+ is low, the routine advances from DIS6 to DIS7. If the result fraction is normalized and the divisor is not zero, this is the final microinstruction of the routine. The JSGN code in the JCOND field of this microinstruc-

tion causes a jump to microinstruction DIS8 if the result fraction has overflowed into the sign bit or if the divisor is zero. The ZTZ code in the FLAG field or DIS7 inhibits any change in the status of the ZERO+ flag so that the status of this flag still indicates whether the divisor fraction is zero, at the time that DIS8 is entered.

Microinstruction DIS8 shifts the result fraction one position to the right in the A and B registers and increments the exponent result in the E register. If the ZERO+ flag is at the low level, this is the final microinstruction of the routine. The JZ code in the JCOND field causes a jump to microinstruction OFL4 if the divisor is zero. This microinstruction establishes the overflow format (all ones in the A and B registers and E register) and sets the interrupt flag. Thus, the attempt to divide by 0 produces the overflow format.

4.5.8.3 Microinstructions DIS3A through DIS8A, OFL4, and UFL2; DIS7B, DIS8B, and UFL2

This branch, which is entered from DIS2 when an out-of-range provisional exponent result is computed in DIS1, is identical to the DIS3 through DIS6 (described in paragraph 4.5.8.2). However, this branch leads to different terminations as appropriate to the out-of-range exponent condition. Whereas the setting of ECRY+ to the high level in DIS5 indicates a conditional overflow condition, the setting of ECRY+ to the high level in DIS5A indicates a conditional underflow (CUFL) condition. (In this case, the provisional exponent excess-128 code of 255 represents the exponent underflow value of -129. This causes a jump from DIS6A to DIS7B in response to the JC code in the JCOND field of DIS6A. The JSGN code in the JCOND field of microinstruction DIS7B causes a jump to microinstruction DIS8B if the result fraction has overflowed into the sign bit position. (Code JSGN causes a jump in response to either a one in the sign bit position of the A register or a high ZERO+ flag signal. In this case, however, it is known that the ZERO+ flag is low. This is because a conditional underflow cannot occur in division when the divisor is zero. If the divisor is zero then its exponent is -128. When this is subtracted from the exponent of the dividend, the smallest result that can be obtained is the in-range value, zero.) Microinstruction DIS8B shifts the result fraction in the A and B registers one position to the right and increments the exponent. The result fraction is now normalized and the exponent is now at the in-range value of -128.

If the result fraction has not overflowed into the sign bit position, then the final result is an underflow. In this case, the routine advances from DIS7B to UFL2. This microinstruction clears the FPP accumulator and sets the interrupt flag.

In the case where ECRY+ is not set high in microinstruction DIS5A, the routine advances from DIS6A to DIS7A. If the divisor fraction is zero, then the JZ code in the JCOND field of DIS7A causes a jump to OFL4. Thus, the attempt to divide by 0 is treated like an exponent overlow. Microinstruction OFL4 loads the overflow value (all ones in the fraction and exponent fields) into the FPP accumulator and sets the interrupt flag. (The constant 2^{47}, used in OFL4, is loaded into the constant register by DIS7A in response to the C47 code in the FLAG field of DIS7A.)

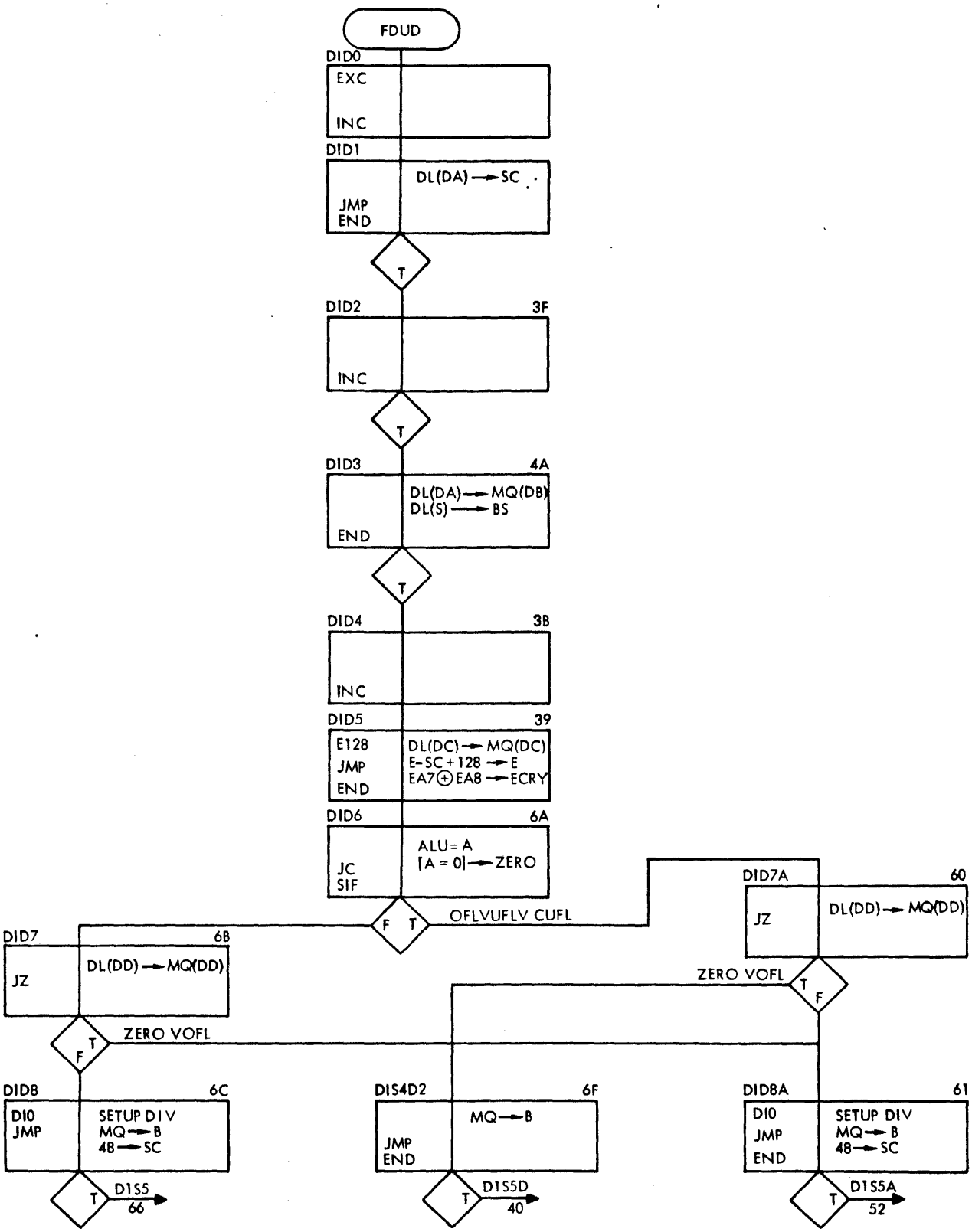
If the divisor fraction is not zero, then the routine advances from microinstruction DIS7A to microinstruction DIS8A. The JESGN code in the JCOND field of this microinstruction causes a jump to microinstruction UFL2 if the most significant bit of the exponent result is a one. This result, indicates that the value in the exponent register is more negative than -128; that is an underflow value. Microinstruction UFL2 clears the FPP accumulator and sets the interrupt flag. If the most significant bit of the exponent result is zero, then the value in the exponent register is more positive than +127; this is an overflow value. In this case, the routine advances from DIS8A to OFL4 and generates an overflow interrupt.

4.5.8.4 Microinstructions DIS4D1 through DIS6D, ZERO3, and OFL2

This branch is entered either from DIS3 or from DIS3A when the dividend fraction is sensed to be zero. The purpose of the branch is to determine whether the divisor fraction is also zero. If the divisor fraction is also zero, then an attempt to divide by zero is occurring. In this case, the routine terminates in microinstruction OFL2 which generates an overflow interrupt. If the divisor fraction is non-zero, then a valid zero result is indicated. In this case, the routine terminates in microinstruction ZERO3 which clears the FPP accumulator. Microinstruction DIS4D1 transfers the divisor fraction from the MQ register to the B register in preparation for the zero test. Microinstruction DIS5D selects the function ALU = B in order to set the ZERO+ flag to the high level if the divisor fraction is zero. Microinstruction DIS6D provides the jump to OFL2 if the ZERO+ flag is high. The C47 code in the FLAG field of DIS6D sets a one into bit 47 of the constants register for use in the OFL2 microinstruction.

4.5.9 FDVD Routine

As illustrated in figure 4-19, the double precision division routine performs functions required for division by a double precision divisor and then jumps into an appropriate microinstruction of the FDV routine. Double precision division is similar to single precision division except that the number of memory words



VTI2-447

Figure 4-19. FDVD Routine Flowchart

in the divisor is four rather than two and the number of quotient bits developed is 48 rather than 24.

Microinstructions DID0 through DID4 are identical to microinstructions MUD0 through MUD4 of the FMUD routine (paragraph 4.5.7). The remaining microinstructions of the FDVD routine are all similar to microinstructions of the FDV routine. DID5 is similar to DIS1 in that it transfers the fraction field of an operand word (in this case the 3rd operand word) from the data latch to the MQ register, subtracts the divisor exponent from the dividend exponent, and sets ECRY+ high if the exponent difference is out of range. DID6 is identical to DIS2. DID7 and DID7A are similar to microinstructions DIS3 and DIS3A in that each transfers the final operand word into the MQ register and each causes a jump if the dividend fraction is zero. DIS4D2 of FDVD is identical to DIS4D1 of FDV. From DIS4D2, a jump into DIS5D of FDV occurs.

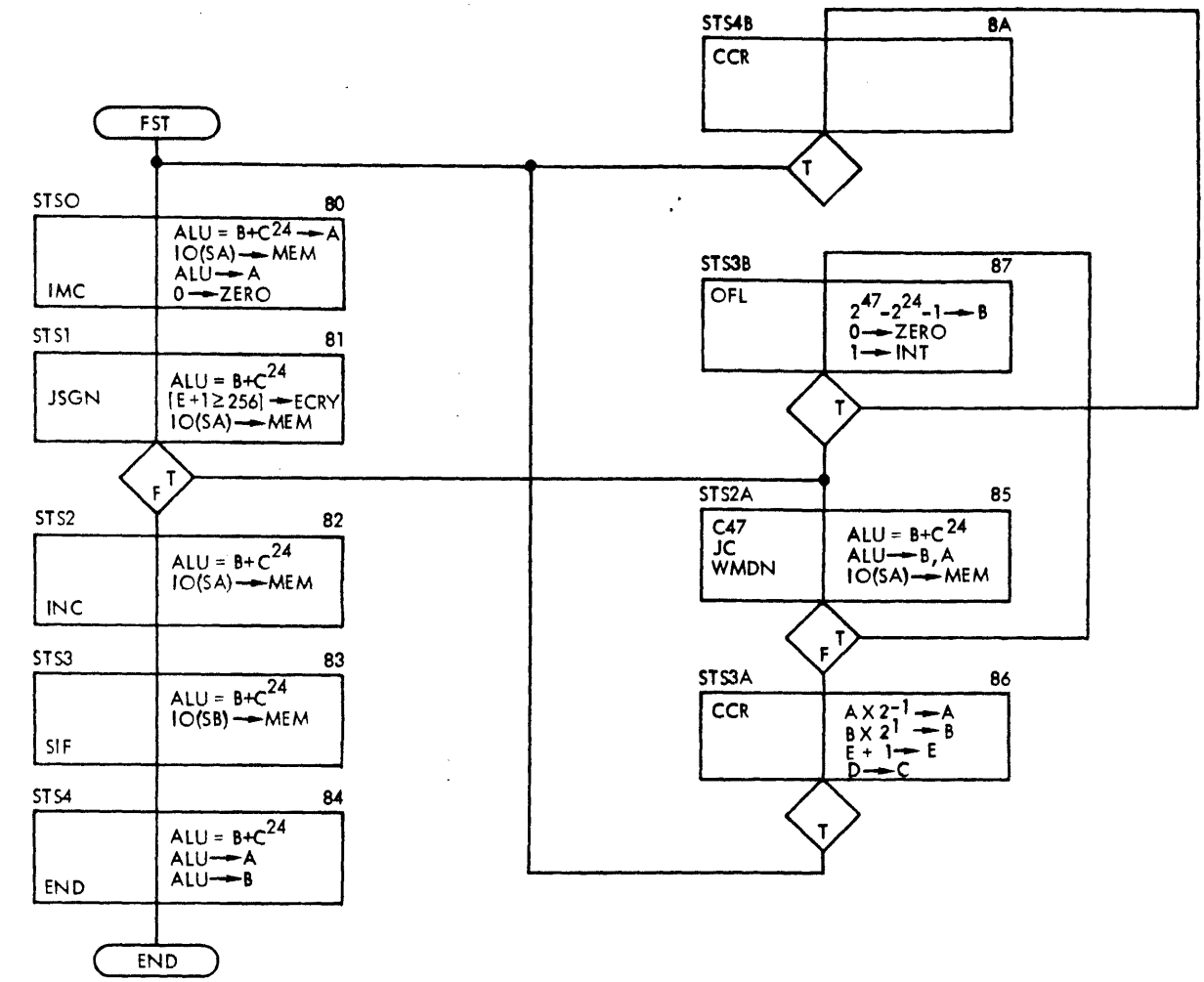
DID8 and DID8A are similar to DIS4 and DIS4A. However, the DID8 and DID8A instructions set 48 rather than 24 into the shift counter. From DID8 and DID8A, jumps occur into DIS5 and DIS5A respectively of FDV.

Double precision fraction division in steps DIS5 through DIS5A is similar to single precision fraction division except for the number of iterations and the destination of the quotient bits. In double precision division, quotient bits are shifted into bit 0 of the MQ register.

4.5.10 FST Routine

The purpose of the single precision store routine is to round off a single precision floating number resulting from a previous arithmetic operation and store this number in memory. Roundoff may cause fraction overflow. In this case, the fraction must be shifted to the right one bit position in order to normalize it. This, in turn, may cause an exponent overflow. If an exponent overflow occurs, the exponent and fraction fields of the number must be set to the maximum values before the number is stored. (Actually, the routine stores the first operand word twice if a fraction overflow occurs. Thus, a corrected copy is stored in place of the original copy if a fraction overflow is detected.)

As illustrated in the flow chart of figure 4-20, round off is implemented by the first microinstruction of the routine (STSO). This is accomplished by adding a binary one in bit position 24, obtained from the constant register (CR) to the copy of the previous result fraction in the B register. The rounded-off result is placed in the A register. (The required constant is loaded into CR prior to the start of the routine when the single precision FST OP code is loaded into the FPP instruction latch.) Microinstruction STSO resets the ZERO flag and selects the first



VTI1-3213

Figure 4-20. FST Routine Flowchart

result word for output to memory. The IMC code in the MEM field of the microinstruction causes the initiation of the memory request required to transfer this first result word.

Microinstruction STS1 repeats the round-off operation provided by STSO and continues to select the first result word for output to memory. Microinstruction STS1 forms the sum (E + 1) in the exponent loop. A carry out of the highest order bit position of the exponent-loop ALU during this summation indicates that (E + 1) is out of range. This sets the ECRY+ flag to the high level, indicating that a fraction overflow will produce an exponent overflow. The JSGN code in the JCOND field of the microinstruction causes a jump to microinstruction STS2A if a fraction overflow into the sign bit position is sensed.

Microinstruction STS2A continues to provide the rounded-off fraction function and continues to select the first result word so that the memory transfer that is in progress can be completed. The IMC code in the MEM field of the microinstruction suspends the termination of the microinstruction until the memory cycle which accomplishes this transfer has been completed. This microinstruction also loads the rounded-off fraction into both the A and B registers. The C47 code in the FLAG field causes a binary one to be loaded into the bit 47 position of the constant register in preparation for the possible requirement to generate an overflow interrupt. The JC code in the JCOND field causes a jump to microinstruction STS3B if ECRY+ is high; that is, if the required incrementation of the exponent would cause an exponent overflow.

If ECRY+ is low, the routine advances to microinstruction STS3A. This microinstruction shifts the result fraction to the right one position in the A and B registers and increments the exponent. The CCR code in the FLAG field resets the constant register. The routine then jumps back to microinstruction STSO to transfer the corrected version of the result to memory. Notice, however, that on the second pass through STSO and STS1, the constant register contains zero. Thus, there is no repetition of the round-off but only the required repetition of the transfer of the first result word to the memory. On this second pass, no overflow can occur and, consequently, the routine advances to microinstruction STS2. STS2 is reached on the first pass if roundoff does not cause fraction overflow. STS2 continues to select the first result word for output to memory. The INC code in the MEM field suspends the termination of the microinstruction until the memory cycle is completed. The INC code also causes the memory address to be incremented and causes another memory cycle to be requested at the time that the microinstruction is terminated.

From STS2, the routine advances to STS3. This microinstruction again provides the rounded-off fraction function. However, now, the second result word is selected for output to memory. The SIF code in the MEM field suspends the termination of the micro-instruction until the memory cycle has been completed. The SIF code also causes control to be returned to the central processor at the time that the microinstruction is terminated. This allows the next instruction fetch to be initiated by the central processor.

From STS3, the routine advances to STS4 where it ends. This instruction loads the rounded-off fraction into the A and B registers. (Actually, the rounded-off value is contained in the 24 most significant bit positions and the 24 least significant bit positions, which are not truncated, contain an excess increment.)

As noted earlier, the sensing of the overflow condition causes a jump from STS2A to STS3B. This microinstruction causes the maximum fraction value to be loaded into the B register. The explicit function used to accomplish this result is to perform the ones complementation of the constant register (which now has a one in the sign bit position and zeros in all of the magnitude positions). The OFL code in the FLAG field of this instruction causes the interrupt flag to be set. The routine then loops back to STSO via STS4B which resets the constant register. With the constant register cleared, no fraction overflow can occur and the routine advances from STSO through STS4 as previously described.

4.5.11 FSTD Routine

The FSTD routine (figure 4-21) is similar to the FST routine. Because of the greater number of significant bits provided in a double-precision result, the constant that is added during round off is different from that used in the FST routine. The appropriate constant value (a binary one in bit position 1) is loaded into the constant register prior to the start of the routine after the FSTD OP code has been loaded into the instruction latch. The most basic difference between the routines is that the FSTD routine must transfer four result words rather than two result words. Thus, where microinstruction STS3 of FST contains a SIF code to return control to the processor after the transfer of the second word, the corresponding FSTD microinstruction (STD3) contains an INC code to increment the memory address and request another transfer after the second word transfer has been completed. Microinstruction STD4 presents the third result word to the memory data bus and waits for this transfer to be completed. It then increments the memory address and initiates the next memory cycle. STD5 presents the fourth result word to the memory data bus and returns control to the central processor after this transfer has been completed.

4.5.12 FIX Routine

This routine converts the floating point result computed by the previous FPP instruction routine into the integer format and stores the integer in memory. The floating point number is truncated during the conversion so that only the integer portion of the floating point number contributes to the final integer value; that is, there is no round-off to the nearest integer value.

The routine first determines whether the exponent of the floating point number is negative. If this is the case, then the magnitude must be less than one. Thus, the routine sets the

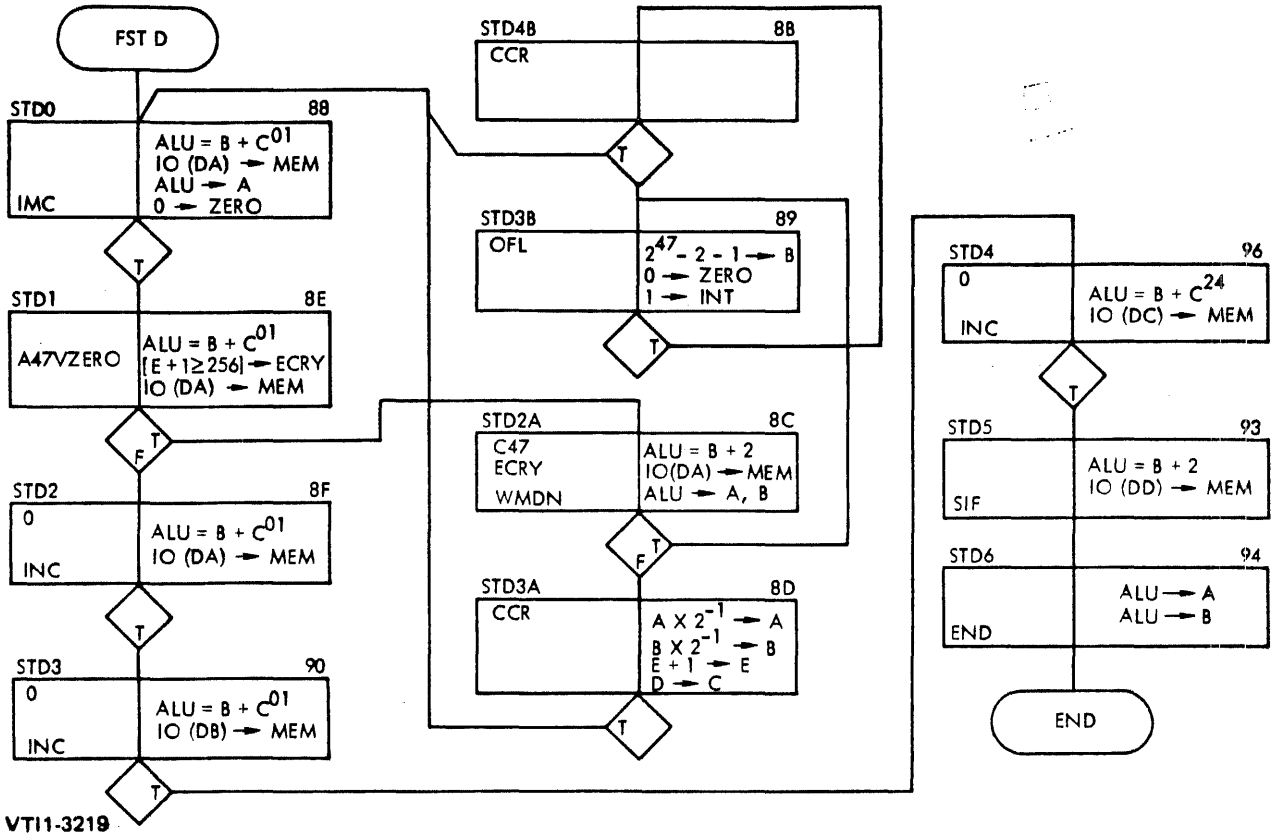


Figure 4-21. FSTD Routine Flowchart

integer to zero and stores this value. If the exponent is zero or positive, the routine subtracts it from 15. If the difference is positive or zero, then the integer portion of the floating point number is definitely within the range that can be represented by the integer format. An exponent value of exactly 15 represents the value 2^{15} X (fraction). Since the most significant bit of the fraction has the weight of 2^{-1}, this means that this bit must have the weight of 2^{14} in the integer format; that is, it must occupy the bit 14 position in that format. This result is achieved without any shifting of the fraction. (During the transfer from the B register to the memory bus, the most significant magnitude bit position of the fraction is connected to the bit 14 line of the memory bus.) If the exponent value is less than 15, then the fraction must be shifted to the right number of positions corresponding to the difference between 15 and the exponent value in order to provide the appropriate alignment for the integer format.

Negative numbers are represented in twos complement form in the integer format. Before the twos complementation can be accomplished, the fraction must be truncated so that only the integer portion of the number is complemented.

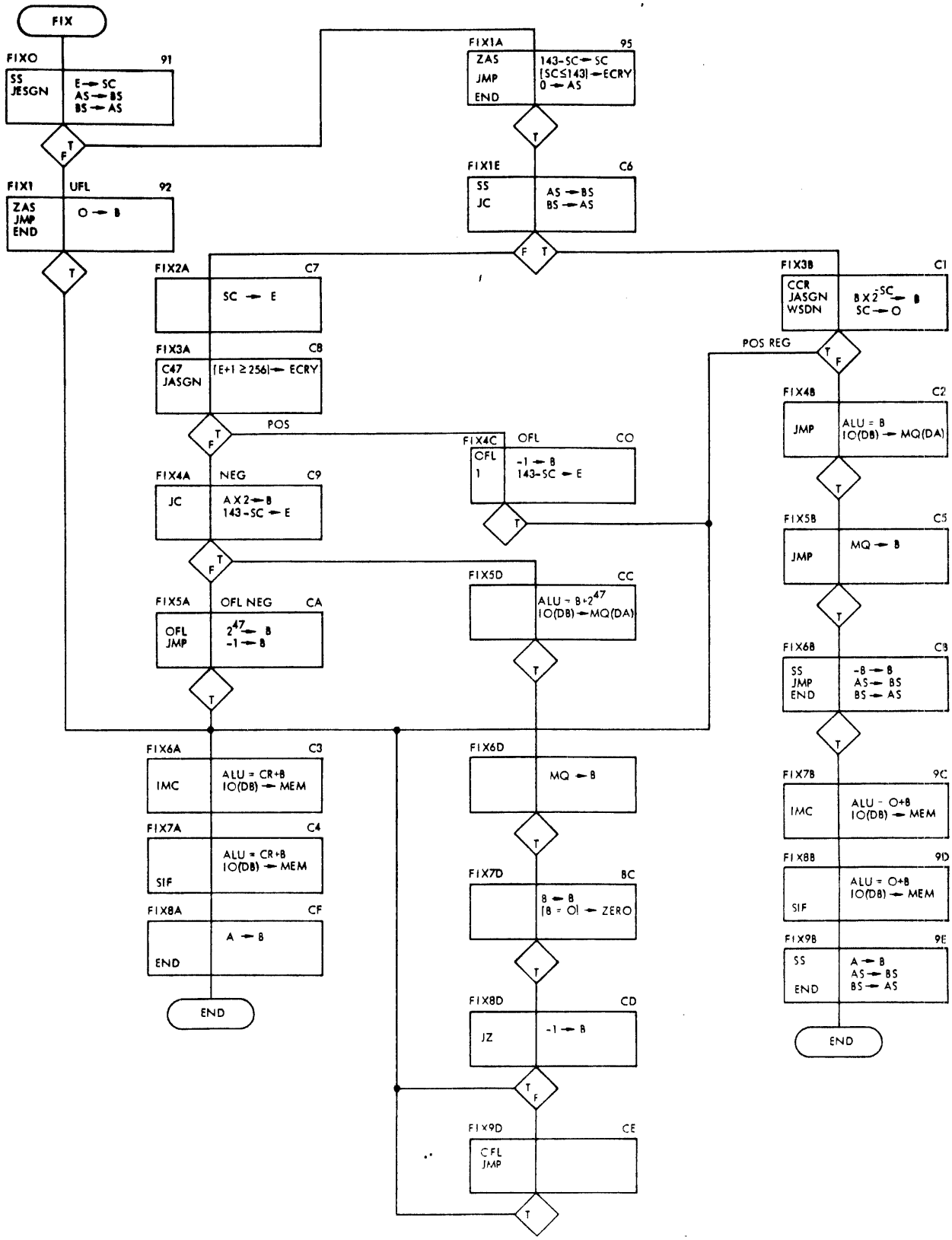
If the difference (15 - exponent) is negative, then the integer portion of the floating point number is either the most negative number that can be represented in the integer format (100000_8) or else it is out of range. The routine first determines the sign of the number. If the sign is positive, then the routine sets the integer to the largest positive in range value and sets the interrupt flag to indicate that an overflow has occurred. If the number is negative, the routine determines whether it is the most negative in-range number or a negative overflow. In either case, the most negative in-range integer value (100000_8) is stored in memory. In the overflow case, the interrupt flag is set.

Microinstruction FIXO (figure 4-22) copies the floating point exponent code into the shift counter in preparation for the subsequent possible subtraction of this exponent code from the constant 143 (the excess-128 code for 15). In response to the SS code in the FLAG field, the sign bits held in the AS and BS flags are swapped. This saves the sign of the floating point number in BS. The JESGN code in the JCOND field causes a jump to microinstruction FIX1A if the most sifnificant bit of the exponent code is a one; that is, if the exponent is zero or positive.

If the exponent is negative, indicating a zero integer, the routine advances from FIX0 to FIX1. FIX1 loads zero into the B register. Also, because of the ZAS code in the FLAG field, it resets the AS flag (making AS+ low as required to indicate a positive sign).

From FIX1, the routine advances to the branch consisting of microinstructions FIX6A through FIX8A. This branch stores the integer in memory. The IMC code in the MEM field of microinstruction FIX6A, causes the required memory cycle to be requested. The SIF code in the MEM field of microinstruction FIX7A suspends the advance to the next microinstruction until the required memory cycle has been completed and then causes control to be returned to the central processor allowing the next instruction fetch to be initiated. During the memory cycle, bits 32 through 47 of the quantity (CR + B), from the data-loop ALU, are connected to the memory data bus. When this path is followed, bits 32 through 47 of both the B register (B) and the constant register (CR) are zeros. Thus, an all zero integer is stored.

Microinstruction FIX8A restores the initial contents of B, which have been saved in A during the routine.



VTI3-325

Figure 4-22. FIX Routine Flowchart

When the exponent is found to be zero or positive in microinstruction FIX0, a jump to microinstruction FIX1A occurs as previously noted. Microinstruction FIX1A forms the difference (143 - SC) and places this difference in the shift counter. If the difference is positive or zero, the ECRY+ flag is set to the high level indicating that the floating point number is definitely within the range that can be represented by the integer format. The ZAS code in the FLAG field of this microinstruction causes the AS flag to be reset (so that AS+ is low). This prepares for use of the AS/BS comparison as a means of determining whether the number is positive or negative.

Microinstruction FIX1E swaps the signs in the AS and BS flags in response to the SS code in the FLAG field. This returns the sign of the number, which was previously saved in BS, to AS and places BS in the reset status (BS+ low) in preparation for the subsequent AS/BS comparison. The JC code in the JCOND field of microinstruction FIX1E causes a jump to FIX3B if ECRY+ is high; that is, if the number is definitely within range.

Microinstruction FIX3B shifts the contents of the B register to the right the number of times corresponding to the difference computed in FIX1A as required to provide the appropriate alignment of the number. The shift count is decremented as each right shift occurs. The WSDN code in the MEM field inhibits the advance to the next microinstruction until the clock time when the shift count reaches zero. The CCR code in the FLAG field clears the constant register. This erases a binary one bit which is automatically loaded into the constant register at the start of the routine and which otherwise could affect the final result that is stored. The JASGN code in the JCOND field causes a jump to microinstruction FIX6A if AS = BA; that is, in this case, if AS+ is low indicating that the number is positive. The termination of the routine when the branch beginning at FIX6A is executed has already been described. Notice, however, that when the branch is entered from FIX3B, the value in the constant register is zero and the integer in B is transferred to memory without modification.

When AS+ is high, indicating that the number is negative, the routine advances into the branch beginning with microinstruction FIX4B. The purpose of this branch is to convert the negative number to twos complement form and then store it. Before the number can be conplemented, it must be truncated. This is accomplished by transferring bits 32 through 47 of the number from B register to the MQ register. Since the remainder of the MQ register contains all zeros both before and after this transfer, the transfer places the truncated version of the number in the MQ register. Microinstruction FIX4B executes this transfer. Microinstruction FIX5B transfers the truncated number from the MQ register into the B register. Microinstruction FIX6B performs the twos complementation (by forming -B and storing it in B) and also swaps the sign bits again. This sign swap

again clears AS. This is necessary in order to avoid ones complementation of the result that is transferred from the data-loop ALU output to the memory bus during the next two microinstructions.

Microinstructions FIX7B and FIX8B, which store the result, are identical to the previously described microinstructions FIX6A and FIX7A. Microinstruction FIX9B copies the floating point fraction which has been held in the A register throughout the routine into the B register and again swaps signs. Since the exponent register still contains its initial contents, the accumulator fields are now in the same status as when the routine was entered.

Now consider the case where the exponent is larger than 15. In this case, ECRY+ remains low when FIX1A is executed so that the routine advances from FIX1E to FIX2A. This microinstruction transfers the negative difference computed in FIX1A from the shift counter to the E register in preparation for a test for the largest in-range negative value.

Microinstruction FIX3A now forms the sum (E + 1) and sets ECRY+ high if a carry is generated out of the most significant bit position of the exponent-loop ALU. This indicates that the difference computed in FIX1A was -1; that is, that the exponent value is 16. This means that if the fraction value is 2^{-1}, and the number is negative, it can still be represented in the integer format. The C47 code in the FLAG field of microinstruction FIX3A causes a binary 1 to be loaded into bit position 47 of the constant register. The subsequent use of this constant depends upon the path that is followed. The JASGN code in the JCOND field causes a jump to microinstruction FIX4C if AS = BS; that is, if the number is positive.

Microinstruction FIX4C loads -1 in twos complement form into the B register (that is, it loads all ones into the B register). It also restores the exponent register to its initial status by subtracting the difference computed in FIX1A, and now held in the shift counter, from 143 and loading the result into the exponent register. The OFL code in the FLAG field sets the interrupt flag. From FIX4C, the routine jumps to the branch beginning at FIX6A. This branch has previously been described. Notice, however, that when this branch is entered from FIX4C, the B register contains all ones and the CR register contains a one in the sign bit position (47). The result, (CR + B), has ones in all bit positions except the sign position. Thus, the integer that is obtained from the bit 32 through bit 47 segment of this function is the largest in-range positive integer value.

When the test of microinstruction FIX3A determines that the number is negative, the routine advances to FIX4A. This microinstruction shifts the fraction one bit position to the left and restores the exponent register to its initial status. The fraction is left shifted by forming A X 2 in the ALU and loading

it into the B register. This left-shifted fraction is used in the test for the largest in-range negative integer value in the event that the exponent is found to be 16. Recall that if the exponent is 16, the ECRY+ flag is set high in FIX3A. The JC code in the JCOND field of FIX4A thus causes a jump into the branch beginning at FIX5D if the exponent value is 16. If the exponent value is greater than 16, then ECRY+ is low and the routine advances from FIX4A to FIX5A. This microinstruction is similar to microinstruction FIX4C in that it sets all ones into the B register, sets the interrupt flag and then leads to the branch starting at microinstruction FIX6A. Thus, in this case also, the largest positive in range integer value is stored.

The branch beginning at FIX5D determines whether the number represents a negative overflow value or the largest in-range negative value. In either case, the number which is stored is the most negative integer value. The only difference is that, in the case of an overflow, the interrupt flag is set.

Microinstruction FIX5D adds the binary one held in the bit 47 position of the constant register (CR) to the left-shifted fraction held in the B register and stores bits 32 through 47 of the result in the MQ register. If the number is the largest in-range integer value, then the truncated result in the MQ register is all zeros. Microinstruction FIX6D transfers this result to the B register. Microinstruction FIX7D sets the ZERO+ flag to the high level if this result is all zeros. Microinstruction FIX8D loads all ones into the B register. The JZ code in the JCOND field of this instruction causes a jump into the branch beginning at FIX6A if ZERO+ is high: that is, if the result is the most negative in-range integer. If the result is an overflow, then microinstruction FIX9D is executed before jumping to FIX6A. The OFL code in the FLAG field of this microinstruction causes the interrupt flag to be set.

When FIX6A is entered from FIX8D or FIX9D, the B register contains all ones and the CR register contains a one in bit position 47. This produces a (CR + B) function which has a zero in the bit 47 position and a one in every other bit position as previously noted. However, in this particular case, AS+ is high. This produces an inversion of the data as it passes from the ALU output through the I/O data multiplexor to the memory data bus so that the data that is stored has a one in the bit 47 position and zeroes in the 32 through 46 bit positions. Thus, the most negative in-range integer (100000_8) is stored in memory.

Figure 4-23 provides a numerical example of the FIX routine.

4.5.13 System Reset/Time-out Routine

This routine, illustrated in figure 4-24, is entered when a system reset (SRST-) signal is received by the FPP or when a time-out occurs. The routine has two branches, the system reset

Microinstruction	Operation	Results	Comments
	Result of previous routine	E [1 0 0 0 0 0 0 1 0] AS+ [L] BS+ [?] A,B [0.1 1 1 1 0 0 ..... 0 0]	$+2^{2} \times (2^{-1} + 2^{-2} + 2^{-3} + 2^{-4})$ $= 2^{1} + 2^{0} + 2^{-1} + 2^{-2}$ $= 2 + 1 + 0.5 + 0.25 = +3.75$
FIX0	E → SC BS → AS; AS → BS JESGN	SC [1 0 0 0 0 0 0 1 0] AS+ [?] BS+ [L]	Exponent copied into SC Signs swapped (Sign saved in BS) Go to FIX1A because MSB of E is 1
FIX1A	143 - SC → SC 0 → AS [SC ≤ 143] → ECRY	SC [0 0 0 0 1 1 0 1] AS+ [L] ECRY+ [H]	1 0 0 0 1 1 1 1 (143) -1 0 0 0 0 0 1 0 (130) 0 0 0 0 1 1 0 1 (13) AS reset in response to ZAS ECRY sets because SC 143
FIX1E	BS → AS; AS → BS JC	AS+ [L] BS+ [L]	Signs swapped (Sign returned to AS) Go to FIX3B because ECRY+ is high
FIX3B	B X 2^{-SC} SC → 0 CCR JASGN	SC (Decimal) B 12 0.011110... 11 0.0011110... 10 0.00011110... . . . . . . 0 0.000000000000011110... CR [0.00.....0]	Shift B right and decrement SC until SC = 0 Reset CR Go to FIX6A because AS = BS
FIX6A FIX7A	ALU = CR + B IO(DE) → MEM	Memory Data Bus 15 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	Non-integral portion of result in B is truncated because only 16 MSBs of B are connected to memory data bus. Integer result stored in memory is 3.
FIX8A	A → B	A, B [0.1 1 1 1 0 0 ..... 0]	Restores initial contents of B

Figure 4-23. FIX Example

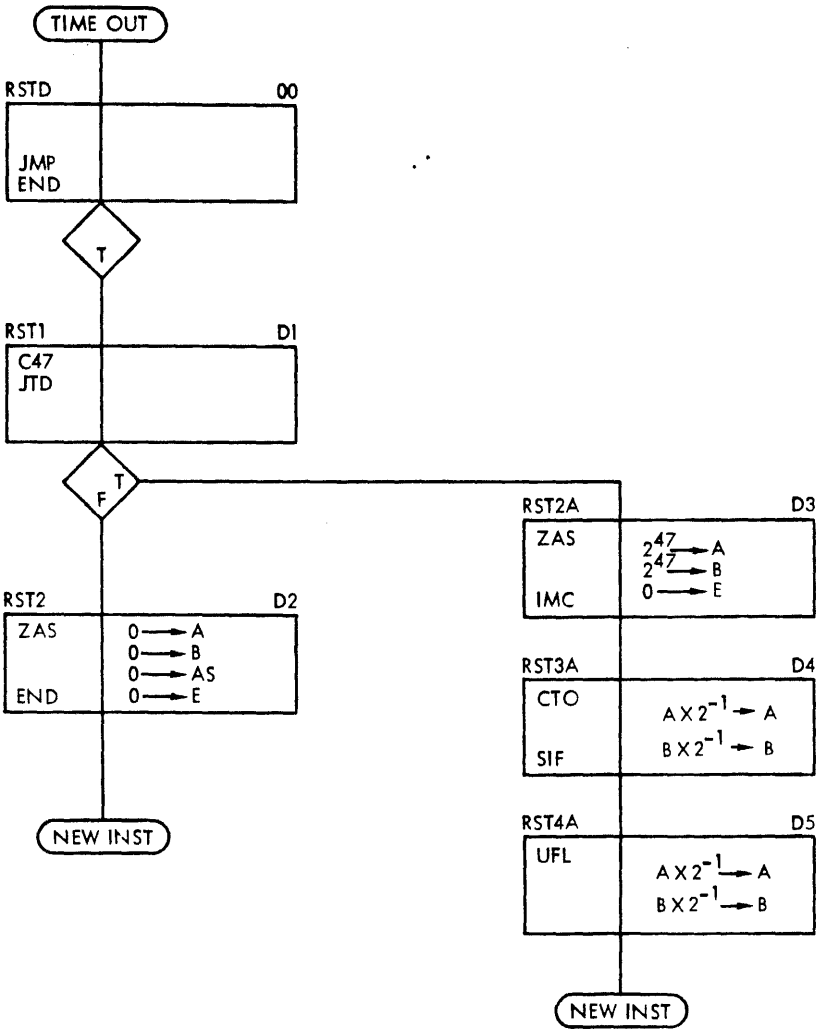


Figure 4-24. System Reset/Time-Out Routine Flowchart

branch (microinstruction RST2) and the time-out branch (microinstructions RST2A through RST4A). The system reset branch loads zero into the fraction, exponent, and sign fields of the FPP accumulator. The timeout branch waits for the completion of any memory cycle that is currently in progress, then returns control to the processor, sets the interrupt flag, and sets an illegal number into the fraction field to serve as an identification of the reason for the interrupt.

Microinstruction RST0, the entry point into the routine, serves no other function than to provide a jump to RST1. The JTO code in the JCOND field of RST1 causes a jump to the timeout branch if the timeout flag (TO+) high. The C47 code in the FLAG field of RST1 loads a one bit into the bit 47 position of the constant register.

Microinstruction RST2A sets the fraction fields in the A and B registers to the constant value loaded in RST1. This places a one in the sign bit position (bit position 47) and zeros in the magnitude bit positions. RST2A also clears the E register. The ZAS code in the FLAG field of RST2A resets the AS flag. The IMC code in the MEM field causes the advance to RST3A to be delayed until the end of any memory cycle that is currently in progress. (The IMC code would normally initiate a memory request. However, with TO+ high, this request function is inhibited.)

Microinstruction RST3A shifts the contents of the A and B registers to the right one bit position. This moves the ones from the sign positions into the most significant magnitude positions which is a valid normalized number (representing 1/2). The CTO code in the FLAG field causes the TO+ flag to be reset to the low level. The SIF code in the MEM field causes control to be returned to the central processor so that the next instruction fetch can be initiated.

Microinstruction RST4A shifts the contents of the A and B registers to the right one bit position. This creates a non-normalized, illegal number. The UFL code in the FLAG field causes the interrupt flag to be set.

When the routine is entered due to a system reset signal, the TO+ flag is at the low level. In this case, the routine advances from RST1 to RST2. RST2 clears the floating point accumulator. The A and B registers and the exponent register are cleared. In response to the ZAS code in the FLAG field, the AS flag is reset.

4.6 DETAILED FUNCTION DESCRIPTION

The remainder of this section describes the functional circuits of the FPP. Each of the functional circuits illustrated as a separate block in figure 4-2 is covered in a separate paragraph.

Subscripts identifying particular members of a logically identical group of control or clock signals are omitted from the mnemonics used in the text and illustrations which follow. For example, the three logically identical clock signals; KKMOK-1, KKMOK-2, and KKMOK-3; are all referred to as KKMOK-.

4.6.1 Central Processor Control

When a floating point instruction is decoded, the central processor control forces the central processor microprogram to a control store location which contains a no-operation (NOP) microinstruction and locks the central processor microprogram at this location until the FPP microprogram reaches a step which specifies an instruction fetch. The central processor control then forces the central processor microprogram to a control store location which contains the start of an instruction fetch routine. In addition to the two valid addresses which central processor control supplies to the central processor control store address lines, central processor control is also capable of supplying repetitive high signal pulses to the address lines during periods when they are not in use. The purpose of these transient signals is to maintain some positive charge on the lines so that they can be placed at the high level more quickly at the start of a valid address period. If the system includes a WCS, then this charging function is provided by the WCS and the transient signal generating capability of central processor control is disabled.

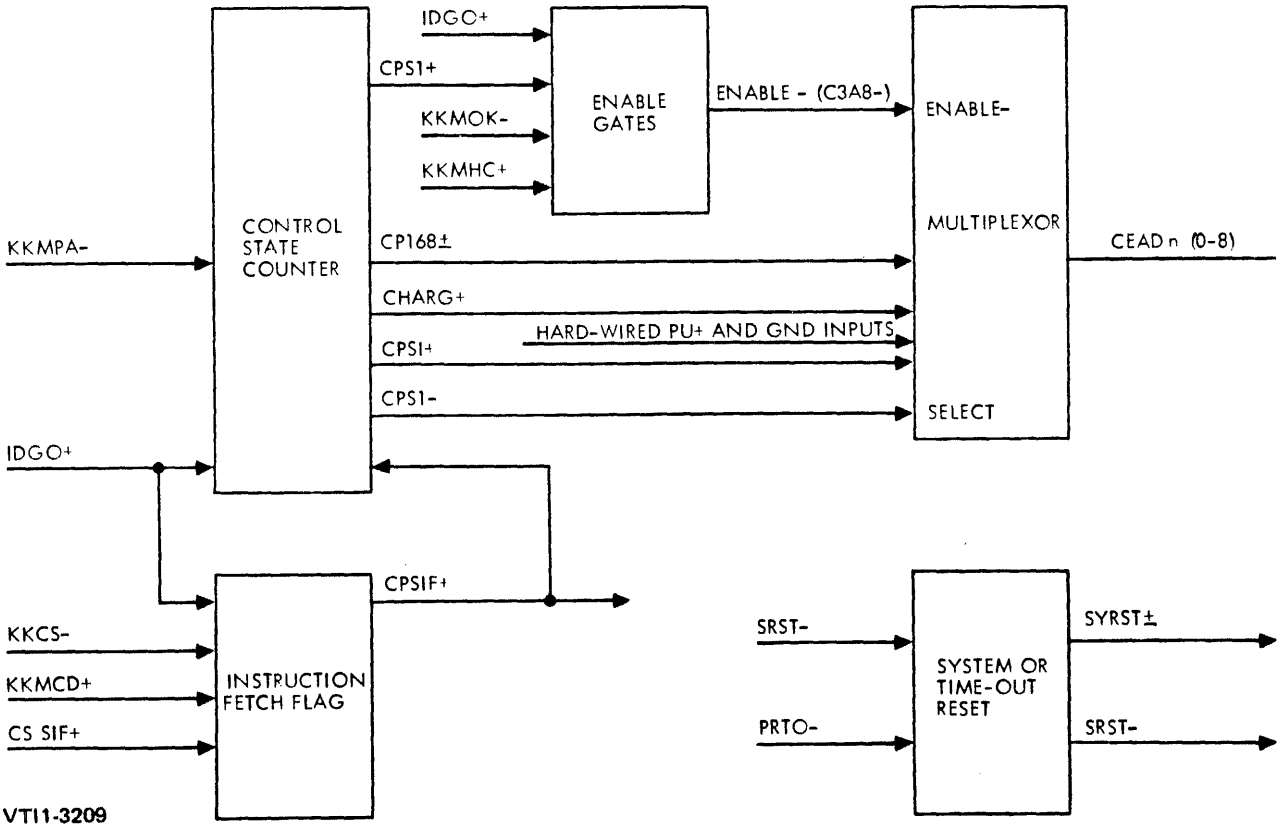


Figure 4-25. Central Processor Control, Block Diagram

Central processor control also supplies system reset signals to other FPP circuits in response to a system reset signal from the central processor or in response to a timeout signal from the priority control.

The Central processor control store address lines are manipulated by means of a 2-to-1 multiplexor and a control state counter. (See figure 4-25.) The control state counter not only switches the multiplexor input channel selection but also switches the levels supplied to certain input lines of each channel. This combination of control functions is used to select hexadecimal 1D8, 168, or 000 for application to the central processor control store address lines. 1D8 is the location containing the NOP microinstruction 168 is the location containing a microinstruction which initiates the next instruction fetch routine. 000 provides all high outputs for use in charging the lines during periods when they are not in use.

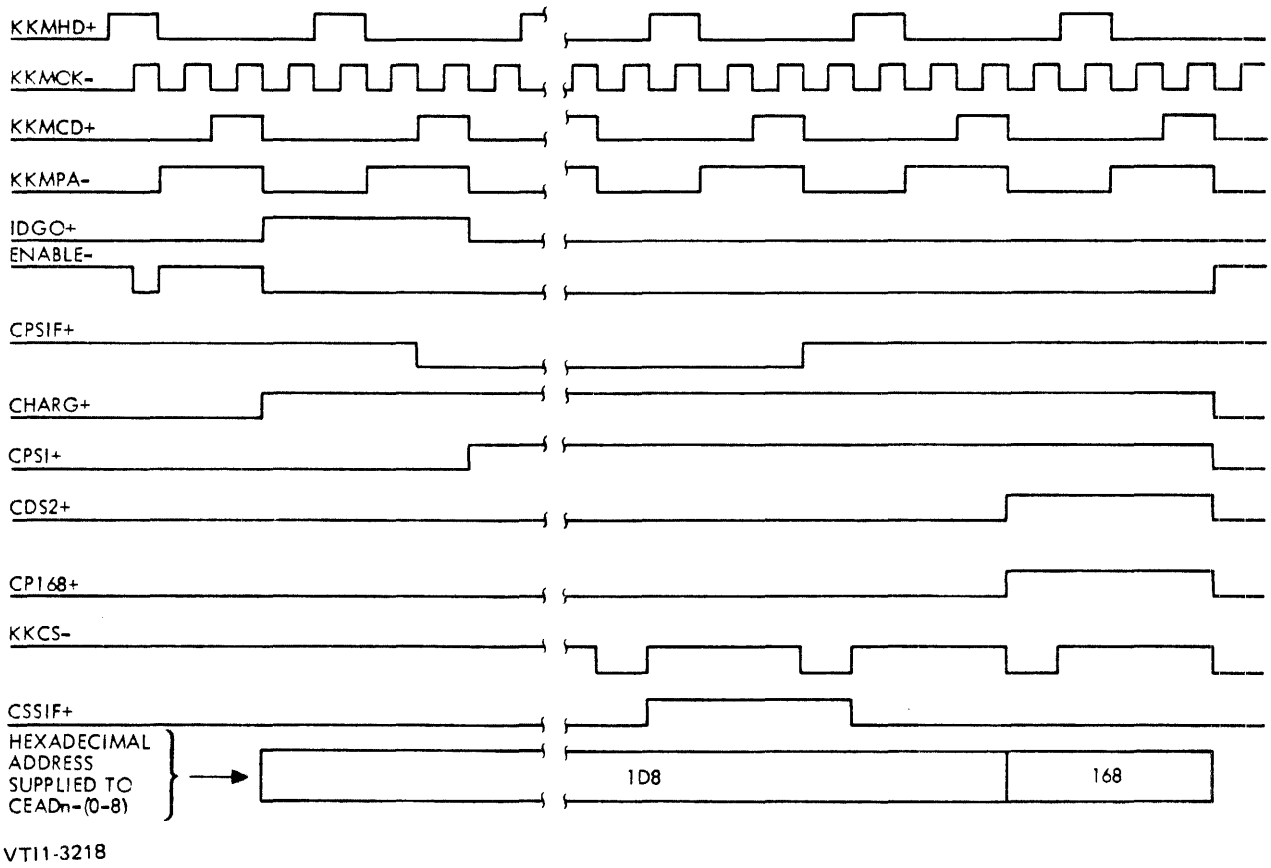


Figure 4-26. Central Processor Control Timing

When the floating point instruction is decoded, the IDGO+ signal from the instruction register is high for one 165-nanosecond clock period. With IDGO+ high, the KKMCD+ pulse from the system clock generator resets instruction fetch flag CPSIF+ to the low level and the KKMPA- pulse from the system clock generator sets CPS1+ of the central processor control state counter to the high level. (Refer to figure 4-26 for central processor control timing.) With IDGO+ or CPS1+ high, the ENABLE- signal (C3A8-) to the multiplexor controlling central processor control store address lines CEADn- (0-8) is low as required to enable the multiplexor. During the period when IDGO+ is high and CPS1+ is low, multiplexor input channel B is selected. Under this condition, the multiplexor output is determined by the status of the CHARG+ signal. This signal is low when either IDGO+ or CPS1+ is high. Under this condition, the address appearing at the multiplexor output is hexadecimal 1D8. When CPS1+ switches to the high level, multiplexor input channel A is selected. Under this condition, the multiplexor output is determined by the status of the CP168+ and CP168- signals. With CPS1+ high and CPS2+ low, the CP168+ signal is low and the CP168- signal is high. Under this condition, hexadecimal 1D8 continues to be supplied to the multiplexor output.

When an FPP microinstruction containing the SIF code in its MEM field is executed, a high CSSIF+ signal is received from the control store decoder. This enables the leading edge of the next KKCS- pulse to set CPSIF+ to the high level. With CPSIF+ high, the next negative-going transition of KKMPA- sets CPS2+ of the control state counter to the high level. With CPS1+ and CPS2+ both high, CP168- is low and CP168+ is high. Under these conditions, the address that is supplied on the CEADn- (0-8) lines is hexadecimal 168. This central processor control store location contains a microinstruction which initiates the next instruction fetch routine. With CP168+ high, the next negative-going transition of KKMPA- resets CPS1+ and CPS2+ to the low level. This returns the ENABLE- signal to the high level, terminating the FPP control of the CEADn- (0-8) lines.

When the ENABLE- signal (C3A8-) is not in a steady-state low status, it is driven low during each period of coincidence of high KKMHD+ and KKMOK- signals, provided that the board is configured to enable the charging function. At this time CPS1+ is low so that channel B of the multiplexor is selected and CHARG+ is high so that an all zeros (all high) set of address signals is supplied as required to allow positive charging of the lines.

When a low SRST- system reset signal is received from the central processor, the SYRST+ signal is placed at the high level and the SYRST- and SRST- lines are placed at the low level as required to produce a general FPP reset terminating any instruction routine that is currently in progress. When a low time-out

signal PRTO- is received from the priority control, the SYRST+ line is placed at the high level and the SYRST- signal is placed at the low level. This produces an FPP reset which is appropriate to terminate the instruction in progress and allow the time-out microprogram routine to be executed.

4.6.2 Priority Control

The priority control exercises control over the interrupt request line (IRQC-A) to the central processor. This allows the priority control to inhibit all interrupts during the period that a string of FPP instructions is being executed. The inhibit is initiated when the first FPP instruction is decoded and remains in force until the FPP stores a result in memory by executing an FST, FSTD, or FIX instruction. A 500-microsecond timeout circuit provides the means of terminating the inhibit in the event that the FPP instruction string is not completed within the expected time. When the timeout occurs, the interrupt-inhibit is terminated and a fault interrupt is initiated.

The priority control also exercises control over the PMA and DMA memory access lines to the central processor (ORQM-A and IRQM-A respectively) and the acknowledge line to the PMA (MAKO+A). This control is used to inhibit PMA and DMA memory access during FPP memory cycles. When the FPP has memory access priority, the priority control places the ORQM-A line to the central processor at the low level. This simulates a pending PMA memory request. Since PMA has higher priority than DMA, this prevents the central processor from giving memory access to the DMA. PMA memory access is inhibited during FPP memory cycles by inhibiting the acknowledge line to the PMA. When a PMA or DMA request is received by priority control, memory access priority is surrendered to the PMA or DMA at the end of the current FPP memory cycle. The priority control then places the PROUT+ signal at the high level until the PMA or DMA completes its transfers and drops its memory request. The high PROUT+ signal inhibits FPP memory access during the period when the FPP does not have memory-access priority.

Figure 4-27 illustrates the various priority control subfunctions. If an interrupt request is enabled, a low IRUX-I signal from the interrupt interface causes interrupt request IRQC-A to switch to the low level in response to the trailing edge of KKMFC- from the system clock generator. The IRQC-A signal remains low until reset during the interrupt service routine executed by the I/O control microprogram of the option board. The reset occurs under the control of the IR007+, IINHC-, and CINTF+ signals from the I/O control on the option board. (The interrupt request reset circuit duplicates a circuit provided on the option board which controls the interrupt request reset function when an FPP is not used.)

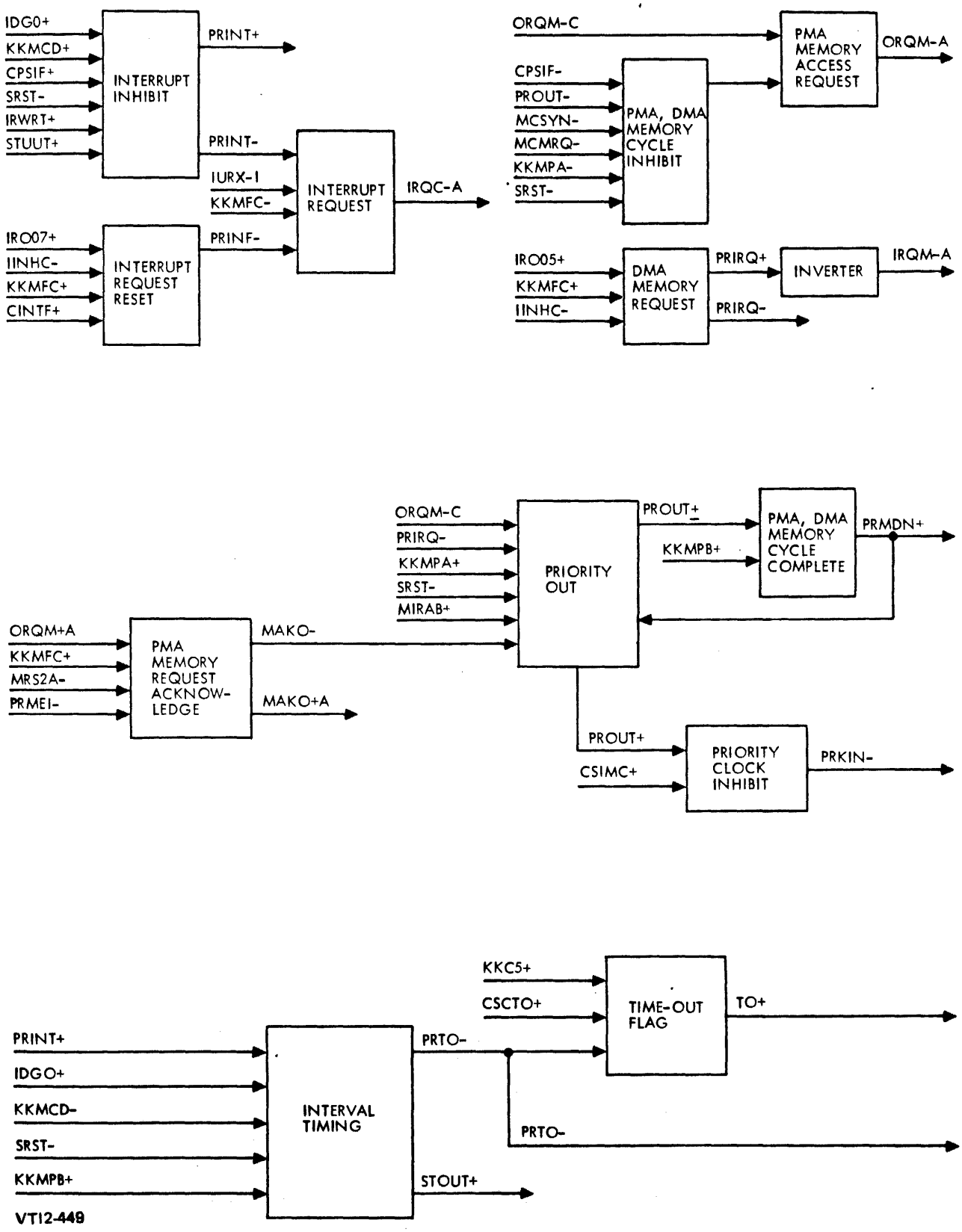


Figure 4-27. Priority Control, Block Diagram

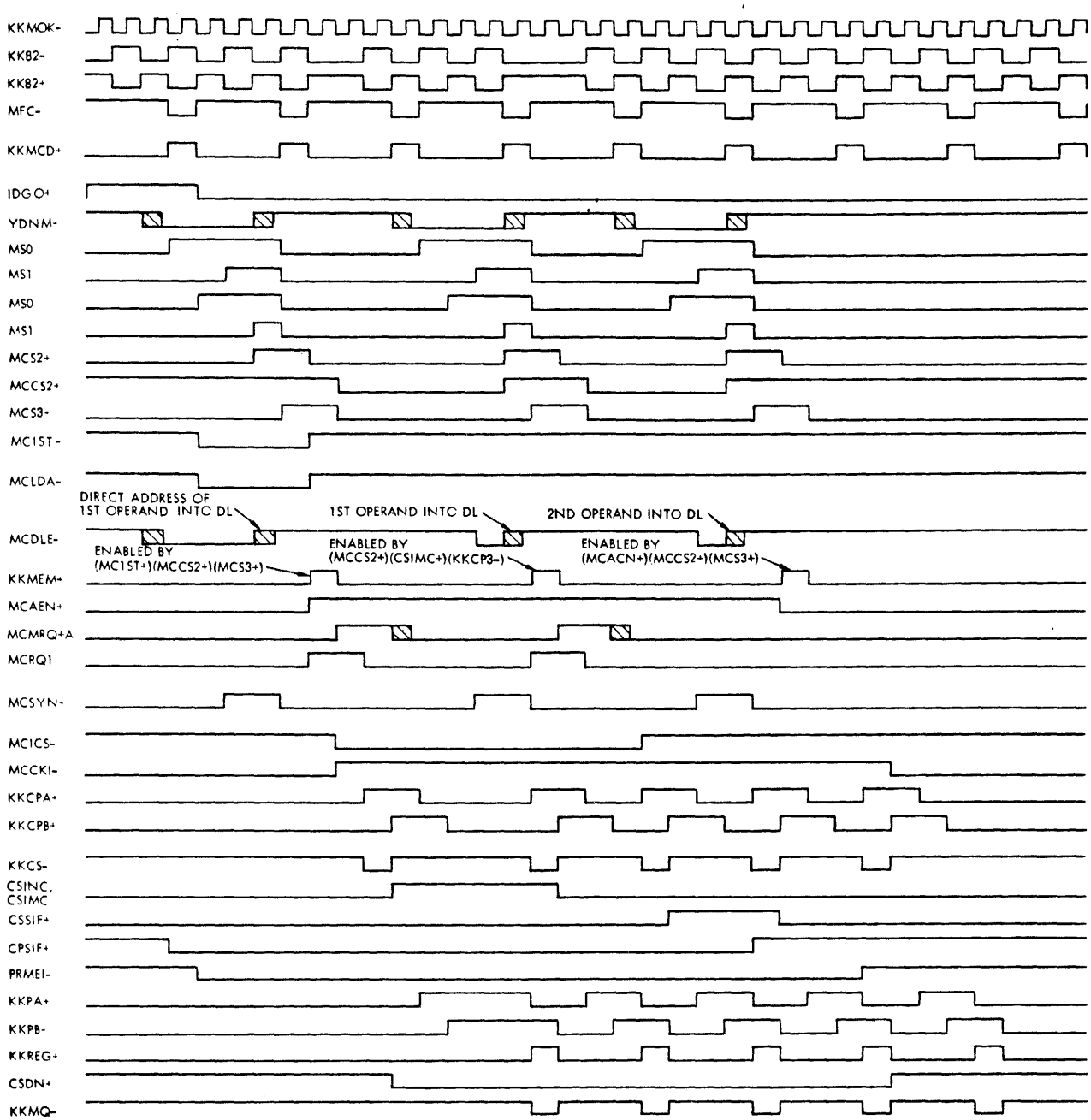
When the first FPP instruction of an FPP instruction string is decoded (IDGØ+ high) the trailing edge of KKMCD+ from the system clock generator sets PRINT- low. This inhibits the setting of IRQC-A to the low level. Once set low, PRINT- normally remains low, inhibiting interrupts, until the FPP transfers the results of its computations to memory. This occurs during the execution of an FST, FSTD, or FIX instruction. These instructions are identified by a high ILWRT+ signal from the instruction latch. After the FPP executes a microinstruction containing the instruction fetch code in its MEM field, CPSIF+ from the central processor control switches to the high level. The coincidence of high ILWRT+ and CPSIF+ signals resets PRINT- to the high level.

The PRINT+ signal of the interrupt-inhibit sub-function is used to initiate the interrupt-inhibit time-out. The positive-going transition of PRINT+ starts the time-out function. The STOUT+ signal is set high at the same time that the PRINT+ signal is set high. (This is timed by IDGO+ and KKMCD-.) After the time-out is completed, KKMCD- resets STOUT+ to the low level. If PRINT+ is still high at this time, the next KKMPB+ pulse is gated to the PRTO-line. The low-going PRTO- pulse sets time-out flag TØ+ to the high level. PRTO- is also supplied to the system reset logic where it is steered onto the SYRST+ line. This results in the termination of the instruction routine that is in progress and the transmission of an interrupt request to the central processor. (With STOUT+ high, the next KKMCD+ pulse resets PRINT- to the high level as required to enable interrupts.) After the FPP microprogram has entered the time-out branch in response to the high TØ+ signal, a high CSCTO+ signal causes the next KKCS+ pulse to reset TØ+ to the low level.

Memory priority is controlled by three flip-flops; PRØUT, PRMEI, and PRMDN.

PRØUT indicates that the FPP has lost memory priority due to a PMA request (ORQM) or a DMA request (IRQM). PRØUT is set by either request and is not reset until both request (ORQM or IRQM) and acknowledge (MAKØ or MIRAB) are reset. PRØUT prevents initiation of a new memory cycle by the FPP and, at completion of the current memory cycle (PRMEI low), disables the FPP memory address and data drivers.

PRMEI indicates the FPP is currently fetching or storing memory words and prevents transfer of priority until the end of the current memory cycle. PRMEI is normally set at the start of an FPP instruction (CPSIF low) and reset when all operands have been transferred (CPSIF high). PRMEI prevents acknowledge to the PMA (it forces MAKØ low) and inhibits acknowledge to the DMA by simulating a PMA request (it forces ORQM to the central processor high). If the FPP loses memory priority (PRØUT high) then PRMEI is reset at the end of the current memory cycle (KKMEM, MCMRQ, and MCSYN low) allowing PMA and DMA memory access and disabling FPP memory address and data drivers. PRMEI is not set again until the FPP regains memory priority (PRØUT low).



NOTE: THIS DIAGRAM IS BASED ON THE FOLLOWING ASSUMPTIONS;

1. THE ADDRESS RECEIVED FOLLOWING THE DECODING OF THE FPP INSTRUCTION IS A DIRECT ADDRESS
2. THE ONLY MEMORY ACCESS REQUESTS OCCURRING DURING THE INTERVAL SHOWN ARE THOSE ASSOCIATED WITH THE TRANSFERS TO THE FFP
3. MEMORY CYCLE TIME IS 330 NSEC

VTI3-323

Figure 4-28. FLD Instruction, Timing Diagram

PRMDN is used to reset FPP memory fetches after the FPP loses and regains priority. PRMDN follows PRØUT by 44 nanoseconds.

To avoid clock skew problems, PRMEI is clocked at the negative-going edge of MCDFC, PRØUT is clocked 82 nanoseconds later, and PRMDN is clocked 41 nanoseconds after PRØUT.

When PRØUT+ is high at the time that the FPP microprogram initiates a memory request (CSIMC+ high), the PRKIN- signal to the clock control is placed at the low level. This inhibits control store and register clocks during the period while the FPP is waiting to regain memory access priority.

4.6.3 Interrupt Interface

When enabled, the interrupt interface stores a floating point processor interrupt request until it can be serviced by the central processor. Figure 4-29 illustrates the interrupt interface as two functional blocks. One of these blocks provides an interrupt storage register and an address encoder which places the assigned interrupt address on the E-bus to the central

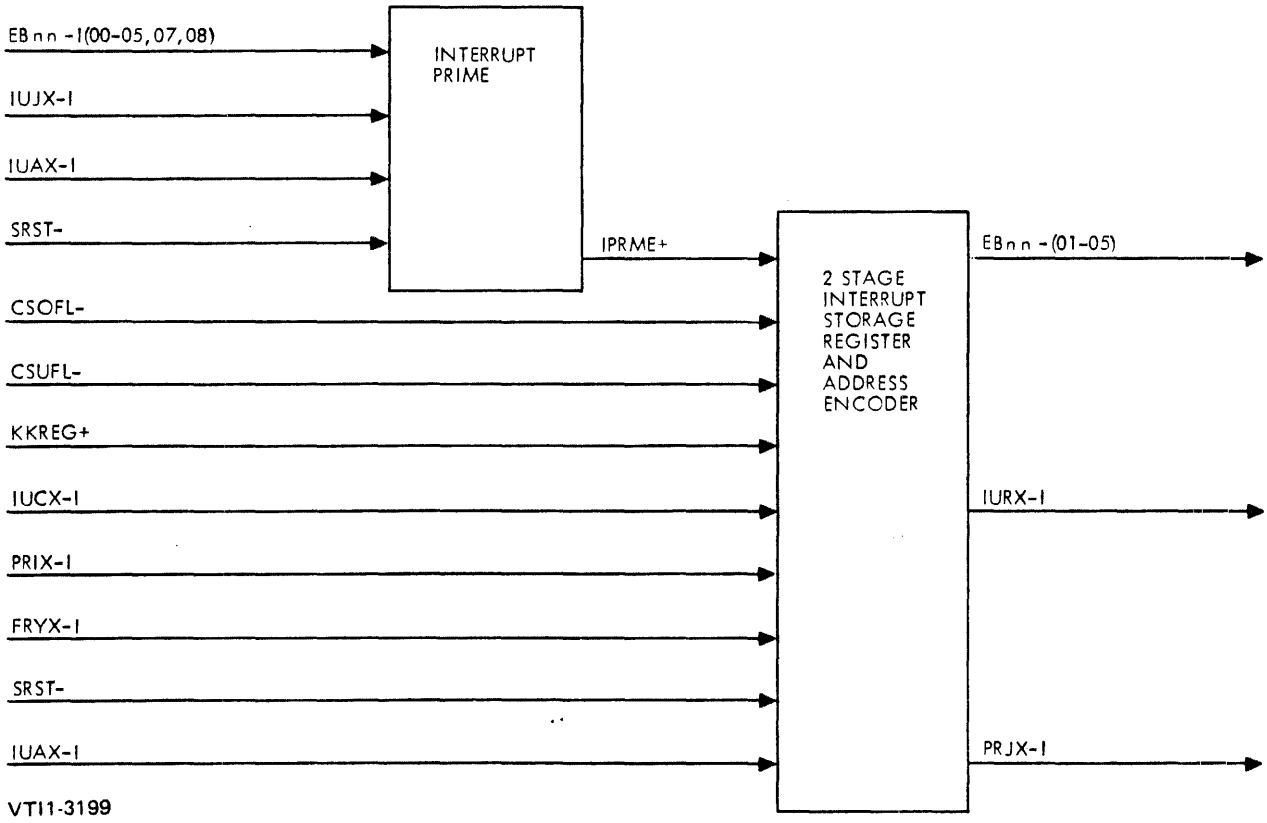


Figure 4-29. Interrupt Interface, Block Diagram

processor when the central processor services the interrupt request. The second block provides the interrupt-enable function.

System reset signal SRST- from the central processor control resets the interrupt-enable function, inhibiting the storage of an interrupt request. In order to enable the storage of an interrupt request, the interrupt-enable function must be set by a command received on the E bus. The interrupt set/reset command code appears on bits 0 through 5 and bit 11 of the E bus (EBnn-I). When bit 7 of the E bus is high, the set/reset command sets the interrupt-enable function. When bit 8 of the E bus is high, the set/reset command resets the interrupt-enable function. The interrupt-enable function is also reset if a low IUJX-I signal is received from the central processor at the same time as the low IUAX-I signal associated with the acknowledgement of an interrupt request.

When the interrupt-enable function is set, the IPRME+ signal is high. This enables the storage of an interrupt request in the interrupt storage register. The storage of an interrupt request occurs in response to the receipt of a low CSØFL- or a low CSUFL- signal from the control store decoder. These signals indicate that an overflow or underflow condition respectively has been detected. When the CSØFL- or CSUFL- signal is low, the KKREG+A signal from the clock control logic clocks the input stage of the two-stage interrupt storage register to the set status. The output stage of the interrupt storage register is updated to the status of the input stage by each interrupt clock signal IUCX-I from the central processor. Three conditions must be satisfied in order to transmit an interrupt request (low IURX-I signal) to the central processor. The interrupt-enable function must be set, the output stage of the interrupt storage register must be set, and the PRIX-I signal must be at the low level (indicating that no higher priority interface is currently requesting an interrupt). When the central processor acknowledges the interrupt request by placing the IUAX-I signal at the low level, the address encoder places the assigned interrupt address (76 octal) on the E bus. This requires driving the EB01- through EB05- lines to the low level. This address is placed on the E bus only if the interface is still the highest priority interface currently requesting an interrupt. If the low IURX-I signal is accompanied by a low FRYX-1 signal, then the input stage of the interrupt storage register is reset as required to terminate the interrupt request at the next IUCX-I time.

When the interface is not currently requesting an interrupt, it passes the PRIX-I signal on the PRJX-I line to the next lower priority interface.

A system reset signal (SRST-) resets both the input and output stages of the interrupt storage register.

4.6.4 Memory Sequencer

The memory sequencer (figure 4-30) supplies the MCDLE- signal used to load data from the memory data bus into the data latch. It also supplies a sequence of pulses used to synchronize FPP memory control functions with respect to memory cycle timing.

A low MM1I- signal from the central processor is steered to the MCDLE- line. This provides the MCDLE- signal which loads instructions or addresses into the data latch during each central processor instruction fetch cycle. This is the means by which the FPP instruction reaches the data latch. It is also the means by which the direct or indirect address of the first operand word (fetched following the instruction word) reaches the data latch. Subsequent transfers into the data latch occur during states MCS1 and MCS2 of the memory sequence, provided that YDNM- and MCA41+ are both high. The high YDNM- signal indicates that a memory cycle is in progress. The MCA41+ signal follows the MCAEN+A signal but is delayed with respect to MCAEN+A by one clock period of KKMØK+ (41 nanoseconds). MCAEN+A is high when a transfer between memory and the FPP is required and the FPP has memory access priority. States MCS1 and MCS2 of the memory sequence are indicated respectively by a low MCS1A- or MCS1B-

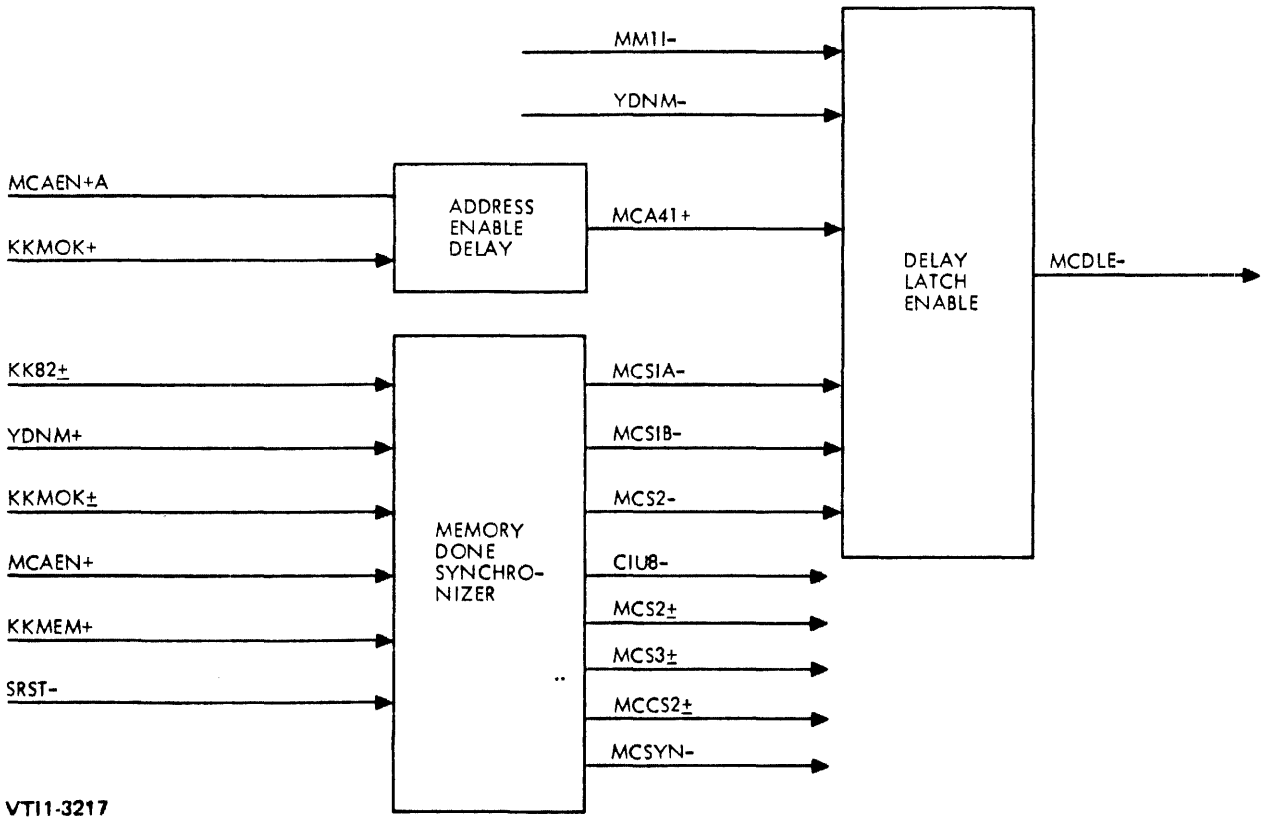


Figure 4-30. Memory Sequencer, Block Diagram

signal and by a low MCS2- signal. MCA41+ is necessary in order to insure that data transferred over the memory data bus during a PMA or DMA memory cycle is not unintentionally loaded into the data latch.

The memory done synchronizer synchronizes the memory done signal (YDNM+) with the system clock (MØCLK+) and provides synchronized timing pulses following the negative-going edge of YDNM+. State flip-flops in the memory done synchronizer are MCSOA, MCSOB, MCS1A, MCS1B, MCS2, MCS3, and KK82. Either MCSOA or MCSOB is set within 41 nanoseconds after the negative-going edge of YDNM+. Then either MCS1A or MCS1B is set between 82 and 124 nanoseconds after the negative-going edge of YDNM+, providing the synchronized memory done signal, MCSYN+. MCS2 follows MCSYN+ by 41 nanoseconds and MCS3 follows MCS2 by 41 nanoseconds. The pulse width of MCSYN+, MCS2, and MCS3 is 82 nanoseconds. Flip-flop KK82 provides 82 nanosecond period clocks to flip-flops MCSOA, MCSOB, MCS1A, and MCS1B. A timing sequence for the memory done synchronizer appears in figure 4-28.

In addition to controlling the timing of the MCDLE-signal, outputs from the memory done synchronizer inhibit the advance of the FPP microprogram, inhibit memory priority changes, and inhibit initiation of the next memory cycle until the current memory cycle is completed.

The MCCS2+ output of the memory sequencer is reset when an FPP memory cycle is initiated. The signal is set at the same time that MCS2+ is set. This signal allows the control store to initiate a memory cycle after the current memory cycle is completed. MCCS2+ also prevents double memory clocks (KKMEM+) between memory cycles.

4.6.5 Memory Control

As shown in figure 4-31 the memory control function includes the following sub-functions:

a. Memory Control Clock Inhibit. This sub-function stops the control store clock (KKCS-) and the register clocks (KKREG-, KKMQ-) when an FPP instruction is complete, and it starts these clocks when the next FPP instruction routine is started.

b. Memory Request. This sub-function controls the memory request line (MRQY-) to the central processor.

c. Memory Control Address Enable. This sub-function provides signals which enable the FPP address to the memory address bus.

d. Memory Control Data Enable. This sub-function controls the gating of data to the memory data bus and controls the write request lines (MWRY+ and MWLY+).

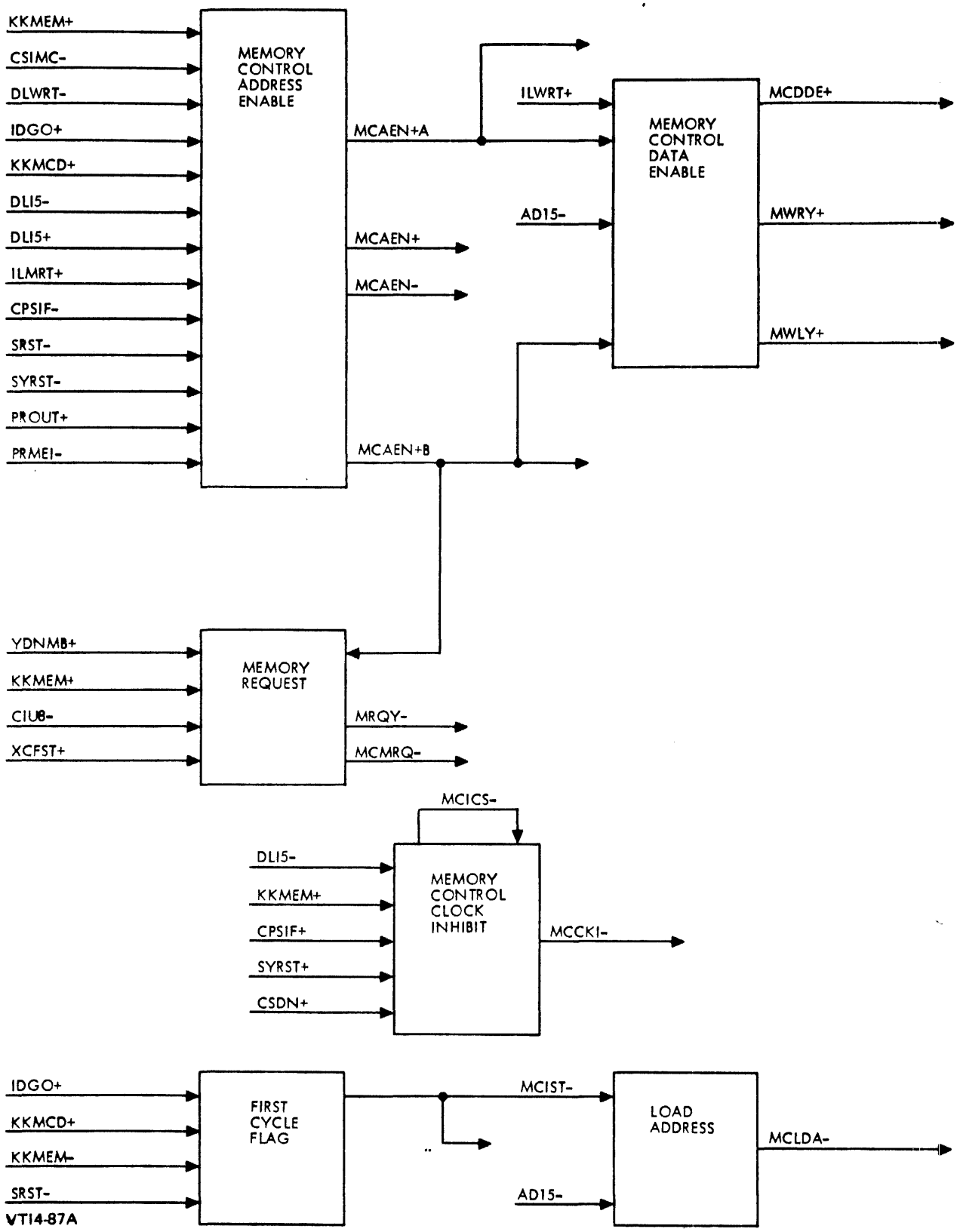


Figure 4-31. Memory Control, Block Diagram

e. Instruction Start Flag. This subfunction provides the MCIST-signal which initiates the first memory clock (KKMEM+) of an FPP instruction and loads the first address into the FPP address register.

f. Load Address. This subfunction provides the MCLDA-signal which loads the addresses into the FPP address register.

Figure 4-28, which shows the timing of the FLD instruction, provides an example of the timing of the various memory control signals.

In order to enable the control store clock generator, MCCKI- must be high. This requires that either CSDN+ or MCICS- be low. CSDN+, from the control store decoder, is the signal which inhibits the control store clock generator at the end of an instruction routine. It is high only for the final microinstruction of an instruction routine. MCICS-, which is generated in the memory control clock inhibit sub-function, is the signal which goes low to initiate each instruction routine. Under quiescent conditions, the final microinstruction of the previous instruction routine resides in the control store register so that CSDN+ is high. MCICS- is also high at this time. When the presence of the direct address of the first operand in the data latch is sensed, MCICS- is set low to initiate the instruction routine. A direct address is identified by a binary 0 in the bit 15 position. Thus, when the data latch contains an address, a high DL15- signal from the data latch indicates a direct address. With DL15- high, MCICS- is set low by the negative-going transition of the KKMEM+ signal from the system clock generator. This places MCCKI- at the high level to enable the control store clock generator. The first control store clock loads the starting microinstruction of the instruction routine being executed into the control store register. This causes the CSDN+ signal to switch to the low level. MCICS- also remains low until the CPSIF+ signal from the central processor control switches to the high level. This occurs following the execution of a microinstruction whose MEM field contains the SIF code. With MCICS- returned to the high level, MCCKI- goes low again when the final microinstruction of the routine is loaded into the control store register causing CSDN+ to switch to the high level.

The memory control address enable sub-function sets the MCAEN+ signal high when there is a requirement for an operand transfer between the FPP and memory. If the FPP currently has memory access priority, then the high MCAEN+ level also places the MCAEN+A and MCAEN+B signals at the high level. These signals connect the address register outputs to the memory address bus.

A high MCAEN+B signal is also required to enable a memory request.

During a non-write instruction, MCAEN+ is set at the start of the first FPP-initiated memory cycle (indirect address of operand fetch) and remains high until the next instruction fetch is initiated by signal CPSIF+.

In the case of a write instruction, MCAEN+ is set at the start of the first FPP-initiated indirect address memory cycle (if indirect addressing is required) and remains high until indirect addressing is completed. MCAEN+ is also set at the start of the first FPP-initiated write memory cycle and remains high until the next instruction fetch is initiated by signal CPSIF+.

The priority inputs to the memory control address enable sub-function are PROUT+ and PRMEI- from priority control. As long as one of these signals is low, the FPP has memory access priority. Under this condition, MCAEN+A and MCAEN+B are placed at the high level when MCAEN+ is high.

A high MCAEN+B signal from the memory control address enable sub-function arms the memory request sub-function to initiate a memory request. The status of MCAEN+ is determined at the leading edge of each KKMEM+ pulse. If MCAEN+ is set high (or remains high) at the leading edge of a KKMEM+ pulse and if the FPP has priority (so that MCAEN+B is also high), then a memory request is normally initiated at the trailing edge of KKMEM+. To make a memory request, the MRQY- signal is placed at the low level. When a memory cycle is initiated in response to the low MRQY- level, the memory acknowledge signal (YDNMB+) switches to the low level. This returns the MRQY- signal to the high level terminating the request. The internal memory request flag is then reset by the C1U8- signal from the memory sequencer. A memory request can only occur if the XCFST+ signal from the WCS option is high.

High MCDDE+A and MCDDE+B signals from the memory control data enable sub-function are used to connect the I/O data multiplexor outputs to the memory data bus. Three conditions are required to obtain these high signals;

a. MCAEN+A and MCAEN+B must be high, indicating that a transfer is required and that the FPP has memory access priority.

b. ILWRT+ must be high, indicating that a write instruction is being executed.

c. AD15- from the address counter must be high, indicating that the memory cycle is not an indirect address cycle.

The memory-write-left-byte and memory-write-right-byte signals (MWLY+ and MWRY+) are also held high during a write transfer. These signals are held low by the FPP only during read transfers (that is, when MCAEN+A and MCAEN+B are high and ILWRT+ is low).

Instruction start flag signal MCIST- is set low when the FPP instruction is decoded (IDGO+ high) by the KKMCD+ pulse from the system clock generator. It is set low by the first KKMEM+ pulse of the instruction routine. The low MCIST- signal is steered to the MCLDA- line by the load address sub-function. The low MCLDA- signal is used to transfer the contents of the data latch to the address counter. If an indirect address is transferred from the data latch into the address counter, then AD15- is low. Under this condition, MCLDA- remains low after MCIST- has been returned to the high level. Thus, each address received from memory is transferred from the data latch into the address counter until a direct address has been loaded into the address counter.

A system reset signal from the central processor or an FPP timeout reset places the SYRST+ signal at the high level and the SYRST- signal at the low level. The high SYRST+ signal resets MCICS and the low SYRST- signal resets read request flip-flop MCRRQ of the memory control address enable sub-function. During a central processor system reset only, the SRST- signal is placed at the low level. This resets MCIST- to the high level and resets MCAEN+ to the low level.

4.6.6 Clock Control

As illustrated in figure 4-32, clock control provides the control store clock, the register clock, and the MQ clock. The trailing edge of each control store clock loads the microinstruction from the currently addressed control store location into the control store register and loads the current control store address plus 1 or a jump address into the control store address register as required to advance the FPP microprogram. The register clock is supplied to the A and B registers of the data loop and to the E and SC registers of the exponent loop. The MQ clock is supplied to the MQ register of the data loop.

When not inhibited, the control store clock generator provides a KKCS-pulse during every fourth KKMØK- period. (Refer to figure 28 for an example of KKCS- timing.) Signals which inhibit the control store clock generator are as follows:

a. MCCKI-. A low MCCKI- signal inhibits KKCS- when the final microinstruction of an instruction routine is executed. With KKCS- inhibited, this final microinstruction remains in the control store register until the start of the next instruction routine when MCCKI- is again set to the high level. MCCKI- is supplied by the memory control.

b. MCCS2- and CSMWT+. The coincidence of high MCCS2- and CSMWT+ signals inhibits the control store clock generator while waiting for the completion of a memory cycle initiated by the FPP

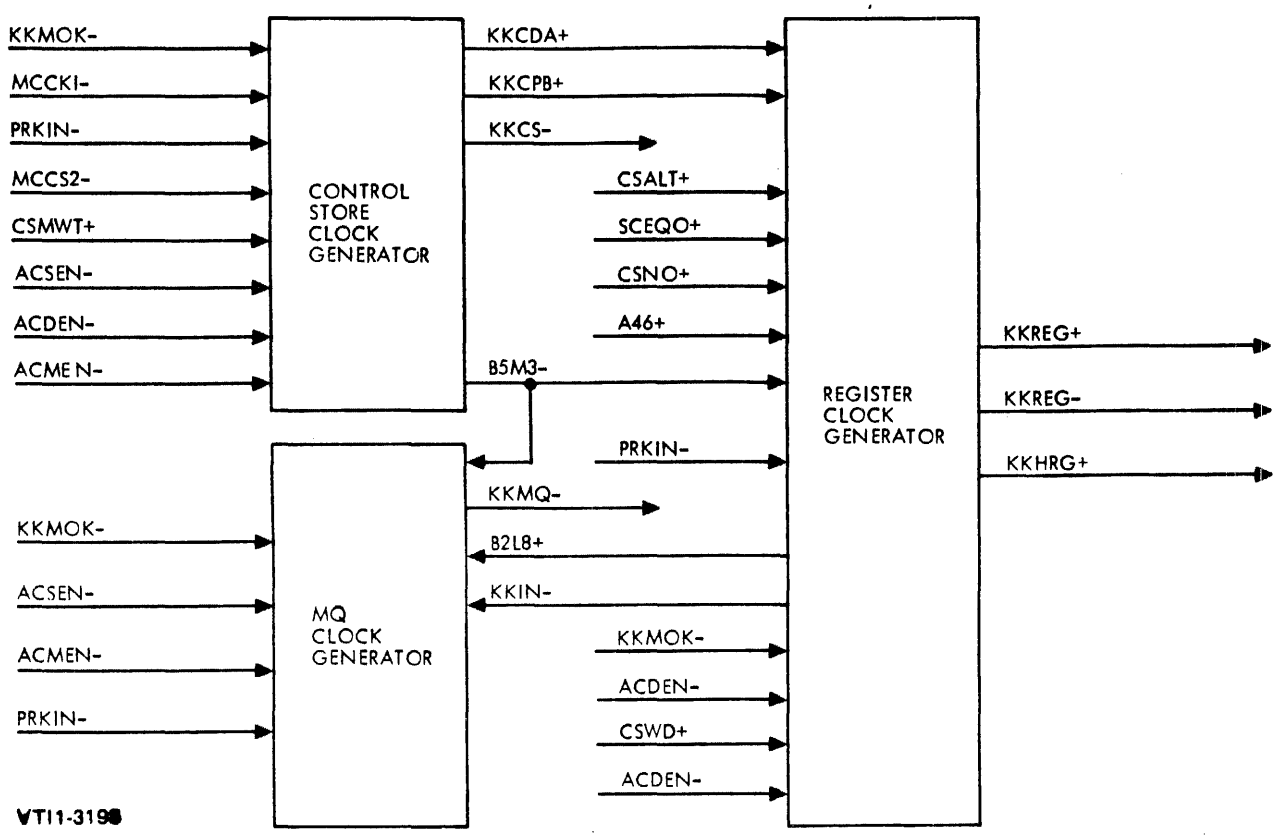


Figure 4-32. Clock Control, Block Diagram

microprogram. MCCS2- from the memory sequencer indicates that the memory cycle has not been completed while CSMWT+ from the control store register indicates that the microprogram has requested the memory cycle.

c. PRKIN-. A low PRKIN- signal from the priority control inhibits the control store clock generator when the FPP microprogram is requesting a memory cycle and the FPP does not have memory access priority.

d. ACSEN-. A low ACSEN- signal from the clock control section of arithmetic control inhibits the control store clock generator during shift operations.

e. ACDEN-. A low ACDEN- signal from the clock control section of arithmetic control inhibits the control store clock generation during the execution of the divide microinstruction.

f. ACMEN-. A low ACMEN- signal from the clock control section of arithmetic control allows the control store clock generator to run but inhibits the output gate supplying KKCS- during the execution of the multiply microinstruction.

The register clock generator has three modes of operation. When ACDEN- and ACSEN- are both high and CSWSD+ is low, the register clock generator follows the control store clock generator. In this mode, the coincidence of high KKCPA+ and KKCPB+ signals (which occurs once for each control store clock generator cycle) starts a register clock generator cycle. The timing produced in this mode is illustrated by figure 4-28. Notice that register clock pulses KKREG+ generally occur in synchronism with KKCS- pulses. However, since the register clock generator is following the control store clock generator to KKREG+ pulse occurs in synchronism with the first of a series of KKCS- pulses and one KKREG+ pulse occurs following the final KKCS- pulse of a series.

When CSWSD+ from the control store decoder is high and ACSEN- remains low, the register clock generator runs independently of the control store clock generator but continues to provide the same timing as it did when following the control store clock generator. This mode is used to provide the KKREG+ pulses during the execution of the multiply microinsturction when the control store clock generator is inhibited. This mode also provides the slower of the KKREG+ clock rates that is required during the divide microinstruction.

When ACSEN- is low, the register clock generator divides the KKMOK- rate by 2 to provide a KKREG+ rate which is double the rate provided in other modes. In other modes, each KKREG+ period is 165 nanoseconds while in this mode it is 82 nanoseconds. The fast mode implements shift operations during clock periods when no ALU function is being implemented. This occurs during some clock periods of the divide microinstruction and also occurs during the normalize and align microinstructions.

A low ACDEN- signal inhibits control store clock. ACDEN- is low during the divide microinstruction.

There are two register clock inhibits that apply to the fast mode. When an align microinstruction resides in the control store register (CSAL+ high) and the shift counter is zero (SCEQO+ high), the register clock generator is inhibited. This prevents shifting of data which is already properly aligned. Similarly, when a normalize instruction resides in the control store register (CSNØ+ high) and the data in the A register is normalized (A46+ high), the register clock generator is inhibited.

An inhibit associated with the mode in which the register clock generator follows the control store clock generator is provided by the B5M3- signal which is low during the coincidence of high MCCS2- and CSMWT+ signals. This stops the register clock generator to wait for the completion of a memory cycle initiated by the FPP microprogram. Similarly, a low PRKIN- signal stops the register clock generator as well as the control store clock generator.

KKHRG+ is a register clock generator output which is used in the arithmetic clock control for switching the status of the ACMEN- signal. KKHRG+ pulses follow KKREG+ pulses by 82.5 nanoseconds.

The MQ clock (KKMQ-) is identical to the register clock (KKREG-) except during the fraction multiply microinstruction. The MQ clock period is 82.5 nanoseconds during fraction multiply.

4.6.7 System Clock Generator

The system clock generator provides buffering and inversion of MFC-, MHC-, MCDFC-, and MØCLK+ signals received from the central processor. (See figure 4-33.) It provides retiming of the 165-nanosecond clock periods in order to eliminate transmission delays for critical cases. It also generates the memory clock pulses which time FPP memory requests and other memory control functions. In addition, it generates 82-nanosecond clock KK82± which is used by the memory sequencer.

Most of the 165-nanosecond timing within the FPP is derived from KKMCD± which is the buffered MCDFC- clock. MCDFC- is a gated version of central processor full clock MFC-. When not inhibited it is identical to MFC-. Central processor full clock MFC- is

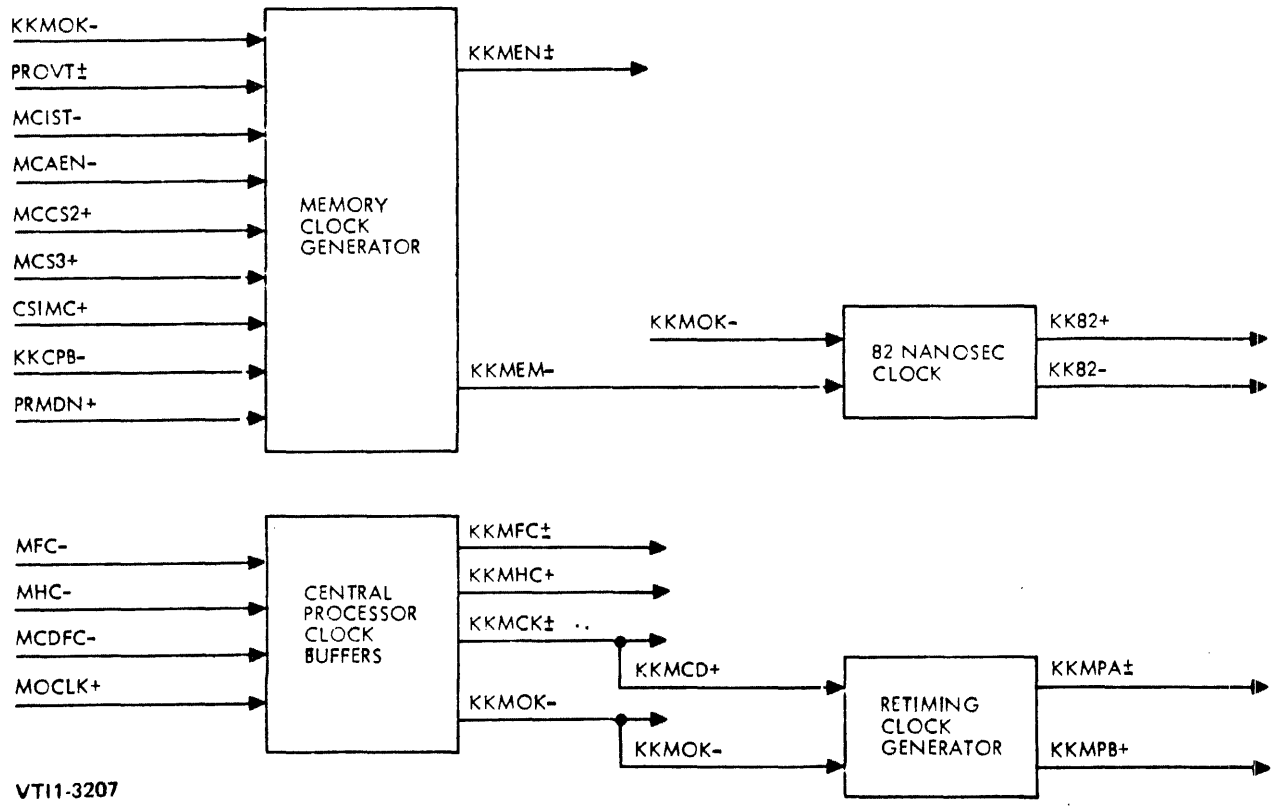


Figure 4-33. System Clock Generator, Block Diagram

inverted to provide KKMFC+ which is inverted to provide KKMFC-. Central processor half clock MHC- is inverted to provide KKMHC+.

The 41-nanosecond central processor MØCLK+ signal provides the fine clock timing for the FPP. Both phases of this clock are provided by the buffers. A single KKMØK+ output is provided and a fanout of KKMØK- signals is provided.

Because the MCDFC- signal, and consequently the KKMCD± signal, is somewhat delayed, a retiming clock generator is used to provide the KKMPA± and KKMPB+ signals which are used for timing in cases where the 165-nanosecond timing is critical. The KKMØK- signal is used to provide the required retiming as shown in figure 4-34. By timing the transitions of KKMPA and KKMPB with KKMØK-, positive-going transitions of KKMPA+ occur a short time before positive-going transitions of MCDFC-, negative-going transitions of KKMPA+ are approximately coincident with positive-going transitions of MHC-, and negative-going transitions of KKMPB+ are approximately coincident with negative-going transitions of MFC-.

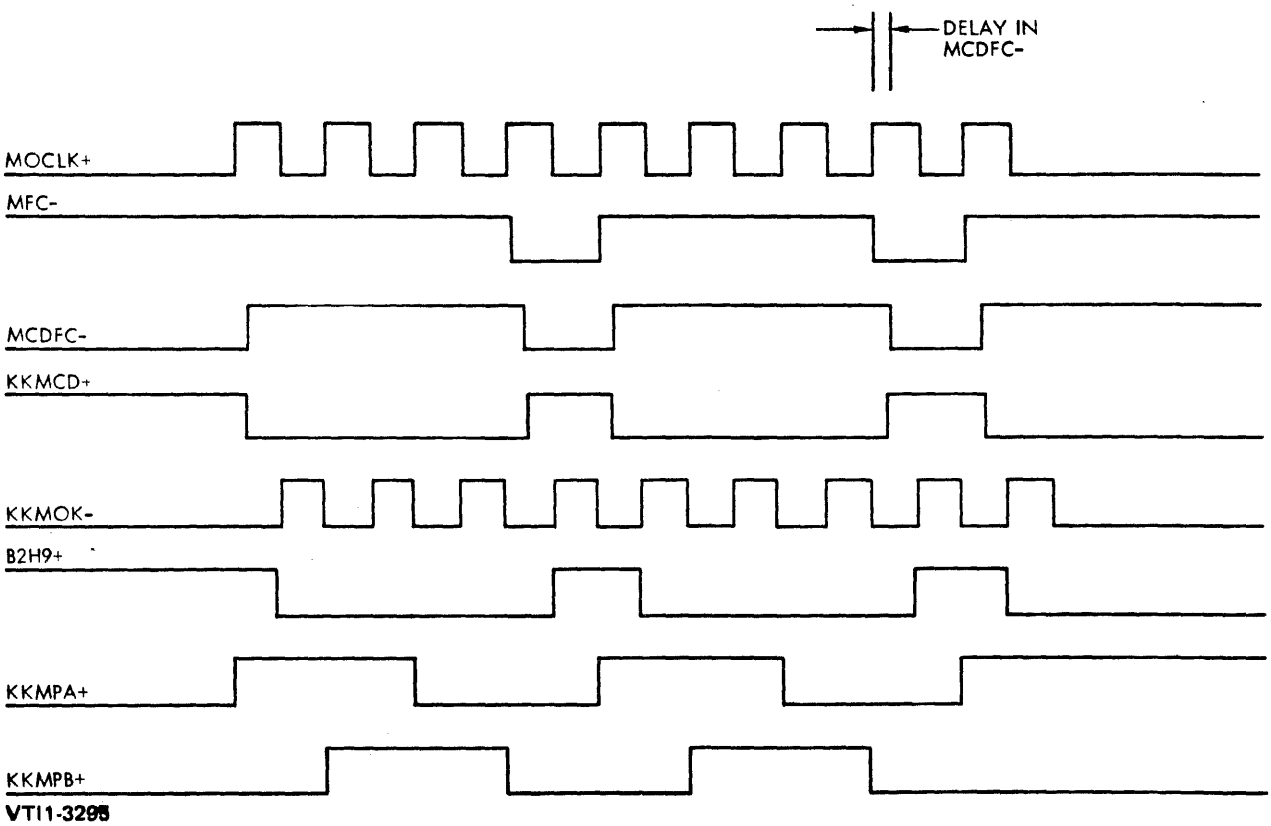


Figure 4-34. Retiming Clock Generator Waveforms

In general, memory clock KKMEM+ is timed to occur at the beginning or end of a memory cycle. At the positive-going edge of KKMEM+, a decision is made as to whether the FPP requires a memory access. If memory access is required, a memory request is initiated by memory control at the negative-going edge of KKMEM+. In order to obtain memory access, the FPP must have memory access priority. If a request from a higher priority device is pending PROUT+ from priority control is high. This inhibits the generation of a KKMEM+ pulse.

If the FPP has memory priority (PRØUT+ low), then KKMEM+ can be set by any of three items:

MCIST AND MCS3+ AND MCCS2+

or MCAEN+ AND MCICS- AND MCS3+ AND MCCS2+

or CSIMC+ AND KKCPB- AND MCCS2+.

In the first term, MCIST+ indicates the start of a FPP instruction and MCS3+ and MCCS2+ occur at the end of the first address fetch memory cycle. If an address fetch (DL15+ high) or an operand fetch (MCRRQ+ high) is required, then the first KKMEM+ pulse clocks the memory address into the address register, sets MCAEN+, and initiates a memory cycle. During a non-write instruction without indirect addressing, the first KKMEM+ pulse clocks the address register without initiating a memory cycle. During FPP instructions with indirect addressing, the second term sets KKMEM+ at the end of each address fetch memory cycle. The resulting KKMEM+ pulse clocks the address into the address register and, if required (DL15+ high or MCRRQ+ high), initiates the next memory cycle. The third term sets KKMEM+ for control store initiated memory cycles.

If the FPP loses memory priority due to a PMA or DMA request (PRØUT+ high), then the setting of KKMEM+ is delayed until the PMA or DMA memory cycles are completed (PRØUT+ low and PRMDN high).

4.6.8 Arithmetic Clock Control Logic

As shown in figure 4-35, the arithmetic control clock logic provides shift-enable signal ACSEN-, multiply-enable signal ACEM-, and divide-enable signal ACDEN-. These signals condition the clock control to provide clock rates or clock inhibits required during certain FPP microinstructions. All transitions of ACSEN- and ACDEN- (except transitions occurring during a system reset) occur at the leading edge of KKREG+. Transitions of ACMEN- occur at the leading edge of KKHRG+.

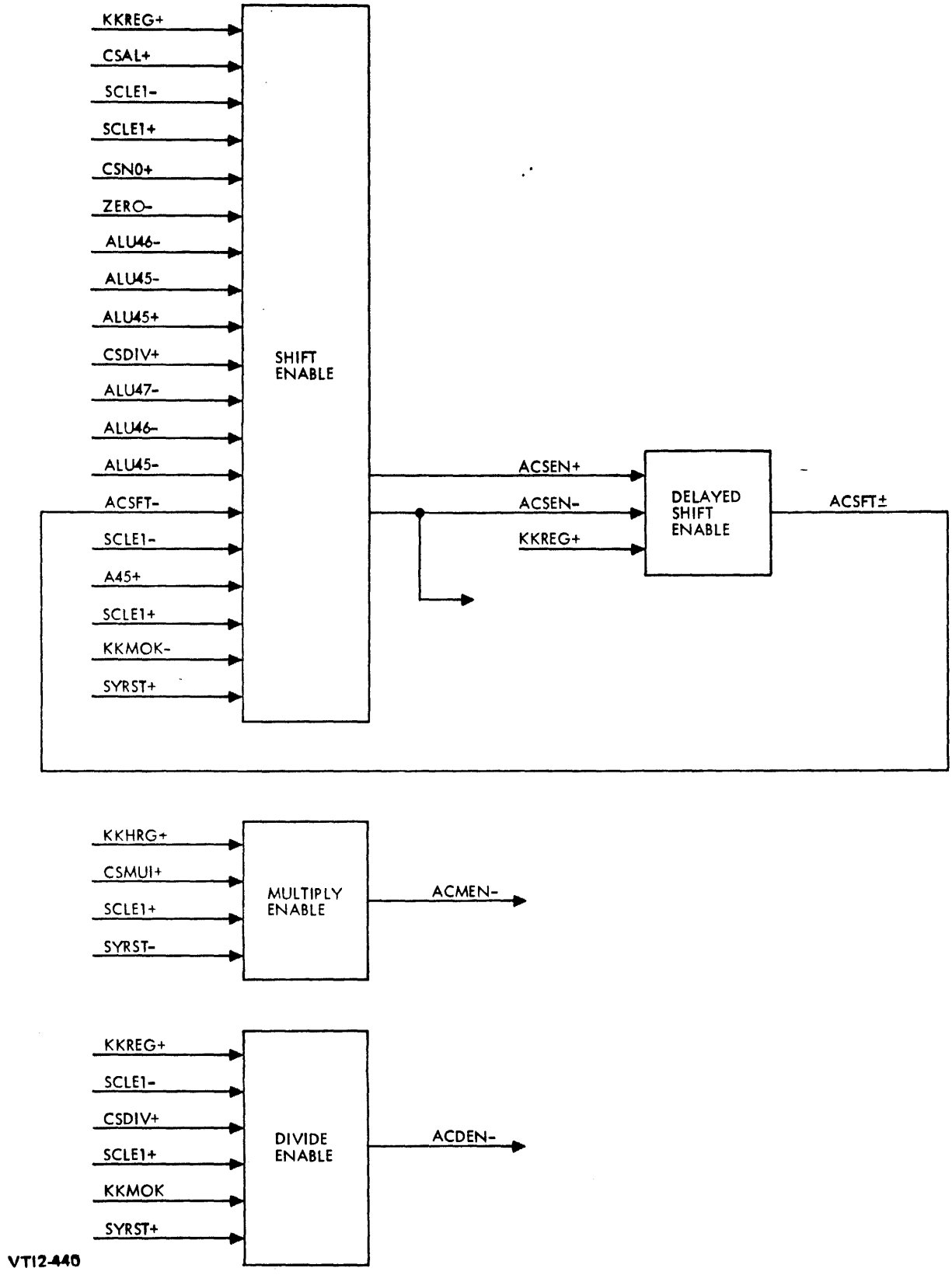


Figure 4-35. Arithmetic Clock Control, Block Diagram

ACSEN- is used during three types of microinstructions to produce 82-nanosecond clocks for shifting and to inhibit the trailing edge of the control store clock until the shifting operating is complete. During an alignment microinstruction (CSAL+ high) ACSEN- is set low during the period when the smaller operand is being aligned with the larger operand prior to addition or subtraction. The number of shifts is determined by the number in the shift counter which is decremented by 1 at each shift clock time. To set ACSEN- low, the shift-count-less-than-or-equal-to-1 signal (SCLE1-) must be high, indicating that at least two shifts are required. (If only one shift is required, the microinstruction has a duration of one clock period and the required shift is produced by the KKREG+ pulse at the end of this clock period. If no shift is required, the microinstruction has a duration of one clock period and the KKREG+ pulse at the end of the clock period is inhibited by the clock control.) Once set, ACSEN- remains set until the final shift pulse occurs. This shift pulse is identified by a high SCLE1+ signal. Figure 4-36 illustrates the timing of the align microinstruction. Notice that the positive-going edge of the KKCS- pulse (which terminates the microinstruction period) is coincident with the negative-going edge of the final shift pulse (KKREG+).

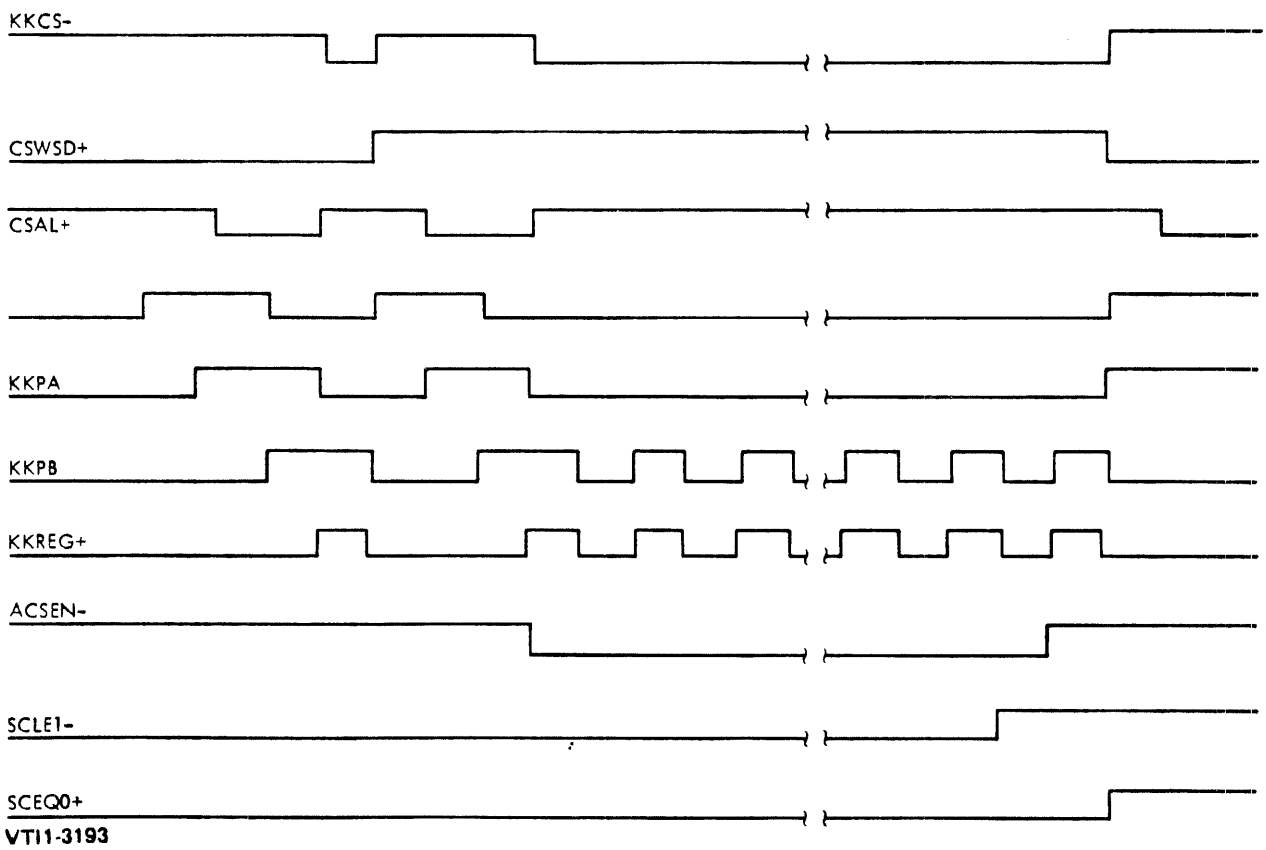


Figure 4-36. Align Microinstruction Timing

During a normalize microinstruction (CSNØ+ high), is set low if ZERØ-, A46-, and A45- are all high. The high ZERØ- signal indicates that the result is not zero and can thus be normalized. The high A46- and A45- signals indicate that the two most significant magnitude bits of the result fraction are zeros so that at least two shifts are required to normalize the result. (The cases where one shift or no shift is required are handled similarly to the corresponding cases during the align microinstruction.) Once set, ACSEN- is reset by the KKREG+ pulse occurring after A45+ becomes high. Thus, the microinstruction is terminated when the most significant binary one of the fraction is shifted from bit position 45 to bit position 46.

In the case of the division microinstruction (identified by a high CSDIV+ signal), the status of ACSEN- is manipulated in accordance with the status of the three most significant bits of the ALU and in accordance with the status of bit 45 of the A register. An exception to this occurs in the case of the final iteration, when ACSEN- is reset to the high level (or remains high) because SCLE1+ is high. The coincidence of high ALU47-, ALU46-, and ALU45- signals indicates a positive difference whose two most significant magnitude bits are zeros. Since the most significant magnitude bit of the divisor fraction is always a one, it follows that at least two left shifts of this difference are required before another positive difference can be obtained. Thus, this coincidence of high levels is used to set ACSEN- low and enable the fast (82-nanosecond) shift clock rate. This rate is maintained until a one is shifted into the most significant magnitude position (A46) of the A register. At the time of this shift, ACSEN- is reset to the high level as a result of A45+ being high (that is, as a result of the 1 that is being shifted from bit position 45 to bit position 46).

Figure 4-37 provides an example of ACSEN- timing for a particular fraction division.

ACSFT is a delayed version of ACSEN. While transitions of ACSEN occur at the positive-going edge of KKREG+, transitions of ACSFT occur at the negative-going edge of KKREG+. ACSFT is used to prevent ACSEN from simultaneously receiving both set and reset inputs.

In the case of the CSDIV+ microinstruction, it is the ACDEN- signal which inhibits the trailing edge of KKCS- until the final interation. With CSDIV+ high, ACDEN- is set low at the positive-going edge of the first KKREG+ pulse and remains low until the positive-going edge of the final KKREG+ pulse when it is reset to the high level because SCLE1+ is high.

During the main multiply microinstruction (identified by a high CSMUI+ signal), ACMEN- is set low by the first KKHRG+ pulse and is reset high by the KKHRG+ pulse occurring during the final iteration period (identified by a high SCLE1+ signal). (See figure 4-38 for ACMEN- timing.) The low ACMEN- signal enables

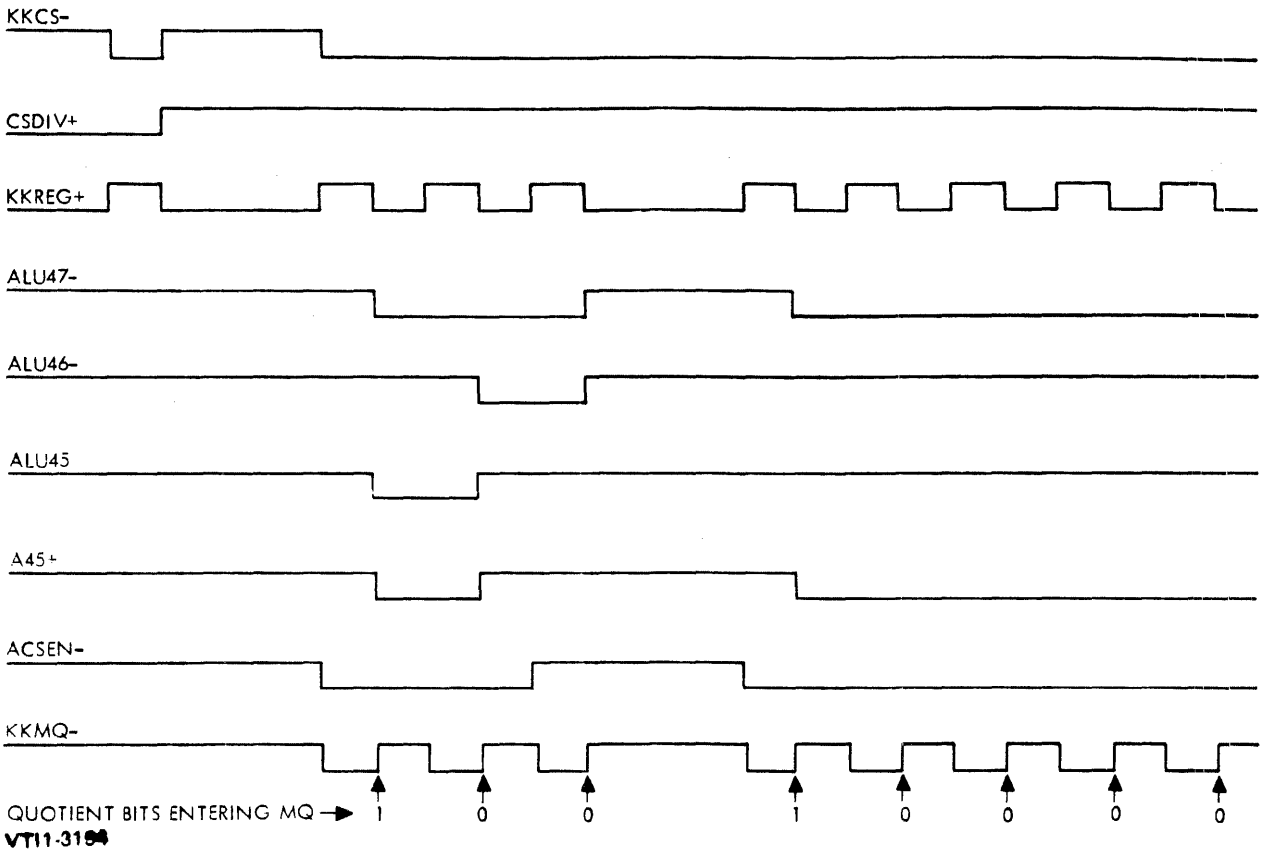


Figure 4-37. Timing at Start of CSDIV Microinstruction $\left(\begin{matrix}0.11111100--\\0.11100000--\end{matrix}\right)$

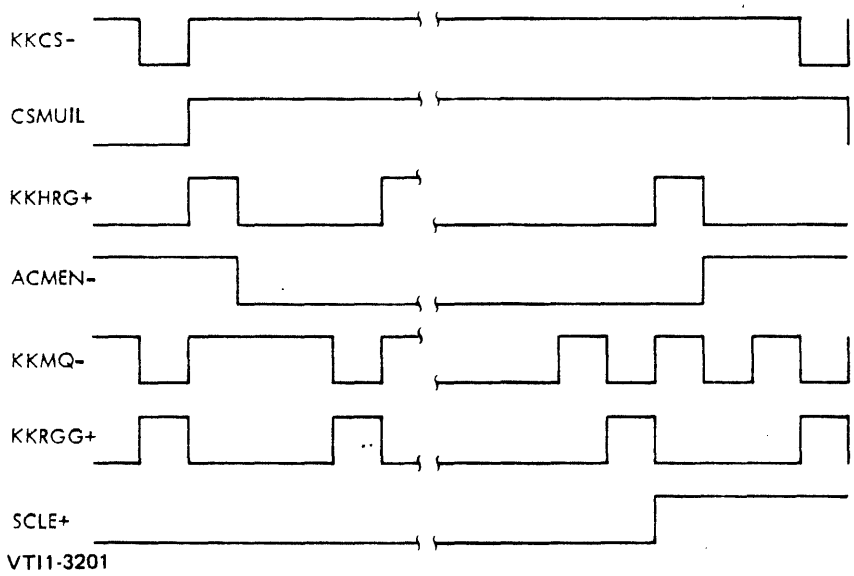


Figure 4-38. Timing of ACMEN-

KKMQ- pulses to occur at the fast (82-nanosecond) rate as required to shift the multiplier two bit positions to the right during each iteration. AMEN- inhibits KKCS- pulses without stopping the KKCS- clock generator. Thus after ACMEN- resets to the high level during the final iteration period, a KKCS- pulse occurs at the normal time and terminates the microinstruction.

System reset clears ACSEN, ACDEN, and ACMEN.

4.6.9 Data Latch and Address Output

The data latch provides buffer storage for each instruction, address, or operand word that is transferred from memory to the FPP. Before operand transfers can begin the direct address of the first operand must be received from memory and loaded from the data latch into the memory address counter. The memory address counter is then incremented under the control of the FPP microprogram following each operand word transfer so that successive operand words are transferred to or from consecutive memory locations. As shown in figure 4-39 the address held in the memory address counter reaches the memory address bus via inverters and gates.

Each word transferred from memory to the FPP is loaded into the data latch in response to a low MCDLE- signal from the memory sequencer. Each address word received from memory is transferred from the data latch to the memory address counter in response to a low MCLDA- signal from the memory control. The address held in the memory address counter is gated to the memory address by high MCAEN+A and MCAEN+B signals from the memory control. Incrementing of the memory address count at the positive-going edge of the KKMEM+ pulse from the system clock generator is enabled by a high CSINC+ signal from the control store decoder.

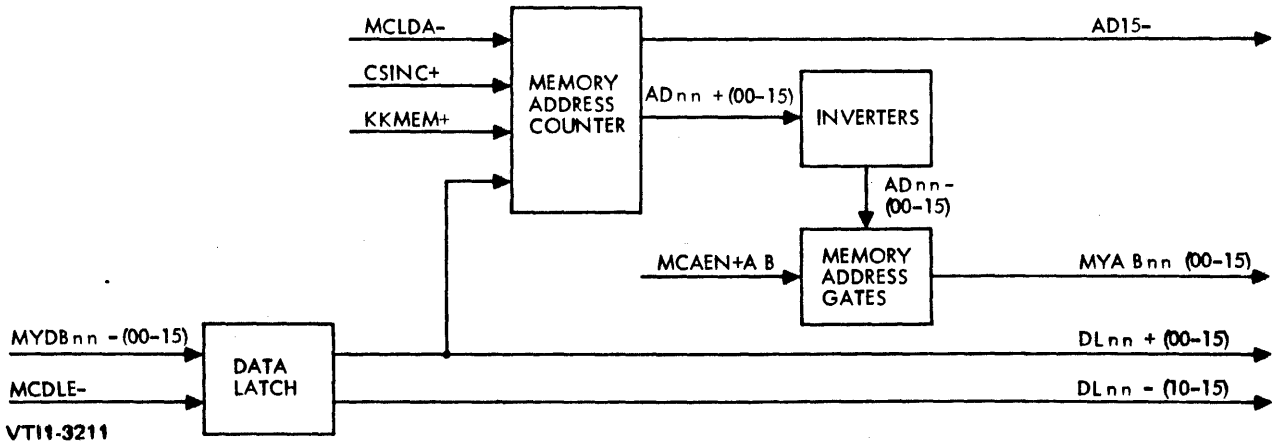


Figure 4-39. Data Latch and Address Output, Block Diagram

4.6.10 Instruction Latch, Instruction Register, and Instruction Decoder

The provision of an instruction latch and an instruction register allows the pipelining of instructions to the FPP. Each FPP instruction is decoded as it resides in the data latch. It is then moved to the instruction latch. While the instruction resides in the instruction latch, the direct address of the first operand word is obtained from memory and loaded into the address counter. In the case of a non-write instruction, the first operand word is obtained from memory and stored in the data latch. These functions are performed independently of the FPP microprogram so that they can be performed while the microprogram for the previous FPP instruction is still running. (Each FPP instruction jams the central processor microprogram at a NOP location until it has completed its final operand word transfer. Thus, there is no contention between the operand transfers associated with one FPP instruction and the fetching of the next instruction and direct operand address.)

Three signals (CDREN+, CEND+, and CAEN+) enable the FPP instruction decoder (see figure 4-40). CDREN+ high indicates a primary decode request from the central processor control store. CEND+ high indicates no interrupt requests pending. CAEN+ high indicates that the instruction in the central processor instruction buffer (and the FPP data latch) is in the standard instruction set or FPP instruction set. (CAEN+ low indicates an instruction in an extended instruction set decoded and executed under control of the WCS.) CAEN+ is clocked into a flip-flop (B5T9+) at the positive-going edge of central processor clock MCDFC-. The instruction decoder is enabled when CDREN+, CEND+, and B5T9+ are high. When the decoder is enabled and an FPP instruction resides in the data latch, the IDGO+ signal is placed at the high level. An FPP instruction is identified by code 1000 1011 in bits 8 through 15 (where bit 15 is the MSB). With IDGO+ high, the eight least significant bits of the instruction are loaded into the instruction latch at the positive-going edge of clock MCDFC-. These bits are received on data latch lines DLnn+ (0-7).

When the final microinstruction of an FPP instruction routine reaches the control store register, the CSDN+ signal from the control store decoder switches to the high level. If the direct address of the first operand of the next FPP instruction has not yet been obtained, then control store clocks are inhibited until this function has been completed. Thus, the final microinstruction of the completed FPP instruction remains in the control store register and the CSDN+ signal remains high. When the direct address of the first operand word for the next FPP instruction is received and loaded into the data latch, control store clocks are again enabled. The trailing edge of the next control

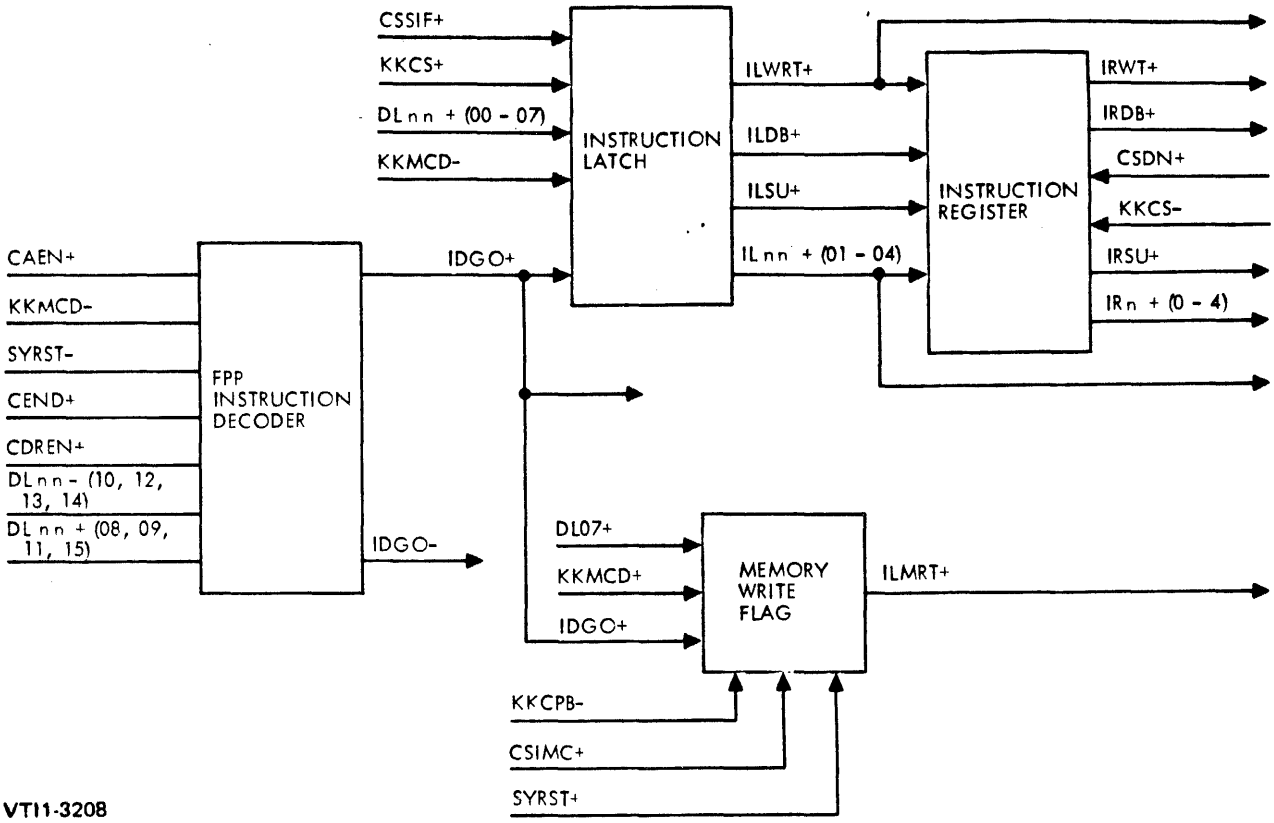


Figure 4-40. Instruction Latch, Instruction Register and Instruction Decoder, Block Diagram

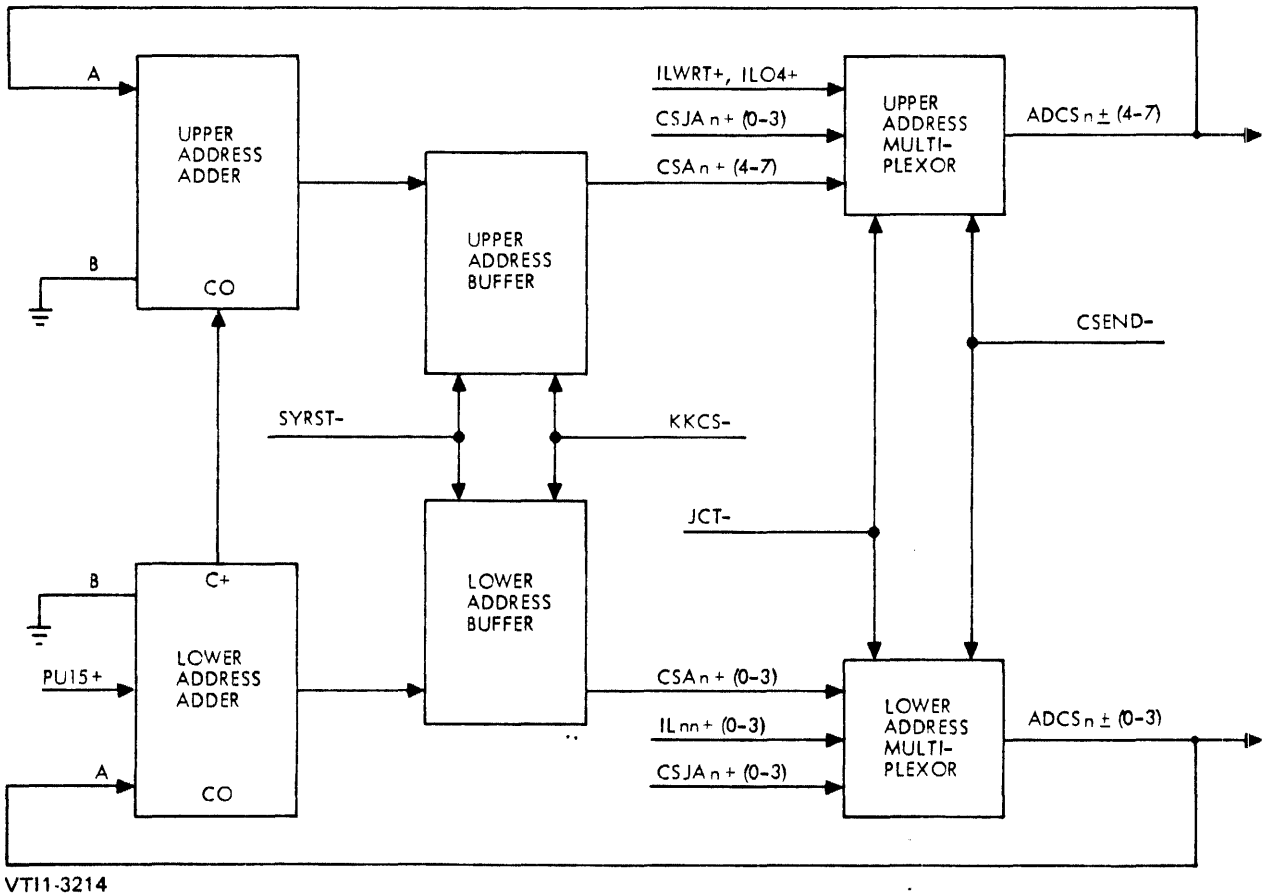
store clock pulse (KKCS-3) loads the instruction from the instruction latch into the instruction register. Bits ILnn+ (00-04) and bit ILWRT+ are supplied via the control store address multiplexor to the control store address lines at this time and thus specify the location of the first microinstruction of the next FPP instruction microprogram. This microinstruction is loaded into the control store register by the same KKCS- transition which loads the instruction latch into the instruction register. The instruction register presents the instruction code bits to the constant storage PROM and to the arithmetic control logic for use during the execution of the microprogram.

The three most significant bits of each 8-bit instruction code are designated to identify the functions they specify. In terms of the instruction latch, these are the ILWRT+, IBDB+, and ILSU+ bits. A high ILWRT+* signal specifies a write instruction (that is an FPP instruction during which operand data is transferred from the FPP to the memory). A high ILDB+ signal specified a double precision instruction. A high ILSU+ bit specifies a subtraction instruction.

Special duplicate storage (memory write flag) is provided for the most significant bit, which specifies a write or non-write instruction. The memory write flag is stored at the same time that the instruction is loaded into the instruction latch. However, the memory write flag is reset to the low level by the coincidence of high KKCPB- and CSIMC+ signals associated with the first memory request initiated by the FPP microprogram. The memory write flag (ILMRT+) is used in the memory control logic in connection with obtaining the direct address of the first operand during a write instruction.

4.6.11 Control Store Address Loop

The control store address loop (figure 4-41) provides the control store address storage and manipulation required to cause the microinstructions of a routine to be executed in the appropriate order. The elements of the control store loop include the upper and lower address adders, buffers, and multiplexors. The starting control store address for a floating point processor operation is obtained from the instruction latch. As indicated by table 4-4, the instruction latch signals are connected to the



VTI1-3214

Figure 4-41. Control Store Address Loop, Block Diagram

Table 4-4. Control Store Address Multiplexor Outputs

Control Inputs		Outputs							
CSEND-	JCT-	ADCS7+	ADCS6+	ADCS5+	ADCS4+	ADCS3+	ADCS2+	ADCS1+	ADCS0+
L	H	ILWRT+	L	L	IL04+	IL03+	IL02+	IL01+	IL00+
H	H	CSA7+	CSA6+	CSA5+	CSA4+	CSA3+	CSA2+	CSA1+	CSA0+
L	L	CSJA3+	CSJA2+	CSJA1+	CSJA0+	CSA3+	CSA2+	CSA1+	CSA0+
H	L	CSA7+	CSA6+	CSA5+	CSA4+	CSJA3+	CSJA2+	CSJA1+	CSJA0+

Note: L = low; H = high

control store address multiplexor outputs when CSEND- is low and JCT- is high. Under this condition, the five least significant bits ILnn+(0-4) from the instruction latch are connected to the four least significant control store address multiplexor output lines, ADCS0+ through ADCS4+, while the most significant bit from the instruction latch, ILWRT+, is connected to the most significant control store address multiplexor output line. Hard-wired low (L) signals are simultaneously connected to the ADCS5+ and ADCS6+ control store address multiplexor output lines.

In addition to the starting address selection, there are three other selections that can be implemented by the control store address multiplexor. When both CSEND- and JCT- are high, the eight address signals from the upper and lower address buffer are selected for connection to the control store address multiplexor output lines. Each KKCS- pulse from the clock control logic clocks the upper and lower adder outputs into the upper and lower address buffers respectively. The adders function to increment the address being received from the control store address multiplexor by 1. Thus, as long as both CSEND- and JCT- are high, consecutive control store locations are addressed in sequence. A low JCT- signal indicates that a jump condition is satisfied. In this case, only half of the control store address is obtained from the buffer register while the other half is obtained from the jump address field of the word currently held in the control store register. If CSEND- is high then the upper address is obtained from the buffer while the lower address is obtained from the jump address field (CSJA0+ through CSJA3+). If CSEND- is low, then the upper address is obtained from the jump address field while the lower address is obtained from the buffer.

4.6.12 Jump Condition Multiplexor

Complement jump-condition-true signal JCT- is obtained from an 8-to-1 multiplexor. When the selected input of the multiplexor is high, the output is low as required to produce the jump. The multiplexor selection inputs are the CSJCn+(0-2) signals from the jump condition field of the word currently held in the control store register. Table 4-5 summarizes the jump condition selections.

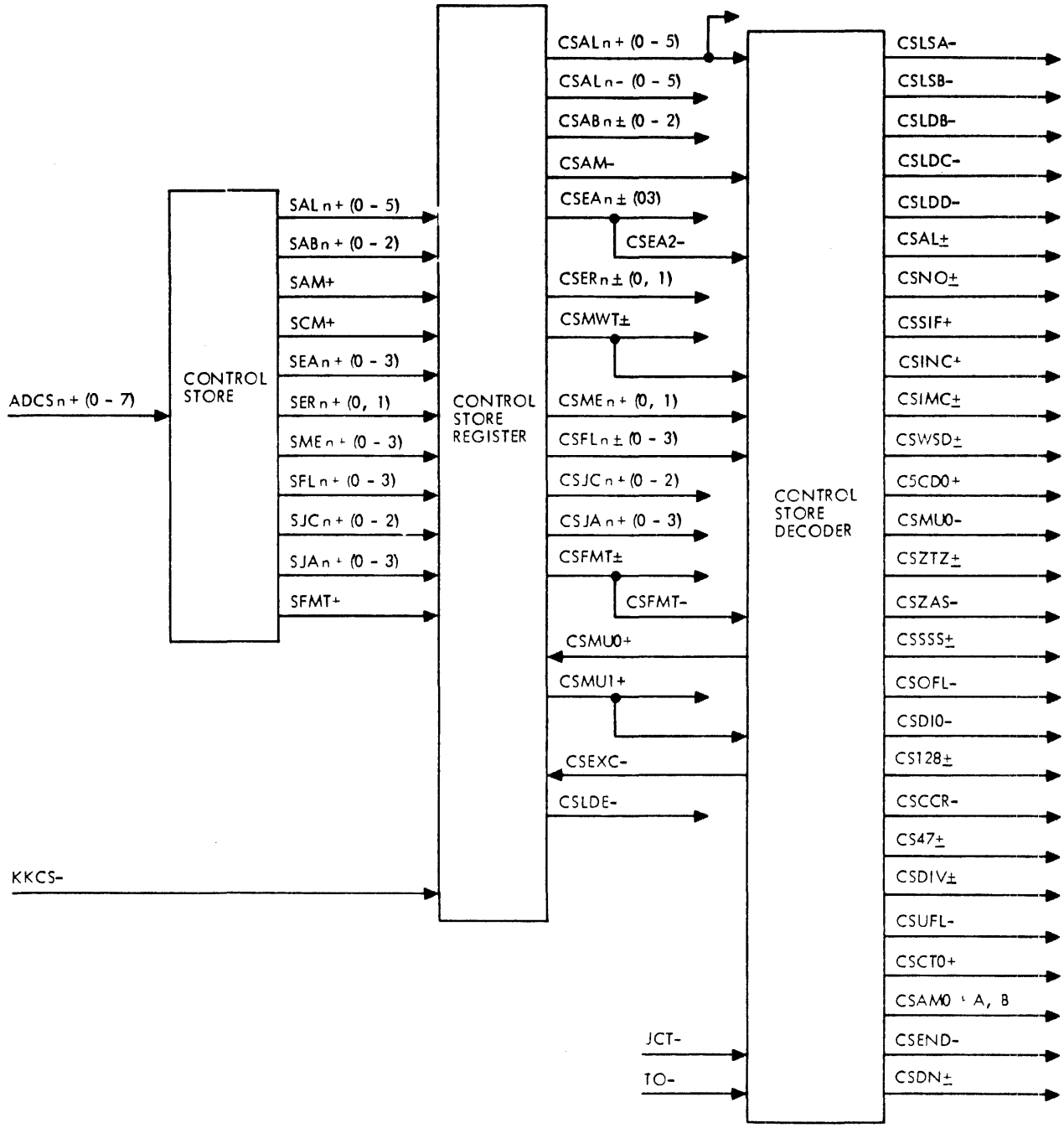
4.6.13 Control Store Memory, Register, and Decoder

The FPP microprogram is contained in a 16-bit, 256-word read-only memory (control store). Associated with the control store are a control store register, which holds each microinstruction while it is being executed, and a control store decoder, which decodes certain fields of each microinstruction held in the control store register. (See figure 4-42.) The trailing edge of each KKCS- pulse from the clock control loads the microinstruction from the control store location currently addressed by the ADCSn+ (0-7) signals (from the control store address loop) into the control store register.

Table 4-5. Jump Condition Selections

Jump Condition			Condition Selected	Function
CSJC2+	CSJC1+	CSJC0+		
L	L	L	Hard-wired Low	No jump
L	L	H	Hard-wired High	Unconditional jump
L	H	L	ECRY+	Jump if exponent carry carry
L	H	H	ZERO+	Jump if ZERO flip-flop is set
H	L	L	ACZ47+	Jump if accumulator is negative or zero
H	L	H	ASEBS+	Jump if AS = BS
H	H	L	E7+	Jump if MSB of exponent is 1
H	H	H	TO+	Jump if time-out flip-flip is set

Note: L = low; H = high



VTI2-442

Figure 4-42. Control Store Memory, Register, and Decoder, Block Diagram

Table 4-6 lists the fields of each microinstruction and lists the signals associated with each field that are supplied by the control store and the control store register.

The control store decoder provides decoding of various IN, ALU, MEN, and FLAG codes as summarized in table 4-7. For information about the significance of each of these decoding signals, refer to table 4-2.

When the decode signal CSEND- is low and the complement jump-condition-true signal (JCT-) from the jump condition multiplexor is high, the CSDN+ signal is placed at the high level. This signal inhibits KKCS- pulses at the end of an FPP instruction microprogram until the next FPP instruction is decoded.

There are two decode signals that are loaded into the control store register. CSMU0+ is loaded into the CSMU1+ stage of the control store register while CSEXC- is loaded into the CSLDE- stage. CSMU1+ identifies the main multiply microinstruction which is executed following the multiply-start microinstruction which contains the CSMU0+ code. The only function performed by CSEXC- is to set the CSLDE- signal low during the next microinstruction period. This allows two FLAG codes to be used to specify conditions for a single microinstruction, the CSEXC- code being used in the preceding microinstruction which does not require any information from the FLAG code field.

Complement decoding signal CSIMC- which is used to initiate a memory request is gated by TØ- from the priority control. This allows the MEM field of one of the microinstructions of the timeout routine to be used to provide a wait-for-memory-done function without initiating a memory request.

4.6.14 I/O Data Multiplexors

The I/O data multiplexors are used to connect operand data from the data latch to the MQ register and E register of the data loop. They are also used to connect result data from the ALU in the data loop and from the E register to the memory data bus of the central processor. As shown in figure 4-43, a data source selection multiplexor selects one of eight data sources while a conditional inversion multiplexor selects either the inverted or non-inverted outputs of the data source selection multiplexor as appropriate. Data words which contain sign bits are inverted when the operand or result being transferred is negative. This provides the required conversion between the format used to store operands in memory and the absolute format required and supplied by the FPP.

Table 4-8 lists each of the eight data words that can be selected by the data source selection multiplexor. Notice that two of these words (selections LLL and LHH) are obtained from the data latch (DLnn+). Selection LHH is used when transferring the first

Table 4-6. Control Store and Control Store Register Outputs

Microinstruction Field	Control Store Signals	Control Store Register Signals
ALU or IN/IO	SALn+ (0-5)	CSALn± (0-5)
AB	SABn+ (0-2)	CSABn± (0-2)
AC	SAM+, SCM+	CSAM-, CSCM±
EADD	SEAn+ (0-3)	CSEAn± (0-3)
EREG	SERn+ (0,1)	CSERn± (0,1)
MEM	SMEn+ (0,3)	CSMEn+ (0,1), CSMWT±
FLAG	SFLn+ (0-3)	CSFLn± (0-3)
JCOND	SJCn+ (0-2)	CSJCn+ (0-3)
JADD	SJAn+ (0-3)	CSJAn+ (0-3)
FMT	SFMT+	CSFMT±

Table 4-7. Control Store Decoder Signals

Microinstruction Field	Control Store Register Bits				Decoder Signal Activated
IN		CSAL5+	CSAL4+	CSAL3+	
		L	L	H	CSLSA-
		L	H	L	CSLSB-
		H	L	H	CSLDB-
		H	H	L	CSLDC-
		H	H	H	CSLDD-
ALU (With CSWSD- signal from MEM field LOW)		CSAL2+	CSAL1+	CSAL0+	
		L	L	H	CSAL±
		L	H	L	CSNØ±
MEM		CSMWT+	CSME1+	CSME0+	
		L	H	L	CSWSD±
		H	L	L	CSIMC±
		H	L	H	CSINC+, CSIMC±
		H	H	H	CSSIF+
FLAG	CSFL3+	CSFL2+	CSFL1+	CSFL0+	
	L	L	L	H	CSCP0+
	L	L	H	L	CSMU0±
	L	L	H	H	CSZTZ±
	L	H	L	L	CSZAS-
	L	H	L	H	CSSS±
	L	H	H	L	CSØFL-

Table 4-7. Control Store Decoder Signals (continued)

Microinstruction Field	Control Store Register Bits				Decoder Signal Activated
FLAG (cont'd)	CSFL3+	CSFL2+	CSFL1+	CSFL0+	
	L	H	H	H	CSDI0-
	H	L	L	L	CS128±
	H	L	L	H	CSEXC-
	H	L	H	L	CSCCR-
	H	L	H	H	CSC47±
	H	H	L	L	CSDIV-
	H	H	L	H	CSUFL-
	H	H	H	L	CSCT0+

Note: L = Low; H = high

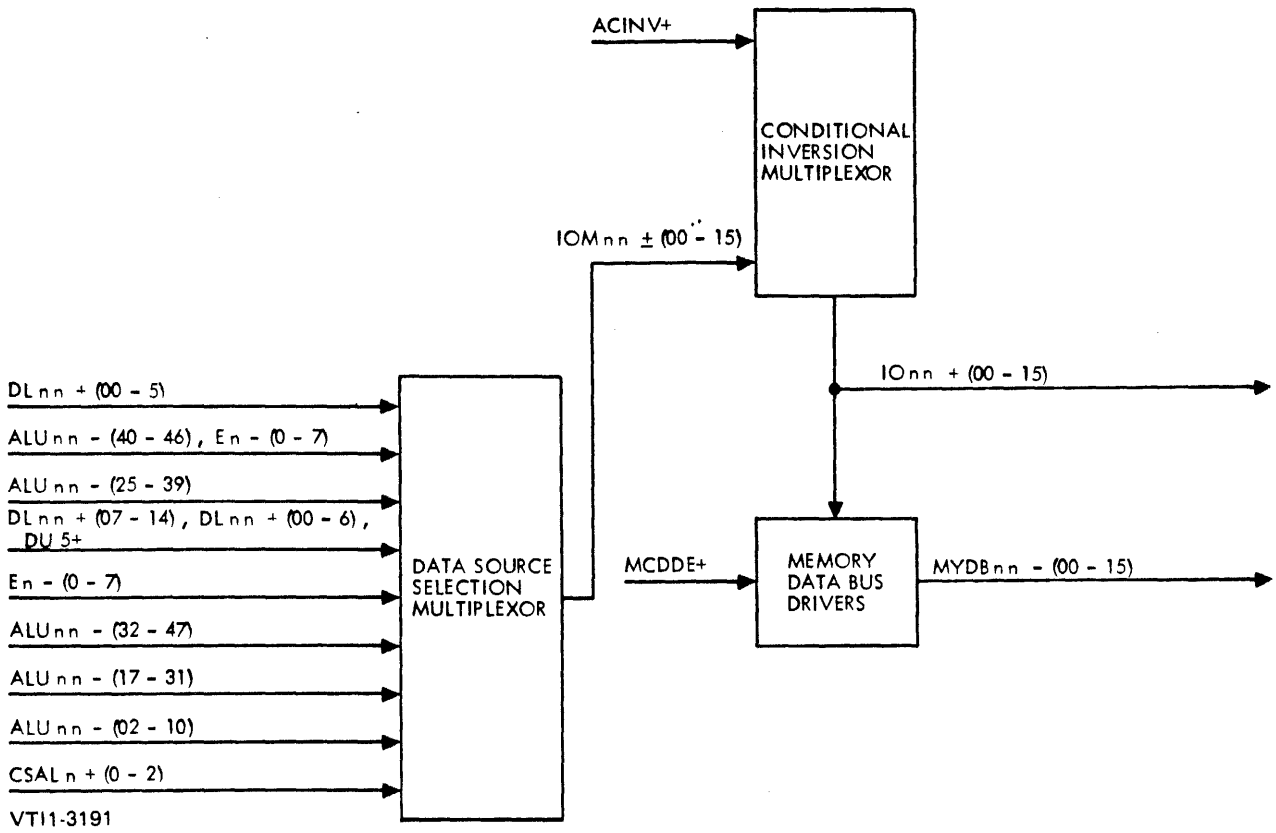


Figure 4-43. I/O Data Multiplexors, Block Diagram

word of a single precision number from the data latch to the data loop. This selection transposes the positions of the exponent and high-fraction fields of the word. Selection LLL is used when transferring all other operand words from the data latch to the data loop.

The remaining five selections are associated with the transfer of result words from the data loop to memory. Selections LLH and LHL are associated respectively with the transfer of words 1 and 2 of a single precision number while selections HLL through HHH are associated respectively with the transfer of the four words of a double precision number.

The data source selection is controlled by the IO field (CSAL0+ through CSAL2+) of the microinstruction word currently held in the control store register.

A conditional inversion is introduced in response to a high ACINV+ signal. (The I/O data multiplexor is the boundary between the negative-true data-loop/exponent-loop domain and the positive-true IOM/IO domain. Therefore, a level inversion to convert between negative-true and positive-true occurs when ACINV+ is low. The high ACINV+ level eliminates this level inversion and produces an inversion of ones to zeros and zeros to ones).

Table 4-8. I/O Data Multiplexor Selections

Selection Inputs - CSALn+ 2 1 0	Source Data Selection							
	IOM15-	IOM14-	IOM13-	IOM12-	IOM11-	IOM10-	IOM09-	IOM08-
L L L	DL15+	DL14+	DL13+	DL12+	DL11+	DL10+	DL09+	DL08+
L L H	PU+	E7-	E6-	E5-	E4-	E3-	E2-	E1-
L H L	PU+	ALU39-	ALU38-	ALU37-	ALU36-	ALU35-	ALU34-	ALU33-
L H H	DL15+	DL06+	DL05+	DL04+	DL03+	DL02+	DL01+	DL00+
H L L	PU+	PU+	PU+	PU+	PU+	PU+	PU+	PU+
H L H	ALU47-	ALU46-	ALU45-	ALU44-	ALU43-	ALU42-	ALU41-	ALU40-
H H L	PU+	ALU31-	ALU30-	ALU29-	ALU28-	ALU27-	ALU26-	ALU25-
H H H	PU+	ALU16-	ALU15-	ALU14-	ALU13-	ALU12-	ALU11-	ALU10-
	Source Data Selection (continued)							
2 1 0	IOM07-	IOM06-	IOM05-	IOM04-	IOM03-	IOM02-	IOM01-	IOM00-
L L L	DL07+	DL06+	DL05+	DL04+	DL03+	DL02+	DL01+	DL00+
L L H	EO-	ALU46-	ALU45-	ALU44-	ALU43-	ALU42-	ALU41-	ALU40-
L H L	ALU32-	ALU31-	ALU30-	ALU29-	ALU28-	ALU27-	ALU26-	ALU25-
L H H	DL14+	DL13+	DL12+	DL11+	DL10+	DL09+	DL08+	DL07+
H L L	E7-	E6-	E5-	E4-	E3-	E2-	E1-	E0-
H L H	ALU39-	ALU38-	ALU37-	ALU36-	ALU35-	ALU34-	ALU33-	ALU32-
H H L	ALU24-	ALU23-	ALU22-	ALU21-	ALU20-	ALU19-	ALU18-	ALU17-
H H H	ALU09-	ALU08-	ALU07-	ALU06-	ALU05-	ALU04-	ALU03-	ALU02-

Note: L = low; H = high

During the transfer of results from the data loop to the memory, the IOnn+ outputs from the conditional inversion multiplexor are gated to the memory bus (MYDBnn-) in response to a high MCDDE+ signal from the memory control.

4.6.15 Data Loop

The data loop performs those data processing operations that involve the fraction portions of operands. As illustrated in figure 4-44, the data loop includes the following components:

a. ALU. This is a 48-bit wide general purpose arithmetic-logic unit with a 2-bit extension that is used only during execution of the multiplication instructions. The particular logic or arithmetic function provided by the ALU depends upon signals supplied to its four selection lines (S0 through S3), to its mode (M) line, and to its carry input (C_n) line. Twelve combinations of inputs are used in the FPP. These inputs and the resultant functions are summarized in table 4-9.

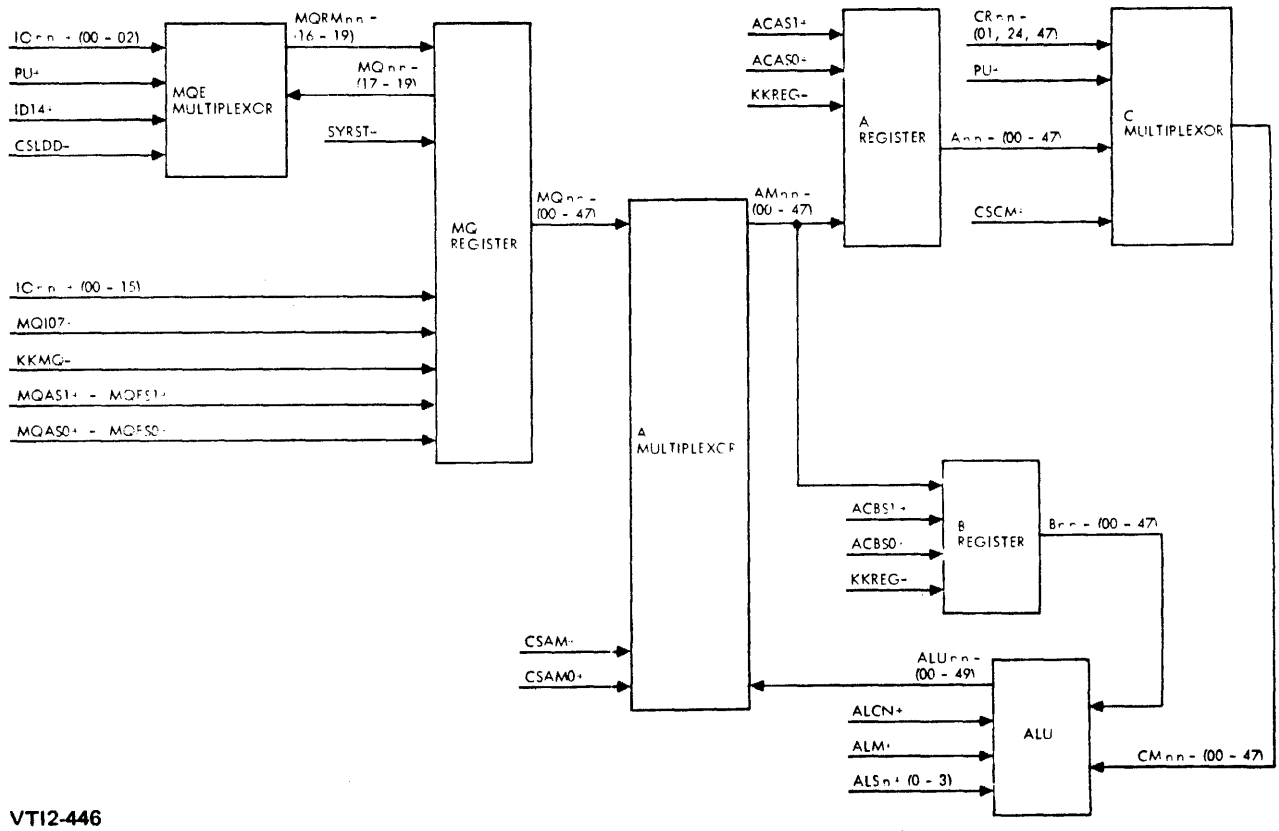


Figure 4-44. Data Loop, Block Diagram

Table 4-9. ALU Functions

Input Code						
S3 ALS3+	S2 ALS2+	S1 ALS1+	S0 ALS0+	M ALM+	C_n ALCN+	Function
L	L	L	L	H	L	F = $\overline{A}$ (Logic complementation of A)
L	L	H	H	H	L	F = 1
L	H	L	H	H	L	F = $\overline{B}$ (Logic complementation of B)
L	H	H	L	L	H	F = A MINUS B
H	L	L	H	L	L	F = A PLUS B
H	L	H	L	H	L	F = B
H	H	L	L	L	L	F = A PLUS A = 2A
H	H	L	L	H	L	F = 0
H	H	H	H	H	L	F = A
L	L	H	H	L	L	F = MINUS 1

Note: L = low; H = high

b. MQ Register. The MQ (multiplier/quotient) register is a 48-bit shift register with parallel-entry, parallel-readout, shift-left, and shift-right capabilities. It holds the multiplier during multiplication and the quotient at the end of division (See figure 4-45.) All operands received by the data loop from memory are initially assembled in the MQ register. Various sections of the MQ register can be addressed separately for parallel entry as required to assemble 16-bit words received from memory into 22-bit single precision or 45-bit double precision operand fractions. The MQ register function executed at MQ clock (KKMQ-) time is determined by the status of the two selection inputs (S0 and S1) as summarized in table 4-10.

For parallel entry purposes, the MQ register is divided into six separately addressable sections (A through F) as illustrated in figure 4-46. The manner in which these sections are addressed when loading single precision and double precision fraction words is also indicated in the figure. For the single precision fraction the loading control signal CSLSA- loads the sign (S) and the seven most significant bits of the fraction (high fraction) into section A of the MQ register. The low CSLSA- signal places the MQAS1+ and MQAS0+ signals at the high level in order to select the loading function for this section. The data that is loaded into the MQ register sections is obtained from the data latch via the I/O data multiplexor. Figure 4-46 indicates the data

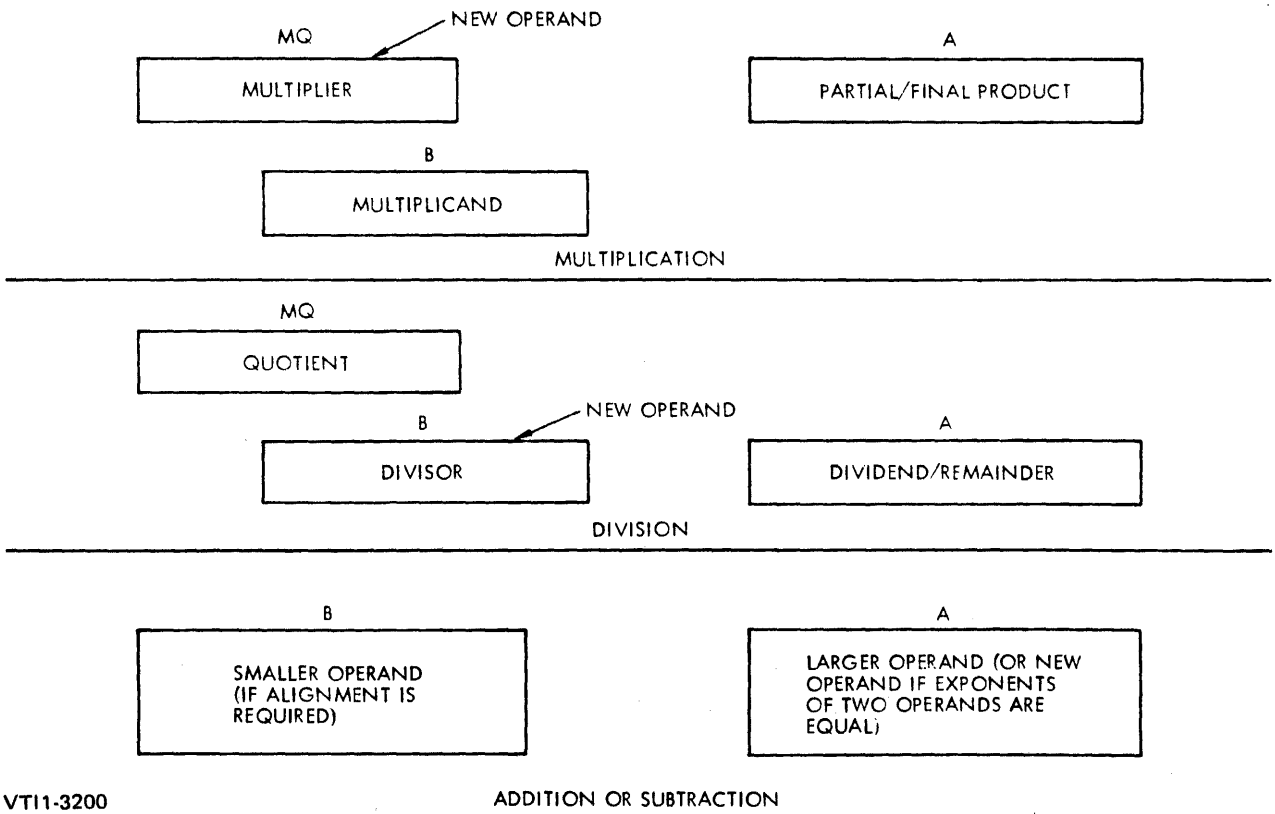


Figure 4-45. Data Loop Register Usages

Table 4-10. MQ, A, and B Register Function Selection

Selection Signals		
S1	S0	Function
L	L	No operation
L	H	Shift Right
H	L	Shift Shift Left
H	H	Load

Note: L = Low; H = High

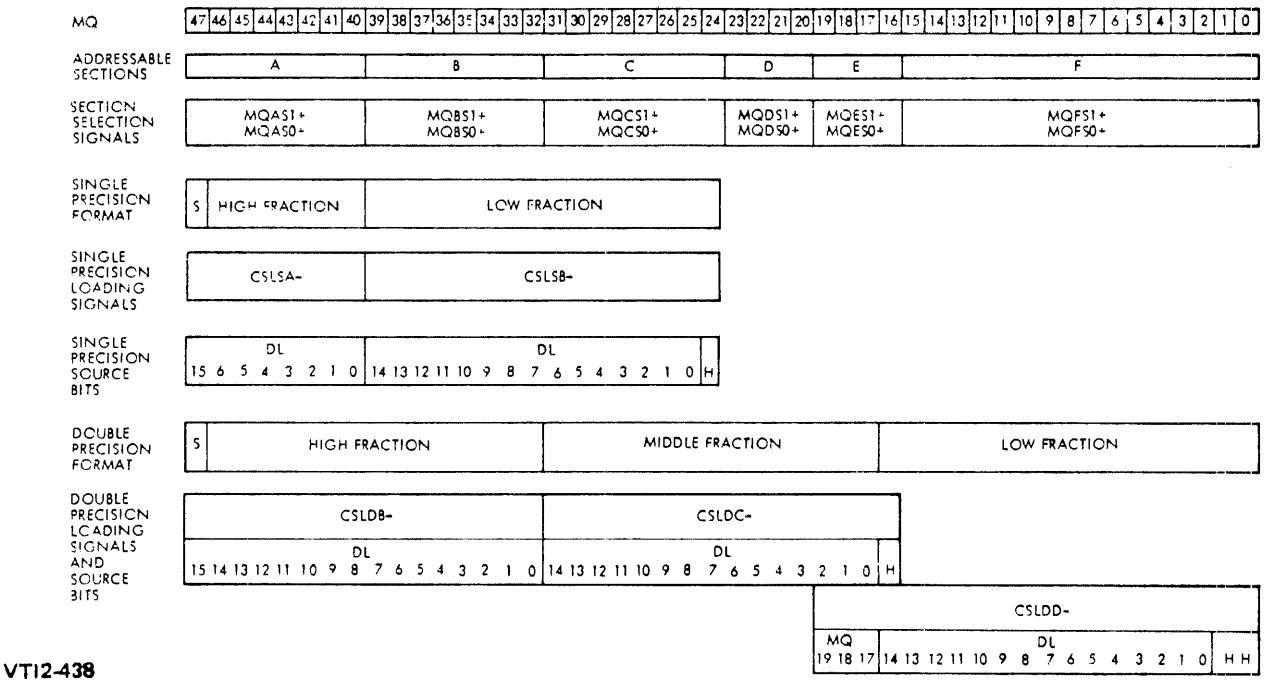


Figure 4-46. MQ Register Loading Formats

latch bit positions from which the data is obtained during each transfer into the MQ register. When section A is loaded in response to CSLSA-, the sign bit is obtained from data latch bit 15 while the high fraction is obtained from bit positions 0 through 6. The CSLSB- signal loads the remaining bits (low fraction) of the single precision fraction into sections B and C of the MQ register. Bits 0 through 14 from the data latch are loaded into bit positions 25 through 39 of the MQ register while a zero (high level) is loaded into bit position 24.

In the case of a double precision fraction, three transfers are required to assemble the complete fraction. The loading signals associated with these transfers are CSLDB-, CSLDC-, and CSLDD-. Section E is loaded twice. CSLDC- loads data latch bits 0 through 2 into bit positions 17 through 19 and a zero into bit position 16. CSLDD- loads MQ bits 17 through 19 back into the same positions in section E and loads data latch bit 14 into position 16 in place of the zero loaded in that bit position by CSLDC-. The selection of the source data loaded into bit positions of section E is accomplished by the MQE multiplexor. A low CSLDD- signal selects the inputs required during the CSLDD- transfer while a high CSLDD- signal selects the inputs required during other transfers.

The bit 24 input to the MQ register (MQIØ7+) is provided by a multiplexor in the MQ control logic. (This is the logic which supplies the selection signals to the MQ register sections.) The multiplexor selects a zero (high level input) for single precision operations or the appropriate output from the I/O data multiplexor for double precision operations.

With the exception of the selections just described, the bit inputs to the MQ register sections are selected by the I/O data multiplexor. The data from the I/O data multiplexor is received on the IOnn+ (00-15) lines.

At the end of each FPP instruction microprogram routine, the MQ register is cleared by loading zeros from the I/O data multiplexor into all sections. In the case of division, where the MQ register contains the quotient at the end of the routine, this result is transferred to the A and B registers at the time that the MQ register is cleared.

Shifting of the contents of the MQ register is described in connection with descriptions of clock control and MQ register control.

c. A and B Registers. The A and B registers are 48-bit, shift-left, shift-right, parallel-entry, parallel-readout registers similar to the MQ register. For addition or subtraction the A and B registers hold the two operands and the result is placed in the A register. During multiplication, the A register is used to accumulate the product while the B register is used to hold and shift the multiplicand. During division, the A register

is used to hold and shift the dividend/remainder while the B register is used to hold the divisor. If the result of an operation is not in the A register at the end of an instruction routine, it is copied into the A register. Every result is also copied into the B register at the end of an instruction. Thus, results reside in the A and B registers between instructions. During addition or subtraction, the B register is used to align the fraction portion of the smaller operand. Thus, if the operand resulting from the previous operation is smaller, the copy in the B register is used and the new operand is loaded into the A register. Conversely, if the new operand is smaller, it is loaded into the B register and the copy of the result of the previous operation held in the A register is used in the new operation.

During multiplication, the copy of the previous result held in the B register is used as the multiplicand. During division, the copy of the previous result held in the A register is used as the dividend. A load instruction places copies of the operand fraction in both the A and B registers. Thus, if the previous instruction was a load instruction, it is still true that the result of the operation is available in both the A and B registers.

The contents of the A register is applied to the A input of the ALU via the C multiplexor. The contents of the B register is applied directly to the B input of the ALU.

Both the A and B registers receive parallel data from the A multiplexor. Data appearing at the output of the A multiplexor can be loaded into either the A or B register or both by appropriate control of the function selection inputs to the two registers.

The A and B register are both clocked by KKREG- from the clock control. The function performed at the positive-going edge of each clock pulse is determined by the selection inputs as defined in table 4-10. The selection inputs to the A register are ACAS0+ and ACAS1+ while those to the B register are ACBS0+ and ACBS1+. These inputs are supplied from the A, B, and multiply control.

d. A Multiplexor. The A multiplexor provides the path by which operands are transferred from the MQ register to the A and/or B register. It also provides three paths by which result data from the ALU can be supplied to the A or B register. One of these paths shifts the ALU data to the right two bit positions (with the data from bits positions ALU49 and ALU48 being loaded into the two most significant bit positions of the A register). This path is used during multiplication to shift each new partial product two bit positions to the right at each clock time. A second path shifts the ALU data one bit position to the left (and puts a zero into the least significant bit

position of the register). This path is used during division to shift the partial remainder one bit position to the left at each clock time. The third path supplies the ALU data to the A or B register without shifting it either to the right or to the left.

The A multiplexor selection is controlled by the CSAM+ and CSAM0+ signals from the control store decoder. The paths selected by these signals are summarized in table 4-11.

e. C Multiplexor. When CSCM+ from the control store decoder is high, the C multiplexor selects the constant being supplied from the constant register. When CSCM+ is low, the C multiplexor selects the contents of the A register. The data selected by the C multiplexor is supplied to the A input of the ALU.

4.6.16 Multiply Control

Multiply control selects the multiplier bit pair to be evaluated during each iteration period of the fraction multiply micro-instruction, performs the evaluation of the selected multiplier bits, and manipulates various control signals in order to add or subtract an appropriate quantity to or from the previous partial product at each clock time.

Two dual 4-to-1 multiplexors and a latch circuit (figure 4-47) are used to select the two multiplier bits to be evaluated during each iteration of the multiplication routine. During the initialization step, the multiplier bits are obtained from the data latch. Two bits are supplied through the multiplier control bit

Table 4-11. A Multiplexor Selections

Selection Inputs		Selection
CSAM+	CSAM0+	
L	L	MQ register
L	H	ALU X 2^{-2}
H	L	ALU
H	H	ALU X 2^{1}

Note: L = low; H = high

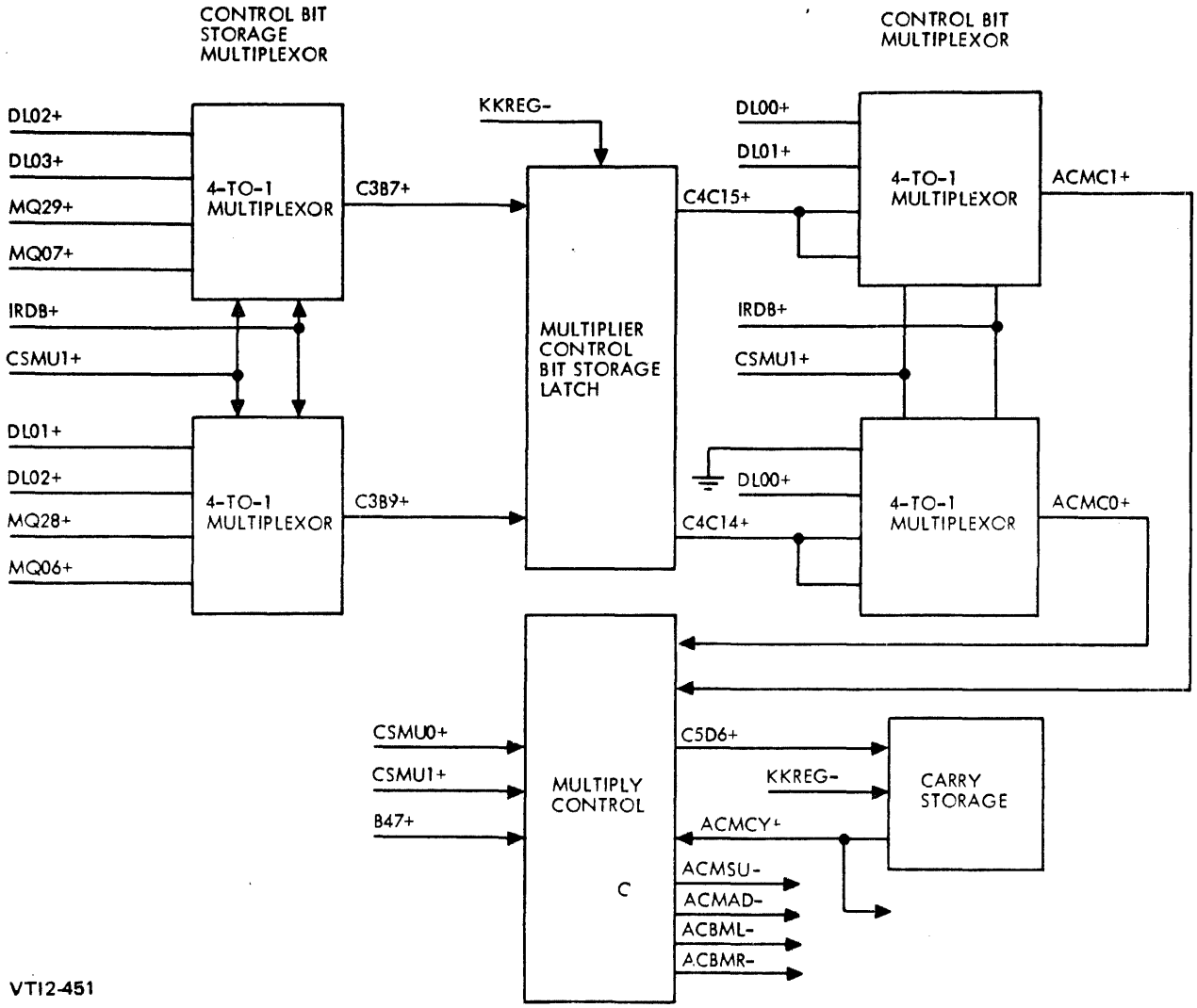


Figure 4-47. Multiply Control, Block Diagram

multiplexor to the ACMCn+(1,0) lines for use in controlling the first iteration. The next two bits are supplied through the multiplier control bit storage multiplexor to the multiplier control bit storage latch for use in controlling the second iteration. During subsequent iterations, the multiplier control bit multiplexor selects the contents of the multiplier control bit storage latch for application to the ACMCn+(1,0) lines while the multiplier control bit storage multiplexor selects two bits from the multiplier-quotient register (MQnn+) for loading into the multiplier control bit storage latch at the end of the iteration. The particular bits selected in each case depend upon whether single or double precision multiplication is being performed.

Both multiplexor selections are controlled by IRDB+ and CSMU1+. The status of IRDB+, which indicates whether single or double precision multiplication is being performed, determines the particular multiplier bit selections while the status of the CSMU1+ signal determines whether the bits are selected from the data latch or from the MQ register and the multiplier control bit storage latch. Table 4-12 summarizes the bit selections.

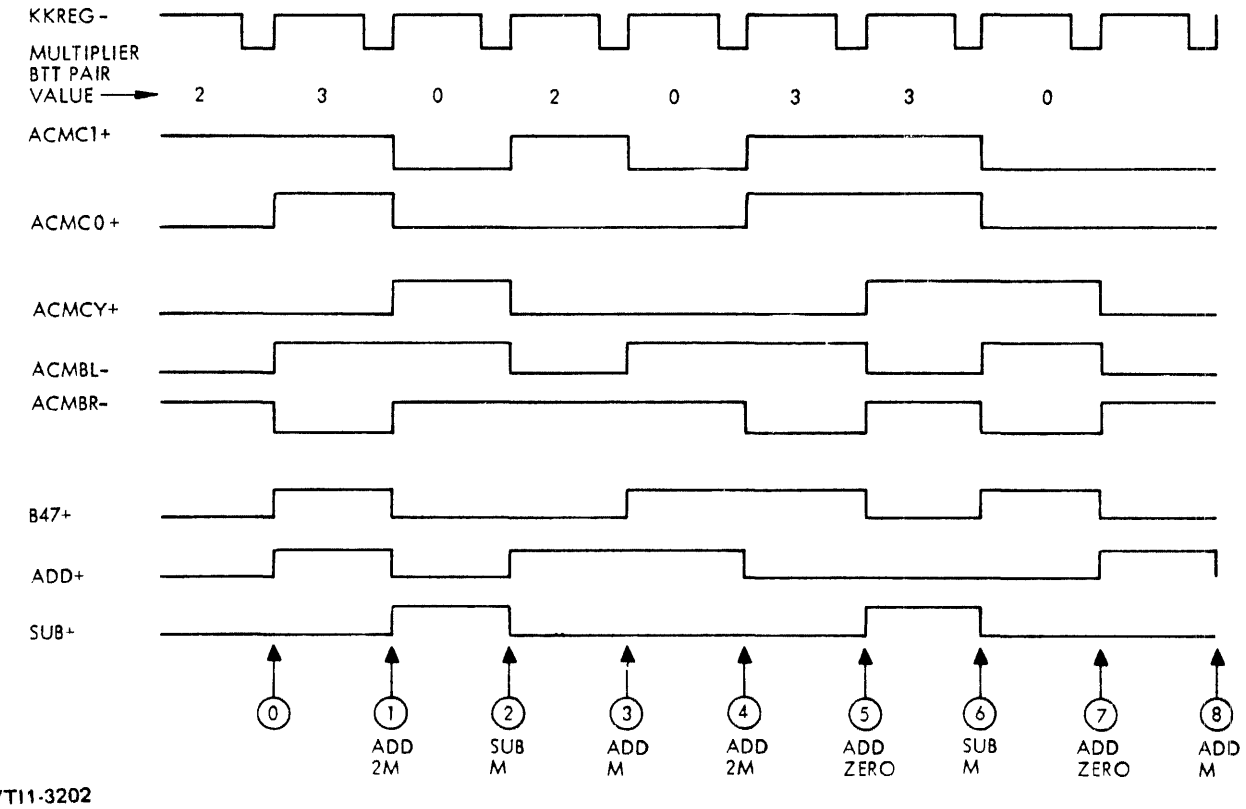
Multiplication is implemented by an add and shift algorithm. The selected pair of multiplier bits is evaluated to determine the quantity to be added to the partial product during each iteration. The bit pair can assume any of the values 00, 01, 10, or 11 (decimal 0, 1, 2, or 3). Thus, the basic requirement is to add 0M, 1M, 2M, or 3M to the partial product during each iteration (where M = multiplicand). However, only M and 2M are explicitly available. (2M is obtained by shifting the multiplicand one bit position to the left in the B register and M is recovered by shifting 2M one bit position to the right.) 3M is added by subtracting M and generating a control bit carry into the next iteration. The carry indicates a component, M, that must be added to partial product during this next iteration. (Since the partial product is shifted to the right two bit positions after each iteration, adding M one iteration later is equivalent to adding 4M during the current iteration. Thus, effectively, 3M is added by adding 4M - M. When the control bit from the preceding iteration is added to the bit pair value for the current iteration, the number of possible values increases to include the new value, 4. However, this value is easily handled by generating a carry into the next iteration time.

The multiply control manipulates signals which specify left shifting of M to provide 2M (ACMBL-) right shifting of 2M to provide M (ACMBR-), addition (ADD+), subtraction (SUB+), and multiplier bit carry into the next iteration period (ACMCY+). The ACMBL- and ACMBR- signals depend upon the current values of ACMC0+, ACMCY+, and B47+. The ADD+, SUB+, and ACMCY+ are stored quantities which depend upon the values of ACMC1+, ACMC0+, and ACMCY+ at the previous clock time. Figure 4-48 provides an example of multiply control timing. Nine clock times are identified by the circled numbers, 0 through 8, in the figure. At register clock time 0, the stored quantities (SUB+, ADD+, and ACMCY+) are set to the values appropriate for the first iteration. No addition or subtraction occurs at this clock time. In the example shown the first multiplier bit pair is 10 (as indicated by the high ACMC1+ signal and the low ACMC0+ signal at clock time 0). This results in the setting of ADD+ to the high level while SUB+ and ACMCY+ remain at the low level. Because the addition of 2 is indicated and B47+ is initially low, ACMBL- is low prior to clock time 0. This causes a left shift of the B register as indicated by the positive-going transition of B47+ at clock time 0. With ADD+ high and with 2M in the B register, the

Table 4-12. Multiplier Control Bit Selections

Step/Mode	CSMU1+	IRDB+	Bits Stored For Next Iteration		Bits Evaluated During Current Iteration	
			C3B7+	C3B9+	ACMC1+	ACMC0+
Initialization, Single Precision	L	L	DL02+	DL01+	DL00+	L
Initialization, Double Precision	L	H	DL03+	DL02+	DL01+	DL00+
Subsequent iterations, Single Precision	H	L	MQ29+	MQ28+	C4C15+	C4C14+
Subsequent iterations, Double Precision	H	H	MQ07+	MQ06+	C4C15+	C4C14+

Note: L = low; H = high



VTI1-3202

Figure 4-48. Example of Multiply Control Timing

addition of 2M to the partial product occurs at clock time 1. At this clock time, ACMCY+ and SUB+ are set high, ADD+ is set low, and the contents of the B register is shifted to the right to recover M. This prepares for the addition of 3M by subtraction of M at clock time 2 and addition of M at clock time 3. The process continues in this manner until every pair of multiplier bits has been processed.

Table 4-13 summarizes the manner in which the control signals depend upon the values of ACMCY+, ACMC1+, and ACMC0+ at each clock time. Notice that the B register shift control signal status depends not only upon the status of the multiplier bits and carry bit but also upon the current status of the B register (as indicated by the status of the B47+ signal). ACMBL- is placed at the low level to enable the left shift only if B47+ is low, indicating that the current contents of the B register is M. Similarly, ACMBR is placed at the low level only if B47+ is high. This inhibits the shifting of the B register when the B register already contains the quantity (M or 2M) required for the addition or subtraction at the next clock time.

The ADD+ and SUB+ flip-flops are in the ALU control logic. The signals from this logic which control the switching of these flip-flops at clock times are the ACMAD- and ACMSU- signals. The C5D6+ signal controls transitions of the ACMCY+ flip-flop which is part of multiply control as illustrated in figure 4-48.

4.6.17 A, B Control Encoder

The A, B control encoder provides the function selection signals to the A and B registers. The function selections can be controlled by the AB selection field of the microinstruction currently held in the control store register. (As indicated in table 4-14, there are seven active function selections that are encoded in response to control store register AB control signals CSABn+ (0-2). When these three signals are all low, the register function selection can be controlled by any of four other inputs. Three of these control the selection function codes supplied to the B register. These are the ACMBL-, ACMBR-, and CSAL- signals. ACMBL- and ACMBR- are supplied from the multiply control and control shifting of the multiplicand in the B register during the multiplication microprogram. CSAL- from the control store decoder provides right shifting of the B register during the alignment microinstruction.

During the fraction divide microinstruction, MQQB- from the MQ control selects either left shifting of the partial remainder in the A register (if the partial remainder is smaller than the divisor) or loading of a new partial remainder from the ALU into the A register (if the partial remainder in the A register is larger than or equal to the divisor).

Table 4-13. Multiply Control

Inputs			Control Outputs at KKRGE					Resultant Flip-Flop Outputs after KKREG-			Direction of Conditional B Register Shift at KKREG-
ACMCY+	ACMC1+	ACMCO+	ACMSU-	ACMAD-	CSD6+ (Next ACMCY+)	ACMBL- (If B47+ is low)	ACBMR- (If B47+ is high)	SUB+	ADD+	ACMCY+	
0	0	0	H	L	L	L	H	L	L	L	Left
0	0	1	H	H	L	H	L	L	H	L	Right
0	1	0	H	H	L	L	H	L	H	L	Left
0	1	1	L	L	H	H	L	H	L	H	Right
1	0	0	H	H	L	H	L	L	H	L	Right
1	0	1	H	H	L	L	H	L	H	L	Left
1	1	0	L	L	H	H	L	H	L	H	Right
1	1	1	H	L	H	L	H	L	L	H	Left

Note: 0 = L = 1 = H = high

Table 4-14. A, B Control Encoder

Encoder Input(s)			B Register Control		A Register Control		
CSAB2+	CSAB1+	CSAB0+	ACBS1+	ACBS0+	ACAS1+	ACAS0+	Function
L	L	L					NOP
L	L	H	L	L	L	H	SRA
L	H	L	L	L	H	L	SLA
L	H	H	L	L	H	H	LDA
H	L	L	H	H	L	L	LDB
H	L	H	L	H	L	H	SRAB
H	H	L	H	L	H	L	SLAB
H	H	H	H	H	H	H	LDAB
MQQB- = L MQQB- = H	(See note 4.)				H H	H L	LDA SLA
CSAL- = L			L	H			SRB
ACMBR- = L			L	H			SRB
ACMBL- = L			H	L			SLB

Notes:
1. L = low; H = high
2. SR = shift right, SL = shift left, LD = load
3. A = A register, B = B register, AB = A and B registers
4. When MQBB- is controlling ACASO+, ACAS1 is held high by low CSAB1- signal.

4.6.18 MQ Control

The MQ control (figure 4-49) provides the function selection signals to the various separately addressable MQ register sections. It provides the path by which quotient bits are shifted into the MQ register during division. Also, it determines, in accordance with the precision selection, the source of the data bit that is loaded into bit position 24 of the MQ register during the assembly of the operand fraction in the MQ register.

Function selection signals MQmS1+ (A-F) and MQmSO+ (A-F) provide the capability of addressing the appropriate sections of the MQ register during assembly of the operand in the MQ register. For example, a low CSLSA- signal places the MQAS1+ and MQASO+ signals at the high level as required to load the sign bit and high

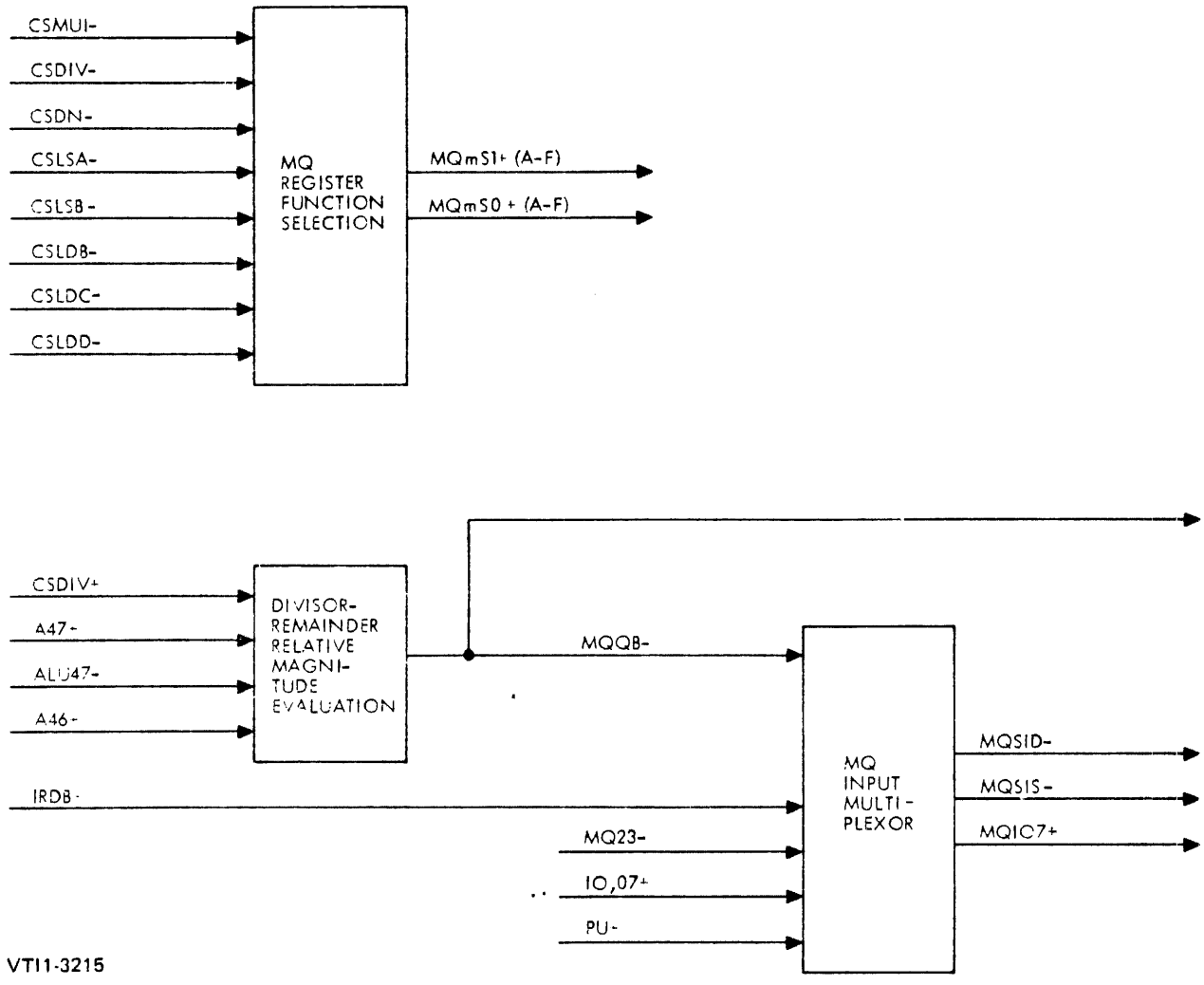


Figure 4-49. MQ Control, Block Diagram

fraction of a single precision operand into register section A. Refer to figure 4-46 for a diagram that illustrates the various loading signals involved in loading the operand words of both single and double precision operands into the MQ register.

In the case of the selection of the MQ shift-left or shift-right function, the function is selected for all sections of the register. The control inputs which select shift functions are CSMU1- and CSDIV-. CSMU1- selects the right-shift function that is required to shift the multiplier to the right during multiplication. CSDIV- selects the left-shift function required to shift quotient bits into the MQ register during division.

A low CSDN- signal selects the load function for all MQ register sections. This causes zeros to be loaded into the MQ register at the end of the final microinstruction of each FPP instruction microprogram. During division, quotient bits are supplied to the MQQB- line by a sub-function which evaluates the magnitude relationship between the partial remainder (or, during the first iteration, the dividend) and the divisor. If the partial remainder is at least as large as the divisor, MQQB- is placed at the low level to produce a binary one quotient bit). The evaluation is enabled during the division microinstruction in response to the high CSDIV+ signal from the control store decoder. There are two evaluation results which place the MQQB- line at the low level. One of these is the coincidence of high A46+ and ALU47- levels. This accounts for the case where the most significant binary one magnitude bit of the remainder is in bit position 46 of the A register (A46+ high) and the subtraction of the divisor from this remainder is producing a positive difference (ALU47- high). The second result which places MQQB- at the low level is a high A47+ signal. This indicates that the most significant binary one magnitude bit of the remainder is in bit position 47 of the A register. In this case, the remainder must be larger than the divisor, since the most significant magnitude bit of the divisor is in bit position 46. (In this case, the status of the ALU47- signal does not indicate the sign of the result of the ALU subtraction because of the presence of the magnitude bit in bit position 47 of the A register.)

The quotient bits appearing on the MQQB- line are supplied to the MQ register via the MQ input multiplexor. If the IRDB+ signal from the instruction register is low, indicating the selection of single precision, then MQQB- is connected to the MQSIS- line. In this case, quotient bits are shifted into bit position 24 of the MQ register. If IRDB+ is high, indicating the selection of double precision, then MQQB- is connected to the MQSID- line. In this case, quotient bits are shifted into bit position 0 of the MQ register. In this case, MQ23- from the MQ register is connected to the MQSIS- line to shift bits from bit position 23 to bit position 24. (When single precision is selected, PU+ is connected to MQSID- to shift zeros into the unused sections of the MQ register.)

The status of the IRDB+ signal also determines the source of the data that is supplied to bit position 24 of the MQ register via the MQIØ7+ line during parallel loading. When single precision is selected, PU+ is connected to MQI07+ to load a zero. When double precision is selected, IØ07+ is connected to MQIØ7+ to load a magnitude bit received from the I/O data multiplexor.

4.6.19 ALU Control

The ALU control (figure 4-50) provides function selection signals ALSn+ (0-3), mode selection signal ALM+, and carry-input signal ALCN+ to the ALU. When the CSFMT- signal from the control store register is high, these signals provide the addition function when the ADD- signal is low or the subtraction function when the SUB+ signal is high. (Refer to table 4-9 for a complete list of the ALU functions provided in response to each combination of control inputs that is used by the FPP.) The status of the ADD- and SUB+ signals is updated at each KKREG- pulse time in accordance with the status of the control inputs. During multiplication, the status of ADD- and SUB+ is determined by the ACMAD- and ACMSU- signals respectively from the multiply control. When ACMAD- is high, ADD- is set low to select the addition function. When ACMSU- is low, SUB+ is set high to select the subtraction function. During division, SUB+ is initially set high, to select subtraction, in response to the low CSDIO- signal from the control store decoder and is maintained high in response to the low CSDIV- signal from the control store decoder. When a microinstruction with the CPO code in its flag field resides in the

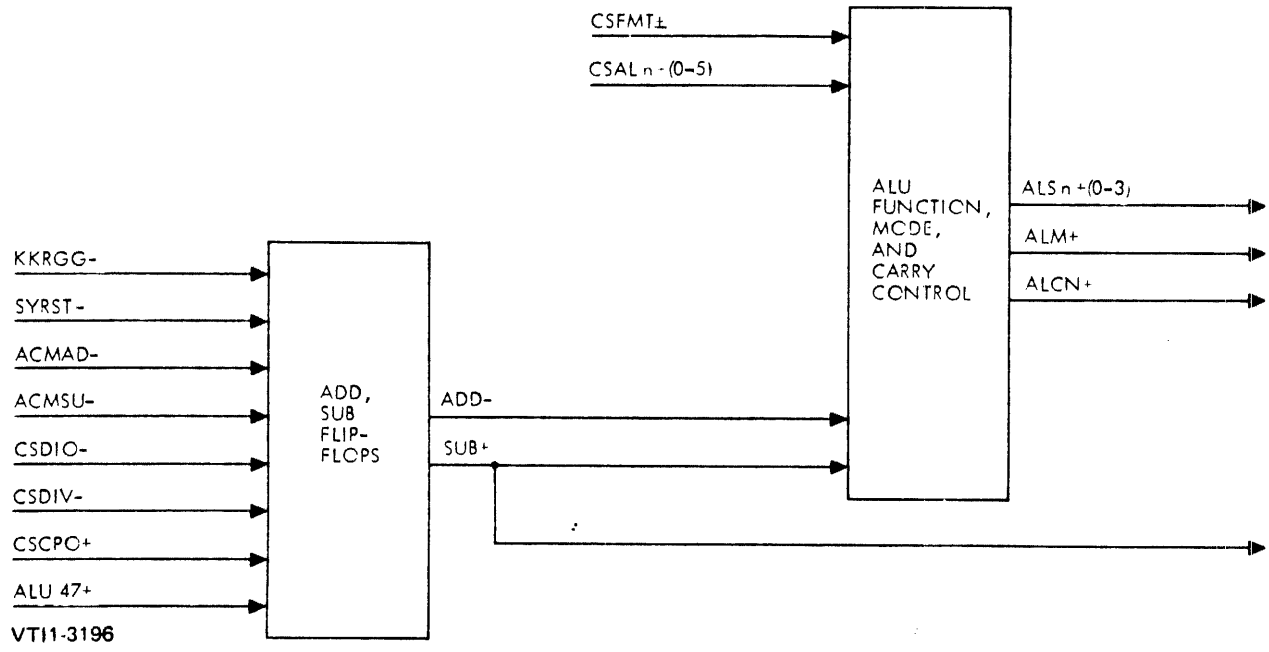


Figure 4-50. ALU Control, Block Diagram

control store register (CSCPO+ high), then SUB+ is set high to provide the subtraction function if ALU47+ is high (indicating that the current ALU function is providing a negative result). This is used to select a subtraction operation for the purpose of inverting a negative result. When none of the above control inputs is providing active control of the ADD- and SUB+ lines and CSFMT- is high, then ADD- is low as required to select the addition function by default.

When CSFMT+ is high, the control lines to the ALU are controlled by the status of the ALU field of the microinstruction in the control store register as follows: CSAL0+ controls ALCN+; CSAL1+ controls ALM+, CSAL2+ through CSAL5+ control ALS0+ through ALS3+ respectively.

4.6.20 Sign and Zero Flags

Sign flag AS (figure 4-51) indicates the sign of the result at the end of an instruction microprogram and between instruction microprograms. Sign flag BS indicates the sign of the new operand that is received during an instruction microprogram. The ZERO flag generally indicates whether the ALU output is zero or non-zero. A subsidiary signal (ACZ47+) indicates either the ZERO condition or the presence of a binary one in the most significant bit position of the A register.

The updating of sign and ZERO flags is synchronized by the KKREG± signal.

The following operations on sign flags are provided:

a. Loading sign of new operand (DL15+) into BS. This operation occurs when the new operand word containing the sign bit is in the data latch (as indicated by a low CSLSA- or CSLDB- signal from the control store decoder). As the sign bit is transferred, it is inverted if a subtraction instruction is being performed (IRSU+ high).

b. Resetting of AS. A low CSZAS- or CSUFL- signal from the control store decoder resets AS providing a positive sign flag for a positive or zero result. (CSUFL- zeroes the floating point accumulator when underflow occurs.)

c. Swap signs. The swap sign decode (high CSSS+, low CSSS-) from the control store decoder swaps the signs in AS and BS. This is used to transfer the new operand sign from BS to AS in cases where the sign of the result is simply the sign of the new operand (for example, addition of a larger new operand to a smaller previous result). Swaps signs is also used to place a sign bit in AS temporarily so that it can be reset.

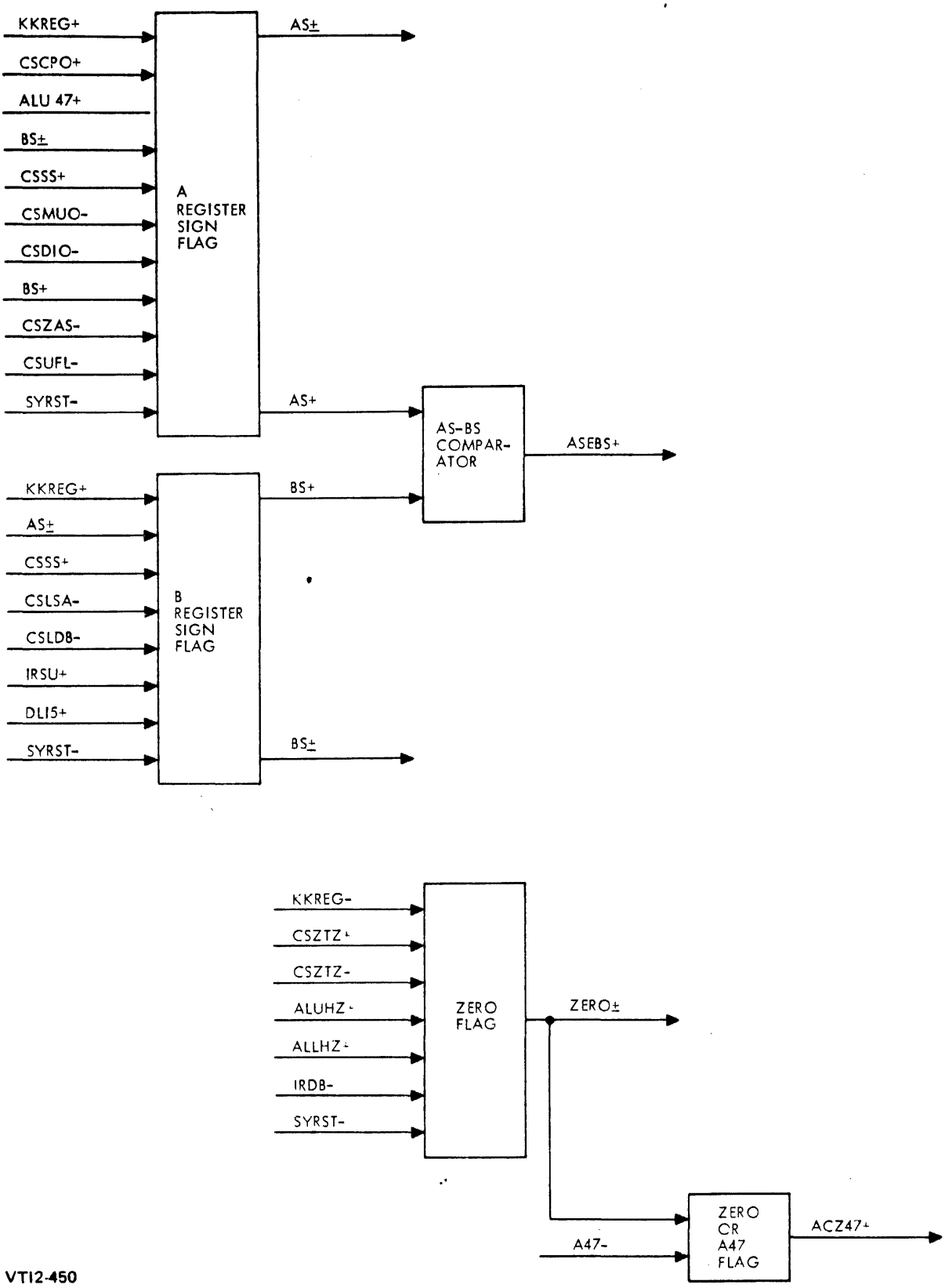


Figure 4-51. Sign and Zero Flags, Block Diagram

d. Toggling of AS. During the multiply or divide set-up microinstruction (CSMUO- or CSDIO- low), AS is toggled if BS+ is high. This provides the exclusive OR function which determines the sign of the result in multiplication or division. The set-up-complement decode (high CSCPO) toggles AS if a fraction subtraction produces a negative result (ALU47+ high). This conditional operation puts the sign of the larger number in AS.

When CSZTZ+ from the control store decoder is low, the ZERO flag is updated by each KKREG- pulse to indicate a zero or non-zero ALU output. High ALUHZ+ and ALLHZ+ signals indicate upper ALU zero and lower ALU zero. For single precision (IRDB- low) the ZERO flag status is determined by ALUHZ+ only. A high CSZTZ+ signal inhibits updating of the ZERO flag.

4.6.21 Constants and Conditional Inverter

The constant register provides constants that are required for various data loop ALU operations. The conditional inverter control provides inversion of I/O data to accomplish the conversion between the inverted sign word format used in memory and the absolute format used in the FPP.

At the first KKCS- time of an instruction microprogram, the CSDN+ signal from the control store register is high (indicating that the final microinstruction of the previous instruction microprogram is still held in the control store register). Under this condition, the KKCS- clock loads a round-off constant into the constant register. The constant loaded depends upon the status of IRDB±. If IRDB- is high, indicating single precision, a binary one is loaded into the CR24- bit of the constants register. If IRDB+ is high, indicating double precision, a binary one is loaded into bit position CR01-. These bits are added to the result fraction during a store instruction in order to round off the result.

A high CS47+ signal from the control store decoder causes a binary one to be loaded into the CR47- bit position of the constant register (provided that CSDN- is low). This constant is used to generate the largest fraction after an overflow has been sensed.

The microprogram can clear the constant register by placing the CSCCR- line at the low level.

During the transfer of the operand word containing the sign bit from the data latch to the MQ register, the ACINV+ signal is placed at the high level if DL15+ from the data latch is high, indicating a negative operand. In this case, the word is in ones complement form. The high ACINV+ signal provides the inversion required to convert the word to absolute form. The transfer of the single precision operand word containing the sign bit is

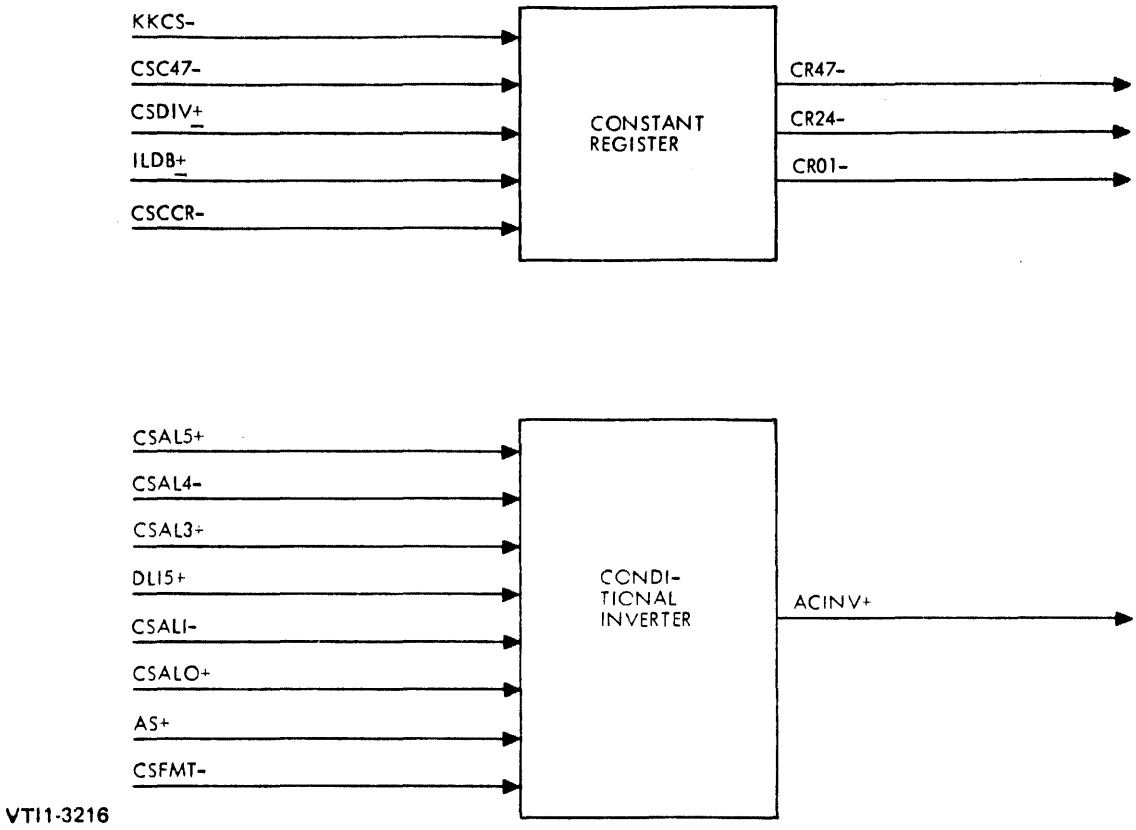


Figure 4-52. Constant and Conditional Inverter, Block Diagram

identified by high CSFMT-, CSAL4-, and CSAL3+ signals and a low CSAL5+ signal. The transfer of a double precision operand word containing the sign bit is identified by the same set of high signals coupled with a low IRDB- signal.

The ACINV+ signal must also be placed at the high level during the transfer of the result word containing the sign bit if the result is negative. This transfer is identified by high CSFMT-, CSAL1-, and CSAL0+ signals. The negative result is identified by a high AS+ signal.

4.6.22 Exponent Loop

As illustrated in figure 4-53, the exponent loop includes the following components:

a. Shift Counter The shift counter is used to count register clocks (KKREG-) to control the number of shifts during alignment or the number of iterations of the fraction multiply or fraction divide step. The shift counter is also used to provide temporary storage for exponents. The exponent ALU output data is loaded into the shift counter at KKREG- time in response to a low

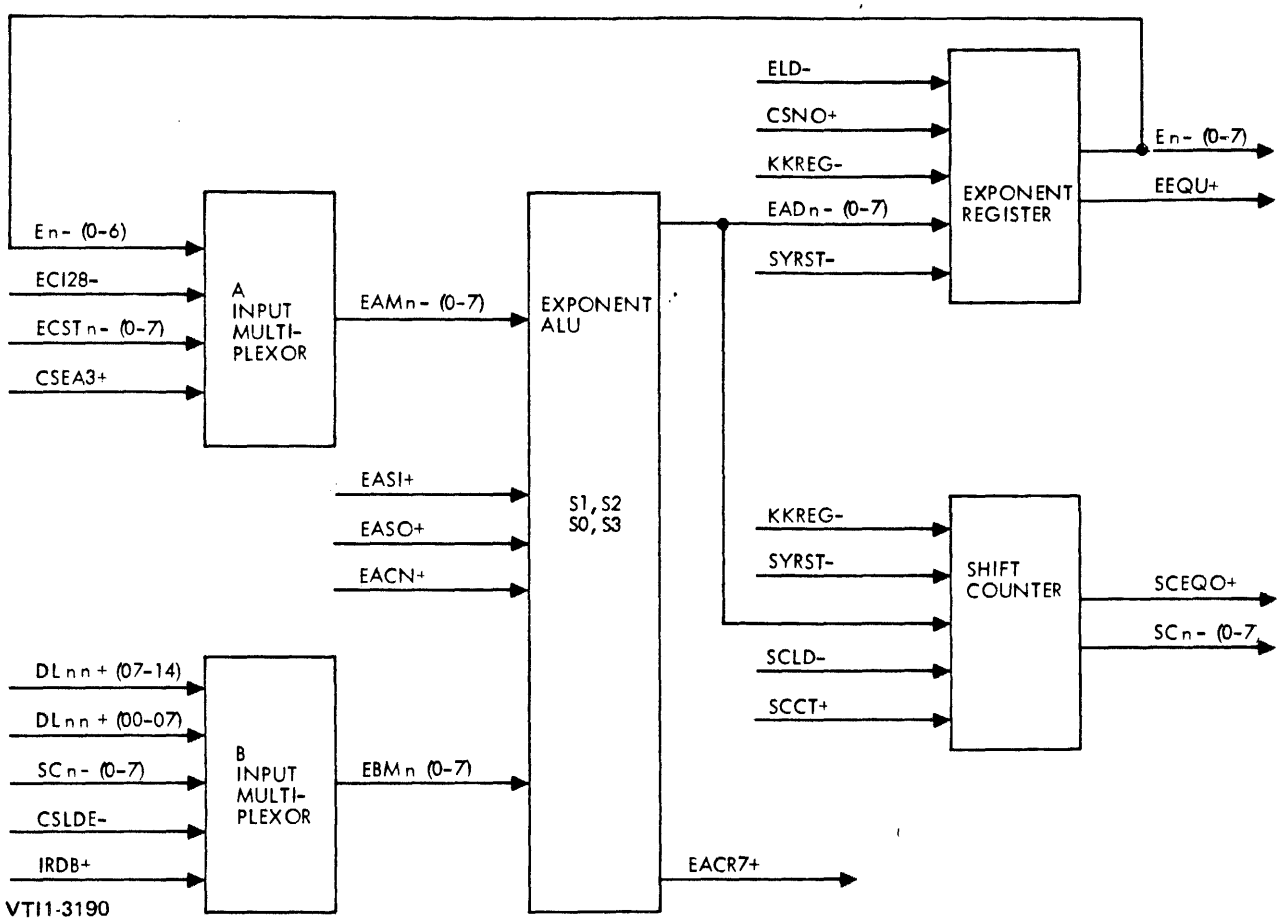


Figure 4-53. Exponent Loop, Block Diagram

SCLD- signal. Counting of KKREG- pulses is enabled by a high SCCT+ level. SCLD- and SCCT+ are supplied by the shift counter control.

b. Exponent Register The exponent register holds the final value of the exponent at the end of an instruction microprogram as well as various intermediate values during the execution of a microprogram. During the normalization microinstruction (CSNO+ high), the contents of the exponent register is counted down at each KKREG- time. (This decrements the exponent value by 1 for each left shift of the fraction.) The exponent ALU output data is loaded into the exponent shift register at KKREG- time in response to a low ELD- signal from the exponent control.

c. Exponent ALU The ALU adds two exponents, subtracts one exponent from a second exponent, increments an exponent by 1, decrements an exponent by 1, or provides twos complementation of an exponent. The function performed by the exponent ALU depends upon the following:

(1) The level of the EASO+ signal (which is applied to the S0 and S3 function selection inputs) and the level of the EAS1+ signal (which is applied to the S1 and S2 function selection inputs).

(2) The level of the EACN+ signal which is supplied to the carry input.

(3) The ALU data inputs selected by the A-input and B-input multiplexors.

The EAS1+, EAS0+, and EACN+ signals are supplied by the exponent control.

d. A-Input Multiplexor When the CSEA3+ signal from the EADD field of the microinstruction currently held in the control store register is high, the A-input multiplexor supplies data from the exponent register to the A input of the exponent ALU. (The most significant bit of the exponent data reaches the A-input multiplexor on the EC128- line from the exponent control. This allows the exponent control to invert this bit as required to add or subtract 128 to or from the exponent value in order to maintain the excess-128 code during addition or subtraction of exponents.) When CSEA3+ is low the data from the constant store and readout is connected to the A input of the exponent ALU. This data is received on the ECSTn-(0-7) lines.

e. B-Input Multiplexor When the CSLDE- signal from the control store register is low, data from one of two sections of the data latch is connected to the B input of the exponent ALU, in accordance with the status of the IRDB+ signal. If a single precision instruction is being executed (IRDB+ low), the DLnn+ (07-14) data is selected. If a double-precision instruction is being executed (IRDB+ high), the DLnn+(00-07) data is selected. In either case, the data selected is the exponent data for the new operand. When CSLDE- is high, shift counter signals SCn-(0-7) are connected to the B input of the exponent ALU.

4.6.23 Operations on Exponents

Operations performed on exponents include the following:

a. Addition of exponents during floating multiply instructions.

b. Subtraction of the divisor exponent from the dividend exponent during floating divide instructions.

c. Subtraction of the exponent of the smaller operand from the exponent of the larger operand during floating add and floating subtract instructions.

d. Decrementing or incrementing of result exponent during normalization.

e. Decrementing or incrementing of exponent to test for marginal overflow or underflow situations.

With one exception, all of these operations are performed using the exponent loop ALU. The exception is the decrementing of the exponent during normalization which is performed by down counting the exponent register.

Exponents are represented in an 8-bit, excess-128 binary format. (Let E = exponent. Then representation is E + 128.) The range of decimal values of (E + 128) is 0 through 255; the range of decimal values of E is -128 through +127.

4.6.23.1 Exponent Operations for Floating Multiplication

Exponent operations for floating multiplication instructions are illustrated in table 4-15. The first step is to add the excess-128 representations of the operand exponents and convert the representation of the sum from excess 256 to excess 128. The conversion is accomplished by subtracting 128 from the sum. If E_1 is the multiplicand exponent, E_2 is the multiplier exponent, and E_S is the sum, $E_1 + E_2$, then the first step is represented by the following equation:

$$(E_S + 128) = E_1 + E_2 + 128 = (E_1 + 128) + (E_2 + 128) - 128$$

All possible exponent sums, $E_1 + E_2$, are divided into five groups as indicated in column 1 of table 4-15. The corresponding excess-128 representations are indicated in column 2. Bit 8 of the sum, $E_1 + E_2 + 128$, is loaded into the exponent carry flip-flop (ECRY) and bits 7 through 0 are loaded into E. (The sign of the sum is neither computed nor stored.) Column 3 indicates corresponding numbers in the E register after E is decremented by one. Column 4 indicates the value of the carry from bit 7 of the sum, E + 1. Column 5 indicates the final result in E if the fraction product is less than 1/2. Column 6 indicates the final result in E if the fraction product is greater than or equal to 1/2.

(Note that after fraction multiplication, the most significant bit of the fraction product is in bit 47 of the A and B registers. If bit 47 is a one, then the fraction product is greater than or equal to 1/2, A and B are shifted right once, and the E register is incremented by 1.)

Table 4-15. Exponent Operations for Floating Multiplication

	1	2			3	4	5		6	
	Sum of Operand Exponents $E_1 + E_2$	Excess 128 Representation of Sum, $E_1 + E_2 + 128$			Decrement E Register	Test for E = 1111 1111	Final Result if $F_1 \times F_2 < \frac{1}{2}$		Final Result if $F_1 \times F_2 \geq \frac{1}{2}$	
	Exponent	Decimal	ECRY	E	E	ECRY	Exponent	E	Exponent	E
Overflow	+254	+382	1	0111 1110	0111 1101	0	+127	1111 1111	+127	1111 1111
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	+129	+257	1	0000 0001	0000 0000	0	+127	1111 1111	+127	1111 1111
Conditional Overflow	+128	+256	1	0000 0000	1111 1111	1	+127	1111 1111	+127	1111 1111
In-Range	+127	+255	0	1111 1111	1111 1110	0	+126	1111 1110	+127	1111 1111
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	-127	+1	0	0000 0001	0000 0000	0	-128	0000 0000	-127	0000 0001
Conditional Underflow	-128	0	0	0000 0000	1111 1111	1	-128	0000 0000	-128	0000 0000
Underflow	-129	-1	1	1111 1111	1111 1110	0	-128	0000 0000	-128	0000 0000
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	-256	-128	1	1000 0000	0111 1111	0	-128	0000 0000	-128	0000 0000

NOTES: E_1 is multiplicand exponent
F_1 is multiplicand fraction
E_2 is multiplier exponent
F_2 is multiplier fraction
E is exponent register
ECRY is exponent carry flag (used for sum bit in column 2)

Three tests are performed to separate the exponent sums into five groups (overflow, conditional overflow, in-range, conditional underflow, and underflow). First, the value of ECRY indicated in column 2 is tested to separate the sums into two groups:

ECRY = 1 Overflow, underflow, conditional overflow

ECRY = 0 In-range, conditional underflow

Second, the value of ECRY indicated in column 4 is tested to separate the two groups into four groups:

ECRY = 1 Conditional overflow

ECRY = 0 Overflow or underflow

ECRY = 1 Conditional underflow

ECRY = 0 In range

The overflow or underflow group is then separated into two groups by incrementing the E register by one (result indicated in column 2) and testing bit 7 of E:

E7 = 1 Underflow

E7 = 0 Overflow

For the overflow group ($E_1 + E_2$ greater than or equal to 129), an overflow interrupt is generated, and 1111 1111 is loaded into E. For the conditional overflow group ($E_1 + E_2 = +128$), the result is either in-range or overflow depending on the fraction product. For the in-range group (-127 less than or equal to $E_1 + E_2$ less than or equal to +127), the exponent in E is the final result if the fraction product is less than 1/2. Otherwise, the exponent in E is incremented by one. For the conditional underflow group ($E_1 + E_2 = -128$), the result is either underflow or in-range depending on the fraction product. For the underflow group ($E_1 + E_2$ less than or equal to -129), and underflow interrupt is generated and zero is loaded into E.

4.6.23.2 Exponent Operations for Floating Division

Exponent operations for floating division instructions are illustrated in table 4-16. These operations are similar to floating point multiplication. The divisor exponent is subtracted from the dividend exponent and the representation of the difference is converted from binary to excess-128 by adding 128:

$$(E_D + 128) = E_1 - E_2 + 128 = (E_1 + 128) - (E_2 + 128) + 128$$

Table 4-16. Exponent Operations for Floating Division

	1	2			3	4		5	
	Operand Exponent $E_1 - E_2$	Excess 128 Representation of Difference $E_1 - E_2 + 128$			Test for E = 1111 1111	Final Result if $F_1 \div F_2 < 1$		Final Result if $F_1 \div F_2 \geq 1$	
	Exponent	Decimal	ECRY	E	ECRY	Exponent	E	Exponent	E
Overflow	+255	+383	1	0111 1111	0	+127	1111 1111	+127	1111 1111
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.
	+128	+256	1	0000 0000	0	+127	1111 1111	+127	1111 1111
Conditional Overflow	+127	+255	0	1111 1111	1	+127	1111 1111	+127	1111 1111
In-Range	+126	+254	0	1111 1110	0	+126	1111 1110	+127	1111 1111
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.
	-128	0	0	0000 0000	0	-128	0000 0000	-127	0000 0001
Conditional Underflow	-129	-1	1	1111 1111	1	-128	0000 0000	-128	0000 0000
Underflow	-130	-2	1	1111 1110	0	-128	0000 0000	-128	0000 0000
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.
	-255	-127	1	1000 0001	0	-128	0000 0000	-128	0000 0000

NOTES: E_1 is dividend exponent
F_1 is dividend fraction
E_2 is divisor exponent
F_2 is divisor fraction
E is exponent register
ECRY is exponent carry flag (used for sum bit in column 2)

where E_D is the exponent difference, E_1 is the dividend exponent, and E_2 is the divisor exponent. Bit 8 of the difference, $E_1 - E_2 + 128$, is loaded into ECRY and bits 7 through 0 are loaded into E, as indicated in column 2 of table 4-16. The carry from bit 7 of sum (E + 1) is loaded into ECRY to test for the value, E = 1111 1111, as indicated in column 3. Column 4 indicates the final result in E if the fraction quotient is less than one. Column 5 indicates the final result in E if the fraction quotient is greater than or equal to one.

The value of ECRY in column 2 is tested to separate exponent differences into two groups:

ECRY = 1 Overflow, underflow, conditional underflow

ECRY = 0 In-range, conditional overflow

These two groups are separated into four groups by testing the value of ECRY in column 3:

ECRY = 1 Conditional underflow

ECRY = 0 Overflow or underflow

ECRY = 1 Conditional overflow

ECRY = 0 In-range

The overflow or underflow group is separated into two groups by testing bit 7 of the E register (column 2).

E7 = 1 Underflow

E7 = 0 Overflow

Final results for each of the groups are shown in columns 4 and 5. A floating point interrupt is generated for the overflow and underflow groups.

4.6.24 Exponent Control

As shown in figure 4-54, the exponent control provides the following:

a. the exponent carry flag (ECRY+) which is used in evaluating exponent loop ALU results in terms of the specific operation being performed so as to identify a negative or out-of-range result,

b. the add/subtract-128 sub-function which inverts the E7-bit from the exponent register during exponent additions and subtractions so as to maintain the required excess-128 format,

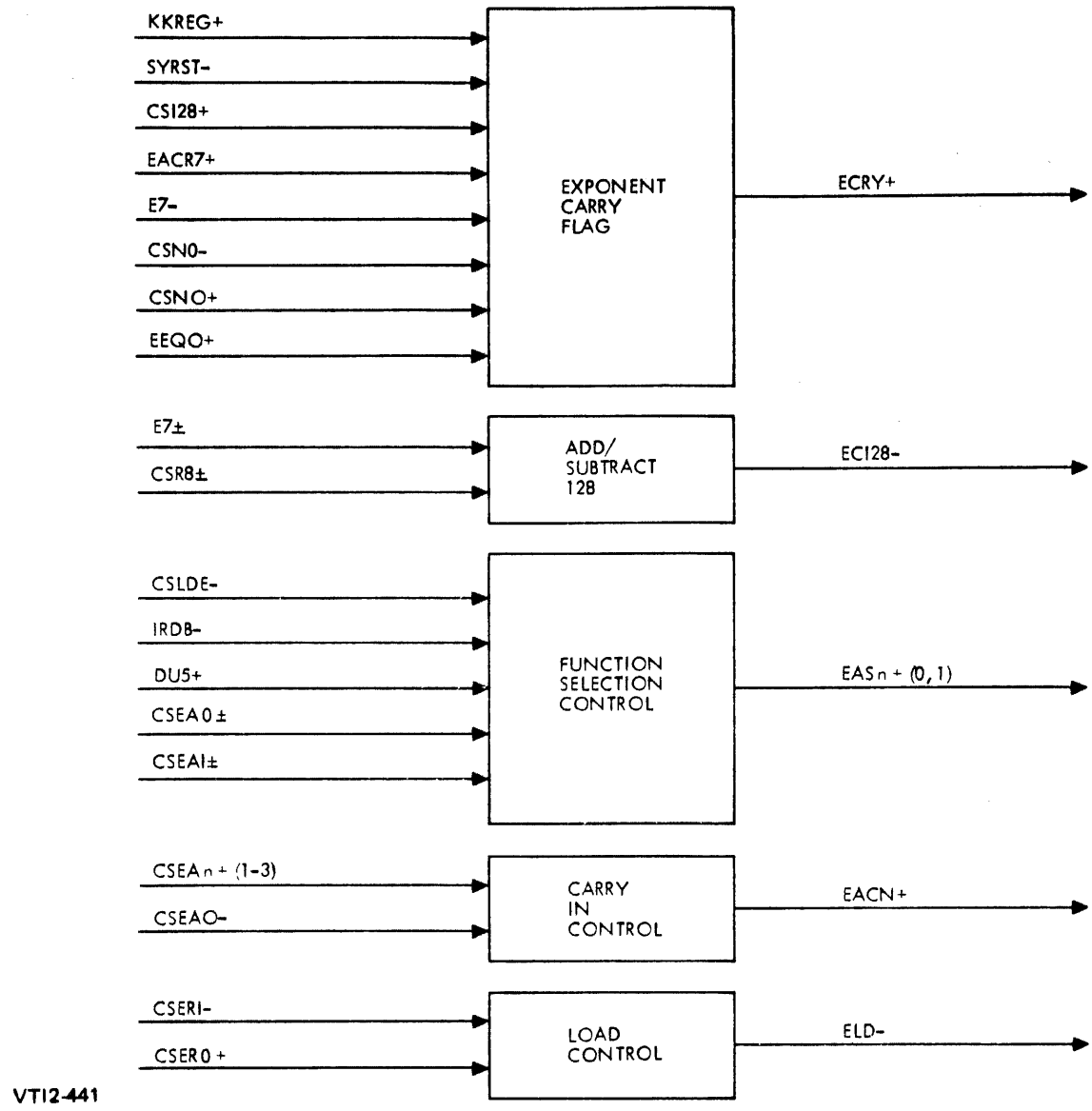


Figure 4-54. Exponent Control

c. the function selection control which determines the function performed by the exponent ALU,

d. the carry-in control which provides the carry-input (EACN+) to the exponent loop ALU,

e. the load control which provides the load control input (ELD-) to the exponent register.

Table 4-17 summarizes the addition and subtraction of exponents that occurs in connection with multiplication and division respectively. During these additions or subtractions, CS128+ is high. Under this condition, ECRY+ is set high at KKREG+ time if an out-of-range exponent value is sensed. If E7- is low (indicating that E_1 is zero or positive) then ECRY+ is set high if EACR7+ is high (indicating a carry or the absence of a borrow). If E7- is high (indicating that E_1 is negative) then ECRY+ is set high if EACR7+ is low (indicating a borrow or the absence of a carry). The high CS128+ signal also causes an inverted version of E7- to be supplied on the EC128- line to the exponent adder A-input multiplexor as required to add or subtract 128 from (E_1 + 128).

When CS128+ is low and a normalize microinstruction is not being executed (as indicated by a low CSNØ+ signal), then ECRY+ is set high by any operation that produces a carry from the exponent loop ALU. The setting of ECRY+ during the implementation of ALU output = (E_1 + 1) thus identifies a value of (E_1 + 1) greater than or equal to 256 during the test for a conditional exponent value. The status of ECRY+ also identifies the sign of the difference obtained during the exponent subtraction at the start of an addition or subtraction instruction. This allows the difference to be complemented if it is negative and also determines the branch of the microprogram that is followed.

When a normalize microinstruction is being executed (as indicated by a high CSNØ+ signal) a high EEQ0+ signal at KKREG+ time causes ECRY+ to be set high to indicate that the exponent value is being decremented into the underflow status.

Function selection control signal EAS0+ controls the S0 and S3 inputs and signal EAS1+ controls the S1 and S2 inputs to the exponent loop ALU. Table 4-18 summarizes the function selections in terms of the inputs to the function selection control.

The carry in control provides signal EACN+ to the carry input of the exponent loop ALU. The coincidence of high CSEA1+ and CSEA0- signals provide the high carry input required for a normal subtraction operation. Notice that this high carry input is not provided in the case where the exponent input is being complemented. Thus, a ones complement function is provided in this case, as required. The coincidence of high CSEA3+ and CSEA2+ signals provides a high carry input in cases where the ALU function, F = A + 1 is required. This is the function that is used to generate (E_1 + 1).

The coincidence of high CSER1- and CSER0+ signals from the EREG field of the microinstruction currently held in the control store register places the ELD- line at the low level as required to load the output data from the exponent loop ALU into the exponent register.

Table 4-17. Exponent Additions and Subtractions

Conditions	ALU Function performed	Overflow/ Underflow
Multiplication $E_1 + 128 \geq 0$ (E7- = L) (E_1 zero or positive)	$E_1 + 128 - 128 + E_2 + 128$ $= E_1 + E_2 + 128$	Carry (EACR7+ = H)
$E_1 + 128 < 0$ (E7- = H) (E_1 negative)	$E_1 + 128 + 128 + E_2 + 128$ $= E_1 + E_2 + 128 + (256)$	No Carry (EACR7+ = L)
Division $E_1 + 128 \geq 0$ (E7- = L) (E_1 zero or positive)	$E_1 + 128 - 128 - (E_2 + 128)$ $= E_1 - E_2 + 128 - (256)$	No borrow (EACR7+ = H)
$E_1 + 128 < 0$ (E7- = H) (E_1 negative)	$E_1 + 128 + 128 - (E_2 + 12)$ $E_1 - E_2 + 128$	Borrow (EACR7+ = L)

NOTES:
1. (256) indicates a component of the result which is lost if a carry occurs or a borrow does not occur
2. (E_1 + 128) is initially in the exponent register as the result of a previous operation
3. (E_2 + 128) is received from the data latch

Table 4-18. Exponent Loop ALU Function Selection

Function Selection Control Inputs		Code Presented to ALU				Function Selected*
CSEA1+	ESEA0+	S3	S2	S1	S0	
L	L	L	L	L	L	F = A or F = A + 1
L	H	H	L	L	H	F = A PLUS B **
H	L	L	H	H	L	F = A MINUS B **
H	H	H	H	H	H	F = A MINUS 1

*Assumes appropriate level on carry input line EACN+

**When CSLDE- is low and IRDB- is high, the addition input code is converted to a subtraction code if DL15+ is high. This is necessary in order to convert the exponent of a negative single precision operand to absolut form (by forming the complement of the complement).

Note: L = low; H = high

4.6.25 Shift Counter Control and Constant Storage

As shown in Figure 4-55, the shift counter control and constant storage provides the following:

a. The shift counter load control (SCLD-) and count enable (SCCT+) signals,

b. The SCLE1± signal which indicates when the shift count is 1 or 0,

c. The constant store and readout which supplies the constants required to specify the number of iterations, establish magnitudes, or test for relative magnitudes.

The bits of the EREG field of the microinstruction currently held in the control store register control SCLD-. If CSER1+ is high and CSER0- is low, SCLD- is placed at the low level as required to load the shift counter. If CSER1+ and CSER0- are both high, then SCLD- is placed at the low level only if ECRY+ is low. This is used to load the two's complement of the shift count into the shift counter if the shift counter contains a negative number in twos complement form.

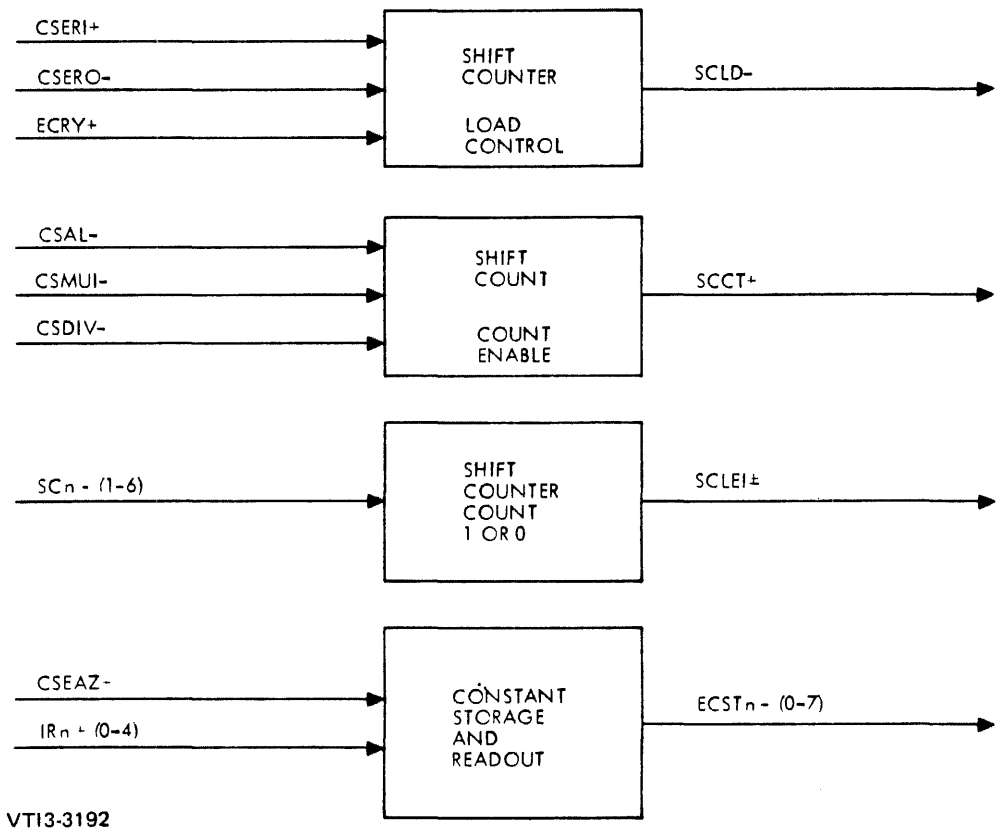


Figure 4-55. Shift Counter Control and Constant Storage

Shift clock counting is enabled for any of three microinstructions; the align microinstruction (CSAL- low), the main multiply microinstruction (CSMU1- low), and the main divide microinstruction (CSDIV- low). In the align microinstruction, shift counting determines the total number of shifts. In the main multiply and divide instructions, it determines the number of iterations.

The SCLE1+ signal is placed at the high level when SCn-(1-6) are all high, indicating that the shift count is either 1 or 0. This provides advance information during the down counting of the shift count that the next shift clock will reduce the shift count to 0.

The reading of a constant is enabled by a high CSEA2+ signal from the EADD field of the microinstruction currently held in the control store register. The constant that is read out depends upon the status of the IRn+ (0-4) signals from the instruction register. The constants associated with each instruction are summarized in table 4-19.

Table 4-19. Stored Constants

Instruction	ROM Address (Binary)	Constant		Function
		Binary Value	Decimal Value	
FAD/FSB	01000	1110 1001	Two's complement of 24	Used to test whether smaller operand will have significant bits after alignment
FADD/FSBD	00011	1101 0001	Two's complement of 48	Used to test whether smaller operand will have significant bits after alignment
FMU	01110	0000 1010	11	Number of iterations of main multiply step
FMUD	00110	0001 0110	22	Number of iterations of main multiply step
FDV	00001	0001 1000	24	Number of iterations of main divide step
FDVD	11101	0011 0000	48	Number of iterations of main divide step
FLT	10001	1000 1111	143	Initial excess-128 exponent code relating integer to floating format
FIX	10101	1000 1111	143	143 minus exponent of floating point number establishes initial value of exponent

SECTION 5
MAINTENANCE

Maintenance personnel should be familiar with the contents of this manual before attempting FPP troubleshooting. A test program (Section 7) is available to verify correct operation and to isolate malfunctions to a particular section of the FPP. Further diagnosis can then be made by referring to this manual.

5.1 TEST EQUIPMENT

The following test equipment and tools are recommended for FPP maintenance:

a. Oscilloscope, Tektronix type 547 with dual-trace plug-in unit or equivalent.

b. Multimeter, Triplett type 630 or equivalent.

c. Soldering iron, 15-watt pencil type.

5.2 CIRCUIT BOARD REPAIR

The FPP board is a wire-wrap circuit board. The ICs contained on the board consist of LSI read-only memories; MSI multiplexors, decoders, ALUs, and registers; and SSI gates and flip-flops.

If it has been determined that circuit board repair is required, it is recommended that the Sperry Univac customer service department be contacted so that a new circuit board can be installed in the user's system and the faulty one returned to the factory for repairs. However, if the user decides to perform his own repairs, extreme caution should be used so that the circuit board is not permanently damaged. Approved repair procedures should be followed such as the ones described in document IPC-R-700A prepared by the Institute of Printed Circuits.

5.3 CIRCUIT-COMPONENT IDENTIFICATION

A system of row and column coordinates provides the means for locating IC components. Major row identifications are A, B, and C. Each of these major rows contains six rows of components

designated 1 through 6. In addition to the component rows within major rows A, B, and C, there are two additional rows designated as X and Y. Columns are designated by single letters of the alphabet. Logic elements are identified by the location coordinates of the IC package. For example, a gate designated as B4U in the FPP logic diagram is in the IC package at major row B, sub-row 4, column U. (IC packages in rows X and Y are identified by the row designations X1 and Y1 even though these are single rows. For example, Y1K is at row Y, column K.)

A parts list in the system documentation package provides a cross reference between Sperry Univac and manufacturers part numbers.

SECTION 6
MNEMONICS

This section presents an alphabetized list of FPP signal mnemonics with definitions.

Plus or minus signs are included at the end of each mnemonic. The plus sign indicates the signal is at a high logical level when its function is being performed. The minus sign indicates the signal is at a low logical level when its function is being performed. A signal that is the logical inversion of another uses the same mnemonic with an opposite sign; these signals are complements of each other.

I/O bus signal mnemonics end with -I.

Letter and/or number subscripts are used after the plus or minus sign to indicate particular members of a family of logically equivalent control or clock signals. For such families of signals, only the base mnemonic is given. For example, KKMOK- is listed for the family that includes KKMOK-1, KKMOK-2, and KKMOK-3.

Mnemonic	Description
Ann-(00-47)	A register outputs
ACASn+(0,1)	Function selection inputs to A register
ACBSn+(0,1)	Function selection inputs to B register
ACDEN-	Arithmetic control divide enable
ACINV+	Invert I/O data
ACMAD-	Multiplication routine ADD selection
ACMBL-	Multiplication routine shift B register left
ACMBR-	Multiplication routine shift B register right
ACMCn+(0,1)	Multiplier bit pair evaluated during current iteration
ACMCY+	Multiplier evaluation bit pair carry
ACMEN-	Multiplication clock enable
ACMSU-	Multiplication routine SUB selection
ACSEN-	Shift clock enable

Mnemonic	Description
ACZ47+	ZERO+ or A47+ is high
ADnn+(00-15)	Memory address counter outputs
ADD-	Selects fraction addition for format 0
ADSn-(0-7)	Control store address
ALCN+	Carry input to data-loop ALU
ALLHZ+	Lower half of data-loop ALU = 0
ALM+	Mode control input to data-loop ALU
ALSn+(0-3)	Function selection inputs to data loop ALU
ALUnn-(00-49)	Data-loop ALU outputs
ALUHZ+	Upper half of data-loop ALU = 0
AMnn-(00-47)	Data-loop A multiplexor outputs
AS+	Sign of result (high = negative)
ASEBS+	AS = BS
Bnn-(00-47)	B register outputs
BS+	Sign of new operand (high = negative)
CAEN+	Central processor microinstruction decoder active
CDREN+	Central processor microinstruction decoder partial enable
CEADn-(0-8)	Address lines to central processor control store
CEND+	Central processor microinstruction decoder partial enable
CIDIO+	I/O instruction decoded (from central processor, FPP, or WCS)
CINTF+	I/O control interrupt flip-flop
CMnn-(00-47)	Constant multiplexor outputs (data loop)
CPSIF+	Start instruction fetch (by return of control to central processor)

Mnemonic	Description
CPSn+(0,1)	Central processor control state counter outputs
CP168+	Central processor control address 168 (hex) selection
CRnn-(01,24,47)	Constant register outputs
CSABn+(0-2)	Control store register, AB field signals
CSAL+	Align microinstruction decoding of ALU field of control store register
CSALn+(0-5)	Control store register ALU field signals
CSAM+	MSB of AC field of control store register
CSAMO+	Multiply or divide selection signal from control store decoder to data-loop A multiplexor
CSAn+(0-7)	Control store address register outputs
CSC47-	Set-constant-register-bit-47 decode of FLAG field of control store register
CSCCR-	Clear-constant-register decode of FLAG field of control store register
CSCM+	Constant multiplexor-enable (LSB of AC field of control store register)
CSCPO+	Change-to-positive decode of FLAG field of control store register
CSCTO+	Reset-time-out-flag decode of FLAG field of control store register
CSDIO-	Set-up-divide decode of FLAG field of control store register
CSDIV-	Main-divide decode of FLAG field of control store register
CSDN+	END and jump condition not true (start or wait for next routine)
CSEAn+(0-3)	Control store register EA field signals
CSEND-	END decode of MEM field of control store register

Mnemonic	Description
CSERn+(0,1)	Control store register EREG field signals
CSEXC-	Exponent-coming decode of FLAG field of control store register
CSFLn+(0-3)	Control store register FLAG field signals
CSFMT+	Control store register format bit
CSIMC+	Initiate memory cycle signal from control store register decoder
CSINC+	Increment-memory-address-and-request-cycle decode of MEM field of control store register
CSJAn+(0-3)	Control store register JADD field signals
CSJCn+(0-2)	Control store register JCOND field signals
CSLDB-	Load double precision middle fraction into MQ register
CSLDC-	Load double precision middle fraction into MQ register
CSLDD-	Load double precision low fraction into MQ register
CSLDE-	Load exponent from data latch into exponent register (or shift counter)
CSLSA-	Load single precision high fraction into MQ register
CSLSB-	Load single precision low fraction into MQ register
CSMUO+	Multiply-set-up decode of FLAG field of control store register
CSMU1+	Main-multiply flag from control store register
CSMWT+	Wait for memory done (MSB of MEM field of control store register)
CSNO+	Normalize microinstruction decode of ALU field of control store register
CSOFL-	Overflow decode of FLAG field of control store register

Mnemonic	Description
CSSIF+	Start-instruction-fetch decode of MEM field of control store register
CSSS+	Swap-signs decode of FLAG field of control store register
CSUFL-	Underflow decode of FLAG field of control store register
CSWSD+	Wait-for-shift-done decode of MEM field of control store register
CSZAS+	Set-AS=0 decode of FLAG field of control store register
CSZTZ-	ZERO-to-ZERO decode of FLAG field of control store register
DLnn+(00-15)	Data latch outputs
DLWRT-	Data latch write output (instruction is a storage type)
En-(0-7)	Exponent register outputs
EACN+	Exponent-loop ALU carry input
EACR7+	Exponent-loop ALU carry output
EADn(0-7)	Exponent-loop ALU outputs
EAMn-(0-7)	Exponent-loop ALU A-input multiplexor outputs
EASn+(0,1)	Exponent-loop ALU function selection control inputs
EBnn-I(00-08)	E bus (I/O bus)
EBMn-(0-7)	Exponent-loop ALU B-input multiplexor outputs
EC128-	Conditionally inverted MSB of exponent register
ECRY+	Exponent-loop ALU carry/out-of-range flag
ECSTn-(0-7)	Gated outputs constant store
EEQO+	Exponent register = 0

Mnemonic	Description
ELD-	Load exponent register
ICPRM+	Clock for interrupt enable
IDGO+	Instruction decoder go (FPP instruction decoded)
IEXCX-	Interrupt interface decode of EXC instruction on E bus
IINT-	FPP interrupt pending
IINHC-	I/O control clock inhibit signal
ILnn+(00-04)	Five LS bits of instruction latch
ILMRT+	Memory-write-instruction flag of instruction latch
ILSU+	Instruction latch subtract bit
ILWRT+	Instruction latch write-instruction bit
IOnn+(00-15)	Conditionally inverted outputs from I/O data multiplexor
IOMnn+(00-15)	I/O data multiplexor selected data
IPRME+	Interrupt master enable
IRn+(0-4)	Five LS bits of instruction register
IR005+	I/O control, control store bit 5 (DMA request bit)
IR007+	I/O control, control store bit 7 (idle bit)
IRDB+	Double precision bit from instruction register
IRQC-A	Interrupt request to central processor
IRQM-A	DMA request to central processor
IRSU+	Subtract bit from instruction register
IUAX-I	E bus acknowledge
IUCX-I	Interrupt clock

Mnemonic	Description
IUJX-I	E bus jump and mark
IURM+	FPP interrupt request
IURX-I	E bus interrupt request
JCT-	Complement jump condition true signal
KK82+	82-nanosecond clock
KKCPA+	Control store clock generator phase A output
KKCPB+	Control store clock generator phase B output
KKCS+	Control store clock
KKHRG+	Register half clock
KKKIN-	Inhibit KKREG+ and KKMQ
KKMCD+	Buffered version of central processor MCDFC
KKMEM+	Memory clock
KKMFC	Buffered version of central processor MFC
KKMHC+	Buffered version of central processor MHC
KKMOK-	Buffered version of central processor MOCLK
KKMPA+	Retiming clock generator phase A
KKMPB+	Retiming clock generator phase B
KKMQ-	MQ register clock
KKPA+	Register clock generator phase A
KKPB+	Register clock generator phase B
KKREG+	Register clock (to all arithmetic registers except MQ)
MAKO+	PMA memory request acknowledge
MCAEN+	Memory control address enable
MCAEN+m(A,B)	Memory control address enabled and memory access priority available
MCCKI-	Memory control store clock inhibit

Mnemonic	Description
MCCS2+	Memory sequencer state 2 or not FPP memory cycle
MCDDE+	Memory control data driver enable
MCDLE-	Memory control data latch enable
MCDFC-	Central processor full clock, gated
MCICS-	Initiate control store clocks
MCIST-	Initiate first KKMEM+ pulse (instruction start)
MCLDA-	Load address register
MCMRQ-	FPP memory request
MCRRQ-	Read request pending
MCSOA+	Memory sequencer state 0A
MCSOB+	Memory sequencer state 0B
MCS1A-	Memory sequencer state 1A
MCS1B-	Memory sequencer state 1B
MCS2+	Memory sequencer state 2
MCS3+	Memory sequencer state 3
MCWRQ+	Write request pending
MFC-	Central processor full clock
MHC-	Central processor half clock
MM1I-	Instruction latch enable (from central processor)
MOCLK-	41 nanosecond clock (from central processor)
MQnn-(00-47)	MQ register outputs
MQmSO+	LSB of function selection code to section m of MQ register (where m = A, B, C, D, E, F)
MQmS1+	MSB of function selection code to section m of MQ register (where m = A, B, C, D, E, F)

Mnemonic	Description
MQI07+	Parallel data input to bit 24 position of MQ register from MQ control
MQMnn-(16-19)	MQ input multiplexor outputs
MQQB-	Quotient bit
MQSID-	Double precision quotient input to MQ register
MQSIS-	Single precision quotient input to MQ register
MRS2A-	Memory sequencing signal (from central processor)
MWLY+	Memory write left byte
MWRY+	Memory write right byte
MYAnn-(00-15)	Memory address lines
MYDnn-(00-15)	Memory data lines
ORQM-A	PMA memory request to central processor
ORQM-C	PMA memory request from PMA
PRIDL-	FPP version of I/O control IEIDLE
PRINF-	FPP version of I/O control IINTF
PRINT+	Interrupt inhibit
PRIRQ+	FPP version of I/O control IRQM
PRIX-I	Interrupt priority input to interrupt interface
PRJX-I	Interrupt priority output from interrupt interface
PRKIN-	Inhibit clocks to wait for memory access priority
PRMDN+	PMA or DMA memory cycle complete flag
PRMEI+	PMA and DMA memory cycles inhibited flag

Mnemonic	Description
PROUT+	FPP memory access priority lost to PMA or DMA
PRTO-	Set time-out flag
PUn+(1-18)	Pull-up resistors
SCn-(0-7)	Shift counter
SCCT+	Shift count decrement enable
SCEQO+	Shift count = 0
SCLD-	Load shift counter
SCLE1-	Shift count less than or equal to 1
SRST-	System reset
STOUT+	Time-out detection enable
SUB+	Selects subtraction function for format 0
SYRST+	System and time-out reset
TO+	Time-out flag
XCFST+	Memory request enable input from WCS
YDNM+	Memory acknowledge (trailing edge is memory done)
ZERO+	ALU = 0 flag

SECTION 7
TEST PROGRAMS

7.1 GENERAL

The FPP test program, which is controlled by the MAINTAIN III Test Executive program, is used to verify correct operation and to isolate malfunctions. Minimum hardware requirements for using the test program consist of a SPERRY UNIVAC 70 series processor with 8K of mainframe memory, an object input device, and a console Teletype (TTY) of equivalent CRT terminal

7.2 TEST PROGRAM ORGANIZATION

The test program includes three tests:

a. Operational Test

b. Fault Test

c. Sequence Test

To thoroughly exercise the FPP, both the operational test and the fault test should be run. The sequence test is used primarily for troubleshooting.

7.2.1 Operational Test

The operational test includes all of the FPP instructions. The test starts with the more simple instructions and progresses to the more complex instructions. Mixed mode arithmetic operations (single and double precision) are performed. Direct and indirect fetching and storing of data are performed. The sequences of instructions and the operand data are varied as the test progresses. Results are checked against known data and, if a discrepancy occurs, the actual and expected values are displayed.

7.2.2 Fault Test

The fault test generates those conditions which produce fault interrupts, and verifies that these interrupts occur or reports their failure to occur.

7.2.3 Sequence Test

The sequence test allows the user to specify a sequence of FPP instructions and to execute this sequence either one time or

repetitively. The operator also specifies the operand data. Results may be displayed if desired. When a fault occurs an error message is displayed.

7.3 PROGRAM-TAPE LOADING

Before loading the FPP test program tape, the MAINTAIN III test executive tape must be loaded into the reader (refer to Test Programs Manual). Included in the test executive is the object tape loader for the FPP test program. After the test executive is loaded, mount the object tape of the FPP test program in the reader. The FPP test program tape is identified by Sperry Univac part number 92U0109-005 punched in the leader portion of the tape. Position the tape in the reader past this area. To load and start the execution of the test program, use the console TTY to type an L followed by a period.

7.4 SENSE SWITCHES

Operation of the FPP test program can be modified by positioning of SENSE switches on the computer control panel. The switch functions are listed in table 7-1.

Table 7-1. SENSE Switch Settings

SENSE Switch	Set	Reset
1	Suppresses error (or result) messages	Prints error (or result) messages
2	Halts program on error (see note 1)	No halts on error
3	Terminates the test	Allows test to continue

Note 1: After the error halt, one of the following operations can be performed:

a. To allow the program to continue to the next error halt, keep SENSE switch 2 set and press START.

b. The program can be made to loop on the sequence which caused the error condition by resetting SENSE switch 2 and pressing START. The program continues to loop until SENSE switch 2 is set; it then continues to run in the halt-on-error mode.

7.5 OPERATING PROCEDURES

When loading is completed, the test program is automatically directed to the starting memory address 0500, and the TTY prints the message:

FLOATING POINT PROCESSOR TEST

USE REAL TIME CLOCK?

If the real-time clock is to be enabled during testing, the operator responds by typing a Y. If the real-time clock is to be inhibited, the operator responds by typing an N.

The program now responds by typing a double asterisk (**) to indicate that it is waiting for an input directive. The operator now selects the test to be performed by typing one of the test selection directives listed in Table 7-2.

Table 7-2. Test Selection Directives

Directive	Program Response
T.	Operational test runs once. Program prints double asterisk upon test completion.
T,C.	Operational test runs continuously until terminated by setting SENSE switch 3.
F.	Fault test runs once. Program prints double asterisk upon completion of test.
F,C.	Fault test runs continuously until terminated by setting SENSE switch 3.
S.	Program prints single asterisk to indicate that it is awaiting Sequence Test commands

Note: The operational test should always be allowed to run to its completion or be terminated by setting SENSE switch 3. This is necessary in order to provide restoration of initial contents of certain memory locations whose contents are altered during the operational test.

In the case of the sequence test, the operator must enter the test commands required to specify operand values and a sequence of up to eight FPP instructions. Specified operand values are stored in software registers. These software registers are memory locations which are dedicated to the storage of operand values. They are addressed by the mnemonics R1, R2, D1, D2, and I; where R1 and R2 are dedicated to the storage of single precision operands, D1 and D2 are dedicated to the storage of double-precision operands, and I is dedicated to the storage of an integer operand. Table 7-3 summarizes the available sequence test commands.

7.6 ERROR MESSAGES

If the operator deviates from the correct format during the entry of a test selection directive or sequence test command, the program responds by printing a question mark after the faulty character and then printing a double asterisk or single asterisk as appropriate to indicate that it is awaiting another test selection directive or Sequence Test command.

When an FPP error is detected by the program, an error-type number is printed on the console TTY. The error type number is followed by the location in the test program where the error was detected. By referring to the test program listing it may be determined which instruction (or sequence of instructions) caused the error. Error type 7 or 8 can occur during the operational test only. Error types 2, 3, 4, 5, or 9 can occur during the fault test only. Error types 1 and 6 can occur during any test. Table 7-4 provides a summary of these error-type messages.

7.7 TTY PRINTOUT EXAMPLE

The printout shown in Table 7-5 occurred during FPP tests run on a V73 processor with 32K of core memory, a model ASR-33 TTY, a high-speed paper tape reader, a PIM module on device address 040, and an FPP.

Table 7-3. Sequence Test Commands

Command Format	Operand or Instruction Specified
Rn,i,j. n = (1,2)	This command loads a single precision operand into software register R1 or R2 (as specified by the value of n). Parameter i specifies the octal value of the exponent and high fraction fields found in the first operand word of a single precision operand. Parameter j specifies the octal value of the low fraction field found in the second operand word of a single precision operand. (See figure 3-3.) EXAMPLE: R1, -40720,0 Here: Rn = R1 specifying software register R1 i = 40720 (octal) = 10000011 1010000 (binary) where 10000011 specifies an exponent value of +3 in excess-128 code and 1010000 specifies a fraction value of .101. j = 0 The minus sign preceding the i specifies a negative operand. Thus, the specified operand is $-2^3 \times .101$ (binary) = -101 (binary) = -5 (decimal) Note: If no sign precedes the i field, a positive operand is specified.
Dn,i,j,k,l. n = (1,2)	This command loads a double precision operand into software register D1 or D2 in accordance with the value of n. Parameter i specifies the octal value of the excess-128 exponent code found in the first operand word of a double

Table 7-3. Sequence Test Commands (continued)

Command Format	Operand or Instruction Specified
Dn,i,j,k,l, n = (1,2) (cont'd)	precision operand while parameters j, k, and l specify the octal values of the high, middle, and low fractions found in the second, third, and fourth words respectively. (See figure 3-5.) EXAMPLE: D2, 202, 52525, 52525, 52525 Dn = D2 specifying software register D1 i = 202 (octal) = 010 000 010 (binary) = 130 (decimal) = +2 in excess -128 code j = k = l = 52525 (octal) = 101 010 101 010 101 (binary)
I,i.	This command loads a signed integer into software register I. Parameter i specifies the octal value of the integer. A minus sign preceding the i specifies a negative number. EXAMPLE: I,377. i = 377(octal) = 011 111 111 (binary) = 255 (decimal)
k,f,r.	Specifies an FPP load or arithmetic instruction and a source register and (optionally) indirect addressing. Parameter k is the mnemonic of the instruction to be performed (FLD, FLDD, FAD, FADD, FSB, FSBD, FMU, FMUD, FVD, or FDVD). Parameter f = 1 specifies indirect address. In this case the first address supplied to the FPP when the instruction is executed is not the direct address of the specified software register but is, instead, and indirect address.

Table 7-3. Sequence Test Commands (continued)

Command Format	Operand or Instruction Specified
k,f,r (cont'd)	This tests the capability of the FPP to request additional memory cycles until a direct address is received. If parameter f = 0 or if parameter f is omitted, direct addressing is provided. r specifies the source register mnemonic. R1 or R2 must be specified for a single precision operation while D1 or D2 must be specified for a double precision operation.
k,f.	Specifies an FPP, float, fix, or store instruction. This is similar to format k,f,r. except that the mnemonics that can be specified are FLT, FIX, FST, and FSTD and that no source register is specified. In the case of a FLT instruction, the operand is automatically taken from the I register. The addresses supplied for the FPP for the FIX, FST, and FSTD instructions are addresses dedicated to buffer storage in connection with printout and are specified by the program. (Results of FIX, FST, and FSTD instructions are printed out unless SENSE switch 1 is set.) Note: Instruction sequences must end with a FIX, FST, or FSTD instruction or a time-out interrupt will occur.
E.	This command initiates a single execution of the previously entered sequence of FPP instructions. The program prints a single asterisk after completing the sequence to indicate that it is waiting for another directive. A repetition of the E command will produce another single execution of the sequence.

Table 7-3. Sequence Test Commands (continued)

Command Format	Operated or Instruction Specified
	Note: Commands which load operands have no affect on previously specified instruction sequences. Thus, the same instruction sequence can be run for different operand values without the need for re-entering the sequence. However, any instruction entry following the execution of an E command erases the previously stored instruction sequence.
E,C.	This command initiates repetitive execution of the previously entered sequence which continues until terminated by setting SENSE switch 3.
G.	This command terminates the sequence test mode. In response, the program prints a double asterisk to indicate that it is now waiting for a test selection directive.

Table 7-4. Error Messages

Error Type Number	Description
1	An actual result differs from the expected result. The actual result is printed on the line following the program location. The expected result is printed on the line following the actual result.
2	FPP fails to interrupt after executing instruction which should produce exponent overflow
3	FPP fails to interrupt after executing instruction which should produce exponent underflow
4	FPP fails to interrupt after executing instructions involving integer overflow.
5	FPP fails to interrupt after executing instruction involving division by zero.
6	FPP interrupts when no fault condition is expected
7	Error in sequence of CPU instructions intermixed with FPP instructions.
8	Real-time clock interrupt occurs during period when FPP should be inhibiting interrupts
9	FPP fails to interrupt when a time-out should have occurred

Table 7-5. Sample TTY Printout

Printout	Comments
THIS IS THE V70/620 or V75 TEST EXECUTIVE MEMORY SIZE IS 32K	MAINTAIN III test executive program identifies itself and the memory size after loading
L. FLOATING POINT PROCESSOR TEST USE REAL TIME CLOCK? N	Operator initiates loading of FPP test program. Program is identified when loading is complete and inquiry about enabling of real time clock interrupts is answered in negative.
**T. **F. **S.	Single passes through the operational test and fault test are requested and successfully completed and the sequence test is entered.
*R1,40540,0. *D2,202,60000,0,0. *FLDD,D2. *FMU,R1. *FSTD,I. *E. 000204 044000 000000 000000	Data is loaded into single-precision register R1 and into double-precision register D2, a sequence of three FPP instructions is entered, the E command is entered, and upon completion of the sequence a double-precision result is printed out (as a result of the double-precision store instruction).
*D1,377,40000,0,0. *D2,377,40000,0,0. *FLDD,D1. *FADD,D2. *FSTD. *E. 6 (000700) 000377 077777 077777 077777	Data is loaded into the two double-precision registers and the instruction sequence required to add these two operands and store the result is entered. The E command is then entered. An exponent overflow occurs as indicated by the type 6 error message. The value in parenthesis is the memory location of the starting instruction of the sequence.

Table 7-5. Sample TTY Printout (continued)

Printout	Comments
*R1,234,8? *R1,2-? *R1,-2,2,? *RO? *R1← *R3? *R2,123456? *G. **	Several invalid sequence test commands are entered and rejected by the program. The G command is then entered, terminating the sequence test mode. The program prints a double asterisk to indicate that it is ready to accept another test selection directive.
FLOATING POINT PROCESSOR TEST USE REAL TIME CLOCK? Y **T. **F. **T,C. ** **	The test program is restarted by using console controls to start the processor at location 0500. On this start, the inquiry about enabling of real-time clock interrupts is answered affirmatively. Successful single passes through the operational test and the fault test are completed. The operational test is then executed in the continuous mode and runs until SENSE switch 3 is set.