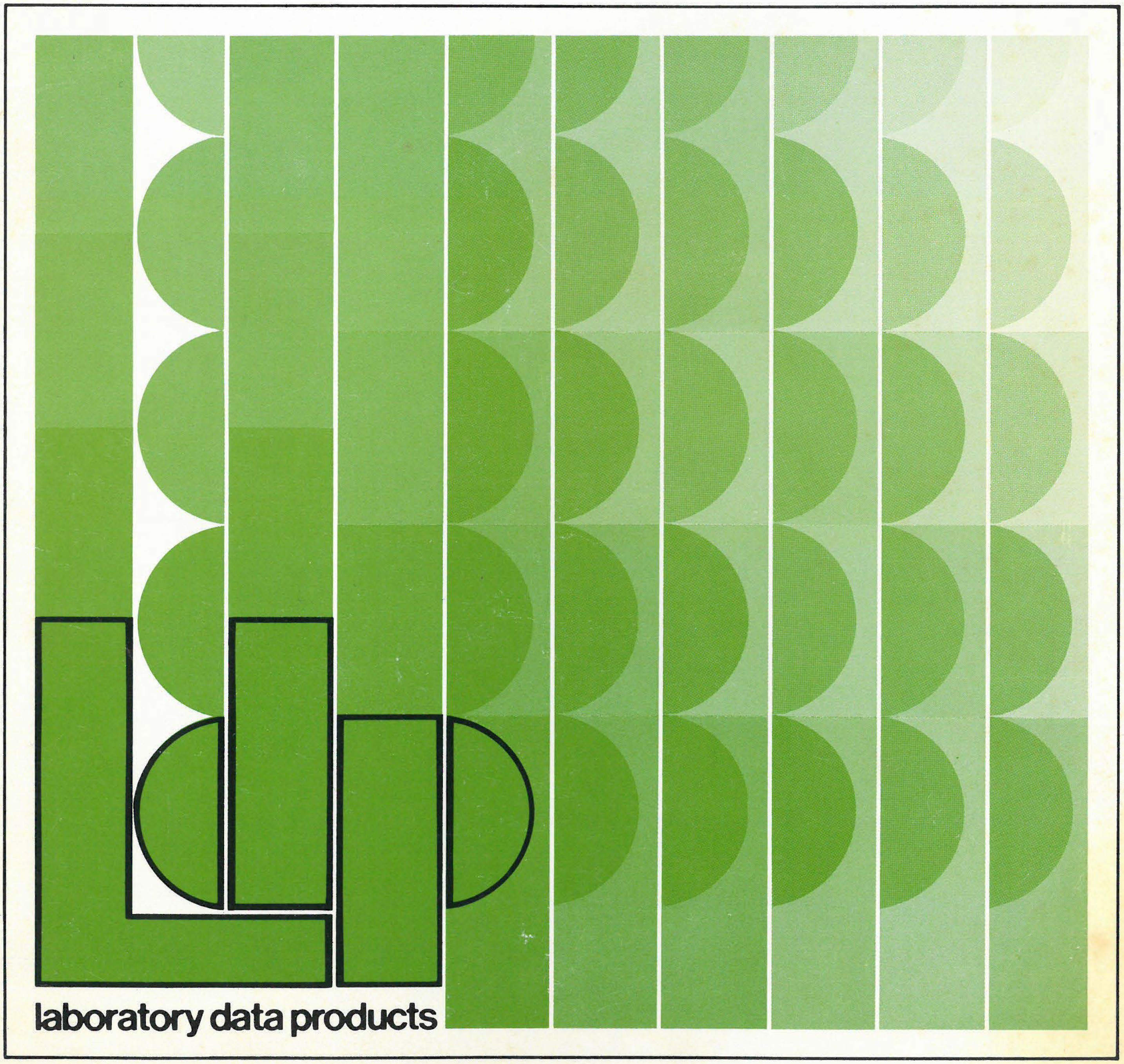
Digital Equipment Corporation
Maynard, Massachusetts

digital

FPP12A floating-point processor user's manual



DEC-12-HFPPA-A-D

FPP12A floating-point processor user's manual

digital equipment corporation · maynard. massachusetts

1st Edition, January 1973
2nd Printing, December 1973

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

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

Printed in U.S.A.

CONTENTS

Page

CHAPTER 1 INTRODUCTION

CHAPTER 2 DESCRIPTION

CHAPTER 3 PROGRAMMING

CONTENTS (Cont)

CONTENTS (Cont)

Page

CONTENTS (Cont)

ILLUSTRATIONS

Figure No.

ILLUSTRATIONS (Cont)

TABLES

TABLES (Cont)

EXAMPLES

CHAPTER 1 - INTRODUCTION

1.1 Scope of the Manual

This manual contains information for programming and servicing the Floating Point Processor (FPP12-A), a peripheral device for the PDP-8 Family, Linc-8, and PDP-12 computers. Chapter 1 will describe the concept of Floating Point Processors in general. Chapter 2 contains a basic system description. Chapter 3 contains information for programming with Chapter 4 having several programming examples. The remaining Chapters 5 through 7 are geared toward maintenance, but contains useful information about the inner workings of the FPP. The installation procedure is in Chapter 8.

1.2 Arithmetic Computation with Mini Computers

As the task for small machines becomes more complex, their performance becomes increasingly inadequate, because their instruction set is too basic for efficient program execution. In many cases, the mini-computer must execute 100 times more code than the computer center machine, for a given task. A way of increasing the efficiency of mini-computers is to implement hardware floating point arithmetic. Several schemes for implementing this hardware are discussed.

In the past, mini-computers have not proved suitable for complex calculations, because they had limited amounts of core and little or no mass storage such as magnetic tapes or disc. Adding extra core or bulk storage devices to the mini-computer was often a tremendous task, because the software supplied by the manufacture was not tape or disc oriented.

Now small computers such as the PDP-12 and PDP-8 are supplied with tape or disc operating systems that take advantage of all extra core beyond the minimum 4 thousand words. Still, the basic mini-computer performs data reduction much slower than the large computer center machine. This performance problem is due in large part to the fact that the limited instruction set of the small computer often requires the mini-conputer to execute 100 instructions for every one executed by the computer center machine for a typical job. The greater efficiency of the large computer instruction set permits the writing of rather sloppy compilers which reduce programming efforts and yet still performs acceptably time wise.

Large computer center machines often dedicate a mini-computer or equivalent to handling I/O; permitting the arithmetic unit to operate for long periods undisturbed by program interrupts. In the mini-computer, one control and arithmetic unit often handles all I/O as well as calculations. The overhead associated with performing several tasks results from the necessity of saving and restoring CPU conditions each time a job is interrupted Even with improved hardware and software monitors that make the programming of interrupts transparent, the fact remains that for the duration of the interrupt service routine the main calculation algorithm is halted.

1.3 Upgrade Mini-Computer By Adding Floating Point Arithmetic

It is clear that adding floating point arithmetic instructions to current mini-computers will speed up the execution of the majority of the data reduction algorithms. The term, floating point, implies a movable binary point in a similar manner to the movable decimal point in scientific notation. As shown in example 1, an exponent is used to keep track of the number of spaces the binary or decimal point is moved. Given a fixed number of bits, it is generally desirable to adjust the fraction to eliminate leading zeros. This retains the maximum number of significant bits for a given amount of space.

Example 1:

$234 = 23.4 \times 10' = 2.34 \times 10^2$

$(1011) = (101.1) \times 2' = (10.11) \times 2^2 = 0.01011 \times 2^5$

It is generally understood that in the performance of floating point arithmetic the hardware will automatically adjust the exponent without the programmers intervention. For instance, if the programmer request the addition of two numbers that have different exponents, the hardware will adjust the fractions of the two numbers such that the exponents are equal prior to the addition.

Generally, floating point instructions are "faked" on mini-computers by the use of software interpreters. These software packages are subroutines written in the computers basic assembly language that interpret instructions written in a more convenient job oriented language. The floating point subroutine supplied by Digital Equipment Corporation for its 12 bit computers, is called as shown in example 2. In example 2, the subroutine float, utilizes the contents of location X + 1 through X + 4 as arguments. These arguments are interpreted as floating point instructions. The argument list can be of almost unlimited length because it has a terminator FEXIT which causes the return from subroutine as part of the argument list.

Example 2:

```
Loc
X JMS Float            /Jump to floating point interpreter
X + 1 FGET A           /Load A into the floating accumulator
X + 2 FMPY B           /Multiply the floating AC by B
X + 3 FBUT A           /Store the results in A
X + 4 FEXIT            /Leave the interpreter and return to X + 5
X + 5 HLT              /Halt program done
```

While these software interpreters are often very flexible they require a large number of machine cycles to perform an arithmetic operation.

Another significant problem is that software floating point interpreters take up core space which is already at a premium in the small computer.

1.4 Implementation of Floating Point Hardware

There are a number of ways to implement floating point hardware on minicomputers. The most straight forward is to design a new computer with the new instructions built-in. This approach has basic defects. First, it could completely obsolute computer installations in the field. Second, a new computer design requires engineering consideration of I/O, memory and console structure which have been well defined in existing units. Third, offering floating point hardware as part of the basic computer increases its price for customers who do not need to perform sophisticated calculations. For a company that has an ever increasing list of similar computers in the field, it is attractive to offer a field installable option that improves the system performance. In this way the customer can gradually convert his software to use the new equipment with a minimum of disruptions to his existing installation.

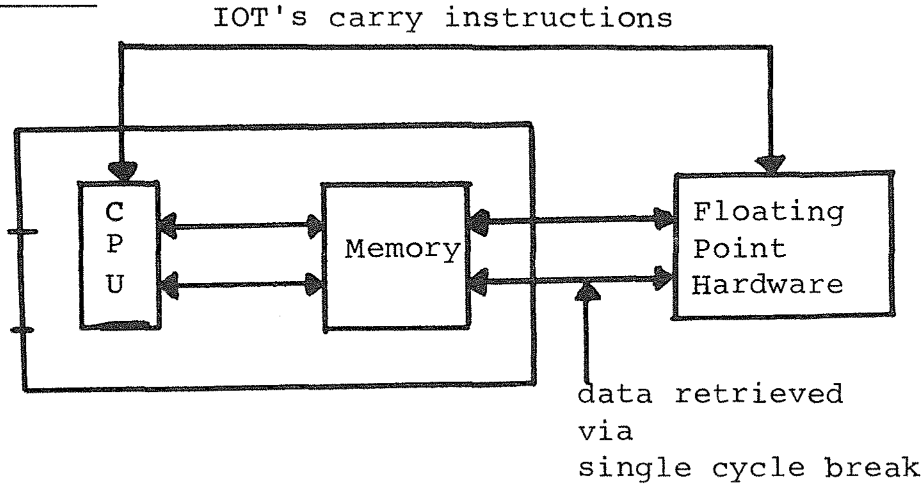
1.4.1 Type of Access to Data

There are two basic paths for communicating with most mini-computers. The most commonly used path for devices such as the teletype involves execution of an instruction by the CPU which transfers the contents of a register to or from the external equipment. This type of operation is called programmed I/O and in the PDP-8 or PDP-12 computer requires the execution of an input/output transfer (IOT) instruction.

Bulk storage devices such as disc units that transfer up to 100,000 words/second to or from the computer are often initialized with IOT instructions, but transfer the bulk of data via a direct access to memory (DMA) or data-break port. The DMA or data break facility, available on almost every mini-computer, permits an external device to access memory without affecting the program counter, accumulator (s), or instruction register in the CPU. In effect, the external device "steals" memory cycles from the CPU. The overhead for this type of memory access in minimal. In its most efficient form, each memory access request by an external device requires only one memory cycle.

A method of implementing floating point hardware (FPH) on a PDP-12 or PDP-8 type computer is diagramed in fig. 1. The instructions and operand addresses could be conveyed to the FPH via IOT instructions while the operand themselves would be retrieved directly from memory.

Figure 1-1

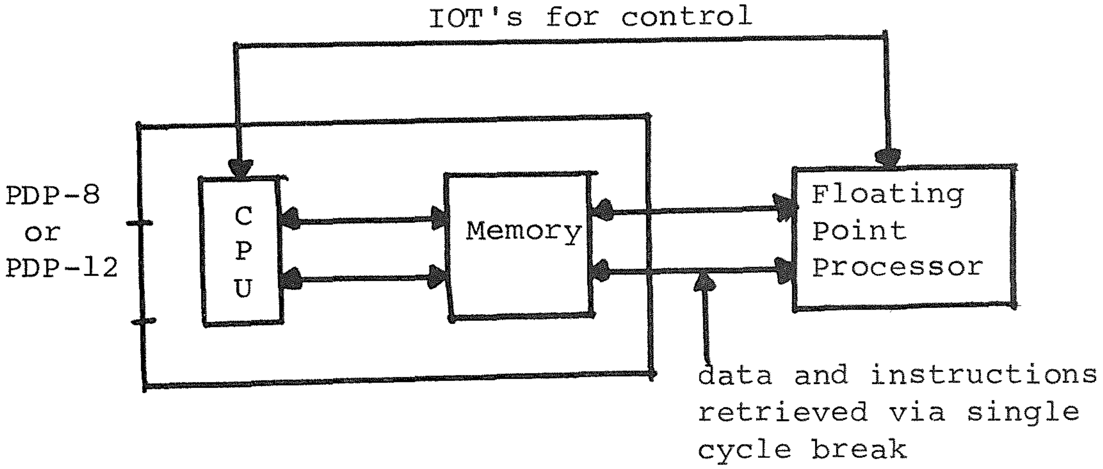


While this design is simple in concept, it does have a number of drawbacks. First it requires the execution of a number of CPU instructions for every floating point operation. Because of this, it does not permit true parallel operation of I/O and calculation. This parallel operation is important both in data acquisition and data display in which the CPU might be called on to write data on tape or disc and up-date or refresh a cathode-ray tube while performing a calculation. In multi-task or time shared environment, input/output is handled by a "monitor". Typically, "user" programs are prevented from issuing I/O instructions by a hardware "trap" that interrupts any IOT issued by other than the "monitor".

Because the issuance of IOT instructions by the time sharing monitor involves complicated record keeping, it is doubtful that floating point hardware that requires an IOT instruction per floating point instruction, would actually be advantageous.

Viewing the handicaps of IOT's to carry instructions, the remaining method was employed to make the floating point processor (FPP12) transfer instructions and data via the direct access to memory as diagrammed in Figure 2.

Figure 1-2



It is activated by the PDP-8 or PDP-12 CPU through the use of programmed input/output transfer (IOT) instructions. Once activated, the FPP12 "steals" memory cycles from the CPU both for fetching instructions and operands. The FPP-12 is a parallel processor with its own instruction set, program counter, and accumulator. The FPP12 and the CPU simultaneously execute instructions.

CHAPTER 2 DESCRIPTION

2.1 System Description

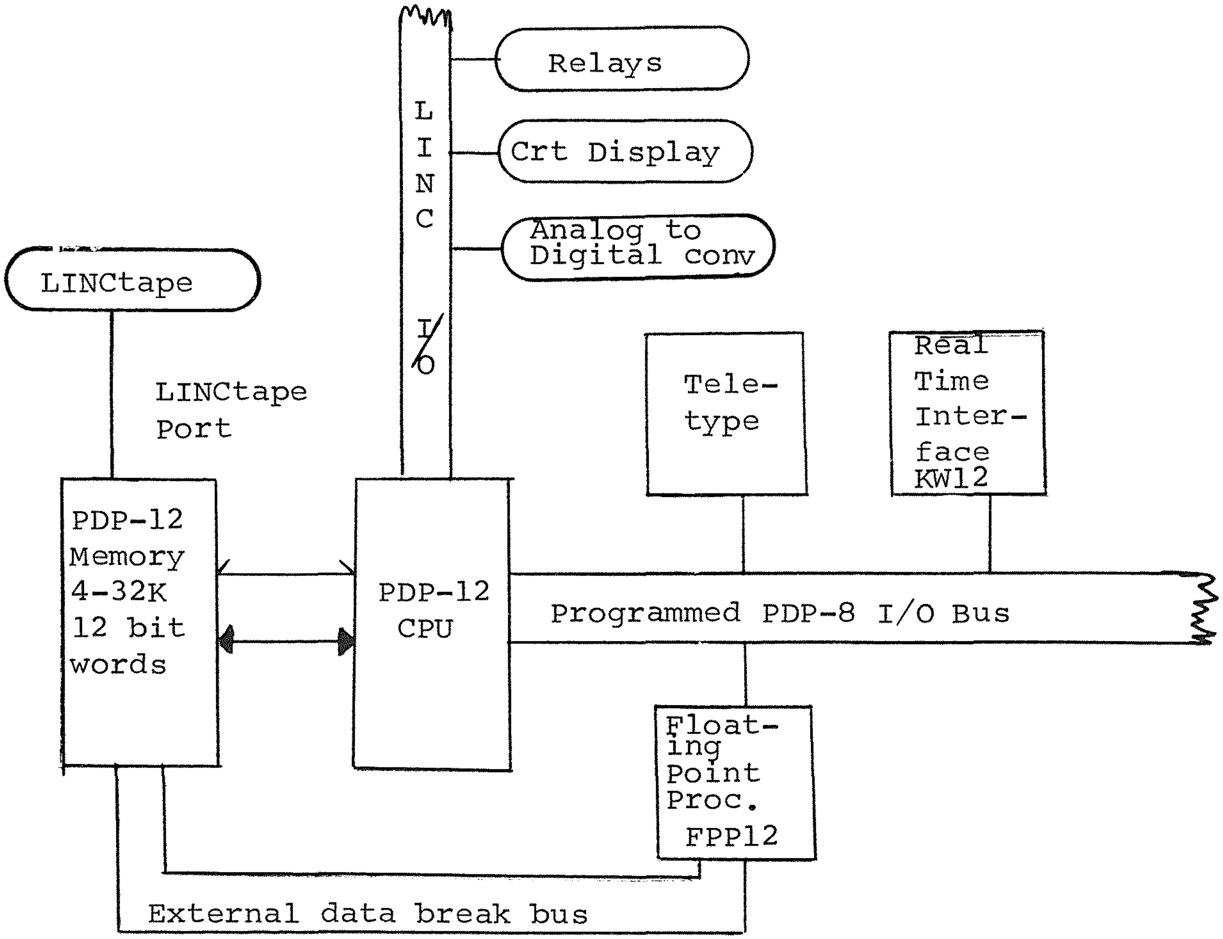
The Floating Point Processor (FPP12) is a programmable, peripheral, digital processor that is attached to the input/output (I/O) bus of any PDP-8 family, LINC-8, or PDP-12 Computer. The FPP12 is initialized and interrogated as to its status via PDP-8 IOT instructions issued on the programmed I/O bus. Once initialized, the FPP12 operates as a processor, fetching instructions and operands via the direct memory access bus.

When activated, the FPP12 steals a maximum of 50 percent of the memory cycles from the PDP-12, LINC-8, or PDP-8 type computers. For the PDP-12 there are two operating modes, parallel and serial.

In parallel mode, the FPP12 steals a maximum of every other memory cycle from the PDP-12, thus permitting the PDP-12 and the FPP12 to operate simultaneously. Once initiated in serial mode, the FPP12 locks out the PDP-12 CPU for the duration of a complete calculation. Serial mode increases the FPP12 calculation speed by approximately 20 percent.

The FPP12 performs arithmetic operations on floating-point numbers 20 to 100 times faster and with 100 to 200 fewer memory cycles than software interpreters. The FPP12 instruction set facilitates the programming of complicated algorithms and the building of compilers for mathematical languages. Variable length instructions are part of a flexible addressing scheme. Direct addressing of 32K of core memory is available using a 24 bit instruction format. A 12-bit instruction format, in which the operand address is relative to a programmable base register, reduces program length and facilitates re-entrant coding. Any eight sequential core locations can be used as an index register to modify operand addresses. Index registers are adjusted prior to use in address modification, to account for the different number of core locations used in the three data format permitted by the FPP12.

A typical system configuration consisting of a PDP-12 and a FPP12 is shown in Figure 2-1. Note that the PDP-12 computer contains two data break ports: one is permanently reserved for the LINC-tape control, the other is available for a device, in addition to the LINCtape, are attached to the PDP-12, a memory multiplexer (DM12) is generally required (see Figure 2-2).



The FPP12 attaches to the EXTERNAL data break and programed I/O bus of the PDP-12 computer without additional hardware.

Figure 2-1 PDP-12 System Configuration

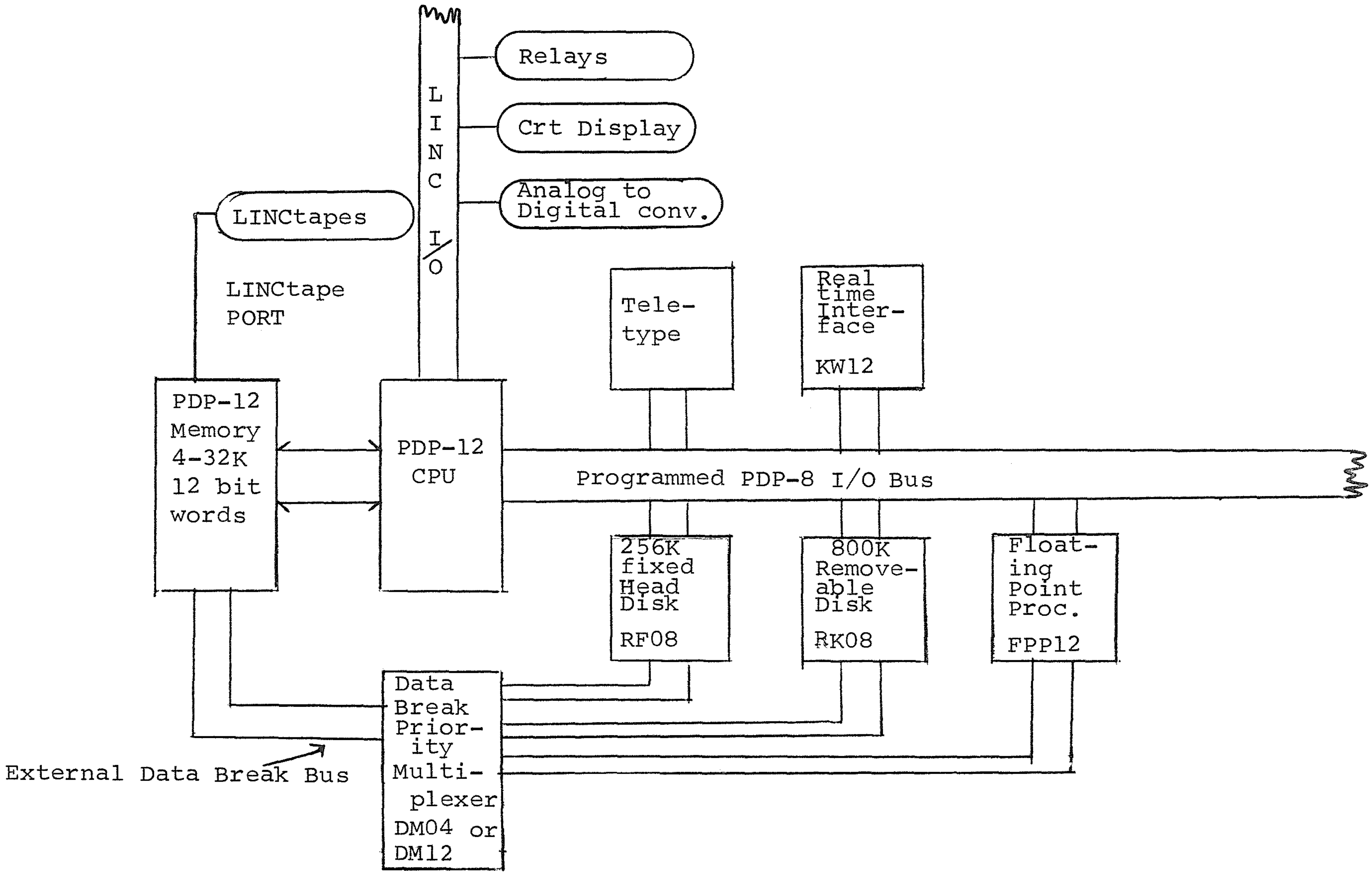


Figure 2-2 Typical Configuration of the PDP-12 Multiple Devices

On the PDP-8, PDP-8/I, and PDP-8/L Computers, only one direct memory access port is available: attaching on FPP12 and DECtape, for example, requires a memory multiplexer (DM01 or DM04). Each data break device has its own memory port on the PDP-8/E computer; therefore, a separate memory multiplexer is not required for up to 12 separate data break devices.

2.2 Floating Point Number System

The term floating point implies a movable binary point similar to the movable decimal point used in scientific notation. An exponent is used to keep track of the number of spaces the binary or decimal point is moved.

Examples of scientific notation:

$234 = 23.4 \times 10^1 = 2.34 \times 10^2$

Examples of binary floating-point notation:

$(1011) = (101.1) \times 2^1 = (10.11) \times 2^2 = (1.011) \times 2^3$

$(1.011) \times 2^3 = 0.1011 \times 2^4 = 0.01011 \times 2^5$

In the example of binary floating-point notation given above, there are four significant bits. However, in the last term, the fraction that multiplies the exponent contains six bits. Given a fixed number of bits, it is desirable to adjust the exponent and the binary point to eliminate insignificant leading ones and zeros. This adjustment retains the maximum numerical significance for a given format length. The FPP12 normalizes as the last step in every floating-point arithmetic operation.

2.3 Floating Point Data Formats

There are two floating-point data formats available; the standard 24 bit format, and the optional 60 bit format referred to as the extended precision mode (EPM).

With the FPP12, the number range is 2^{+2047} to 2^{-2048}; precision is maintained at 24 bits or 60 bits through the number range. Exceeding the upper limit, 2^{+2047}, causes the FPP12 to interrupt the PDP-12 CPU and set its exponent overflow status bit. A calculation resulting in a exponent smaller than 2^{-2048} is an exponent underflow that can cause a program interrupt. At initialization, the programmer has the option to request that the underflow trap be ignored, in which case the result of calculation in which underflow occurred is set to 0.

Floating Point 24-bit Format

The floating-point data format used by the FPP12 in Fig 2-3 below, is identical to the format used by the PDP-8 floating-point system (DEC-08-YQYB-D). There is a 12-bit signed 2's complement exponent and a 24 bit signed 2's complement mantissa.

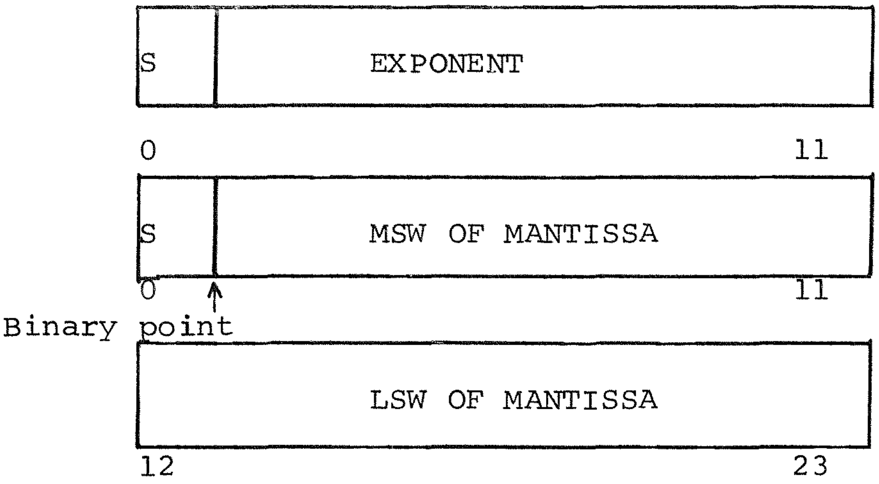
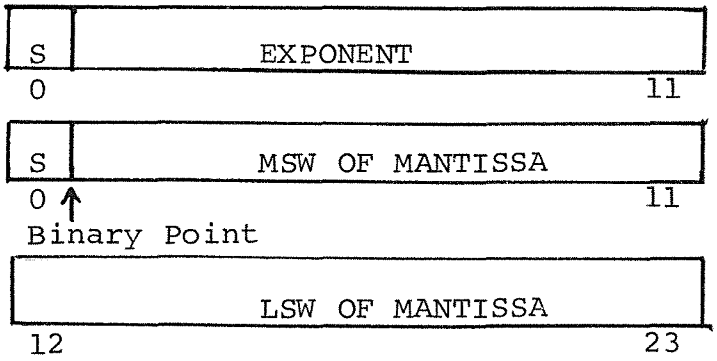


Figure 2-3 24-Bit Data Format

The FPP12 carries all calculations to 28 bits of precision, then rounds up to 24 bits after normalization. After rounding, the result is rechecked for proper normalization prior to completing the instruction.

Floating-Point 6Ø-bit format

The 60-bit data format is referred to as the extended precision mode (EPM). As shown in Fig. 2-4 below, there is a 12-bit signed 2's complement exponent and a 60-bit signed 2's complement mantissa.



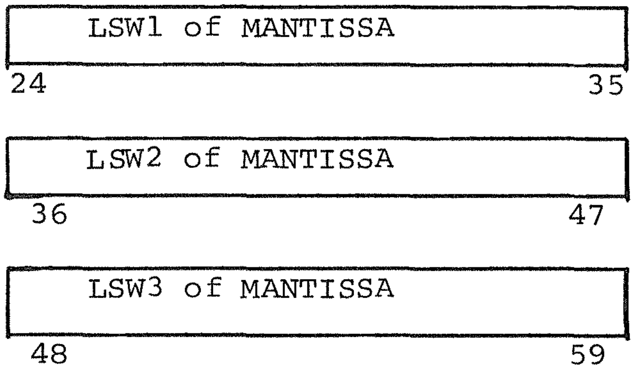


Figure 2-4 6Ø-Bit Data Format

Unlike the standard 24-bit format, the extend precision does not carry calculation beyond its determined word length of 6Ø bits. Typically, the extended presision mode parallels the operation of the standard 24-bit format with the exception of rounding. The differences will be viewed throughout this manual between the standard floating point and the optional EPM.

2.4 Fixed Point Numbers

In fixed point arithmetic, the precision of a number varies with the numbers magnitude. For those calculations where full 24-bit precision is not necessary and where core space is at a premium, the FPP12 can be used in fixed-point 24-bit mode.

2.5 Fixed-Point 24-Bit Data Format

Each operand consists of a 24-bit signed 2's complement fraction as shown below.

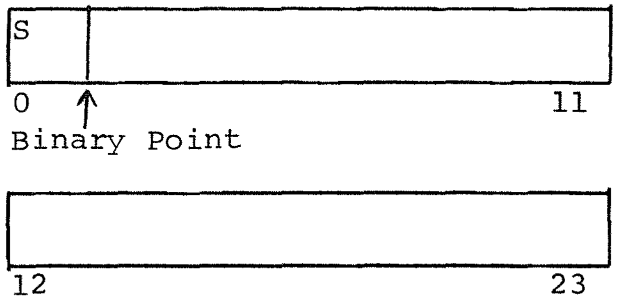


Figure 2-5 Fixed Point 24-Bit Data Format

Each calculation is carried to 28 bits of precision and rounded to 24 bits but no normalization is performed. Therefore, leading zeros occur which reduce the precision of subsequent calculations. Calculations resulting in a fraction overflow cause the FPP12 to initiate a program interrupt with the fraction overflow status bit set to 1.

2.6 Active Parameter Table

The Active Parameter Table (APT) (refer to table 2-1) contains information necessary for starting or restarting an FPP12 program. The APT is defined as 8 consecutive core locations (standard floating point) or 11 consecutive core locations (extended precision mode). The APT pointer is set to point at the first location of the APT. The initialization procedure for the FPP12 includes two IOT instructions that; sets up the command register, and sets the 15-bit APT pointer to the first location of the APT, shown as location P in table 2-1. Following the second IOT, the FPP12 picks up the contents of the APT via data breaks. When the FPP12 performs an EXIT, the current contents of the APT overlay the initial APT contents.

The APT performs three services for the programmer.

a. It reduces the number of IOT's necessary to initialize the FPP12. This reduces the CPU program overhead which is critical in multitask and time-shared environments.

b. It automatically saves the status of interrupted FPP12 programs.

c. It provides convenient access to the information necessary for debugging FPP12 programs and determining the cause of FPP12 "error" exits such as exponent overflow, underflow, or attempted division by 0.

With the exception of the operand address, all parameters contained in the APT are picked up when initializing the FPP12. The operand address is stored for the use of the CPU program when the FPP12 exits.

ACTIVE PARAMETER TABLE FORMAT TABLE 2-1

LOCATION	CONTENTS			
P	Field Bits of Operand Address	Field Bits of Base Register	Field Bits of Index Register Location	Field Bits of FPC
P + 1	Lower 12 bits of FPC			
P + 2	Lower 12 bits of Index Register 0 location X 0			
P + 3	Lower 12 bits of Base Register			
P + 4	Lower 12 bits of operand address			
P + 5	Exponent of FAC			
P + 6	MSW of FAC			
P + 7	LSW of FAC			
P + 8	LSW1 of FAC		These locations only accessed in EPM	
P + 9	LSW2 of FAC			
P + 1Ø	LSW3 of FAC			

NOTE: APT address points to location P.

2.7 FPP12 Register Organization

There are eight registers in the FPP that are of interest to the programmer. The functions of these registers, named below, will be discussed through out this manual.

Register	Function
Floating Point Accumulator (FAC)	36-bit register split into 12 bit exponent and 24 bit fraction, or 72 bit register split into a 12 bit exponent and 60 bit fraction if equipped with the extended precision logic.
Index Register Address Pointer (XO)	Contains the 15 bit core location of index register 0.
Base Register (PØ)	Contains the 15 bit base address used in calculating single word addresses.
Floating Point Program Counter (FPC)	Contains the 15 bit address that is the location of the next FPP12 instruction.
Active Parameter Table Pointer (ADRS)	Contains the 15 bit address of the first location of the active parameter table (APT). This 15 bit address is loaded via two IOT's.

REGISTER	FUNCTION
Active Parameter Table Pointer (ADRS) continued -	The first IOT must be FPCOM for loading the field bits of the 15 bit address. The second IOT must be FPST which loads bits 03-15 of the 15 bit address. The IOT FPST then starts the FPP.
Command Register (CR)	The command register is loaded with an IOT instruction. The command register selects FPP12 operating modes, sets the FPP12 interrupt enable, chooses the important parameters to be saved in the APT, and fixes the most significant 3 bits of the 15 bit APT pointer.
Status Register (SR)	The status register may be interrogated by the CPU to determine the cause of an exit operation by the FPP12. The status register also indicates if the FPP12 is in the run or (run) $\wedge$ (FPAUSE) state.

REGISTER	FUNCTION
Operand Address Register (OP ADDRS)---	The operand address register is deposited in the APT and contains one of the following. a. If the last address-bearing instruction prior to the exit was of the data reference class, the operand address register contains the 15 bit address of the least significant word of the operand. b. If the last address-bearing instruction prior to the exit was an executed jump instruction, the operand address register contains the jump address. c. If after initialization an exit is performed prior to the execution of a jump or data reference instruction, the operand address register contains the FPC originally set by the APT. d. The instructions SET BASE (SET B) and SET X0 REGISTER (SET X) have no effect on the operand address register.

3.1 DESCRIPTION

The FPP12 is initialized and interrogated by PDP-8 type IOT instructions. Once started, the FPP12 operates much like an actual processor, fetching instructions and operands and storing results in the PDP-8 or PDP-12 core memory. Data breaks or "stolen" memory cycles are generally requested by the FPP12 as needed. The maximum number of breaks requested is generally one per regular PDP-8 or PDP-12 instruction. This means that while the FPP12 is operating, PDP-8 or PDP-12 programs can be run simultaneously at 50 to 70 percent of normal speed. Typically LINCtape, display, analog data acquisition, and other forms of I/O can be performed by the PDP-12 Computer while the FPP12 is calculating.

An optional mode is available to the FPP12 attached to a PDP-12 Computer. For calculations where the maximum FPP12 program speed is required, setting the proper command register bit (refer to Table 3-2) locks out the PDP-12 processor during FPP12 program execution. Using the "lock out" mode on the PDP-12 speeds up FPP12 programs by 15 to 20 percent (refer to Table 3-1).

TABLE 3-1

Instruction Execution Times*

Instruction	Octal Code	Serial Mode			Parallel Mode**		
		Fixed-Point Execution Time (μS)	Floating Point Execution Times (μS)		Fixed-Point Execution Time (μS)	Floating Point Execution Times (μS)	
			24-BIT	6Ø-BIT		24-BIT	6Ø-BIT
FLDA	0200+X	12	14	23	14	16	27
FADD	1200+X	13	19	28	14	21	32
FSUB	2200+X	13	19	28	14	21	32
FDIV	3200+X	26	3Ø	52	27	32	55
FMUL	4200+X	25	29	53	27	3Ø	56
FADDM	5200+X	17	26	44	22	29	5Ø
FSTA	6200+X	12	14	23	14	16	27
FMULM	7200+X	29	35	68	32	39	75

*All times were measured using the single-word direct reference format. Timing tolerance is ±20%.

**For these measurements the PDP-12 was performing mostly single cycle instructions.

3.2 Serial vs Parallel Processing

The most efficient use of resources occurs when the CPU and FPP12 are programmed to operate in parallel. For instance, in the display oriented research analysis (DORA) program which faciliates display interactive manipulation of data files, the PDP-12 refreshes a CRT display, performs Teletype®, LINCtape, and disk I/O, and samples knob and sense switch positions while the FPP12 is performing floating-point arithmetic. Because the FPP12 and the CPU access the same core memory, the communication methods are virtually unlimited; either processor can alter the other's program or data. Usually the CPU is assigned the job of scheduling and I/O, while the FPP12 performs complex arithmetic. However, in the DORA program, the FPP12 schedules I/O by passing parameters to the PDP-12 CPU.

There are occasions when it is desirable to complete an FPP12 calculation between operations performed by the CPU. Setting the appropriate command register bit in the FPP12 permits serial operation with the PDP-12 Processor. In serial mode, the PDP-12 CPU is locked out from the executing instructions while the FPP12 is operating.

There is no provision for a true serial mode for an FPP12 on a PDP-8 type processor. The fastest wait loop for a PDP-8,

®Teletype is a registered trademark of Teletype Corporation.

PDP-8/I, LINC-8 or PDP-8L Computer consists of a JMP instruction with the programmed interrupt facility enabled, because data breaks can occur only between complete instructions. On the PDP-8/E Computer, the data break facility is structured so that data breaks may occur after any major state or multistate instructions. Therefore, the particular CPU program in progress does not affect the FPP12 instruction execution time on a PDP-8/E Computer.

3.3 Initialization

To execute the first instruction of any program, the FPP12 must have the 15-bit core address of the first instruction that is contained in the first two locations of the APT. The contents of other locations of the APT are often useful in starting a program and essential in restarting an interrupt task. Once the appropriate parameters are placed in an APT table by the CPU, two IOTs must be issued. FPCOM (6553) loads a command register and the most significant 3 bits of the APT pointer. The significance of the bits in the command register is shown in Table 3-2. FPST (6555) loads the least significant 12 bits of the APT pointer and starts the FPP12. Once initiated, the FPP12 will execute instructions until:

a. An error condition, such as exponent overflow, occurs.

b. An FEXIT instruction is encountered.

c. An FPHLT IOT is issued by this CPU.

d. An I/O preset is issued by the CPU.

e. The CPU encounters any type of halt.

3.4 IOT Instructions

A complete list follows of IOT instructions with device code 55 that apply to programming the FPP12. IOT instructions with device code 56 are relegated to maintenance programs with the exception of the 6567 load shift counter instruction which has been expanded to select the extended precision mode if implemented. The use of maintenance IOTs is presented in Chapter 7. If a conflict exists between the FPP device select codes and the device select codes of another peripheral, the conflict must be resolved in the hardware by altering wired connections in either the FPP12 or the conflicting device. It is recommended that the FPP12 device codes not be altered because of the necessity of changing extensive diagnostic and system software. However, the logic to be altered in changing device codes is found on Prints FPP12-0-CI1, FPP12-0-CI2, and FPP12-0-CI3.

3.5 IOT List

Mnemonic	Octal Code	Function
FPINT	6551	Skip when the FPP12 Interrupt Reqeust flag is set.
FPICL	6552	Unconditionally reset the FPP12 including all flags. To the FPP12, the IOT FPICL is the same as I/O preset.
FPCOM	6553	If the FPP12 is not in a Run state and the FPP12 Interrupt Request flag is not set, the FPP12 command register is loaded with the contents of the AC*. If the FPP12 is in a Run state, or if the Interrupt Request flag is set, the FPCOM instruction is ignored. See Table 3-2.
FPHLT	6554	Force the FPP12 to exit, dump its status in the APT, and set the Interrupt Request flag at the end of the current instruction. The FPHLT instruction is used to abort an FPP12 program in a multijob environment or in software debugging. The following special features apply to the FPHLT instruction. a. If FPHLT is issued prior to the FPST instruction, the FPP12 will execute only one instruction after initiation and then exit with the FPC pointing to the succeeding instruction. This facilitates single stepping through an FPP12 program under CPU control. b. If the FPP12 is in a Pause state, the FPP12 will exit with the FPC pointing at the pause instruction. This means that if a job was aborted in a Pause state it will be resumed in a Pause state.

c. Normally, if an exit is forced by FPHLT, AC02 will be set to a 1 when either read status FPRST or FPIST is issued. However, if the forced exit causes the FPP12 program to abort while an FEXIT instruction is being executed, the CPU forced exit flag is cleared. Thus, the CPU forced exit flag is an absolute indicator that a program was prematurely aborted.

FPST	6555	If the FPP12 is not running and the Interrupt Request flag is not set, the least significant 12 bits of the APT pointer are set to the contents of the AC and the FPP is started. If the FPP12 is in a Run state, but paused, the FPST instruction will cause the FPP12 to continue. Otherwise, the FPST instruction has no affect on the FPP12. If the FPST instruction causes the FPP12 to start or continue, the CPU will skip the instruction following FPST.
FPRST	6556	Read the FPP12 status register into the AC. FPRST may be issued at anytime.
FPIST	6557	Skip if the FPP12 Interrupt Request flag is set. If the skip occurs, read the FPP12 status register in the AC and clear the status flags and the Interrupt Request.
Select Extended Mode	6567	Selects the extended precision mode provided the AC=4ØØØ, the FPP is not in the Run state, and the FPP is equipped with the extended precision. The AC is cleared at the completion of this instruction. The 6567 command must be executed after the FPCOM (6553) if the EPM is desired as the FPCOM selects either Fixed-Point or Floating Point (24-bit) modes which will reset the EPM flop. The 6567 is also used as a Maintenance IOT.

Table 3-2

Command Register Setting

AC Bit*	Function when AC Bit Set to 1
AC bits 0-11 have the following function when the FPCOM IOT is issued	
0	Select fixed-point mode upon initiation.
1	Exit if exponent underflow occurs. Otherwise, set result of calculation and continue.
2	Forbid access to 4K memory fields other than the field that is occupied by the last location of the APT.
3	Enable CPU program interrupt when FPP12 Interrupt Request flag is set. Skip is always enabled.
4	Do not store operand address on exits. The operand address is never retrieved on initiate.
5	Do not store the address of index register 0 from or in the APT.
6	Do not store the base register from or in the APT.
7	Do not store the FAC from or in the APT.
8	Lock out the PDP-12 processor during FPP12 program execution. Unused on PDP-8 FPP12 systems.
9	Most-significant 3 bits of APT pointer.
10	
11	
Note: Setting bits 4-7 of the command register speeds up exit operations. Setting command register bits 4-7 does not alter the relative position of items on the APT. In multijob environments, command register bits 4-7 are typically set to zero.	

*AC refers to the PDP-12 or PDP-8 accumulator while FAC refers to the FPP12 accumulator.

Table 3-3

AC After Read Status Instruction

AC Bit	Function if AC Bit Set to 1
0	Fixed-point mode.
1	Trapped instruction caused exit.
2	FPHLT instruction caused exit.
3	Attempted divide by 0 caused exit. The FAC was not altered.
4	Fraction overflow in fixed-point mode caused exit.
5	Exponent overflow caused exit.
6	Exponent underflow has occurred. Exit on exponent underflow is optional.
7	Unused
8	Unused
9	Extended-precision mode.
10	The FPP12 is currently paused.
11	The FPP12 is currently in a run state.

3.6 Index Registers

Any core location may be used as an index register. The core address of the current index register 0 is stored in the X0 register. The X0 register is initially set from the APT, but may be altered by the SET X instruction. Index register X is in location X0+X, where X = 0,....,7.

Accessing an array of data points requires incrementing the address of the current point by the data length to yield the address of each successive point. Index registers are used to accomplish this address modification. The index register is incremented by one to access each successive point, but it is multiplied by the data length, six for extended precision, three for floating point and two for fixed point. This quantity is used as the displacement from the address specified in the instruction to yield the address of the current point. Adjusting of index registers simplifies "skipping" through data arrays and permits a single index register to be used as both a loop counter and address modifier (see Example 4-2). Pre-incrementing is selected by bit 5 of data reference instructions types a and c. Instructions are available for setting, testing, and performing arithmetic on index registers. In particular, the instruction

ADDX, which adds the contents of bits 12-23 of the instruction to the contents of the index register specified by bits 9-11 of this instruction, is useful in manipulating "push-down stacks".

3.7 Instruction Set

The FPP12 instruction set is divided into two basic classes: data reference instructions and special instructions. Data reference instructions are those that operate on the three data formats specified in Paragraph 3.7.1. Data reference instructions include the basic arithmetic operations plus load and store. All other instructions are special instructions that include index registers modifiers, jumps, pointer moves, and the operate-type instructions.

Three types of data reference instructions are available:

a. 24-bit instruction with a 15-bit absolute address.

b. 12-bit instruction with 7-bit relative address.

c. 12-bit instruction with a 3-bit relative address that specifies a 15-bit indirect address.

Full indexing capability is available for types a and c. The determined operand address points at the exponent of the operand in floating-point mode and at the most significant word of the operand in fixed-point mode.

The instruction set is presented in detail in the following paragraphs. The instruction format follows each group of instructions. Unless otherwise noted, instructions executed in the extended precision mode perform exactly to the description given for floating-point mode.

3.7.1 DATA REFERENCE INSTRUCTIONS*

OP Code	Mnemonic	
0	FLDA	C(Y)→FAC
1	FADD	C(Y)+C(FAC)→FAC
5	FADDM	C(Y)+C(FAC)→Y
2	FSUB	C(FAC)-C(Y)→FAC
3	FDIV	C(FAC)/C(Y)→FAC
4	FMUL	C(FAC)*C(Y)→FAC
7	FMULM	C(FAC)*C(Y)→Y
6	FSTA	C(FAC)→Y

Data Reference Instruction Formats

OP CODE	1	0	+	X	ADDRESS
0 2	3	4	5	6 8	9 11

ADDRESS
12 23

DOUBLE-WORD DATA REFERENCE INSTRUCTION

Y = C (bits 9-23) + M * (C(X + X0) + C (bit 5)) & (X)

*In fixed-point mode the exponent of the FAC is never altered.

OP CODE	0	1	OFFSET
0 2	3	4	5 12

SINGLE-WORD DIRECT REFERENCE

Y = C (base register) + 3 (offset)

OP CODE	1	1	+	X	OFFSET
0 2	3	4	5	6 8	9 11

SINGLE-WORD INDIRECT REFERENCE

Y + C (bits 21-35 of C (base reg.) +3* offset))
+ (M) * (C(X + X0) + C (bit 5)) δ (X)

δ (X) = 1 if X $\neq$ 0 and 0 if X = 0

M = 2 if fixed-point mode
3 if floating-point mode
6 if extended-precision mode

3.7.2 Special Format 1

OP Code	Mnemonic	
2	JXN	The index register X is incremented if bit 5 = 1 and a jump is executed to the address contained in bits 9-23, if index register X is nonzero.
3 4 5 6 7	Trapped Instructions	The instruction-trap status bit is set and the FPP12 exits causing a PDP interrupt. The unindexed operand address is dumped into the APT.

OP CODE	0	0	+	X	ADDRESS
0 2	3	4	5	6 8	9 11

ADDRESS
12 23

SPECIAL FORMAT 1

3.7.3 Special Format 2

OP Code	Extension	Mnemonic	Function
0	10	LDX	The contents of the index register specified by bits 9-11 are replaced by the contents of bits 12-23.
0	11	ADDX	The contents of bits 12-23 are added to the index register specified by bits 9-11.

3.7.4 Conditional Jumps

Jumps, if performed, are to the location specified by bits 9-23 of the instruction.

OP Code	Extension	Mnemonic	Function
1	0	JEQ	Jump if FAC = 0
1	1	JGE	Jump if FAC ≥ 0
1	2	JLE	Jump if FAC ≤ 0
1	3	JA	Jump always

OP Code	Extension	Mnemonic	Function
1	4	JNE	Jump if FAC ≠ 0
1	5	JLT	Jump if FAC < 0
1	6	JGT	Jump if FAC > 0
1	7	JAL	Jump if impossible to fix the floating-point number contained in the FAC; i.e., if the exponent is greater than $(23)_{10}$.

NOTE: In EPM, the jumps look at a 6Ø-bit FAC.

3.7.5 Pointer Moves

OP Code	Extension	Mnemonic	Function
1	10	SETX	Set X0 to the address contained in bits 9-23 of the instruction.
1	11	SETB	Set the base register to the address contained in bits 9-23.
1	13	JSR	Jump and save return. Jump to the location specified in bits 9-23 and the return is saved in bits 21-35 of the first entry of the data block.
1	12	JSA	An unconditional jump is deposited in the address and address + 1, where address is specified by bits 9-23. The FPC is set to address + 2.

OP CODE	0	0	EXTENSION	F
0 2	3	4	5 8	9 11

Y
12 23

SPECIAL FORMAT 2

3.7.6 Special Format 3

OP Code	Extension	Mnemonic	Function
0	1	ALN	The mantissa of the FAC is shifted until the FAC exponent equals the contents of the index register specified by bits 9-11. If bits 9-11 are zero, the FAC is aligned so that the exponent = $(23)_{10}$.* In fixed-point mode, an arithmetic shift is performed on the FAC fraction. The number of shifts is equal to the absolute value of the contents of the specified index register. The direction of shift depends on the sign of the index register contents. A positive sign indicates a shift toward the least significant bit, while a negative sign indicates a shift toward the most significant bit. The FAC exponent is not altered by the ALN instruction in fixed-point mode.
0	2	ATX	The contents of the FAC are fixed and the least significant 12 bits of the mantissa, bits 12-23, are loaded into the index register specified by bits 9-11. In fixed-point mode the least significant 12 bits of the FAC, bits 12-23 are loaded into the specified index register by the ATX instruction.

*Setting the exponent = $(23)_{10}$ integerizes or fixes the floating-poin number. The JAL instruction tests to see if fixing is possible.

OP Code	Extension	Mnemonic	Function
0	3	XTA	The contents of the index register specified by bits 9-11 are loaded right-justified into the FAC mantissa, bits 12-23. The FAC exponent is loaded with $(23)_{10}$ and then the FAC is normalized. This operation is typically termed floating a 12-bit number. In fixed-point mode, the FAC is not normalized. The least significant three word of the FAC, bits 24-59 are cleared.

NOTE: The ALN, ATX and XTA instructions, when in the extended precisic mode, will fix or float the FAC based on $(23)_{1\text{Ø}}$ not $(59)_{1\text{Ø}}$. The entire FAC will be shifted during the ALN and ATX instructions.

0	4	NOP	The single-word instruction performs no operation.
0	5	STARTE	Start extended-precision mode.
0	6-7	reserved	These codes are reserved for instruction set expansion and should not be used.
0	12-17		
1	14-17		

3.7.7 Operate Group - Special Format 3

OP Code	Extension	9-11 Bits	Mnemonic	Function
0	0	0	FEXIT	Dump active registers into the APT, reset the FPP12 RUN flip-flop to the 0 state, and interrupt the PDP-12 processor.

OP Code	Extension	9-11 Bits	Mnemonic	Function
0	0	1	FPAUSE	Wait for synchronizing signal. IOT FPST (6555) will restart the instruction following FPAUSE.
0	0	2	FCLA	Zero the FAC mantissa and exponent.
0	0	3	FNEG	Complement FAC mantissa.
0	0	4	FNORM	Normalize the FAC. In fixed-point mode FNORM is a NOP.
0	0	5	START F	Start floating-point mode. If issued in extended precision mode, the C(FAC) is rounded to 24 bits.
0	0	6	START D	Start fixed-point mode.
0	0	7	JAC	Jump to the location specified by the least significant 15 bits of the FAC mantissa.

OP CODE	0	0	EXTENSION	F
0 2	3	4	5 8	9 11

SPECIAL FORMAT 3

CHAPTER 4 FPP12 PROGRAMMING EXAMPLES

4.1 INTRODUCTION

Programming examples for the Floating Point Processor and a procedure for initializing the FPP12 are contained in this chapter. Several examples are provided that utilize index registers. A re-entrant sine subroutine illustrates a technique for writing re-entrant code. Program debugging techniques are discussed in detail. The mnemonics and syntax used in this chapter are consistant with those of the FPP assembler. A complete description of the assembler can be found in the manual entitled, FPP Assembler Manual, DEC-12-AQZA-D. A math package for the FPP is described in a manual entitled FPP Support Library (DEC-12-YEXA-D). There is also a Real-Time Programming System (RTPS) with ASA Standard Fortran IV available for the FPP12. This system approaches or exceeds the performance of many larger systems. RTPS Fortran IV is an extension to the OS/8 system software (see OS/8 Software Support Manual DEC-08-MEXB-D) and as such uses many of the existing OS/8 programs, particularly the keyboard monitor, command decoder and editor. The reader interested in the RTPS should acquire the RTPS Fortran IV users Manual (DEC-08-LRTPA-A-D) which describes both the RTPS Fortran IV operating system and the RTPS Fortran IV language with many programming examples.

4.2 PROGRAM INITIALIZATION

Each FPP12 program consists of one or more instructions and an Active Parameter Table (APT). Upon initialization, the APT (refer to Table 2-1) contains the initial setting of important FPP12 registers. Whenever the FPP12 finishes or aborts a program, the APT is updated before the CPU is interrupted.

The CPU program in Example 4-1 starts the FPP12 in floating point mode with the APT pointer set to location 01000, which is word 1000 of field 0. The FPP12 normally does not recognize page or field boundaries. If the APT started in location 07777, the least significant 12 bits of the FPC would be found in location 10000.

In Example 4-1, the FPP12 will pick up locations 02000, 02001, 02002, 02003, 02005, 02006, and 02007 of the APT. Note that the operand address, location 02004 in this example, is never retrieved from the APT by the FPP12. After retrieving the contents of location 02007, the FPP12 will fetch its first instruction from location 01000. The 4 in the second digit of the contents of location 01000 indicates that the instruction is a 2-word, direct addressing, data reference instruction. The 0 in the first digit of location 01000 indicates that the instruction is an FLDA. Bits 9-23 of the instruction specify the address, which is not indexed when bits 6-8 are all zero. After fetching the address, the FPP12 will break to 12000, 12001, and 12002 to load the operand into the FAC.

After retrieving the least significant word of the FAC from location 12002, the FPP12 will fetch another instruction from location 01002. The instruction in location 01002 is an FEXIT, equivalent to a halt instruction for the CPU. Prior to stopping, the FPP12 dumps the current APT over the initial APT, beginning with the least significant word of the FAC in location 02007 and ending with location 02000. The APT at the completion of the FEXIT instruction is shown in Table 4-1.

```
/Sample program to initialize FPP12

                  ORG       00020      /Psuedo OP sets assembler orgin
                                       at location (20)  of field 0.
00020     2000    APTPT,    APT        /Pointer to APT 8

                  ORG 200
                  BEGIN,    CLA        /Clear AC
                            FPCOM      /Load 0's to FPP12 command register
                            TAD APTPT
                            FPST       /St APT points to 02000 and start
                            HLT        /if no skip FPP12 is not ready
                            FPINT      /Wait             to be started
                            JMP. -1
                            HLT        /Program done

/ A Sample
/FPP12 Program is below

Loc       Contents                     ORG 02000

01000     0401                         FLDA TAG     /Load contents of
01001     2000                                      /location TAG into
                                                     FAC
01002     0000                         FEXIT        /Dump APT
                                                    /into core and
                                                    interrupt CPU

/ Active parameter table

Loc      Contents                      ORG 02000

02000       0                        APT,  0000     /most sig bits
02001    1000                              1000     /FPC
02002    3000                              3000     /XO
                                           4000     /Base
                                           ----     /Operand address
                                           ----     /FAC EXP
                                           ----     /FAC MSW
                                           ----     /FAC LSW
ORG 12000
12000    0002    TAG,    3.0                        /constant (3.0)10
12001    3000
12002    0000
```

Example 4-1 Sample FPP12 Program

Table 4-1

APT After FEXIT is Example 4-1

```
02000          1000          /current Field Bits
02001          1003          /current FPC
02002          3000          /XØ
02003          4000          /Base
02004          2002          /Operand address
02005          0002          /exponent
02006          3000          /MSW
02007          0000          /LSW
```

Only after dumping the APT is the FPP12 Skip or Interrupt flag set. In Example 4-1, the CPU executes a WAIT loop while the FPP12 is operational. It would be far more efficient for the CPU to perform some other task, such as tape or Teletype I/O, while the FPP12 is calculating.

4.3 INDEX REGISTERS AS ADDRESS MODIFIERS AND LOOP COUNTERS

The FPP12 program in Example 4-2 moves a list of $(200)_8$ floating point numbers from an area of core starting at location ALPHA to an area starting at location BETA. Note that index registers are used both for loop counting and address modification. Index register 1 is set to -1 and index register 0 is set to -200 using the LDX instruction. Index register 1 is incremented prior to use as an address modifier for the FLDA instruction at location LOOP. Index register 0 is used as a loop counter by the JXN instruction.

```
BEGIN,      LDX  -1,1         /set index register 1 =-1
            LDX  -200,0       /set index register Ø = (-2ØØ)8
LOOP,       FLDA ALPHA, 1+    /first C(1 = XØ_ = C(1 +XØ)
                              +1 Then load FAC̄ from loc.
                              ALPHA + C(1 + XØ) *3
            FSTA BETA, 1      /Store FAC in loc BETA + C
                              (1 + XØ) *3
            JXN LOOP, 0+      /first C (XØ + Ø) = C(XØ + Ø) Ø
                              +1
                              /then go to loop if C (XØ + Ø)±Ø
            FEXIT             /trap to CPU

            ORG 4000
ALPHA,      ------
            ORG 6000
BETA,        0
```

Example 4-2 Move List from ALPHA to BETA Using Index Registers

It is possible to use the same index register as a loop counter and as an address modifier, because of the method used in the FPP12 hardware to calculate indexed addresses. In the process of formulating an address, the FPP12 checks to see if indexing is required. If indexing is required, the contents of the specified index register are retrieved and "adjusted" by the appropriate multiplier, which is 6 for extended-precision mode, 3 for floating-point mode and 2 for fixed-point mode. Then the adjusted index register is added to the unindexed address and the resulting addition, initially performed with 24 bits of precision, is truncated to 15 bits by dropping the 9 most significant bits of the result. Example 4-3 illustrates the standard method of indexed address calculation. If it is necessary to use index register 5 as a loop counter, additional care must be used in selecting the pointer to list A contained in the instruction. Consider the case where the loop counter is set to $(-200)_8$. Then the pointer to list A must be modified to be A + M(C (I) + $(10000)_8$. C(I) is the initial setting of the index register and M is the number of 12-bit bytes in the data word. Example 4-4 is similar to Example 4-2; however, only one index register is used.

Initially X0 = 14003

C(X0+5) = 0001

Instruction FLDA A,5 0451
2003

A is location 12003

Address calculation proceeds as follows:

1. The contents of (X0 + 5) are retrieved and multiplied by three.
2. The "adjusted" index register is added to 00012003 the unindexed address to yield 00012006.
3. This address is truncated to 12006.

Example 4-3 Indexed Address Calculation

4.4 USE OF INDEX REGISTERS TO CREATE PUSH-DOWN STACKS

The subroutines in Example 4-5 illustrate the use of the ADDX instruction in creating push-down or last-in-first-out lists. The PUSH routine is called with an argument in the accumulator. The POP routine returns with elements removed from the stack in the accumulator. These subroutines are designed to be called with the JSA instruction which places an unconditional jump to the return in the first two locations of the subroutine. The PUSH and POP subroutines in Example 4-5 are valid for either fixed-point, floating-point mode or extended-precision mode, as long as second and additional calls are in the same mode as the very first call.

```
BEGIN,        LDX-COUNT, 1

LOOP,         FLDA ALPHA      -( M* COUNT-K),1
              FSTA BETA + M*  (COUNT-K),1
              JXN LOOP, 1+

M = 3, if floating point mode
    2, if fixed point mode

K = 10000
```

Example 4-4 Index Register 1 is Used as Both an Address Modifier and Counter

```
  PUSH,      0
             0
             FSTA STACK, 2+       /Place contents of AC in stack
             JA PUSH              /Return from subroutine

  POP,       0
             0
             FLDA STACK, 2        /Retrieve item from stack
             ADDX  -1,  2         /Decrement stack pointer
             JA POP               /Return from subroutine
```

Example 4-5 Push-Down Stacks

4.5 BRANCH OR JUMP ON CONDITION INSTRUCTIONS

Seven conditional jump instructions are provided in addition to the JXN instruction. Six of these, JEQ, JGE, JLE, JNE, and JGT, test the FAC mantissa, The seventh, JAL, executes a jump if the FAC cannot be represented as a $(24)_{10}$ bit binary number. This occurs when the FAC exponent is greater than $(23)_{10}$ or 27_8.

4.6 WRITING RE-ENTRANT SUBROUTINES

A re-entrent subroutine is one in which the code is not altered during execution. This property permits the interruption of a task which is executing a given re-entrant subroutine and the starting of another task that uses the same subroutine. The advantage of re-entrant coding is that two or more jobs can use the same subroutine without concern as to when a given job is interrupted.

The single-word data reference instructions and a re-entrant jump to subroutine facilitate the writing of re-entrant codes. With the JSR instruction, the return address is saved in bits 21-35 of the location pointed at by the contents of the base register. If it is necessary to store temporary values during subroutine execution, single-word instructions should be used. This will force addressing to be relative to the base register setting. Each task will have a unique base register setting; therefore, the effective addresses for temporary storage for each task will have a unique base register setting; therefore, the effective addresses for temporary storage for each task will be unique, even though the offsets for the data instructions are never charged in the pure subroutine. The return from the re-entrant subroutine consists of the two instruction sequence, FLDA ALPHA JAC, shown in Example 4-6. JSR causes the return address to be deposited into the first location of the data block, ALPHA, which is defined by the base register. The return address is deposited into the FAC with the instruction FLDA ALPHA. The JAC instruction actually executes the return jump by setting FPC equal to bits 9-23 of the FAC mantissa.

```
/ Main Prog

MPROG,  JSR SUB          /Jump to sub prog.
        FEXIT

SUB,    FLDA ALPHA       /Load return address
        JAC              /Jump to the address contained in
                         /bits 9-23 of the FAC fraction
        Base ALPHA
ALPHA,  ------
```

Example 4-6 Return from Re-Entrant Subroutine

4.7 USE OF THE FPHLT INSTRUCTION

The FPHLT IOT (6554) permits the CPU to force the abortion of a running FPP12 program or to force the FPP12 to execute one instruction each time it is initialized. In a multitask or time-shared environment, it is often necessary to suspend a calculation prior to completion. When debugging a program, it is often desirable to examine the results of each instruction's execution.

If FPHLT is issued while the FPP12 is executing a program, that program, will be aborted at the end of the current FPP12 instruction. The FPP12 will dump the current APT in core and then cause a CPU program interrupt. If the current instruction is anything except FEXIT, status bit 02 will be set to 1 if FPHLT forced the FPP12 to stop program execution.

Issuing FPHLT prior to FPST will cause the FPP to initialize, execute one instruction, then exit. By repeating this procedure the CPU can force the FPP12 to single step thnough a program.

4.8 DEBUGGING FPP12 PROGRAMS ON UNITS ATTACHED TO PDP-12 COMPUTERS

The PDP-12 console (described in the PDP-12 System Reference Manual)is a powerful tool for debugging FPP12 programs. Using the switches, one can single step through FPP12 programs, observing the transfers between the FPP and the PDP-12 memory on the console lights. Alternatively, the FPP12 program can run until a specific memory address is accessed, in which case the computer will halt, permitting the console light to be examined. While the computer is halted, memory may be examined and altered with the switch register without disturbing the program counters associated with either the CPU or the FPP12. IOT instructions may be issued with the console switches that examine registers within the FPP12.

If the stop switch is raised during the execution of a FPP12 program, the PDP-12 will stop at the end of a complete instruction or a data break caused by some external device such as the FPP12. Depressing the continue switch with the stop switch raised causes the execution of one CPU instruction or one data break for each actuation of the continue switch. Operating in this mode, the FPP12 will receive one data break for each CPU instruction. This means that every other time the continue switch is depressed a data break will occur. Whenever the break indicator light is lit, the MA and MB lights on the console refer to the data break address and memory buffer contents associated with the FPP12 program*. The single step switch causes similar results, except the halts occur at the end of each major state of the CPU instructions. The single step switch is useful when the CPU program that runs in parallel with the FPP12 program contains tape instructions. The stop switch has no affect for the duration of LINC tape instructions, or more exactly, if the inprogress light is lit. (IP)

If bit 8 of the FPP12 command register is set to 1, the CPU will be locked out while FPP12 programs are executing. This is reflected in the fact that the break light will stay on

* For the sequence of breaks for instructions and major states, see Chapter 7.8.

continously as the continue switch is actiated.

4.9 USING THE EXECUTE STOP SWITCH

If the execute stop switch is raised the PDP-12 will halt whenever the memory location whose address is contained in the left switches is accessed during any cycle except a CPU fetch cycle. Setting the left switches to the first location of the next APT to be used and raising the execute stop switch causes the PDP-12 to halt following the first FPP12 data break following FPST IOT.

4.1Ø CARE NECESSARY IN THE USE OF EXAMINE AND DEPOSIT SWITCHES

Some care is necessary when using the examine and deposit switches, if they are to be used while a FPP12 program is temporarily halted. Problems arise because of the logical implementation of the break field register within the PDP-12. The 4K memory field examined on the first push of the examine switch following a program may be the field into which the FPP12 was breaking when the program stop occurred. To be sure that the proper data for an examine operation is displayed in the MB register, the examine switch should be actuated twice for the first operation following a program stop. When the computer system is restarted, the first PDP-12 cycle following an examine or deposit operation will be a break cycle if the FPP12 is requesting a data break. To ensure that the FPP12 breaks to the proper 4K memory field, the last operation after any series of examines and deposits must be a fill; fill-step. This sequence should be addressed to a non existant memory field or a unimportant core location.

4.11 ADDITIONAL PROGRAMMING HINTS

4.11.1 Illegal Mantissa

In the 2's complement number system the number consisting of a one followed by twenty-three zeros is an illegal number because it and its 2's complement are both equal to -1. The FPP12 logic will not allow this number to be generated as the result of any calculation. For instance, if -1/2 is added to -1/2 the result shows up in the FPP as -1/2 *2 or -1. It is possible for this number to arise in other than calculations. For instance, it is possible to intentionally place a number into core memory from the CPU's switches. The routine in Example 4-7 illustrates a test for the illegal fraction.

```
/The value in location A possibly has an illegal fraction.

BEGIN,     FLDA      A     /Get C (A)
           JGE       GOOD  /If C(A) 0 all is OK
           FNEG            /Form 2's complement of fraction

           JLT BAD

GOOD,      FEXIT           /Number is OK
BAD,       FEXIT           /Number has illegal fraction
```

Example 4-7 Test for Illegal Fraction 100000000...000

Example of Re-Entrant Sine and Exponential Subroutines

Examples 4-8 and 4-9 contain the FPP code for calculating SINE (X) and X (X**Y). The comments indicate what each step of the routines is doing. Both subroutines are written in the mnemonics and syntax of the FPP assembler.

```
            / SINE USES THE 1ST 3 ENTRIES IN
            / THE BLOCK AND INDEX REG. 0,1&2
            / X IS PASSED THROUGH THE 2ND ENTRY
            / IN THE BLOCK AND SIN(X) IS RETURNED
            / THROUGH THE SAME LOCATION
                    ORG 10500
                    BASE 0
                    X=1*3
                    XSQR=2*3
            / CALCULATE ABSOLUTE VALUE OF X.INDEX
            / REG 0 SET TO 0 INDICATES SIGN OF X
            / WAS NEGATIVE
10500 0201 SINE,    FLDA X           /INITIATE INDEX REG 1
10501 0100          LDX -1,0
10502 7777
10503 1061          JGT CAL          /GO TO CAL IF X IS POSITIVE
10504 0512
10505 1001          JEQ DONE         /GO TO DONE IF X IS 0
10506 0603
10507 0003 MOD,     FNEG             /NEGATE FAC
10510 0100          LDX 0,0          /SET INDEX REG TO ZERO
10511 0000
            / REDUCE X TO 1ST CYCLE USING THE
            / IDENTITY SIN(X)=SIN(N*2*PI*X)
10512 3401 CAL,     FDIV TWOPI       /DIVIDE X BY 2*PI
10513 0607
10514 6201          FSTA X
10515 1071          JAL ERROR        /X IS TOO LARGE
10516 0606
10517 0010          ALN 0
10520 0004          FNORM            /GET INTEGER PART
10521 2201          FSUB X
10522 0003          FNEG             /GET FRACTIONAL PART
10523 1001          JEQ DONE         /SIN(2*PI*N) IS ZERO
10524 0603
10525 4401 REM,     FMUL TWOPI       /NORMALIZE TO BETWEEN 0 AND 2*PI
10526 0607
10527 6201          FSTA X
            / REDUCE X TO 1ST HALF CYCLE USING
            / THE IDENTITY SIN(X)=-SIN(X-PI) FOR
            / PI<X<=2*PI
10530 2401          FSUB PI
10531 0612
10532 1051          JLT PCHECK       /IF X IS LESS THAN PI GP TO PCHECK
10533 0543
10534 6201          FSTA X           /SET X TO X-PI
10535 2101          JXN RESET,0+     /IF INDEX REG 0 WAS -1 SET TO 0 AND
10536 0541
10537 1031          JA PCHECK+1      /GO TO PCHECK+1
10540 0544
10541 0100 RESET,   LDX -1,0         /IF INDEX REG 0 WAS 0 SET IT TO -1
10542 7777
```

Example 4-8 SINE Routine (Sheet 1 of 2)

```
ØØ52                 / REDUCE X TO 1ST QUARTER CYCLE USING
ØØ53                 / THE IDENTITY SIN(X)=SIN(PI-X) FOR
ØØ54                 / PI/2<X<=PI
ØØ55 1Ø543 Ø2Ø1 PCHECK, FLDA X
ØØ56 1Ø544 24Ø1         FSUB PIBY2       /IF X IS LESS THAN OR EQUAL TO PI/2
     1Ø545 Ø615
ØØ57 1Ø546 1Ø21         JLE PALG         /GO TO PALG
     1Ø547 Ø555
ØØ6Ø 1Ø55Ø Ø4Ø1         FLDA PI
     1Ø551 Ø612
ØØ61 1Ø552 22Ø1         FSUB X           /REPLACE X WITH PI-X
ØØ62 1Ø553 1Ø31         JA PALG+1
     1Ø554 Ø556
ØØ63 1Ø555 Ø2Ø1 PALG,   FLDA X
ØØ64 1Ø556 34Ø1         FDIV PIBY2
     1Ø557 Ø615
ØØ65 1Ø56Ø 62Ø1         FSTA X           /NORMALIZE X TO BETWEEN Ø & 1
ØØ66 1Ø561 42Ø1         FMUL X
ØØ67 1Ø562 62Ø2         FSTA XSQR        /CALCULATE X**2
ØØ7Ø 1Ø563 Ø1Ø1         LDX -4,1         /SET UP INDEX REG 1
     1Ø564 7774
ØØ71 1Ø565 Ø1Ø2         LDX -1,2         /SET UP INDEX REG 2
     1Ø566 7777
ØØ72 1Ø567 ØØØ2         FCLA
ØØ73                 / CALCULATE SIN(X)=((((C9*(2*X/PI)**2
ØØ74                 / +C7)*(2*X/PI)**2+C5)*(2*X/PI)**2
ØØ75                 / +C3)*(2*X/PI)**2+PI)*2*X/PI
ØØ76 1Ø57Ø 1521 LOOP,   FADD C9,2+       /ADD C9 ON 1ST PASS, C7 ON
                                          2ND PASS, ECT.
     1Ø571 Ø62Ø
ØØ77 1Ø572 42Ø2         FMUL XSQR        /MULTIPLY PARTIAL SUM BY X**2
Ø1ØØ 1Ø573 2111         JXN LOOP,1+      /GO TO LOOP 4 TIMES
     1Ø574 Ø57Ø
Ø1Ø1 1Ø575 14Ø1         FADD PIBY2
     1Ø576 Ø615
Ø1Ø2 1Ø577 42Ø1         FMUL X
Ø1Ø3 1Ø6ØØ 2ØØ1         JXN DONE,Ø       /GO TO DONE IF X WAS POSITIVE
     1Ø6Ø1 Ø6Ø3
Ø1Ø4 1Ø6Ø2 ØØØ3         FNEG             /NEGATE ANSWER
Ø1Ø5 1Ø6Ø3 62Ø1 DONE,   FSTA X           /STORE ANSWER
Ø1Ø6 1Ø6Ø4 Ø2ØØ         FLDA Ø
Ø1Ø7 1Ø6Ø5 ØØØ7         JAC              /RETURN TO CALL
Ø11Ø 1Ø6Ø6 ØØØØ ERROR,  FEXIT            /EXIT ON ERROR
Ø111 1Ø6Ø7 ØØØ3 TWOPI,  3.1415926*2.Ø
     1Ø61Ø 311Ø
     1Ø611 3756
Ø112 1Ø612 ØØØ2 PI,     3.1415926
     1Ø613 311Ø
     1Ø614 3756
Ø113 1Ø615 ØØØ1 PIBY2,  3.1415926/2.Ø
     1Ø616 311Ø
     1Ø617 3756
Ø114 1Ø62Ø 7764 C9,     +1.514819ØE-Ø4
     1Ø621 2366
     1Ø622 5615
Ø115 1Ø623 7771 C7,     -4.6737656E-Ø3
     1Ø624 5466
     1Ø625 6317
Ø116 1Ø626 7775 C5,     +7.9689679E-Ø2
     1Ø627 2431
     1Ø63Ø 5Ø53
Ø117 1Ø631 ØØØØ C3,     -6.4596371E-Ø1
     1Ø632 5325
     1Ø633 Ø42Ø
```

Example 4-8 SINE Routine (Sheet 2 of 2)

```
                     / EXP USES THE 1ST 6 ENTRIES IN
                     / THE BLOCK
                     / INDEX REG, Ø MUST BE SET TO THE
                     / POSITION OF THE EXPONENT OF THE
                     / 5TH ENTRY IN THE BLOCK
                     / X IS PASSED THROUGH THE 2ND ENTRY
                     / IN THE BLOCK AND EXP(X) IS RETURNED
                     / THROUGH THE SAME LOCATION
                             ORG 1Ø5ØØ
                             BASE Ø
                             X=1*3
                             F=1*3
                             FSQR=2*3
                             TEMP=3*3
                             IDXØ=4*3
                     / CALCULATE THE ABSOLUTE VALUE OF X
                     / INDEX REG 3 SET TO Ø INDICATES THAT
                     / THE SIGN OF X WAS NEGATIVE
1Ø5ØØ  Ø2Ø1  EXP,       FLDA X -         /GET X
1Ø5Ø1  Ø1Ø3            LDX -1,3         /INITIATE INDEX REG 3 TO -1
1Ø5Ø2  7777
1Ø5Ø3  1Ø41            JNE NƵRO         /GO TO NƵRO IF X IS NOT EQUAL TO ƵERO
1Ø5Ø4  Ø511
1Ø5Ø5  Ø4Ø1            FLDA K1          /SET FAC TO 1
1Ø5Ø6  Ø6Ø1
1Ø5Ø7  1Ø31            JA RETURN        /RETURN TO CALL
1Ø51Ø  Ø573
1Ø511  1Ø61  NƵRO,      JGT GTƵERO       /GO TO GTƵERO IF X WAS POSITIVE
1Ø512  Ø516
1Ø513  Ø1Ø3            LDX Ø,3          /SET INDEX REG 3 TO Ø TO INDICATE
                                          X WAS NEGATIVE
1Ø514  ØØØØ
1Ø515  ØØØ3            FNEG             /NEGATE THE FAC
1Ø516  44Ø1  GTƵERO,    FMUL LG2E        /MULTIPLY X BY LOG2(E)
1Ø517  Ø576
1Ø52Ø  62Ø3            FSTA TEMP        /STORE RESULT TEMPORARILY
1Ø521  Ø4Ø1            FLDA K1
1Ø522  Ø6Ø1
1Ø523  62Ø4            FSTA IDXØ        /SET IDXØ TO 1=2**1*1/2
1Ø524  Ø2Ø3            FLDA TEMP
1Ø525  ØØ1Ø            ALN Ø            /FAC=N=INTEGER PART OF X*LOG2(E)
1Ø526  ØØ2Ø            ATX Ø            /IDXØ=2**N*1/2
1Ø527  Ø11Ø            ADDX 1,Ø         /IDXØ=2**(N+1)*1/2=2**N
1Ø53Ø  ØØØ1
                     / THE 5TH ENTRY IN THE BLOCK
                     / CONTAINS 2**N WHERE N IS THE
                     / INTEGER PART OF X*LOG2(E)
                     / FIND F=FRACTIONAL PART OF X*LOG2(E)
1Ø531  ØØØ4            FNORM            /FAC CONTAINS INTEGER PART OF
                                          X*LOG2(E)
1Ø532  22Ø3            FSUB TEMP
1Ø533  ØØØ3            FNEG             /FAC CONTAINS FRACTIONAL PART OF
                                          X*LOG2(E)
1Ø534  62Ø1            FSTA F
```

Example 4-9   Exponential Subroutine (Sheet 1 of 2)

```
0054                / CALCULATE 2**F=1+2*(A-F+B*F**2+
0055                /              C/(D+F**2)
0056 10535 4201           FMUL F
0057 10536 6202           FSTA FSQR         /FSQR=F**2
0060 10537 1401           FADD D
     10540 0620
0061 10541 6203           FSTA TEMP         /TEMP=D+F**2
0062 10542 0401           FLDA C
     10543 0615
0063 10544 3203           FDIV TEMP         /FAC=C/(D+F**2)
0064 10545 2201           FSUB F
0065 10546 1401           FADD A
     10547 0607
0066 10550 6203           FSTA TEMP         /TEMP=A-F+C/(D+F**2)
0067 10551 0401           FLDA B
     10552 0612
0070 10553 4202           FMUL FSQR
0071 10554 1203           FADD TEMP
0072 10555 6203           FSTA TEMP         /TEMP=B*F**2+A-F+C/(D+F**2)
0073 10556 0201           FLDA X
0074 10557 4401           FMUL K2           /FAC=2*F
     10560 0604
0075 10561 3203           FDIV TEMP
0076 10562 1401           FADD K1           /FAC=1+2*F/(B*F**2+A-F+C/(D+F**2))
     10563 0601
0077                / CALCULATE EXP(X)=2**(X*LOG2(E))=
0100                /                 (2**N)*(2**F)
0101 10564 4204           FMUL IDX0
0102 10565 2031           JXN RETURN,3      /GO TO RETURN IF X WAS POSITIVE
     10566 0573
0103                / CALCULATE EXP(-X)=1/EXP(X)
0104 10567 6201           FSTA X
0105 10570 0401           FLDA K1
     10571 0601
0106 10572 3201           FDIV X
0107 10573 6201 RETURN,   FSTA X            /STORE RESULT IN X
0110 10574 0200           FLDA 0
0111 10575 0007           JAC               /RETURN TO CALL
0112 10576 0001 LG2E,     1.442695
     10577 2705
     10600 2434
0113 10601 0001 K1,       1.0
     10602 2000
     10603 0000
0114 10604 0002 K2,       2.0
     10605 2000
     10606 0000
0115 10607 0007 A,        9.954596E+01
     10610 3070
     10611 5703
0116 10612 7774 B,        3.465736E-02
     10613 2157
     10614 5161
0117 10615 0012 C,        6.179723E+02
     10616 2323
     10617 7434
0120 10620 0007 D,        8.741750E+01
     10621 2566
     10622 5341
```

Example 4-9   Exponential Subroutine (Sheet 2 of 2)

CHAPTER 5 HARDWARE DESCRIPTION

5.1 GENERAL

The FPP12 is a peripheral processor that attaches to both the programmed I/O bus and the data break I/O bus. Figure 2-1 shows a typical configuration of an FPP12 attached to a PDP-12,with several other peripherals. It is of major importance to fully understand the difference between the I/O bus of the PDP-12, LINC-8, PDP-8, PDP-8/I, PDP-8/L, and PDP-8/E Computers. All of DEC's 12-bit computers share a compatible I/O structure. Most peripherals such as the FPP12 are nearly plug-in compatible with all of these computers. Major differences are listed below:

a. PDP-8/L, PDP-8/E, PDP-12, and some PDP-8/I's have what is referred to as a positive I/O bus which implies that the I/O signal levels are TTL compatible. The PDP-8, LINC-8, and some PDP-8/I's have a negative I/O bus which implies that the I/O signal levels are 0 and -3V with reference to chassis ground. Bus driver and receiver modules in the FPP12 are selected for either positive or negative bus computers.

b. The sense of the IOP pulses is inverted on those computers with a negative I/O bus. To account for this, certain wiring changes must be made on FPP12 logic to convert it to negative bus units. These changes are detailed in Chapter 8. If the original purchase order for the FPP specifies an FPP12-AN, the negative bus changes will be factory installed.

c. Data break timing on the PDP-12 Computer differs slightly from data break timing on the PDP-8 type computers. On the PDP-12, the trailing edge of ADDRESS ACCEPT indicates memory buffer strobe; on the PDP-8s, PDP-8/I, PDP-8/L, and PDP-8/E the leading edge of BUFFERED TIME STATE 3 indicates memory buffer strobe. The line that carries the signal BUFFERED TIME STATE 3 on the PDP-8 type computer is the same one that carries BUFFERED TIME STATE 5 on the PDP-12 Computer. Therefore, the FPP12 wired in the positive-bus PDP-8/I, PDP-8/L, and PDP-8/E configuration will operate on a PDP-12 but will not achieve optimum performance.

d. Raising pin N16V2 of the I/O cable on a PDP-12 Computer will lock out the CPU. The FPP12-AP will utilize this option when command register bit 8 is set to 1. Time comparisons are shown in Table 3-1. On computers other than the PDP-12, pin N16V2 is used for different purposes. Therefore, run F05U2 - B03V2 is deleted in the FPP12 when it is configured for processors other than the PDP-12.

Data break timing diagrams for the PDP-12 and other Family of 8 Computers are shown in Figures 5-1 and 5-2.

5.2 ORGANIZATION OF HARDWARE COMPONENTS

Floating Point Processor System organization is shown in Figures 5-3, 5-4, and 5-5. The User IOT Decoder System (see Figure 5-3) describes the simplest communication path between the CPU and the FPP12. IOT (device code 55) instructions of interest to systems programmers are described in detail in Chapter 3. Maintenance IOT's (device code 56) are described in Paragragh 7.9.

The FPP12 Timing and Enable system are shown in Figure 5-4. The Major State and Time Slot generator provides up to 2Ø major time states in any 8 major or enable states. Each major time state contains 4 mini states; therefore, a total of 416 time slots are provided by the FPP12 timing system. The timing diagram for the state generator is shown in Figure 5-6. Figure 5-7 displays the extended-precision timing diagram. Typically at any one instant of time, one or two gates in one of the 8 state enable sections is qualified. A qualified gate in the

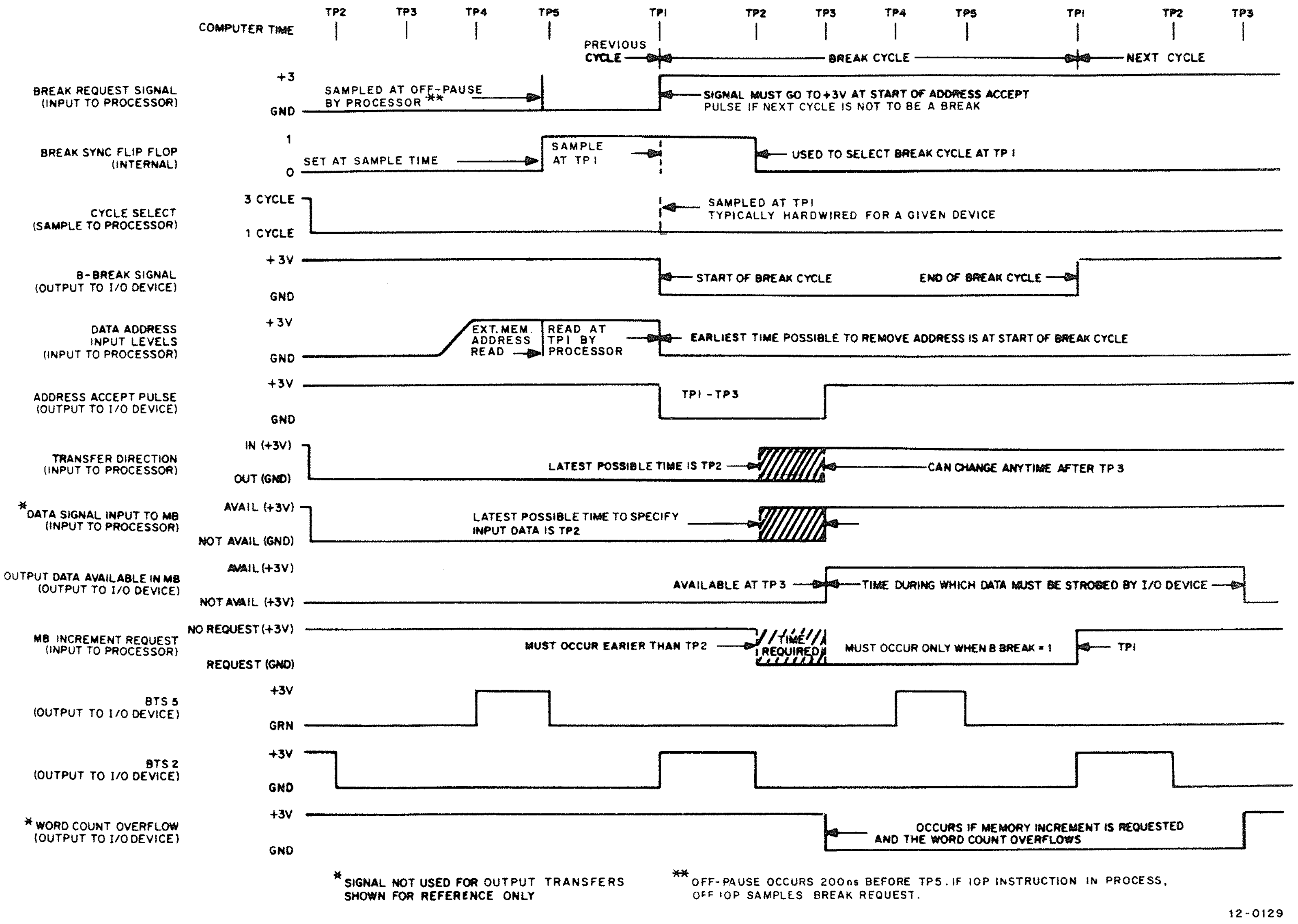
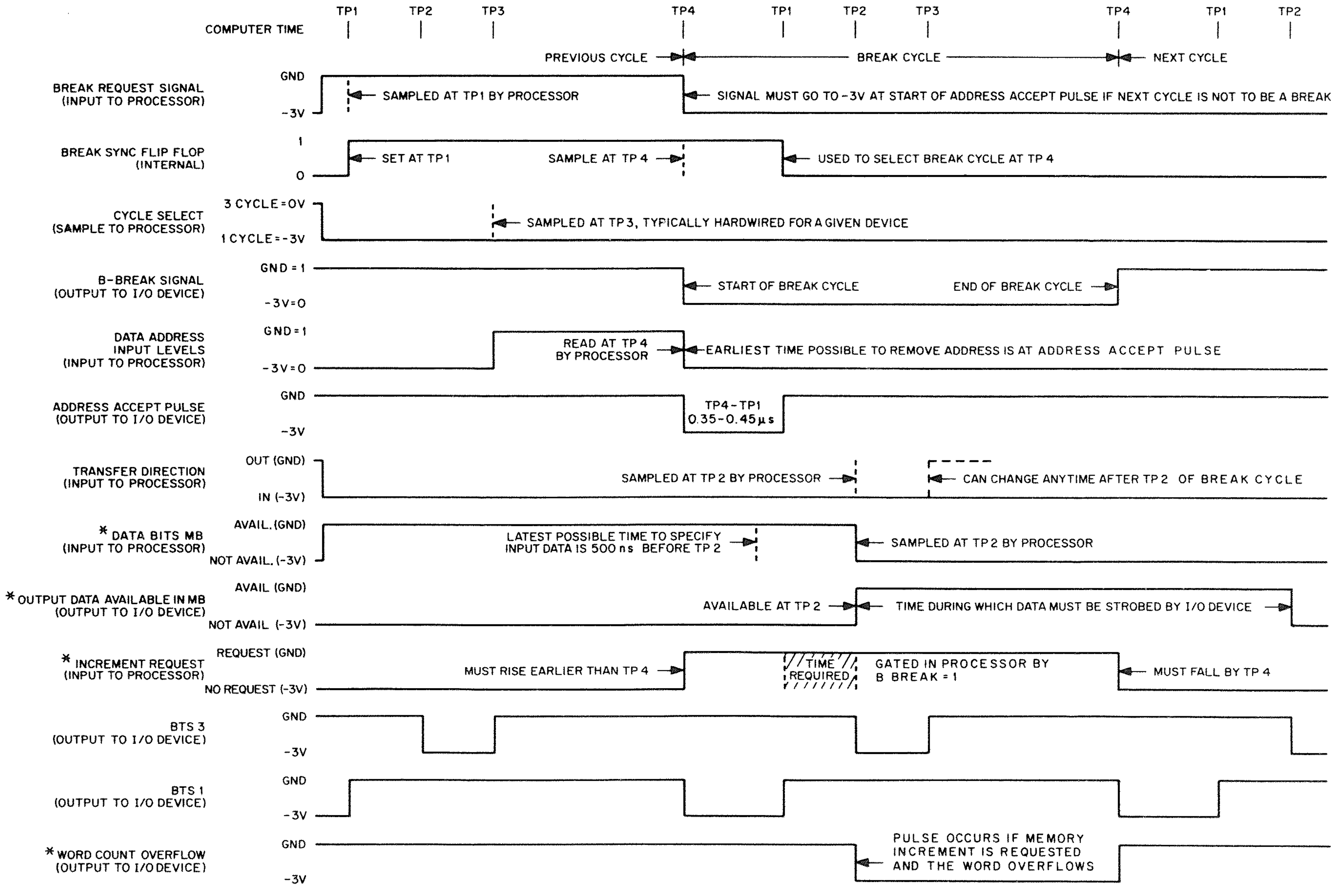

COMPUTER TIME
TP2
TP3
TP4
TP5
TP1
TP2
TP3
TP4
TP5
TP1
TP2
TP3
PREVIOUS CYCLE
BREAK CYCLE
NEXT CYCLE
BREAK REQUEST SIGNAL (INPUT TO PROCESSOR)
+3
GND
SAMPLED AT OFF-PAUSE BY PROCESSOR **
SIGNAL MUST GO TO +3V AT START OF ADDRESS ACCEPT PULSE IF NEXT CYCLE IS NOT TO BE A BREAK
BREAK SYNC FLIP FLOP (INTERNAL)
1
0
SET AT SAMPLE TIME
SAMPLE AT TP1
USED TO SELECT BREAK CYCLE AT TP1
CYCLE SELECT (SAMPLE TO PROCESSOR)
3 CYCLE
1 CYCLE
SAMPLED AT TP1
TYPICALLY HARDWIRED FOR A GIVEN DEVICE
B-BREAK SIGNAL (OUTPUT TO I/O DEVICE)
+3V
GND
START OF BREAK CYCLE
END OF BREAK CYCLE
DATA ADDRESS INPUT LEVELS (INPUT TO PROCESSOR)
+3V
GND
EXT. MEM. ADDRESS READ
READ AT TP1 BY PROCESSOR
EARLIEST TIME POSSIBLE TO REMOVE ADDRESS IS AT START OF BREAK CYCLE
ADDRESS ACCEPT PULSE (OUTPUT TO I/O DEVICE)
+3V
GND
TP1 - TP3
TRANSFER DIRECTION (INPUT TO PROCESSOR)
IN (+3V)
OUT (GND)
LATEST POSSIBLE TIME IS TP2
CAN CHANGE ANYTIME AFTER TP 3
*DATA SIGNAL INPUT TO MB (INPUT TO PROCESSOR)
AVAIL (+3V)
NOT AVAIL (GND)
LATEST POSSIBLE TIME TO SPECIFY INPUT DATA IS TP2
OUTPUT DATA AVAILABLE IN MB (OUTPUT TO I/O DEVICE)
AVAIL (+3V)
NOT AVAIL (+3V)
AVAILABLE AT TP 3
TIME DURING WHICH DATA MUST BE STROBED BY I/O DEVICE
MB INCREMENT REQUEST (INPUT TO PROCESSOR)
NO REQUEST (+3V)
REQUEST (GND)
MUST OCCUR EARIER THAN TP2
TIME REQUIRED
MUST OCCUR ONLY WHEN B BREAK = 1
TP1
BTS 5 (OUTPUT TO I/O DEVICE)
+3V
GRN
BTS 2 (OUTPUT TO I/O DEVICE)
+3V
GND
* WORD COUNT OVERFLOW (OUTPUT TO I/O DEVICE)
+3V
GND
OCCURS IF MEMORY INCREMENT IS REQUESTED AND THE WORD COUNT OVERFLOWS
* SIGNAL NOT USED FOR OUTPUT TRANSFERS SHOWN FOR REFERENCE ONLY
** OFF-PAUSE OCCURS 200ns BEFORE TP5. IF IOP INSTRUCTION IN PROCESS, OFF IOP SAMPLES BREAK REQUEST.
12-0129

Figure 5-1 PDP-12 Single Cycle Data Break Timing

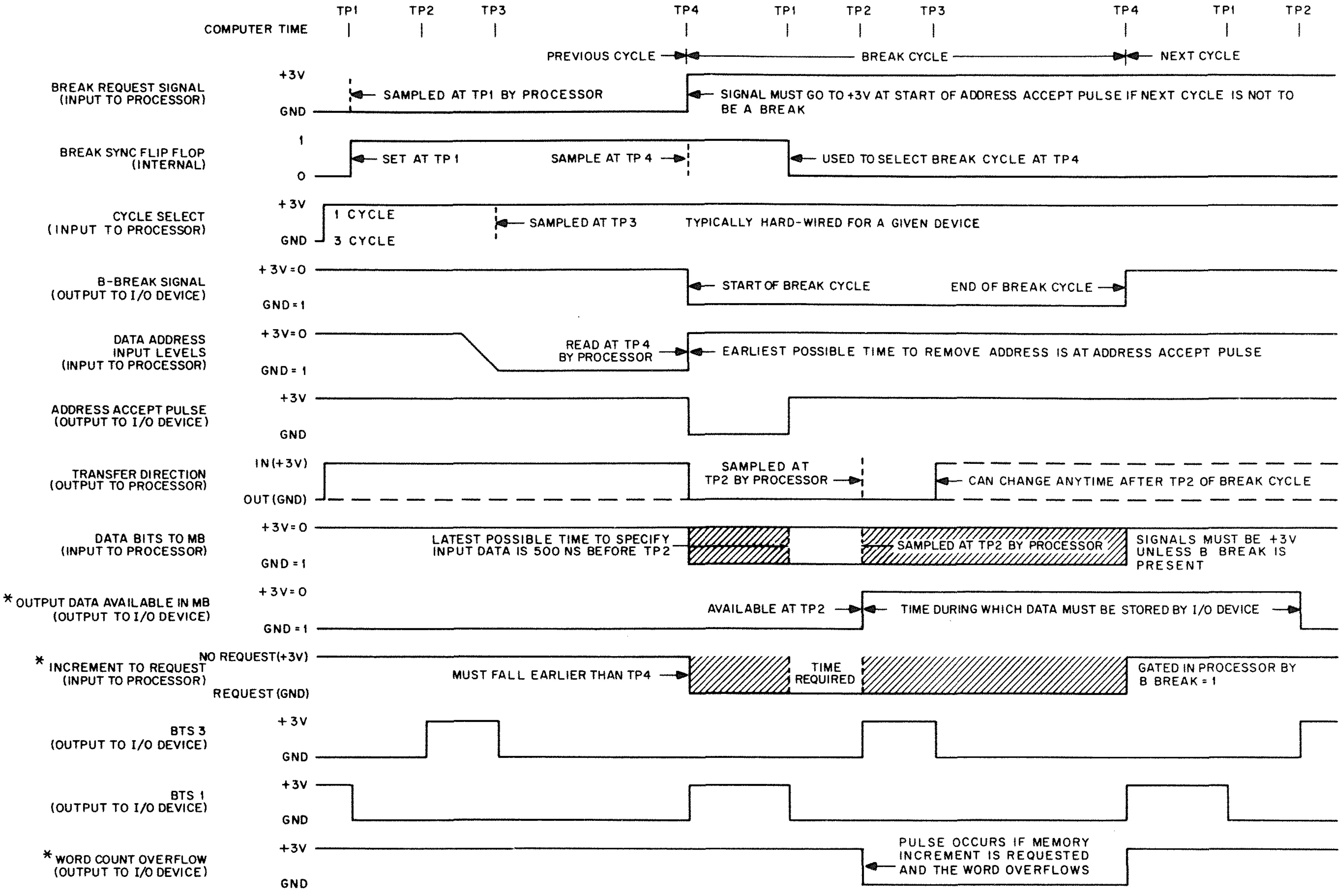

COMPUTER TIME
TP1
TP2
TP3
TP4
TP1
TP2
TP3
TP4
TP1
TP2
PREVIOUS CYCLE
BREAK CYCLE
NEXT CYCLE
BREAK REQUEST SIGNAL (INPUT TO PROCESSOR)
GND
-3V
SAMPLED AT TP1 BY PROCESSOR
SIGNAL MUST GO TO -3V AT START OF ADDRESS ACCEPT PULSE IF NEXT CYCLE IS NOT TO BE A BREAK
BREAK SYNC FLIP FLOP (INTERNAL)
1
0
SET AT TP1
SAMPLE AT TP 4
USED TO SELECT BREAK CYCLE AT TP 4
CYCLE SELECT (SAMPLE TO PROCESSOR)
3 CYCLE=0V
1 CYCLE=-3V
SAMPLED AT TP3, TYPICALLY HARDWIRED FOR A GIVEN DEVICE
B-BREAK SIGNAL (OUTPUT TO I/O DEVICE)
GND=1
-3V=0
START OF BREAK CYCLE
END OF BREAK CYCLE
DATA ADDRESS INPUT LEVELS (INPUT TO PROCESSOR)
GND=1
-3V=0
READ AT TP 4 BY PROCESSOR
EARLIEST TIME POSSIBLE TO REMOVE ADDRESS IS AT ADDRESS ACCEPT PULSE
ADDRESS ACCEPT PULSE (OUTPUT TO I/O DEVICE)
GND
-3V
TP4-TP1 0.35-0.45µs
TRANSFER DIRECTION (INPUT TO PROCESSOR)
OUT (GND)
IN (-3V)
SAMPLED AT TP 2 BY PROCESSOR
CAN CHANGE ANYTIME AFTER TP 2 OF BREAK CYCLE
* DATA BITS MB (INPUT TO PROCESSOR)
AVAIL. (GND)
NOT AVAIL. (-3V)
LATEST POSSIBLE TIME TO SPECIFY INPUT DATA IS 500 ns BEFORE TP 2
SAMPLED AT TP 2 BY PROCESSOR
* OUTPUT DATA AVAILABLE IN MB (OUTPUT TO I/O DEVICE)
AVAIL (GND)
NOT AVAIL (-3V)
AVAILABLE AT TP 2
TIME DURING WHICH DATA MUST BE STROBED BY I/O DEVICE
* INCREMENT REQUEST (INPUT TO PROCESSOR)
REQUEST (GND)
NO REQUEST (-3V)
MUST RISE EARLIER THAN TP 4
TIME REQUIRED
GATED IN PROCESSOR BY B BREAK = 1
MUST FALL BY TP 4
BTS 3 (OUTPUT TO I/O DEVICE)
GND
-3V
BTS 1 (OUTPUT TO I/O DEVICE)
GND
-3V
* WORD COUNT OVERFLOW (OUTPUT TO I/O DEVICE)
GND
-3V
PULSE OCCURS IF MEMORY INCREMENT IS REQUESTED AND THE WORD OVERFLOWS

*SIGNAL NOT USED
SHOWN FOR REFERENCE ONLY

8/I-0139

Negative I/O Bus & Logic

Figure 5-2 PDP-8 Single Cycle Data Break


COMPUTER TIME
TP1
TP2
TP3
TP4
TP1
TP2
TP3
TP4
TP1
TP2
PREVIOUS CYCLE
BREAK CYCLE
NEXT CYCLE
BREAK REQUEST SIGNAL (INPUT TO PROCESSOR)
+3V
GND
SAMPLED AT TP1 BY PROCESSOR
SIGNAL MUST GO TO +3V AT START OF ADDRESS ACCEPT PULSE IF NEXT CYCLE IS NOT TO BE A BREAK
BREAK SYNC FLIP FLOP (INTERNAL)
1
0
SET AT TP 1
SAMPLE AT TP 4
USED TO SELECT BREAK CYCLE AT TP4
CYCLE SELECT (INPUT TO PROCESSOR)
+3V
GND
1 CYCLE
3 CYCLE
SAMPLED AT TP3
TYPICALLY HARD-WIRED FOR A GIVEN DEVICE
B-BREAK SIGNAL (OUTPUT TO I/O DEVICE)
+3V=0
GND=1
START OF BREAK CYCLE
END OF BREAK CYCLE
DATA ADDRESS INPUT LEVELS (INPUT TO PROCESSOR)
+3V=0
GND=1
READ AT TP4 BY PROCESSOR
EARLIEST POSSIBLE TIME TO REMOVE ADDRESS IS AT ADDRESS ACCEPT PULSE
ADDRESS ACCEPT PULSE (OUTPUT TO I/O DEVICE)
+3V
GND
TRANSFER DIRECTION (OUTPUT TO PROCESSOR)
IN(+3V)
OUT(GND)
SAMPLED AT TP2 BY PROCESSOR
CAN CHANGE ANYTIME AFTER TP2 OF BREAK CYCLE
DATA BITS TO MB (INPUT TO PROCESSOR)
+3V=0
GND=1
LATEST POSSIBLE TIME TO SPECIFY INPUT DATA IS 500 NS BEFORE TP2
SAMPLED AT TP2 BY PROCESSOR
SIGNALS MUST BE +3V UNLESS B BREAK IS PRESENT
*OUTPUT DATA AVAILABLE IN MB (OUTPUT TO I/O DEVICE)
+3V=0
GND=1
AVAILABLE AT TP2
TIME DURING WHICH DATA MUST BE STORED BY I/O DEVICE
*INCREMENT TO REQUEST (INPUT TO PROCESSOR)
NO REQUEST(+3V)
REQUEST(GND)
MUST FALL EARLIER THAN TP4
TIME REQUIRED
GATED IN PROCESSOR BY B BREAK = 1
BTS 3 (OUTPUT TO I/O DEVICE)
+3V
GND
BTS 1 (OUTPUT TO I/O DEVICE)
+3V
GND
*WORD COUNT OVERFLOW (OUTPUT TO I/O DEVICE)
+3V
GND
PULSE OCCURS IF MEMORY INCREMENT IS REQUESTED AND THE WORD OVERFLOWS

*SIGNAL NOT USED FOR INPUT TRANSFERS: SHOWN FOR REFERENCE ONLY

Positive I/O Bus & Logic

8I-0137

Figure 5-2 PDP-8 Single Cycle Data Break Timing

state enable system may conditionally qualify any number of Register Gates. A conditionally qualified register gate causes a register transfer on the next clock pulse. It is appropriate to observe that the FPP12 logic is fully clocked, i.e., all flip-flops change state on the occurrence of a pulse from the system clock generated by a free-running RC oscillator adjusted to a frequency of 5 mHz (200 ns).

The FPP12 data flow system is shown in Figure 5-5. In some respects FPP12 architecture is similar to the PDP-8 in that major registers are multiplexed through a central arithmetic logic unit. However, FPP12 logic design is based on the use of medium-scale integrated circuit technology (MSI). The Operand Address Register, Program Counter (FPC), and APT pointer are formed from 4-bit binary up/down counters. This permits the incrementing of address registers and the performance of arithmetic operations on data variables simultaneously. The arithmetic logic unit consists of seven or sixteen (EPM) 24-pin MSI devices that can each perform all 16 Boolean and 16 different arithmetic functions on two variables. Full carry-look-ahead permits the addition or subtraction of two variables in under 100 ns.

FPP12 USER IOT DECODER SYSTEM

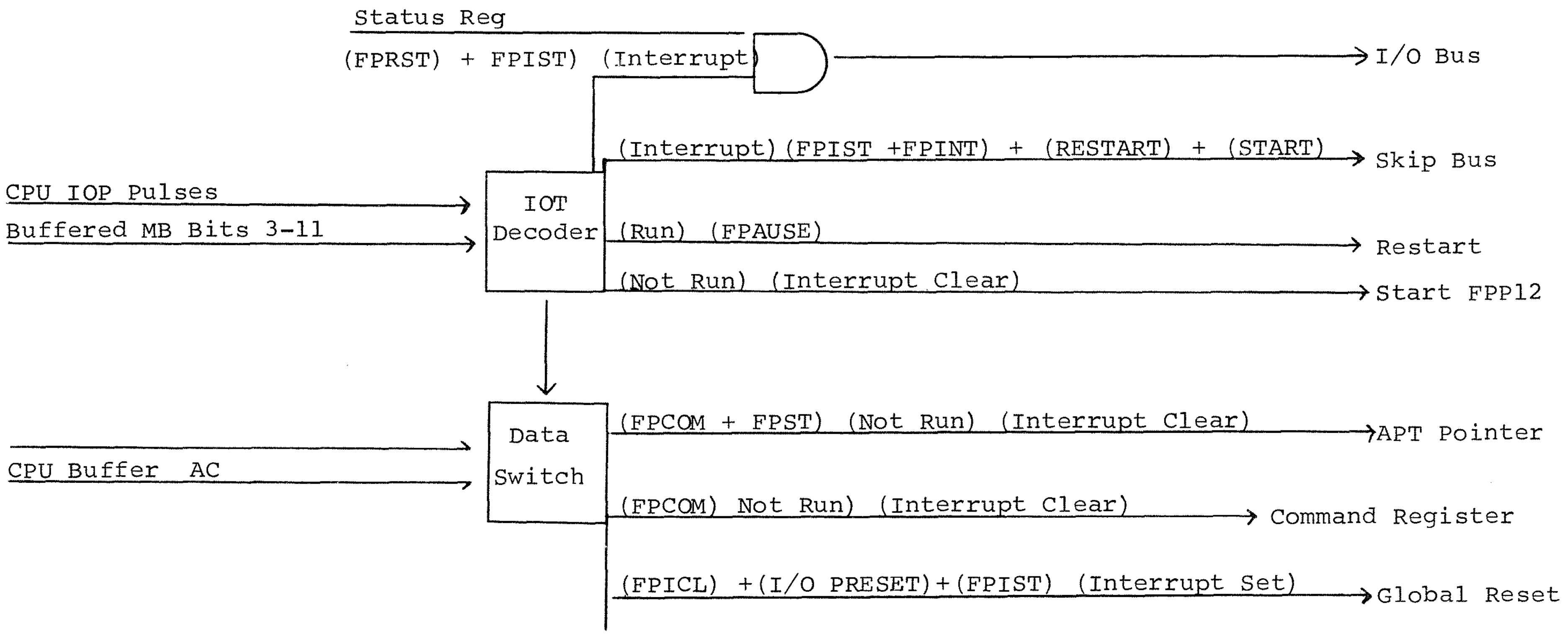

Status Reg
(FPRST) + FPIST) (Interrupt)
I/O Bus
(Interrupt) (FPIST +FPINT) + (RESTART) + (START)
Skip Bus
CPU IOP Pulses
Buffered MB Bits 3-11
IOT Decoder
(Run) (FPAUSE)
Restart
(Not Run) (Interrupt Clear)
Start FPP12
Data Switch
CPU Buffer AC
(FPCOM + FPST) (Not Run) (Interrupt Clear)
APT Pointer
(FPCOM) Not Run) (Interrupt Clear)
Command Register
(FPICL) +(I/O PRESET)+(FPIST) (Interrupt Set)
Global Reset

Figure 5-3 FPP12 User IOT Decoder System

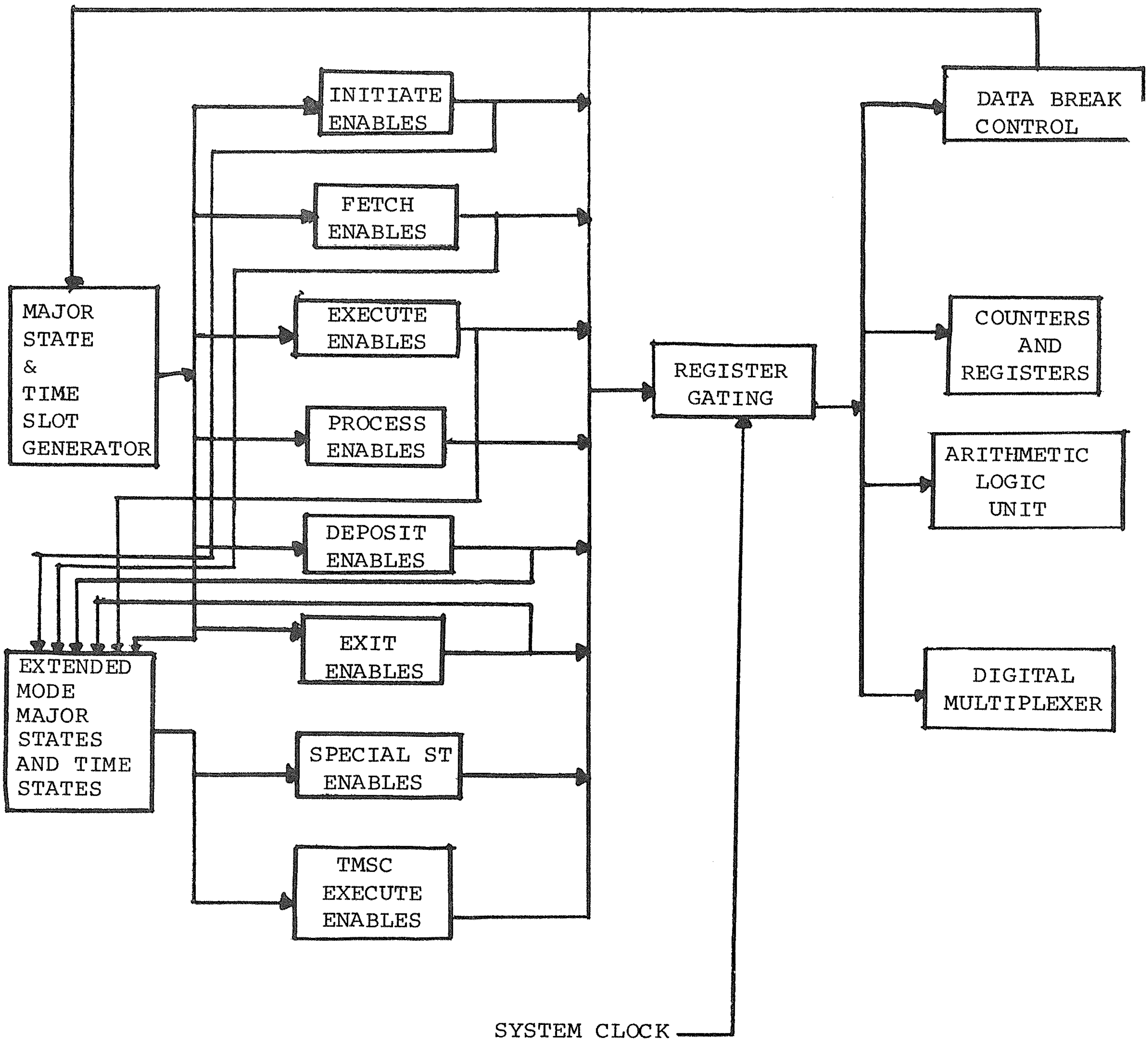


Figure 5-4 Timing and Enable System in FPP12

FPP12 DATA FLOW SYSTEM

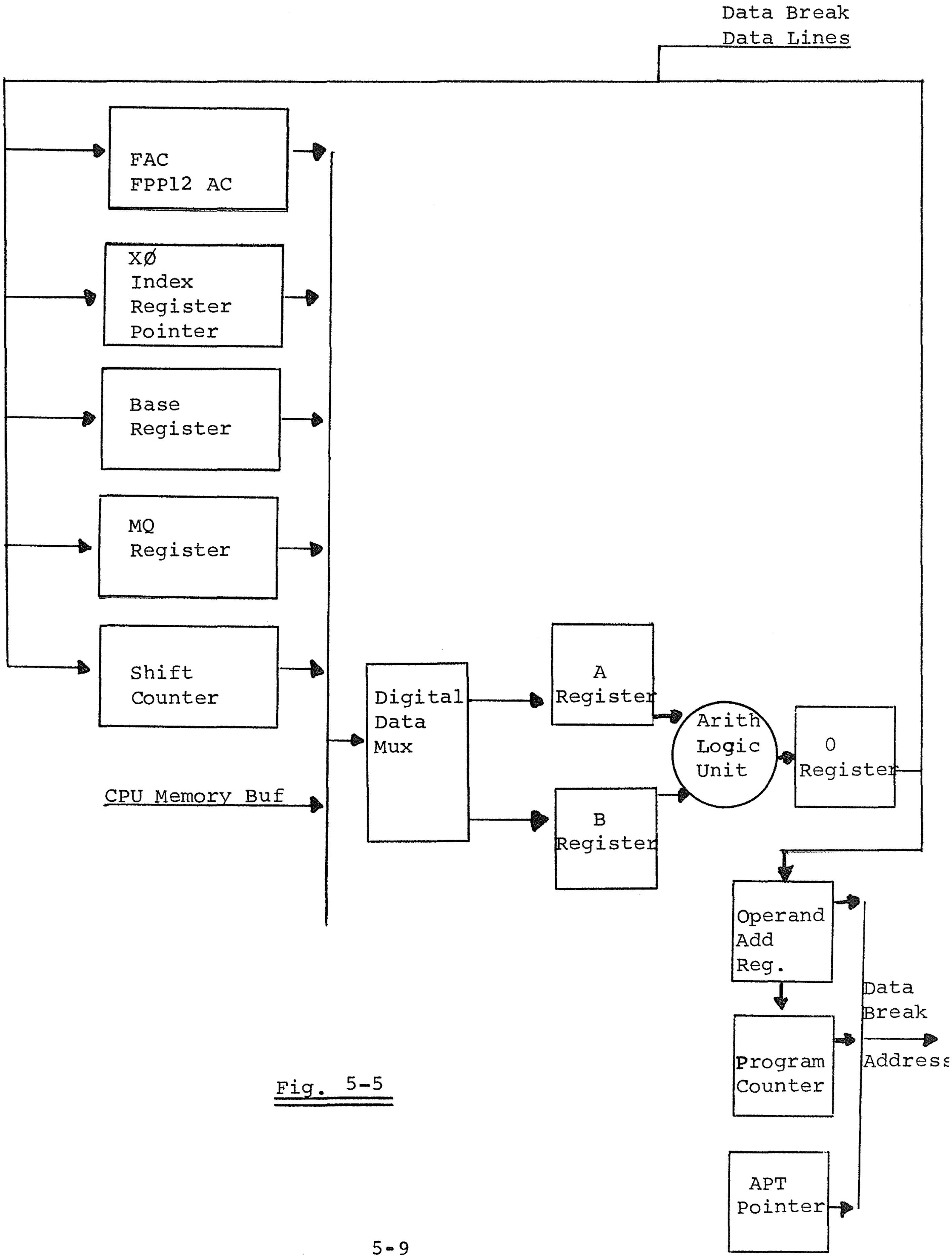


Fig. 5-5

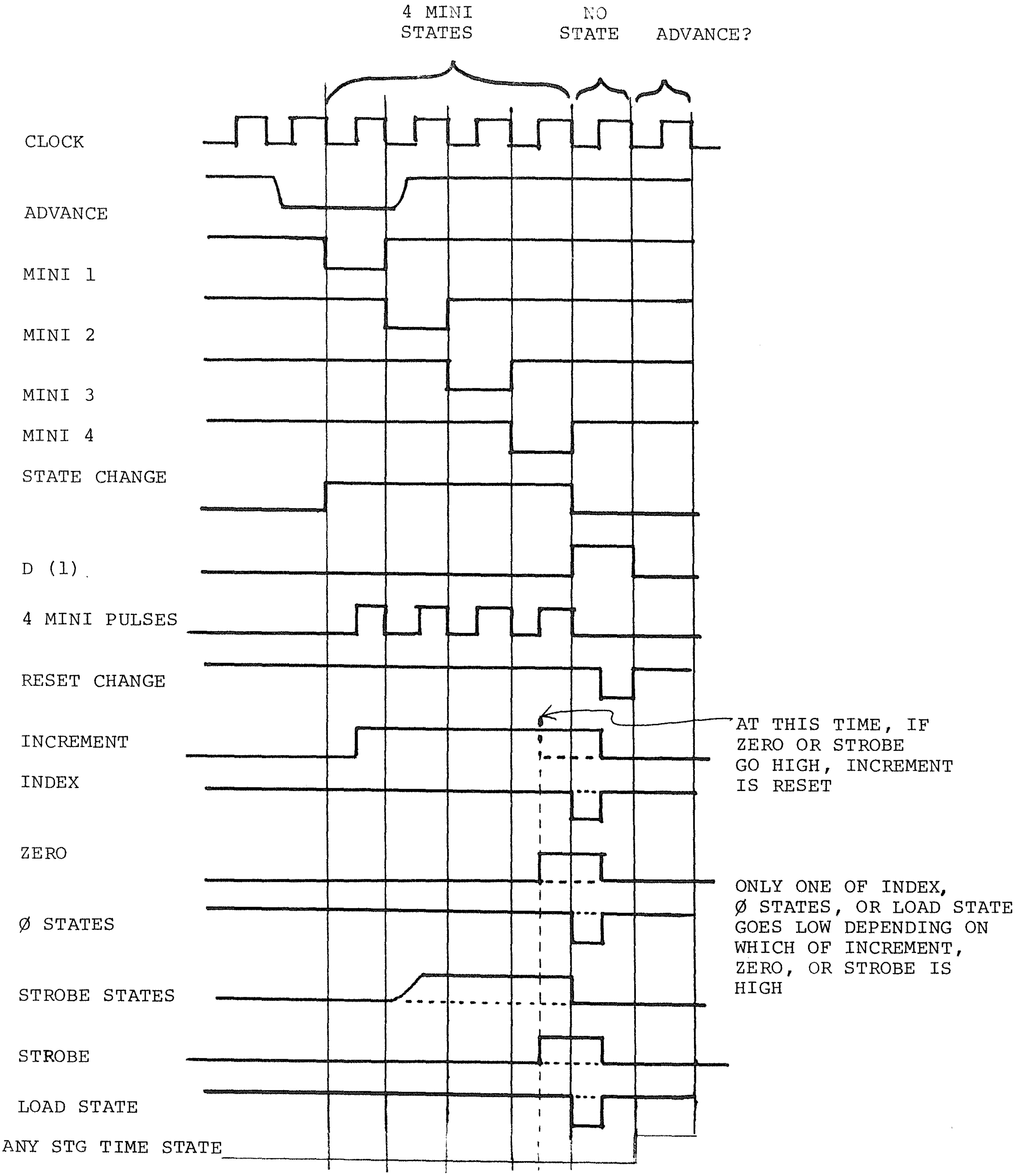


Fig. 5.6

Timing diagram indicating relationship between mini states, major time states, and the system clock.

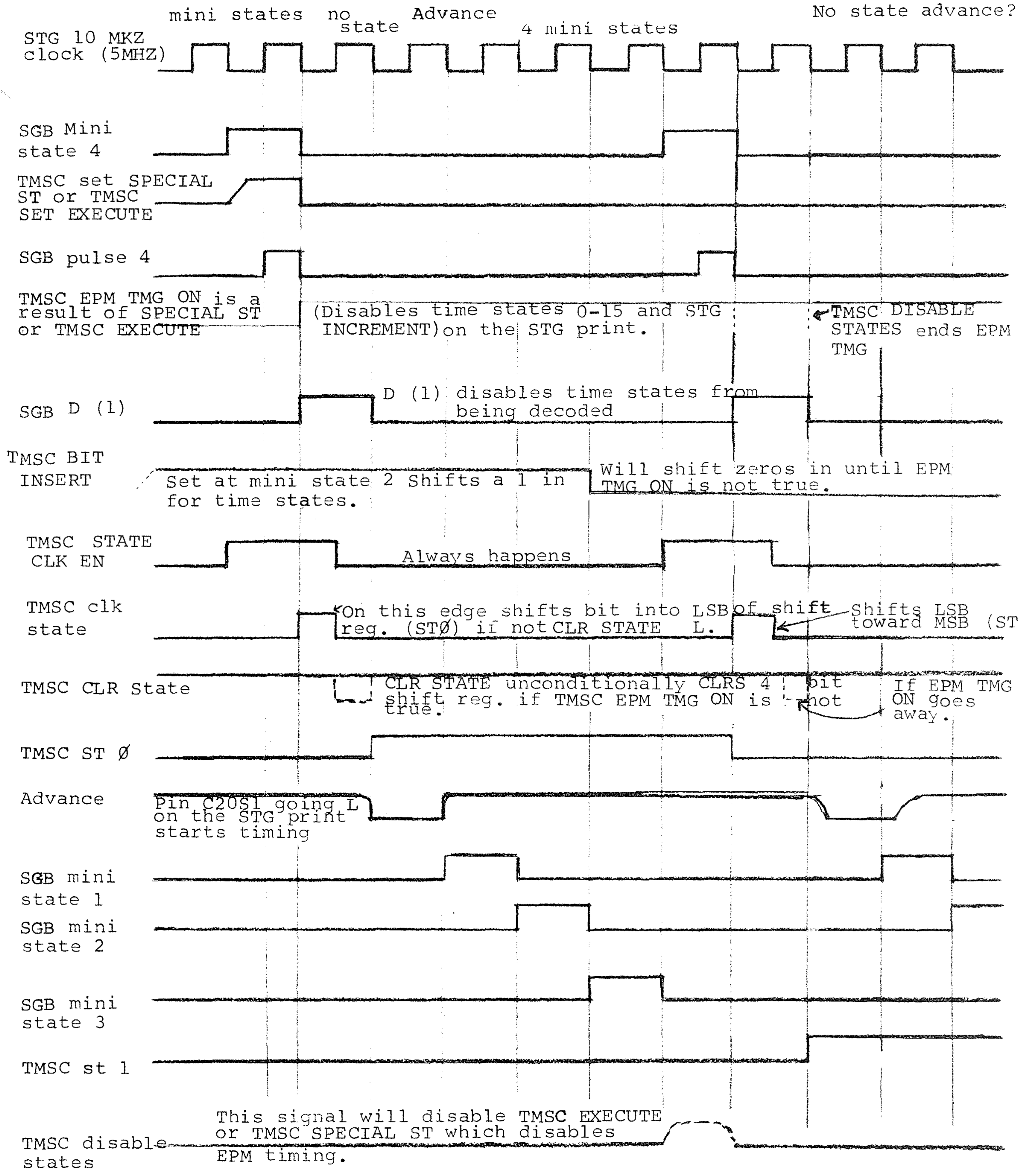

mini states
no state
Advance
4 mini states
No state advance?
STG 10 MKZ clock (5MHZ)
SGB Mini state 4
TMSC set SPECIAL ST or TMSC SET EXECUTE
SGB pulse 4
TMSC EPM TMG ON is a result of SPECIAL ST or TMSC EXECUTE
(Disables time states 0-15 and STG INCREMENT) on the STG print.
TMSC DISABLE STATES ends EPM TMG
SGB D (1)
D (1) disables time states from being decoded
TMSC BIT INSERT
Set at mini state 2 Shifts a 1 in for time states.
Will shift zeros in until EPM TMG ON is not true.
TMSC STATE CLK EN
Always happens
TMSC clk state
On this edge shifts bit into LSB of shift reg. (STØ) if not CLR STATE L.
Shifts LSB toward MSB (ST
TMSC CLR State
CLR STATE unconditionally CLRS 4 bit shift reg. if TMSC EPM TMG ON is not true.
If EPM TMG ON goes away.
TMSC ST Ø
Advance
Pin C20S1 going L on the STG print starts timing
SGB mini state 1
SGB mini state 2
SGB mini state 3
TMSC st 1
TMSC disable states
This signal will disable TMSC EXECUTE or TMSC SPECIAL ST which disables EPM timing.

FIGURE 5-7 EPM TIMING DIAGRAM

5.3 MAJOR STATES

The FPP12 logic is organized into eight major states:

INITIATE

FETCH

PROCESS

EXECUTE

DEPOSIT

EXIT

TMSC EXECUTE }
SPECIAL ST } EPM

With one exception, if any major state flip-flop on print CNR is set, the FPP12 will be actively calculating. If all major state flip-flops are reset, the FPP12 will be inactive. The single exception has to do with the instruction FPAUSE, which causes the FPP12 to wait for a synchronizing signal before proceeding.

The FPP12 operations that occur in each major state are detailed below:

INITIATE: INITIATE major state begins at the trailing edge of IOP 4 when IOT instruction 6555 is issued by the CPU, if the FPP12 is not running and the FPP12 Interrupt Request flag is reset. During INITIATE, the contents of the APT are retrieved.

NOTE: Only the first two locations of the APT must be used as these contain the 15-bit initial setting FPP12 program counter (FPC). The fifth location of the APT, the operand address, is not retrieved during INITIATE.

Following the completion of INITIATE,the FPP12 always proceeds to FETCH time state 0.

Schematic drawings for INITIATE are found on prints ARS2 and SSG.

FETCH During FETCH, preliminary decoding of instructions occurs. Instructions that require only one major time state to be completed, such as FCLA, are completely finished during FETCH State 0. Special instructions that require more than one major time state, such as ALN, or more than one memory cycle, such as ATX, cause the FPP12 logic to go from FTECH State 0 to PROCESS State 1. All data reference instructions require FETCH to continue beyond FETCH State 0 in order to calculate the operand address. At the end of the FETCH cycle for all data reference instructions, a transfer is made to EXECUTE State 0, with the Operand Address Register appropriately loaded. FETCH begins in State 0 and ends when the address calculation is complete. FETCH schematics are found on prints FTH1 through FTH3.

PROCESS Most special instructions that require more than one major time state or more than one memory cycle are completed in PROCESS. With the exception of NORM and XTA, the FPP12 returns to FETCH State 0 after the completion of PROCESS. Processing for XTA and NORM is completed in DEPOSIT. PROCESS begins in major time state 1. PROCESS schematics are found on prints SPI1 through SPI3.

EXECUTE The execution of all data reference instructions begins during EXECUTE. FLDA and FSTA are completed during EXECUTE. For all other data reference instructions the FPP12 proceeds to DEPOSIT at the completion of EXECUTE if no EXECUTE error is encountered. EXECUTE errors are defined as:

a. An attempt to divide by zero.

b. A fraction overflow in fixed-point mode.

EXECUTE

At the completion of EXECUTE for instructions other than FLDA and FSTA, the un-normalized result of any calculation is stored in the O register and the exponent is located in the MQLSW. The exponent contained in the MQLSW is the operand exponent for FMUL, FMULM, and FDIV and the resultant exponent before normalization for FADD, FADDM, and FSUB. The shift counter is 0 if no fraction overflow occurred, and 1 if a floating-point fraction overflow occurred. EXECUTE schematics are found on prints ASTO through AST3.

DEPOSIT

DEPOSIT begins in major time state 11 and ends in major time state 15. As DEPOSIT is the only function performed during these time states, DEPOSIT enables shown on prints DEP1 through DEP3 are not gated with the DEPOSIT flip-flop found on the CNR print. During DEPOSIT the following functions are performed in the order listed:

a. The results of all floating-point arithmetic calculations are normalized. The number of shifts is stored in the shift counter.

b. The normalized result is rounded to 24 bits. No rounding in EPM.

c. For FMUL and FMULM the FAC exponent is added to the MQLSW.

d. For the FDIV instruction the MQLSW is subtracted from the FAC exponent.

e. The contents of the shift counter are added to the un-normalized exponent.

f. If the exponent resulting after normalization is within bounds, -2048 to +2047, the resultant answer is stored in the FAC for all operations except FADDM and FMULM. For FADDM and FMULM, the resultant answer is stored in the addressed location. After storing the resultant answer, the FPP12 returns to FETCH State 0, unless the IOT FPHLT was issued by the CPU during the current instruction.

g. If the exponent is not within bounds after normalization the appropriate status bit is set and the FPP12 enters EXIT State 0. DEPOSIT schematics are found on prints DEP1 through DEP3.

EXIT During EXIT the current APT is deposited into core over the initial APT. Only the first two locations of the APT must be deposited; the other locations are optional according to the command register setting. The items in the APT are always located in the same position relative to one another. If the programmer chooses not to deposit the operand address the fifth location of the APT is simply skipped. The field bits of the base register, X0, and FPC are the first retrieved on INITIATE and last deposited during EXIT.

EXIT is entered for any of the following conditions:

a. A FEXIT instruction is encountered.

b. A fraction overflow occurs in fixed-point mode.

c. An exponent overflow or underflow occurs in floating-point mode. If EXIT is entered for an exponent underflow command register bit 1 is tested. If it is set to 1, the EXIT is continued. If it is set to 0, the result of the previous calculation is set to 0 and the FPP12 returns to FETCH State 0. If an exponent underflow occurs, status bit 6 is set as an indicator, even if command register bit 1 is set to 0.

d. An attempt to divide by 0 is made.

e. A FPHLT IOT is issued by the CPU.

At the end of EXIT the FPP12 halts in major time state 0 with all major state flip-flops reset. The FPP12 skip flag is set and the CPU program interrupt is actuated if command register bit 3 is set to 1.

TMSC EXECUTE All data reference instructions set TMSC EXECUTE during CNR EXECUTE in the extended precision mode. This will occur at major time state Ø, 1 or 2 of CNR EXECUTE depending on the instruction. The sole purpose of TMSC EXECUTE is to activate the gating necessary to pick up or store in core the least significant three words of the operand or FAC, bits 24-59. At the completion of TMSC EXECUTE, which is always at the end of TMSC ST 2, control returns to the next major time state in CNR EXECUTE with the exception of the STR instruction which returns to FETCH state zero.

SPECIAL ST When in the extended precision mode,SPECIAL ST can be set during INITIATE, FETCH, DEPOSIT and EXIT major states. Below is a description of the uses of SPECIAL ST for the major states indicated above:

a. INITIATE - SPECIAL ST is set at the end of INITIATE state 5 to activate the necessary gating to pick up, from the APT, FAC bits 24-59. Control is returned to INITIATE state 6 at its completion.

b. FETCH - SPECIAL ST is set at the end of FETCH state 6. At this point the contents of the specified index register have been multiplied by 3 or by six in EPM. At the completion of SPECIAL ST TMSC ST Ø control is returned to FETCH state 7.

c. DEPOSIT - SPECIAL ST is set at the end of DEPOSIT state 13 to allow bits 24-59 of the result to be stored in core when doing an FADDM or FMULUM instruction. Control returns from SPECIAL ST TMSC ST 2 to DEPOSIT state 14.

d. EXIT - SPECIAL ST can be set during EXIT if in EPM for two different reasons as follows:

   1. EXIT and SPECIAL ST are set simultaneously when the exit occurs for reasons other than exponent underflow not trapped. SPECIAL ST allows the FAC, bits 24-59, to be stored in the APT provided command registers bit 7 is not set. After the four TMSC time states control is returned to EXIT state zero.

   2. SPECIAL ST is set at the end of EXIT state 1 when exponent underflow is the cause of the exit and the underflow is not trapped and it was an FADDM or FMULM instruction that resulted in the exponent underflow. SPECIAL ST will allow for the result, bits 24-59, to be zeroed in memory.

Note that both TMSC EXECUTE and SPECIAL ST are functions of the extended precision mode.

5.4 DESCRIPTION OF REGISTERS

Most major registers of the FPP12 are described in Paragraph 2.7. The MQ, shift counter, A register, B register, and O register are hidden from the programmer. Their characteristics are as follows:

MQ

The MQ is a 28-bit or 6Ø-bit* parallel-serial input, parallel output, left-right shift register. During multiplication, the MQ contains the absolute value of the multiplier mantissa. During division, the MQ temporarily contains the un-normalized absolute value of the quotient. When entering deposit, the MQLSW contains the un-normalized resultant exponent for FADD, FADDM, and FSUB and the operand exponent for FMUL, FMULM, and FDIV. The MQ is found on print MQR1.

SHIFT COUNTER

The SHIFT COUNTER is a 7-bit binary up/down counter. As its name implies, it is used to count shifts. The SHIFT COUNTER is found on print CAR6.

A REGISTER
AND
B REGISTER

These 28-bit or 6Ø bit* registers are inputs to the arithmetic logic unit (ALU) of the M190. The A register is a 28-bit or 6Ø bit* storage register; the B register is a 28 bit or 6Ø bit* parallel-serial input parallel output shift register that shifts towards the least significant bit.

O REGISTER

The O register is the 28-bit or 6Ø-bit* output buffer for the M190. It is a parallel-serial input, parallel output shift register that shifts towards the most significant bit.

The A, B, and O Registers are found on the AMSW, ALSW, EXT, ALS1*, ALS2* and ALS3* prints.

The following registers (listed below with the prints on which they are found) were described in Paragraph 2.7.

FAC	CAR 2, CAR3, EAC 1* and EAC 2*
XO REGISTER	CAR 4
BASE REGISTER	CAR 5
FPC	CAR 1
APT POINTER	CAR 1
OPERAND ADDRESS REGISTER	CAR 1
STATUS REGISTER	CAR 8
COMMAND REGISTER	CAR 8
FIR	CAR 7

*EPM

The operand address register, the FPC, and the APT pointer provide the addresses for data breaks. These registers are attached to a digital multiplexer that drives the EXT address lines of the CPU. The FAC, XO register, base register, O register, MQ, and shift counter feed the digital multiplexer that loads the A and B Registers. The FIR is the floating instruction register which holds the instruction that was loaded at FETCH state Ø.

5.5 REGISTER GATING SYSTEM

The OR gates in the register gating system found on prints RG1 through RG10 funnel enables from many sources into signals that, when added with a clock pulse, cause a register action. This action can be a load, shift, count, or a clear. The signals that actuate the data multiplexer are found on prints RG7 through RG10, and MXEN. The multiplexer gates are enabled for at least the duration of a mini time state. The time from the beginning of the mini time state until the clock pulse, shown on Figure 5-6, is allowed for the data to settle on the register inputs.

5.6 DATA BREAK CONTROL

The FPP12 accesses core memory via the single-cycle data break facility. The data break control serves the following functions:

a. Channels data break and direction requests to the CPU from the state enables.

b. Gates the proper address register onto the EXT ADD bus of the CPU.

c. Gates the proper data register onto the EXT DATA bus of the CPU if an input break is required.

d. Synchronizes the FPP12 time state generator to the particular CPU memory timing to which the peripheral is attached.

The synchronizing logic for the data break control is shown on print DBC1. The signal DBC1 REQUEST BREAK L requests the data break from the processor. This signal is actuated by the conditon:

$$\text{(REQ BRK CYCLE (1) H)(ADD ACCEPT (Ø)H)} \dashrightarrow \text{PDP-12}^{S}$$

OR

(REQ BRK CYCLE (1)H) (CI1 BREAK (Ø) H)---> PDP-8^{s}

The first term is the output of a flip-flop that permits the FPP12 to remember that it is currently requesting a data break. The second term is the signal from the CPU that a break is not in progress. Once the break cycle begins, noted by the disqualification of Break (0) H, the FPP12 break request must be removed. This is the sequence for PDP-8^{s}. ADD ACCEPT is used on PDP-12^{s} to permit the CPU lock out mode to function.

In the lower right-hand side of the DBC1 prints there is a signal DBC1 ENAB DATA H. The equation for this signal is:

(BREAK (1) H) (DBC1 REQ BRK CYCLE (1) H)+C12 MAINT READ L

The signal DBC1 ENAB DATA H permits the placing of data on the I/O bus during the break cycle requested by the FPP12. The DONE flip-flop, which is clocked by the trailing edge of ADDRESS ACCEPT in the PDP-12 or the rising edge of BTS3 in the PDP-8, restarts the FPP12 timing chain.

A data break may be initiated by any of the function enables placing a low level on the input of three sets of OR gates found on DBC2. These OR gates funnel break requests to the DBC1 REQ BRK CYCLE flip-flop and choose which of three address registers to use for the break address.

A data break for the purpose of retrieving data from core memory is an OUTPUT BREAK. A data break for the purpose of storing data in core memory is an INPUT BREAK. If a core memory location is incremented an INCREMENT BREAK is performed. The FPP12 data source for INPUT BREAKS is selected on DBC3.

There are seven data sources used for INPUT BREAKS as shown on prints DBC3 and DBL. They are: the field bits of the APT, the operand address register, the least significant word of the B multiplexer, and the most significant word of the A multiplexer, and in EPM the least significant three words of the B multiplexer.

If the data source selected for an INPUT BREAK is either the A or B multiplexer, an additional eight possibilities exist. The actual data source is resolved on prints RG7 through RG10 and DBL.

5.7 MODULES INTRODUCED IN THE FPP12

There were three etch boards and five new modules introduced for the FPP12. The following list shows the module number and the function of these modules.

Module No.	Function
M155	One of 16 decoders using 74154 IC decoders.
M190	4-bit arithmetic logic module using 74181 arithmetic logic unit integrated circuit.
M191	Two carry look-ahead 74182 ICs for the 74181.
M238	Two separate 4-bit synchronous binary up/ down counters with separate up and down clocks. Uses two 74193 ICs.

M245 Two separate, 4-bit parallel-serial input, parallel serial output shift registers. Uses two 8271 ICs.

The M191, M238, and M245 use a common etch board, the 50089 12, which is a mount for 2-16 pin dip packages with pin 16 reserved for +5V and pin 8 reserved for ground. The M155 is constructed on a 24-pin DIP mount, the 5008908. The M190 is a unique module layout containing 8 ICs including the 24-pin arithmetic logic unit.

Full specifications for new MSIs may be found in either the DEC specification file or from the manufacturer's catalog. The 74182, 74181, 74193, and 74154 ICs are listed in catalog number CC301 by Texas Instruments Inc. The 8271 and the 8291 (74197 from TI) are listed in Signatic's MSI Specifications Handbook, DCL Vol. II.

All five of the modules introduced with the FPP12 are tested on Digital Equipment Corporation's computerized module tester.

CHAPTER 6 - OPERATIONAL GUIDE USING FLOW DIAGRAMS

6.1 Using Flow Diagrams

The flow diagrams are the key to troubleshooting the FPP12 logic. In order to understand the flow diagrams, it is necessary to understand the timing generator and the register structure. It is recommended that the reader thoroughly study Chapter 5 of this manual before attempting to understand this section. Also there are additional aids in Chapter 7, Paragraph 7.6.

6.1.1 Timing

A brief review of the timing generator is presented here. The timing generator has the following properties:

1. There are 2Ø possible major time states. These are named State 0 through State 15 on the STG print and TMSC STØ through ST 3 on the TMSC print if the EPM logic is implemented.
2. The timing generator can be forced to jump to any state; that is, if the timing generator is in State 3, the next state could be State 11 if the proper enables are generated. If EPM timing is activated, a signal called TMSC EPM TMG ON will disable the time state generator on the STG print and the effective time state is then the TMSC time state shown on the TMSC print.

3. During the time a state is enabled, four different enabling pulses are generated; they are called Mini State 1 through Mini State 4. These pulses occur sequentially and are one clock cycle long. The end of the major time state occurs at the end of Mini State 4 and the next major time state is enabled at the trailing edge of the next clock cycle. When a state is entered, the timing generator stops until it receives a timing advance. When the advance is enabled, the four mini states are generated. During an output break cycle, Mini State 1 is used to enable the clocking of the data from the MB into the appropriate register.

6.1.2 Adder Module

The characteristics of the M190 are important also in understanding the flow diagrams. Recall that there are three registers; the outputs of the A and the B registers are conected to the two inputs of the ALU, DEC 74181. The output of the ALU is connected to the input of the O register. The A, B, and O registers each have unique properties. The A register can be loaded with the true or the complemented value of the inputs. The B register is a shift register that can be shifted toward the least significant bit; the O register that can be shifted toward the most significant bit. It is important to realize the many functions that the DEC74181 can perform. The functions used in the FPP12 are listed in Table 6-1, along with the enables that are required to perform the function.

Table 6-1

Functions Performed by DEC74181

	S_3	S_2	S	S_0	Inhibit Carry	Carry IN
A minus B → O	L	H	H	L	L	H
A plus B → O	H	L	L	H	L	L
A plus A → O	H	H	L	L	L	L
A → O	H	H	H	H	L	L
A plus 1 → O	H	H	H	H	L	H
Logical 1 → O	L	L	H	H	H	-
B → O	H	L	H	L	H	-
Logical 0 → O	H	H	L	L	H	

Note that the function B minus A cannot be performed. In order to do this the 1's complement must be loaded in A, A plus B with a carry insert enabled, and the result loaded into O.

6.1.3 Mnemonic Variations

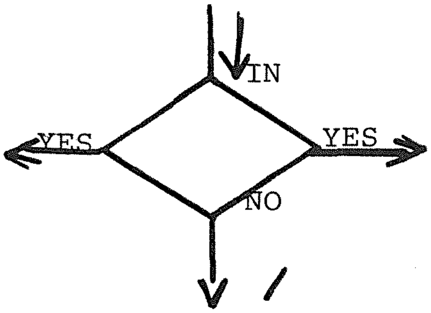
There are a few variations between the mnemonics given on the FPP12 instruction card, manual, etc. and those of the flows and prints. Consult table 6-2 for the proper equivalencies.

Table 6-2
Equivalence Between Instructions and Flow Routines

Instruction	Flow Diagrams
FEXIT	EXT
FPAUSE	PSK
FCLA	CLR
FNEG	NEG
FNORM	NRM
STARTF	STF
STARTD	STD
JAC	RTN
ALN	ALN
ATX	ATX
XTA	XTA
FNOP	NOP's
LDX	LDX
ADDX	ADX
JEQ	JMPS
JGE	JMPS
JLE	JMPS
JA	JMPS
JNE	JMPS
JLT	JMPS
JGT	JMPS
JAL	JMPS
SETX	MUX
SETB	MVP
JSA	JSB
JSR	JMK
JXN	JXN

6.1.4 Symbols and Terms

For one who is not familiar with the FPP12 Flow diagrms, there may be some terms, symbols, or operations that should be reviewed which have substantial importance in understanding the flows. A general list is supplied below with their meaning.

	A decision for "yes" or "no" depending on the test labeled inside the diamond.
/	Slash indicates a note describing current operation or branch or decision.
→	TO
~ —	Not or compliment
>	Greater than
≥	Greater than or equal to
<	Less than
≤	Less than or equal to
X(N) → Y(N-1)	The number in register X goes to register Y shifted left by one.
X(N) → Y(N+1)	The number in register X goes to register Y shifted right by one.
A	A register } feed the arithmetic
B	B register } element
O	O register contains the result of the arithmetic element.
SC	Shift counter register
XØ	Index register pointer
PØ	Base register pointer
FPC	Floating-point program counter
FAC	Floating-point accumulator
EPM	Extended-precision mode
FB	Field bits
FIR	Floating-point instruction register
MQ	Multiplier/quotient register
ADRS	APT address register
BRK	Break cycle
MSW	Most significant word
LSW	Least significant word
OP	Operand
ALU	Arithmetic logic unit
INC	Increment
DEC	Decrement
ADDR ADDRS	Address
SIGN EXTEND	Takes the state of the most significant bit (bit 12) of the least significant

word and forces that state to the most significant word.

Examples:

	MSW	LSW
A	ϕϕϕϕ	2ϕϕϕ
B	7777	4ϕϕϕ

RESULTS TO MEM Refers to the FADDM or FMULM instructions which are to put the answer (result) into memory.

6.2 General Instruction Flow

Before tackling the flow diagrams for the instructions, examine figure 6-1. This chart was provided for a better understanding of the relationships between major states and instructions. Using the chart, one may observe any instruction from fetch, through the sequential major states that follow until the instruction terminates.

6.3 Flow Diagrams - Major States

In this segment of Chapter 6, we will discuss the major states INITIATE, FETCH, DEPOSIT, and EXIT. The major states PROCESS and EXECUTE are not discussed since they are best described in conjunction with the instructions that use those states. It is possible to give a word by word description of each major state, mini state and clock pulse, however, it appears that this information is easily available on the flows. We therefore, will discuss the reasoning behind the operations illustrated on the flows. In addition, simplified flow diagrams are included in this manual for further support.

CHART 6-1

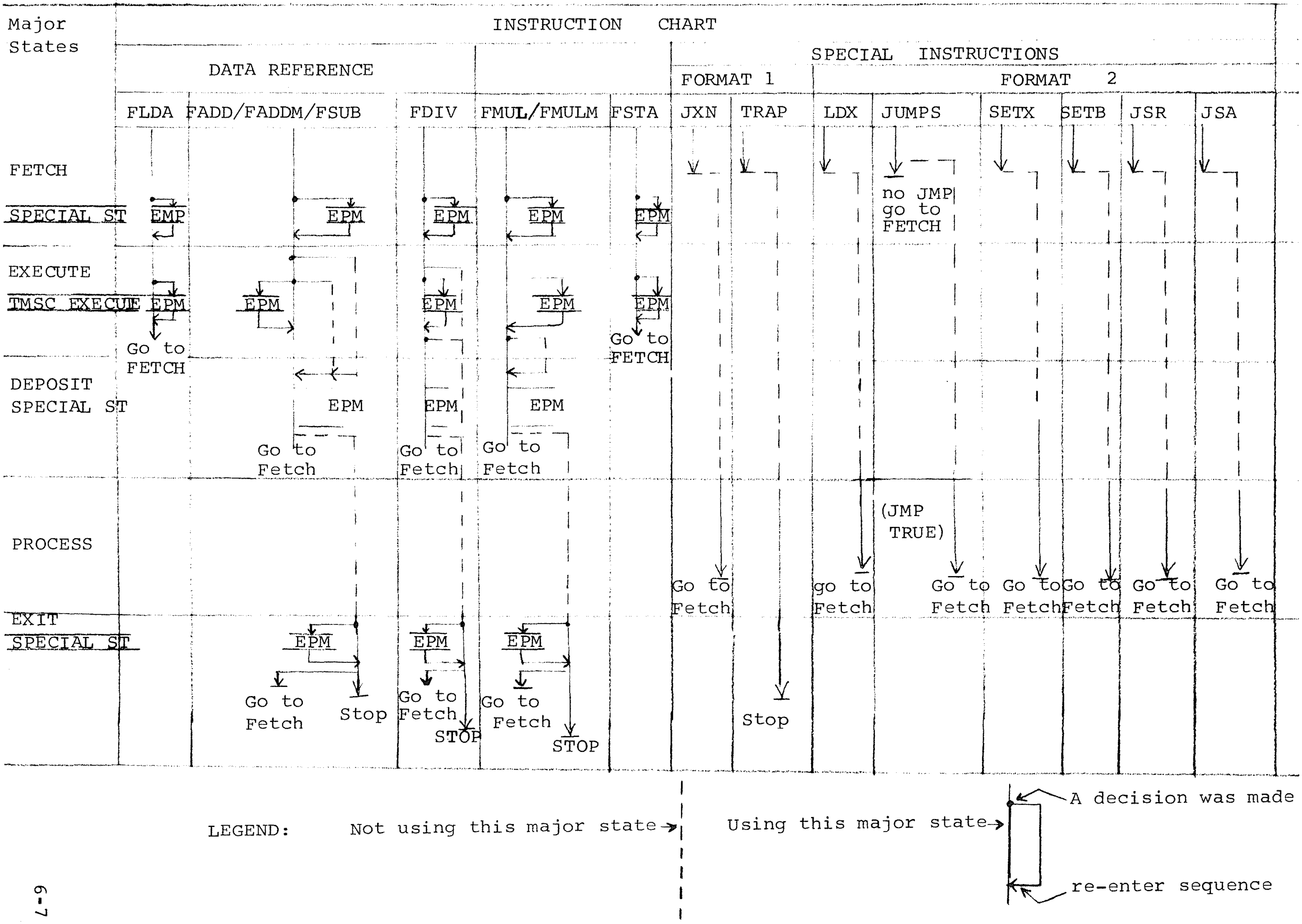
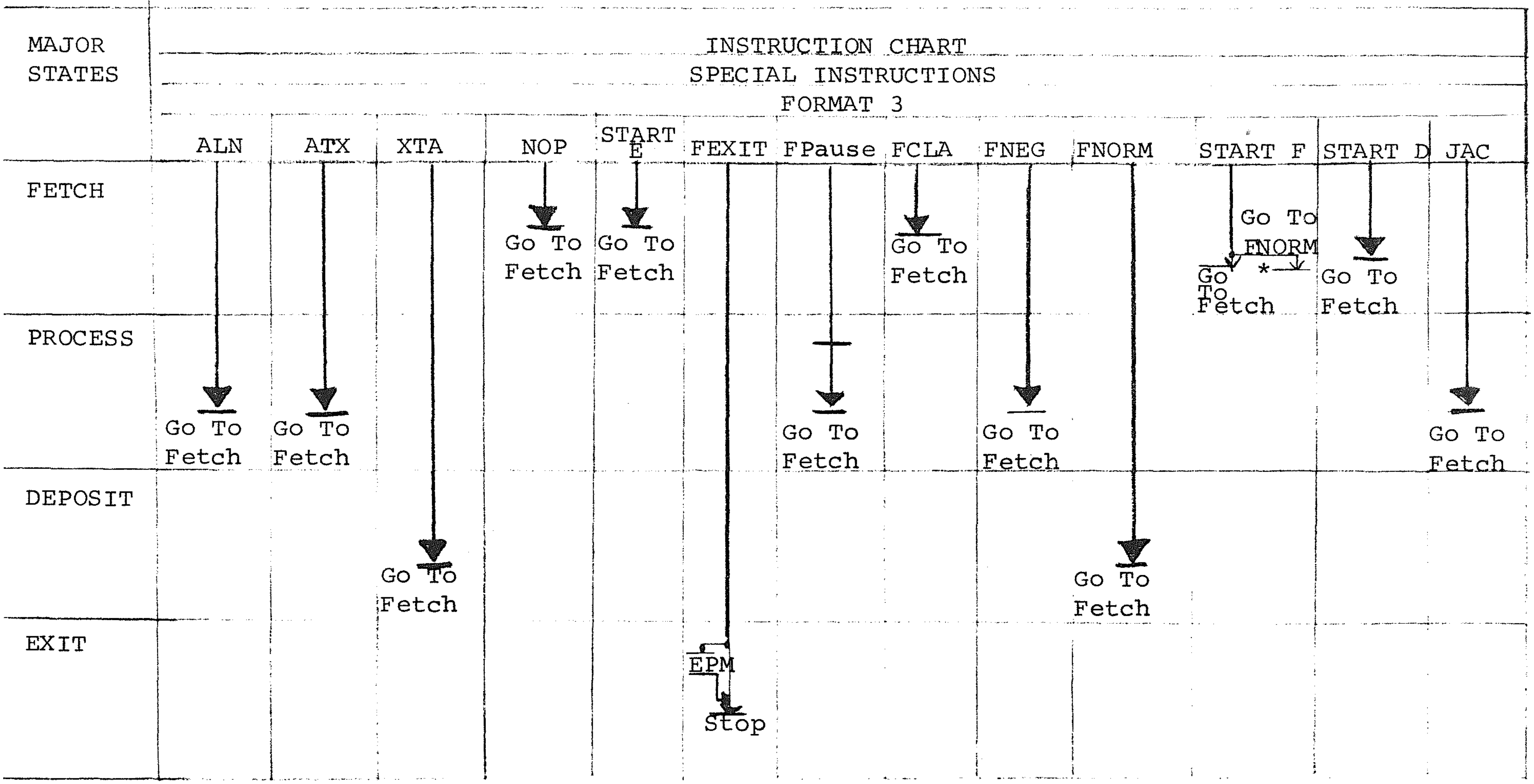

Major States
INSTRUCTION CHART
DATA REFERENCE
SPECIAL INSTRUCTIONS
FORMAT 1
FORMAT 2
FLDA
FADD/FADDM/FSUB
FDIV
FMUL/FMULM
FSTA
JXN
TRAP
LDX
JUMPS
SETX
SETB
JSR
JSA
FETCH
SPECIAL ST
EXECUTE
TMSC EXECUTE
DEPOSIT SPECIAL ST
PROCESS
EXIT SPECIAL ST
EMP
EPM
Go to FETCH
Go to Fetch
no JMP go to FETCH
(JMP TRUE)
go to Fetch
Stop
STOP
LEGEND:
Not using this major state
Using this major state
A decision was made
re-enter sequence

FIGURE 6-1 (Cont)



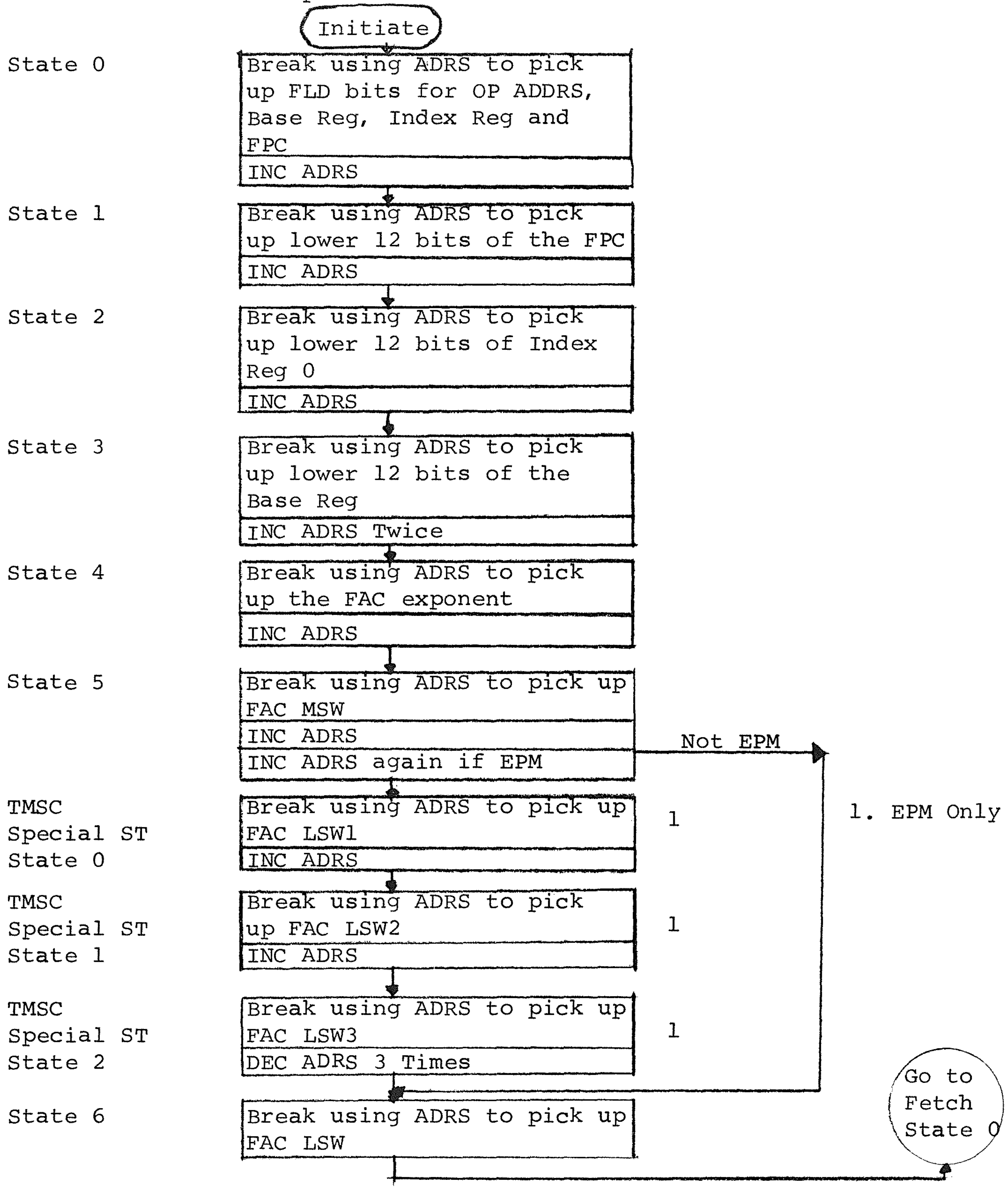
* If in EPM, the hardware forces a FNORM instruction. Control goes to FNORM in Process State 1. The purpose will be to round up FAC Bit 24 in the Deposit State.

6.3.1 Initiate

A simplified INITIATE flow is shown in Figure 6-2. The INITIATE flow is perhaps one of the most straight-forward in the package. There is only one decision made, and that test is performed at the end of state 5 for the extended precision mode. The object of INITIATE will be to do 7 or 10 (EPM) out-breaks for data, to be loaded into the so called "active registers" in the FPP. The first break will pick up the field bits for the operand address, base register, index register and FPC. Note that only the base and index register field bits are loaded from the MB during time state ∅. The FPC field bits are loaded from the AMSW to the OP ADDRS register and FPC register during state 1. The ADRS register is incremented to point to the core location that contains the lower 12 bits of the FPC. In state 1 a break to this location picks up its contents and loads them to the ALSW. Note the reason for the described data flow is because there is no method of loading the FPC directly from the MB. The entire A register is then transferred to the O register, then to the OP ADDRS, and finally to the FPC (CAR1 print). The ADRS is incremented to point to the core location that contains the index register pointer. State 2 will break to transfer the contents of MB to the index register (CAR4 print). The ADRS is incremented to point to the core location that contains the base register pointer. State 3 will break and transfer the contents of the MB to the base register (CAR5 print). The ADRS is incremented twice to point to the core location that contains the FAC EXP. This is done because the operand address if picked up, would have no meaning because it will be calculated for each instruction. The break in state 4 will transfer the contents of the MB to the FAC EXP. The ADRS is incremented to point to the core location which contains the FAC MSW. The break in state 5 will load the MB to the AMSW. Since the MB cannot be loaded directly to the FAC, the data will be loaded in the following fashion: MB to the respective A register. A register to the O register and O register to the FAC FRACTION. These last two operations are accomplished in INITIATE STATE 6. A decision is also made in state 5 which directs the FPP to pick up the least significant three words of the FAC if in the EPM or just to pick up the FAC LSW. In the event of EPM, the ADRS register is incremented twice to point to the memory location that contains the first word of the least significant three of the 60-Bit FAC. TMSC SPECIAL ST along with TMSC ST ∅, 1 and 2 request breaks to get data for the FAC'S least significant three words. The ALS 1, 2 and 3 will hold the contents of the respective MB. At the end of TMSC ST 2, the ADRS is decremented by three to point to the core location that contains the FAC LSW and TMSC SPECIAL ST is zeroed and control returns to INITIATE STATE 6. The break in state 6 will load the contents of the MB to ALSW, take the entire A register (AMSW, ALSW, ALS1, ALS2 and ALS3) and load it in the O register and finally the entire FAC FRAC is loaded from the O register. Control now goes to Fetch state ∅.

FIGURE 6-2

Simplified Initiate Flow


Initiate
State 0
Break using ADRS to pick up FLD bits for OP ADDRS, Base Reg, Index Reg and FPC
INC ADRS
State 1
Break using ADRS to pick up lower 12 bits of the FPC
INC ADRS
State 2
Break using ADRS to pick up lower 12 bits of Index Reg 0
INC ADRS
State 3
Break using ADRS to pick up lower 12 bits of the Base Reg
INC ADRS Twice
State 4
Break using ADRS to pick up the FAC exponent
INC ADRS
State 5
Break using ADRS to pick up FAC MSW
INC ADRS
INC ADRS again if EPM
Not EPM
TMSC Special ST State 0
Break using ADRS to pick up FAC LSW1
INC ADRS
1
1. EPM Only
TMSC Special ST State 1
Break using ADRS to pick up FAC LSW2
INC ADRS
1
TMSC Special ST State 2
Break using ADRS to pick up FAC LSW3
DEC ADRS 3 Times
1
State 6
Break using ADRS to pick up FAC LSW
Go to Fetch State 0

6.3.2 Fetch

Since the primary function of FETCH, other than strobing the instruction from core to the FIR, is one of address calculation, it is suggested that one familiarize himself with the different addressing modes described in Chapter 7, Paragraph 7.5. Aided with this understanding and the simplified FETCH flow, Figure 6-3, the FETCH flow diagram should be much more meaningful.

FETCH state ɸ requests an out-break using the contents of the FPC. The contents of the MB represent the instruction which is loaded into the FIR. In state ɸ FIR bits 3 and 4 are decoded for special instructions (3&4=Ø) or double word instructions (3=1&4=ɸ). Multible state special instructions go to PROCESS state 1 at the end of fetch state zero. Double word instructions jump to FETCH state 4 as there is no offset calculations to be done. The FAC EXP is tested for being greater than, $(27)_8$ to be used later by the JAL instruction in PROCESS state one if the current instruction is a JAL. Fetch state 1 will multiply the offset by three. The offset for single-word direct is made up of FIR bits 5-11 and the offset for single-word indirect is made up of FIR bits 9-11. State 2 will add this offset to the base pointer and put the result in the OP ADDRS. The OP ADDRS is then incremented for one of the following reasons:

a) MUL/DIV instructions pick up the MSW of the operand before the exponent. Remember the base points to a table of three consecutive word quantities determined by the offset. The order of data is EXP - MSW - LSW.

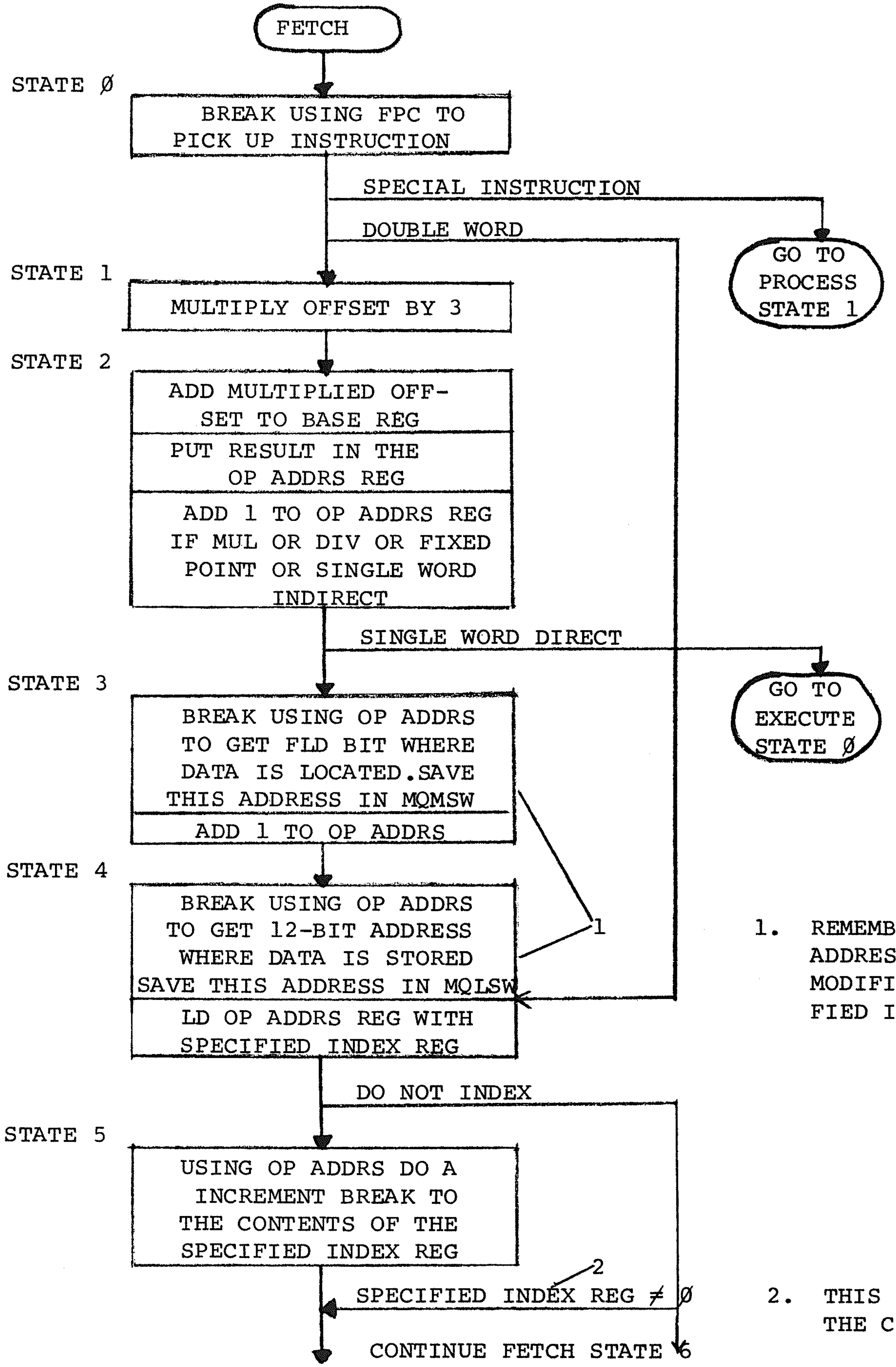
b) FIXED-POINT mode does not touch the exponent

c) Single-word indirect instructions will use the last two words of the three word quantity in the base as a base operand address.

Single-word direct instructions (FIR 4=1 and FIR 3=Ø) go to EXECUTE at the end of state 2. State 3 will break using the OP ADDRS to pick up the field bits (MBØ9-MB11) of the base operand address. This value is saved in the MQ register for possible adjustment by the specified index register.
The specified index register is also calculated in state 3. This is done by adding FIR bits 6-8 to the index register pointer. State 4 will break using the OP ADDRS to pick up the last word of the three word quantity and this data will make up the lower 12-bits of the operand address.
The OP ADDRS register is loaded with the specified index register. State 5 will index the contents of the specified index register if FIR 5=1. State 6 will pick up the contents of the specified index register provided the specified index register is nonzero. These contents are multiplied by 2, 3 or 6 depending on the mode. Two if fixed-point, three if floating-point and six if extended-precision mode. If EPM, the result is doubled again in TMSC SPECIAL ST $\wedge$ TMSC STATE Ø. This has the effect of multiplication by six. Control is then transferred to FETCH state 7. State 7 will take the multiplied contents of the index register and add this to either the contents of the MQ (single-word indirect) or the contents of the address indicated by FIR bits 9-11 and the contents

the memory location one greater than the instruction. (double-word). This result is loaded to the OP ADDRS and is the operand address which points to the data. In the case of MUL/DIV the OP ADDRS wants to be pointing to the MSW of data and not the exponent. Control now transfers to EXECUTE state ϕ.

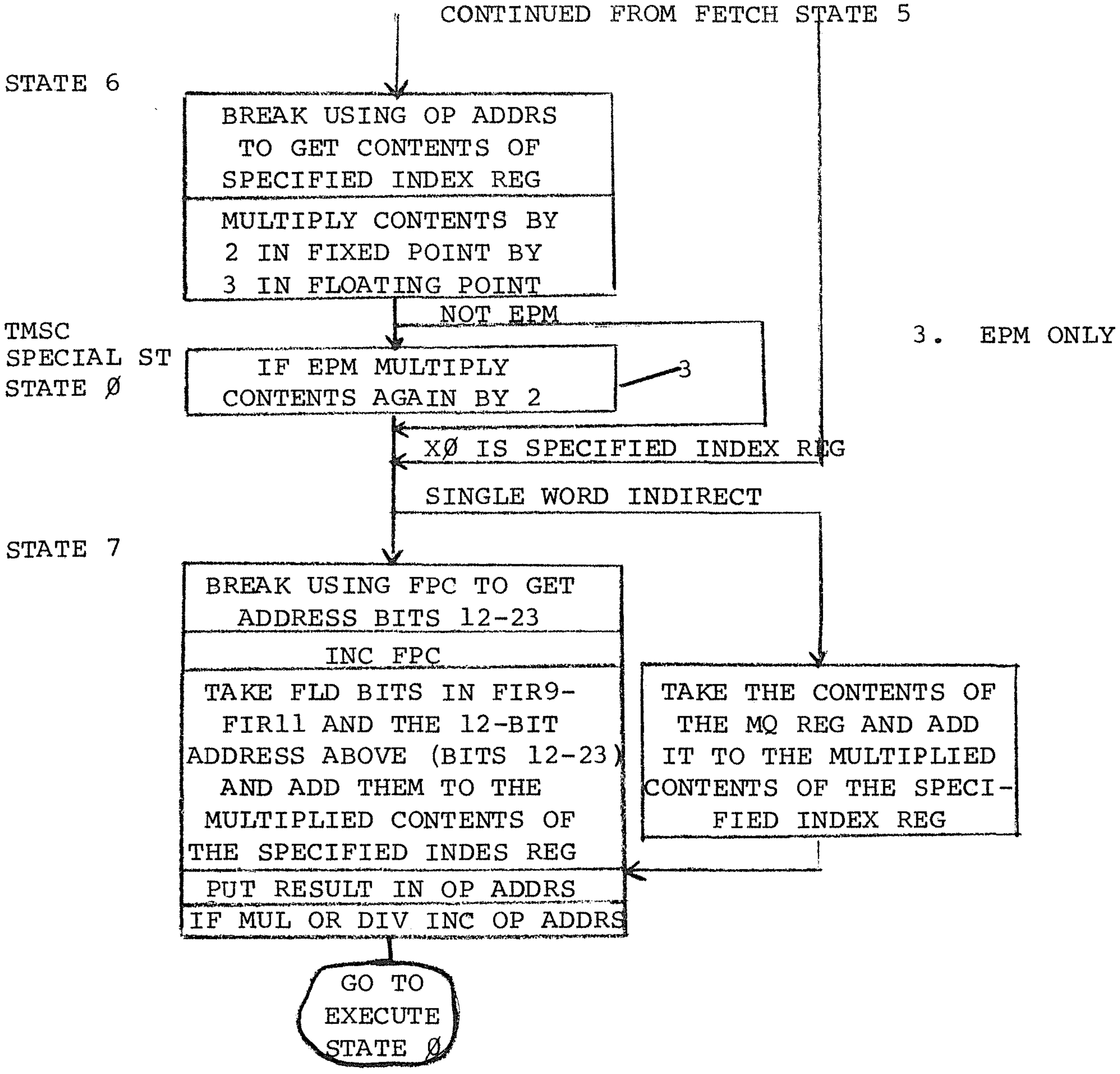
FIGURE 6-3
SIMPLIFIED FETCH FLOW


FETCH
STATE Ø
BREAK USING FPC TO PICK UP INSTRUCTION
SPECIAL INSTRUCTION
GO TO PROCESS STATE 1
DOUBLE WORD
STATE 1
MULTIPLY OFFSET BY 3
STATE 2
ADD MULTIPLIED OFFSET TO BASE REG
PUT RESULT IN THE OP ADDRS REG
ADD 1 TO OP ADDRS REG IF MUL OR DIV OR FIXED POINT OR SINGLE WORD INDIRECT
SINGLE WORD DIRECT
GO TO EXECUTE STATE Ø
STATE 3
BREAK USING OP ADDRS TO GET FLD BIT WHERE DATA IS LOCATED. SAVE THIS ADDRESS IN MQMSW
ADD 1 TO OP ADDRS
STATE 4
BREAK USING OP ADDRS TO GET 12-BIT ADDRESS WHERE DATA IS STORED SAVE THIS ADDRESS IN MQLSW
1
LD OP ADDRS REG WITH SPECIFIED INDEX REG
DO NOT INDEX
STATE 5
USING OP ADDRS DO A INCREMENT BREAK TO THE CONTENTS OF THE SPECIFIED INDEX REG
2
SPECIFIED INDEX REG ≠ Ø
CONTINUE FETCH STATE 6

1. REMEMBER THIS 15-BIT ADDRESS MAY LATER BE MODIFIED BY THE SPECIFIED INDEX REGISTER.

2. THIS DOES NOT MEAN THE CONTENTS

FIGURE 6-3 (Continued)
SIMPLIFIED FETCH FLOW


CONTINUED FROM FETCH STATE 5
STATE 6
BREAK USING OP ADDRS TO GET CONTENTS OF SPECIFIED INDEX REG
MULTIPLY CONTENTS BY 2 IN FIXED POINT BY 3 IN FLOATING POINT
NOT EPM
TMSC SPECIAL ST STATE Ø
IF EPM MULTIPLY CONTENTS AGAIN BY 2
3
3. EPM ONLY
XØ IS SPECIFIED INDEX REG
SINGLE WORD INDIRECT
STATE 7
BREAK USING FPC TO GET ADDRESS BITS 12-23
INC FPC
TAKE FLD BITS IN FIR9-FIR11 AND THE 12-BIT ADDRESS ABOVE (BITS 12-23) AND ADD THEM TO THE MULTIPLIED CONTENTS OF THE SPECIFIED INDES REG
PUT RESULT IN OP ADDRS
IF MUL OR DIV INC OP ADDRS
TAKE THE CONTENTS OF THE MQ REG AND ADD IT TO THE MULTIPLIED CONTENTS OF THE SPECIFIED INDEX REG
GO TO EXECUTE STATE Ø

6.3.3 Exit

On the Exit major state, there are two major directions of flow.

a) Exit for exponent-underflow not trapped.

b) All other exits except the above.

We will not discuss the flow for exponent-underflow not trapped. This flow is best described in the simplified EXIT flow for exponent underflow Figure 6-5. Figure 6-4 supports the description given for EXIT major state in general.

The first thing the EXIT major state must do is determine if in the EPM. If so, the TMSC SPECIAL ST flip-flop is set and a series of three in-breaks is commenced for the purpose of storing the least significant three words of the FAC in core. This sequence only takes place after the ADRS register has been incremented by 3, (Done in TMSC ST ɸ) as it will always point to the memory location of the FAC LSW after INITIATE, and provided the command register bit 7 is not set. If command register bit 7 is set, which specifies not to store the FAC, timing is advanced just to state 1, then to state 2 and finally to state 3. The ADRS is decremented in each time state of EXIT except state 7. If the FAC is stored, non EPM exits will store the FAC in core starting with the FAC LSW in state ɸ of EXIT. Timing will then advance through state 1 and 2 to deposit the FAC MSW and FAC EXP. Unlike INITIATE, the OP ADDRS location in the table is accessed in state 3, provided the command register bit 6 is not set. The break will take the current contents of the OP ADDRS to the MB and advance to state 4. State 4, if command register bit 5 is a zero, will request a break and deposit the current contents of the base (P4) register to the current core location. Timing will again advance to state 5 where the contents of the index register will be stored in core provided command register bit 4 is not set. The remaining two time states 6 and 7 will always request breaks to store the contents of the FPC and field bits respectively. At the end of state 7 the interrupt request flag is set and all activity stops in the FPP.

FIGURE 6 - 4

SIMPLIFIED EXIT FLOW

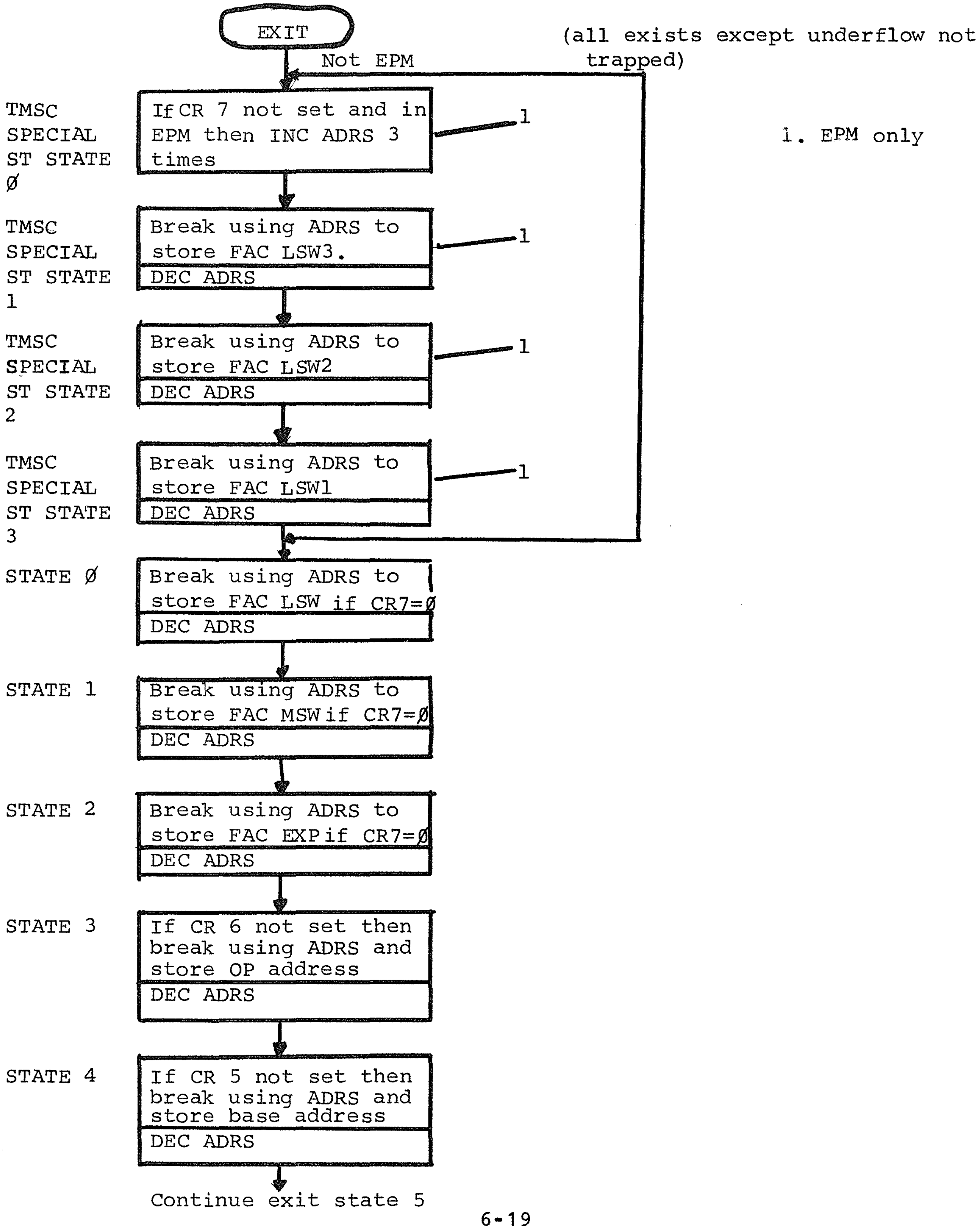

EXIT
Not EPM
(all exists except underflow not trapped)
1. EPM only
TMSC SPECIAL ST STATE Ø
If CR 7 not set and in EPM then INC ADRS 3 times
1
TMSC SPECIAL ST STATE 1
Break using ADRS to store FAC LSW3.
DEC ADRS
1
TMSC SPECIAL ST STATE 2
Break using ADRS to store FAC LSW2
DEC ADRS
1
TMSC SPECIAL ST STATE 3
Break using ADRS to store FAC LSW1
DEC ADRS
1
STATE Ø
Break using ADRS to store FAC LSW if CR7=Ø
DEC ADRS
STATE 1
Break using ADRS to store FAC MSW if CR7=Ø
DEC ADRS
STATE 2
Break using ADRS to store FAC EXP if CR7=Ø
DEC ADRS
STATE 3
If CR 6 not set then break using ADRS and store OP address
DEC ADRS
STATE 4
If CR 5 not set then break using ADRS and store base address
DEC ADRS
Continue exit state 5

FIGURE 6-4 (Continued)

Simplified Exit Flow

Continued from exit state 4

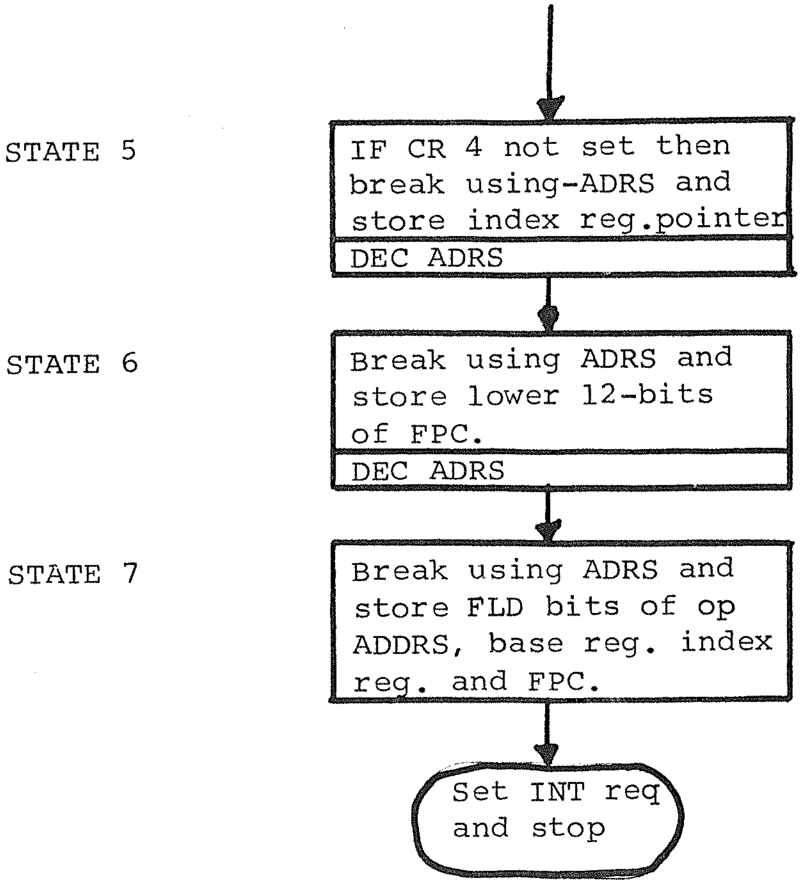
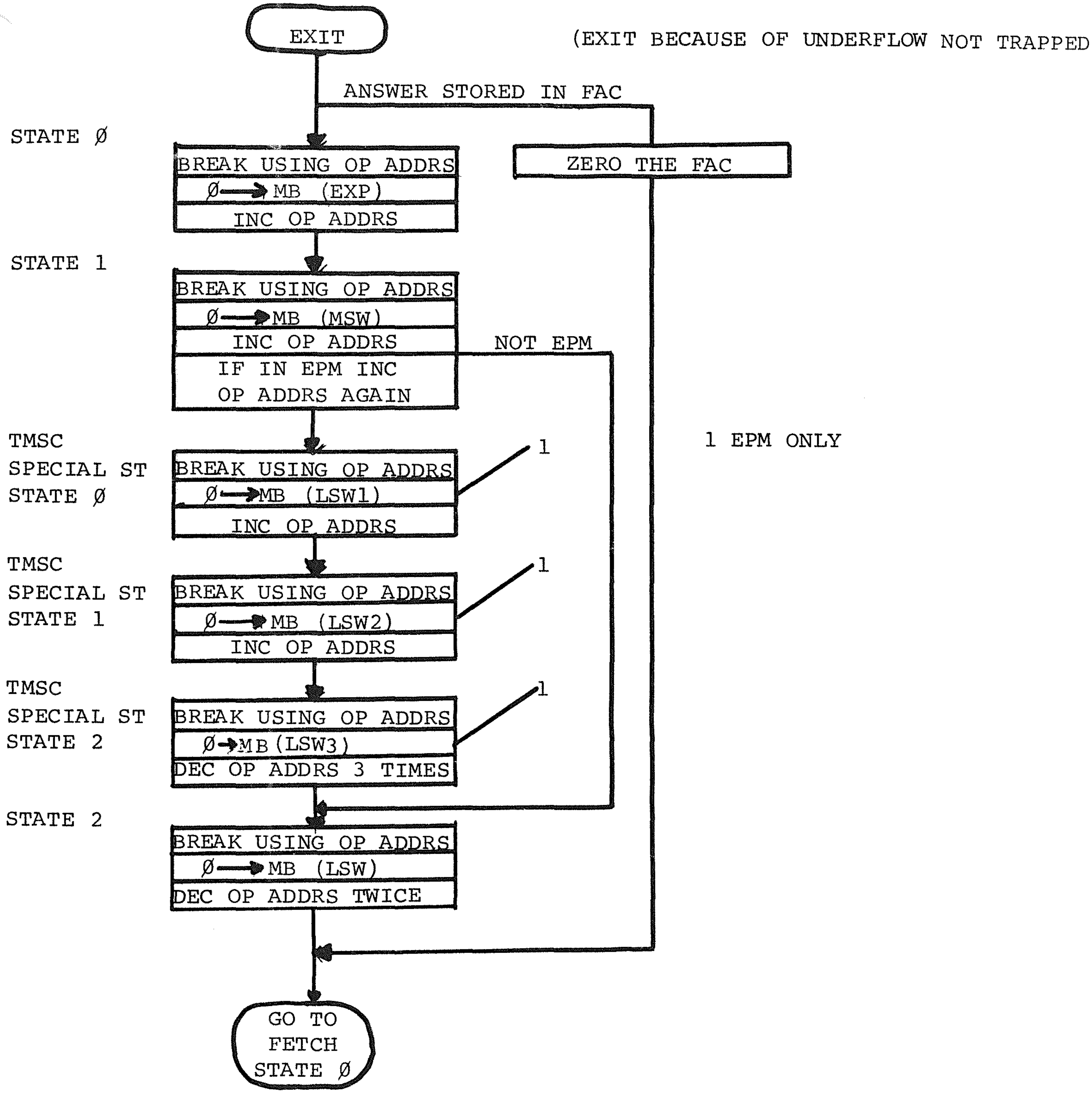

STATE 5
IF CR 4 not set then break using-ADRS and store index reg.pointer
DEC ADRS
STATE 6
Break using ADRS and store lower 12-bits of FPC.
DEC ADRS
STATE 7
Break using ADRS and store FLD bits of op ADDRS, base reg. index reg. and FPC.
Set INT req and stop

FIGURE 6-5

SIMPLIFIED EXIT FLOW



6.3.4 Deposit

At the end of any arithmetic calculation, the FPP can perform any number of the following operations depending on the mode and instruction:

a) Normalization

b) Rounding off the result

c) Checking for fractional overflow

d) Calculating the exponent

e) Storing the result in memory

f) Checking for exponent overflow or underflow

These operations are done in the DEPOSIT Major state as shown by the simplified DEPOSIT flow Figure 6-6. The DEPOSIT major state is enabled by causing the timing generator to jump to State 11.

When state 11 is enabled, the O register (Contains result) is shifted toward the most significant bit until the number is normalized: (See Chapter 7, Paragraph 7.3). This is indicated in the flows by O(N)⇒O(N-1). The 2's complement of the number of shifts required is tallied in the SC. When the number in the O register is normalized, the timing generator is advanced and the four mini states are generated. At this point, if bit 24 in the BEXT (EXT print) is set, it will be rounded up to bit 23. Note -- if in DEPOSIT because of the START F instruction in EPM, rounding will take bit 24 of the FAC and round up via bit 24 of the BEXT. After rounding a test for overflow is made. The only case where overflow can occur is if the number before rounding is 3777 7777. By adding one to the least significant bit (Bit 23), the result would be 4000 0000. A sign change from positive to negative has occurred. This causes an exit in fixed-point mode and in floating-point mode; the result would be shifted right one position and one would be added to the exponent. (Actually the SC temporarily holds this adjustment.) At the end of time state 11, the time state generator is advanced to one of the following:

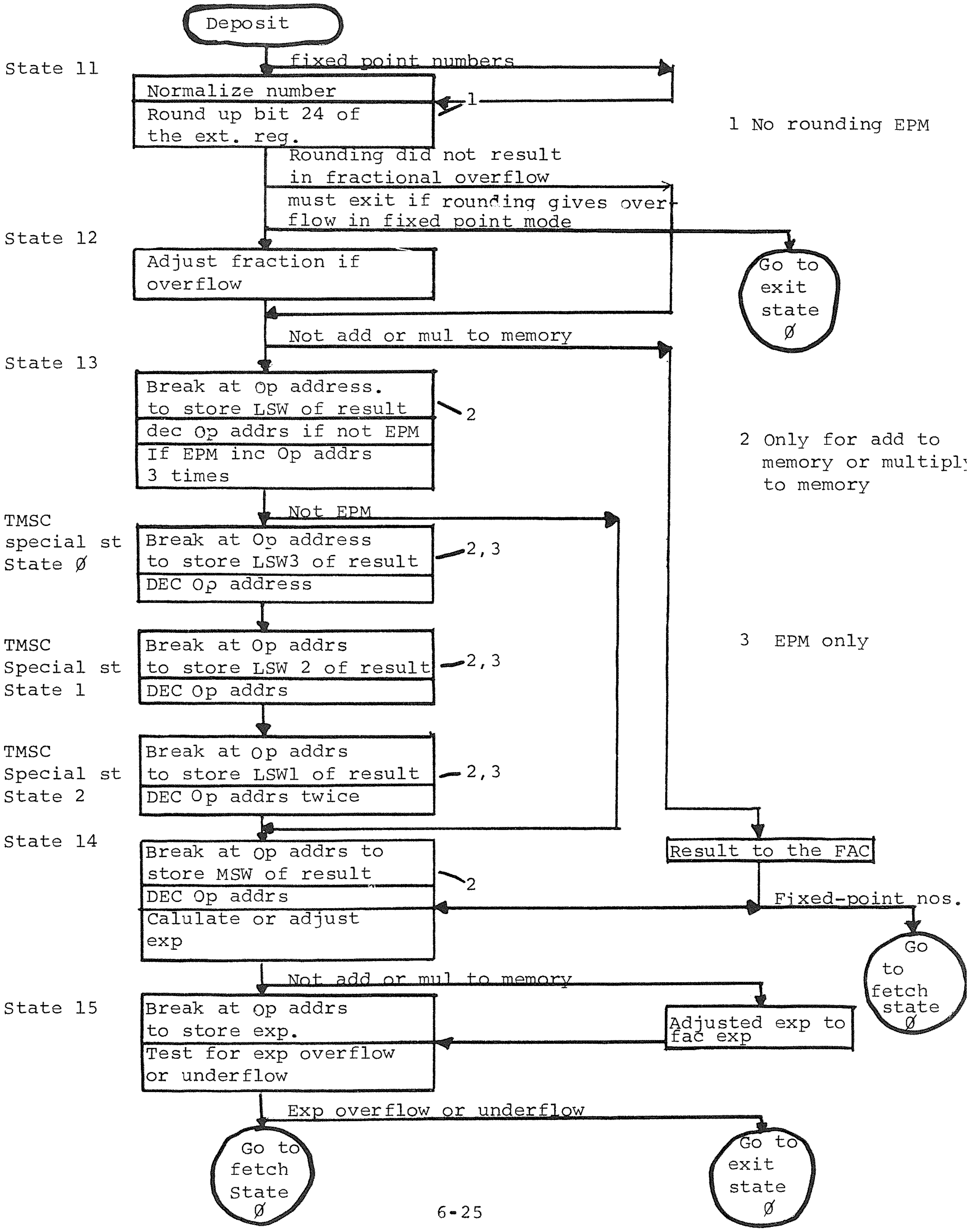
a) Time state 12 if an overflow is detected from rounding.

b) Time state 13 if results to memory.

c) Time state 14 if none of the above.

State 13 will break to memory at the OP ADDRS to deposit the least significant word of the result if doing an FADDM or FMULM instruction. If in the EPM, the next break will be to store the LSW 3 of the result. This is done by enabling TMSC SPECIAL ST ∧ TMSC ST Ø which requests a break at the OP ADDRS which was incremented by 3 in state 13. Following TMSC ST Ø, TMSC ST 1 and TMSC ST 2 will force breaks to store the LSW 2 and LSW1 of the result at the adjusted OP ADDRS. After the LSW 1 is stored in core, control returns to state 14 of DEPOSIT with the OP ADDRS pointing at the memory location to which the MSW of the result will be stored. A break to this location is performed to store the MSW. DEPOSIT major state is completed for fixed-point numbers at this point. So fixed-point mode will go to FETCH state Ø at the end of state 14. With floating-point numbers, the exponent is calculated for MUL/DIV and the contents of the SC are added to the exponent. State 15 either stores the exponent in core, if result to memory, or in the FAC EXP. It also will detect exponent overflow or exponent underflow. This means that the resultant exponent has exceeded the most positive or most negative value. (See Chapter 2, Paragraph 2.3). If this value is exceeded, control is transferred to EXIT state zero. If no overflow or underflow, control is transferred to FETCH state zero.

FIGURE 6-6 SIMPLIFIED DEPOSIT FLOW


Deposit
State 11
fixed point numbers
Normalize number
Round up bit 24 of the ext. reg.
1
1 No rounding EPM
Rounding did not result in fractional overflow
must exit if rounding gives overflow in fixed point mode
State 12
Adjust fraction if overflow
Go to exit state Ø
Not add or mul to memory
State 13
Break at Op address. to store LSW of result
dec Op addrs if not EPM
If EPM inc Op addrs 3 times
2
2 Only for add to memory or multipl to memory
Not EPM
TMSC special st State Ø
Break at Op address to store LSW3 of result
DEC Op address
2,3
TMSC Special st State 1
Break at Op addrs to store LSW 2 of result
DEC Op addrs
2,3
3 EPM only
TMSC Special st State 2
Break at Op addrs to store LSW1 of result
DEC Op addrs twice
2,3
State 14
Break at Op addrs to store MSW of result
DEC Op addrs
Calulate or adjust exp
2
Result to the FAC
Fixed-point nos.
Go to fetch state Ø
Not add or mul to memory
State 15
Break at Op addrs to store exp.
Test for exp overflow or underflow
Adjusted exp to fac exp
Exp overflow or underflow
Go to fetch State Ø
Go to exit state Ø

6.4 Flow Diagrams - Instructions

In the following paragraphs we will discuss the flow diagrams for the data reference instructions. It is felt that the special instructions are more easily understood than the data reference instructions, therefore, those flows will not be covered.

6.4.1 LDA and STR

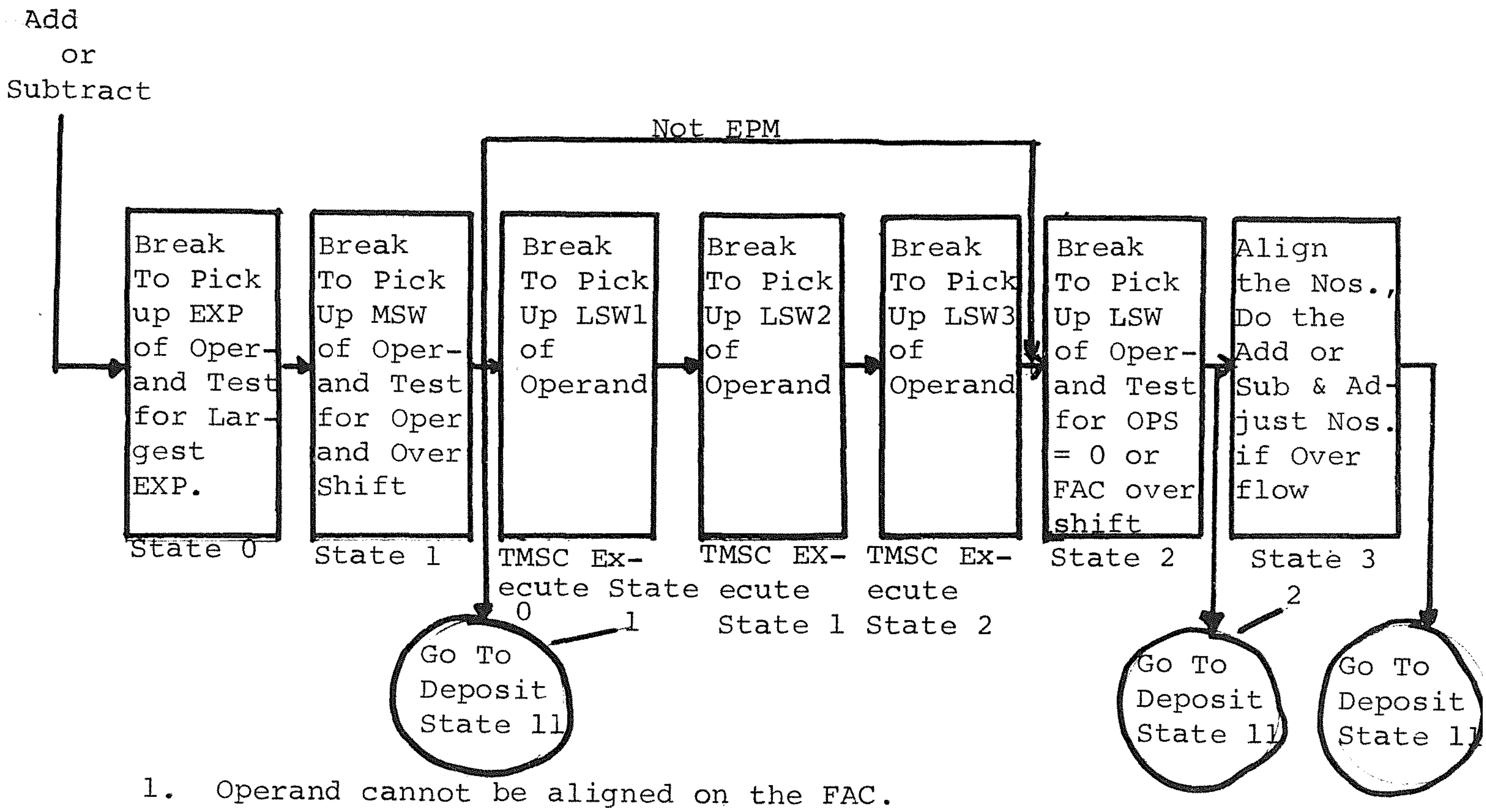
The LDA flow diagram is executed during the EXECUTE major state. If the FPP is in floating-point mode the contents of the three sequential memory locations defined by the contents of the OP ADDRS are loaded into the FAC. In fixed-point mode, only the contents of two sequential memory locations are loaded into the FAC. In the extended precision mode the contents of six sequential memory locations are loaded into the FAC. Since the MB cannot be directly loaded into the FAC FRAC, it must be assembled into the respective A register 12-bits at a time. During the last break cycle in state 2 of EXECUTE the entire A register is then loaded into the O register and then the O register is loaded into the FAC FRAC. The STR flow diagram is basically the reverse process of the LDA. The STR flow diagram is implemented during the EXECUTE major state. In this case, the appropriate 12-bit bytes of the FAC are stored in the memory location defined by the OP ADDRS register.

6.4.2 ADD/SUB (floating-point)

A block diagram of the FPP's ADD/SUB is shown in Figure 6- . The flow diagram is implemented during the EXECUTE major state of an FADD, FADDM, or FSUB instruction, when the FPP12 is in floating-point mode. In order to add or subtract two floating-point numbers, the exponents must be aligned; that is, the fractional part of the number with the smallest exponent must be shifted right and the exponent incremented until the two exponents are equal.

FIGURE 6-7

Add/Subtract Block Diagram


Add or Subtract
Not EPM
Break To Pick up EXP of Operand Test for Largest EXP.
State 0
Break To Pick Up MSW of Operand Test for Oper and Over Shift
State 1
Break To Pick Up LSW1 of Operand
TMSC Execute State 0
Break To Pick Up LSW2 of Operand
TMSC EXecute State 1
Break To Pick Up LSW3 of Operand
TMSC Execute State 2
Break To Pick Up LSW of Operand Test for OPS = 0 or FAC over shift
State 2
Align the Nos., Do the Add or Sub & Adjust Nos. if Over flow
State 3
1
2
Go To Deposit State 11
Go To Deposit State 11
Go To Deposit State 11

1. Operand cannot be aligned on the FAC.
2. FAC cannot be aligned on the operand, or one of the operands equals zero.

Since the A register cannot be shifted, the fraction of the number with the smallest exponent must be loaded into the B register. This is done in the following fashion. During State 0 Mini State 1 and 2 the difference between the FAC exponent and the operand exponent is calculated and stored in the O. The operand exponent is stored in the MQLSW for future reference. The sign of the difference is used to determine which exponent is larger and, hence, which fraction must be loaded into the B register. The absolute value of the difference is also loaded into the SHFT CNTR which is subsequently used to determine the number of times the B register is shifted. During State 0 Mini State 3 and 4, the number of shifts required is checked to determine if it is more than $(27)_8$; that is, if the fraction to be shifted will be completely shifted out of the B register. The OVERSHFT flip-flop, (AST1 NO SHFT) which is set during State 1 Mini State 2, is used to indicate that the required number of shifts is greater than $(27)_8$.

State 1 and State 2 are used to fetch the most significant two words of the operand fraction and TMSC State ϕ, 1 and 2 get the least significant three words. In the case of FSUB, where the operand exponent is greater than the FAC exponent, the 1's complement of the operand fraction is loaded into the A. This is necessary since the ALU cannot perform the operation B minus A.

Before the B is shifted, the fraction of the number with the largest exponent (which is stored in the A register) is checked to make sure that it is nonzero. This prevents the loss of significance when a number with a nonzero exponent and a zero fraction is added to or subtracted from another number. If the fraction of the number with the largest exponent is zero, the fraction stored in the B register is loaded into the O register and its associated exponent is loaded into MQLSW and the DEPOSIT major state is enabled.

When State 3 is enabled, the B register is shifted toward the least significant bit. The number of positions is determined by the number contained in the SHFT CNTR. When shifting is completed, the timing is advanced and Mini State 1 is used to perform the required operation between the A and B registers. If the operation causes an overflow condition, adjust the fraction one bit position right and set the SHFT CNTR to 1.

At the end of State 3, the result of the addition or subtraction of the aligned fractions is stored in the O register, MQLSW contains the value of exponent of the aligned fractions, and SHFT CNTR contains a 0 or contains a 1 in the case when the fraction of the result was shifted right. The DEPOSIT major state is entered to perform the normalization, rounding, exponent calculation, and storage of the result.

6.4.3 ADD/SUB (Fixed-Point)

This flow diagram refers to the EXECUTE state of an FADD, FADDM, or FSUB instruction, when the FPP12 is in fixed-point mode. If the result is greater than $37777777)_8$ or less than $40000001)_8$ the fraction overflow bit (bit 4 of the status register) is set and the EXIT major state is enabled causing the FPP12 to halt.

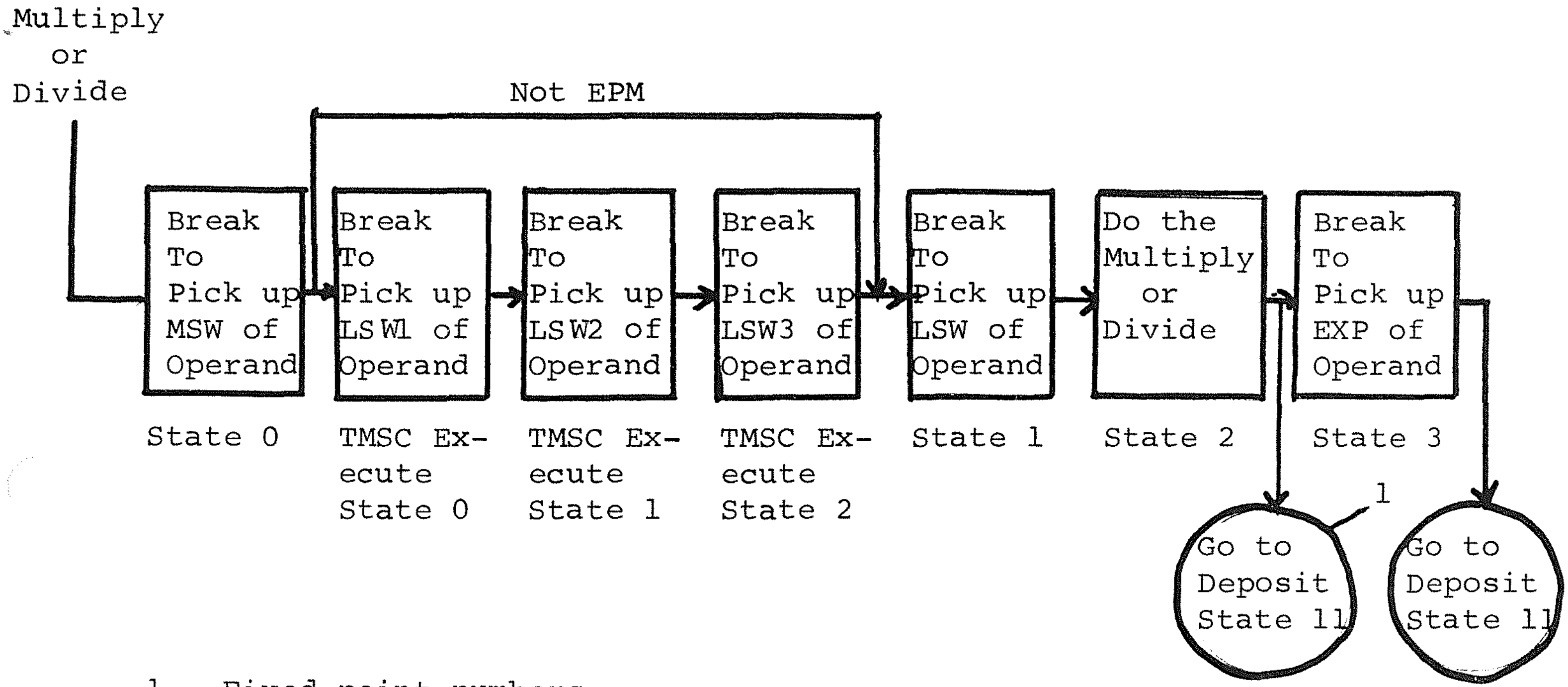
6.4.4 MULTIPLY

A block diagram of the multiply is shown in Figure 6-8. The MULTIPLY flow diagram details the algorithm used by the FPP12 to perform floating-point and fixed-point multiplications. The absolute values of the two fractions are multiplied together to give a positive result which is negated if either but not both of the fractions are negative. The absolute value of the operand fraction is loaded in the MQ during State 0 and State 1. If the FAC fraction is negative, the complement is loaded in A and a CARRY INSERT is generated when A and B are added during the multiply cycle so that the 2's complement of the FAC fraction is added to the contents of the B register.

The multiplication of the two fractions is performed in State 2. Each cycle of the algorithm requires two clock pulses. The first clock pulse is used to load A plus B into the O register and to decrement the SHFT CNTR. The second clock

FIGURE 6.8

Multiply/Divide Block Diagram


Multiply
or
Divide
Not EPM
Break To Pick up MSW of Operand
State 0
Break To Pick up LSW1 of Operand
TMSC Execute State 0
Break To Pick up LSW2 of Operand
TMSC Execute State 1
Break To Pick up LSW3 of Operand
TMSC Execute State 2
Break To Pick up LSW of Operand
State 1
Do the Multiply or Divide
State 2
Break To Pick up EXP of Operand
State 3
1
Go to Deposit State 11
Go to Deposit State 11

1. Fixed point numbers

pulse also shifts the MQ toward the least significant bit, which brings the next binary bit of the multiplier into the 23rd or 59th position for testing on the next cycle. The final product is stored in the B register.

The remaining time states used in MULTIPLY store the product in O and load the operand exponent in the MQLSW for use in the DEPOSIT cycle.

6.4.5 DIVIDE

A block diagram of the FPP's DIVIDE is shown in Figure 6-8. The Divide algorithm used in the FPP12 is shown in the DIVIDE flow diagram. Again, the divide routine makes both the divisor and the dividend positive, calculates the quotient, and negates it if either but not both of the divisor or the dividend was negative.

During State 0 and State 1 and TMSC STϕ, 1 and 2, the FAC and the absolute value of the operand fraction are loaded into the A and the B registers, respectively. During State 2, the division of the fractions occurs. Again, like the MULTIPLY algorithm, two clock cycles are required for a shift cycle of the divide algorithm. During the first half of the cycle, the divisor is subtracted from the current remainder (which is stored in the A register) and, if the result is positive (CARRY OUT = 1), the difference is loaded into the O register and a 1 is shifted into the least significant bit of the MQ. During the second half of the cycle, the new remainder is multiplied by two and stored in the A register. The O register is also shifted left so that the contents of the A and O registers are the same. If the result of the trail subtraction is negative (CARRY OUT=0), a zero is shifted into the least significant bit of MQ and the current remainder, which is stored in both the O and the A registers, is multiplied by two. This multiplication is done by shifting the O register. The second half of the cycle then loads the contents of O register into the A register. The 28-bit or 60-bit (EPM) quotient is found in the MQ when division is complete. The remaining time states used in the divide routines load the MQ into the O, and the operand exponent into the MQLSW. During State 2 Mini State 3, the quotient is divided by two and a one is loaded into the shift counter if the first subtraction gave a positive result. This will occur if the dividend is greater than the divisor.

6.4.6 SPECIAL INSTRUCTIONS

The remaining flow diagrams are those associated with the instructions that use Special Format 1, 2, and 3. Four of these instructions (FEXIT, FCLA, STARTF, and STARTD) are performed completely in FETCH State 0. All of the conditional jumps are also completed in FETCH State 0, if the condition is not true; that is, if the jump is not performed. In all other cases, the PROCESS major state is entered at the beginning of State 1. Table 6-2 shows the equivalence between the instruction mnemonic and its corresponding flow diagram heading.

CHAPTER 7 -- MAINTENANCE GUIDE

7.1 INTRODUCTION

This chapter is designed to assist a technically involved individual in solving hardware problems who may or may not of had training on the FPP12. There are some very important concepts and operation that must be understood before one can effectively fix an FPP12. These will be discussed in the following paragraphs along with several maintenance tips.

7.2 INTEGERS AND FLOATING POINT NUMBERS

It is very important for the user to understand the binary floating point (Binary point) number system and how the FPP adapts itself to this system. One of the most important concepts is, that a number in floating point format is a fraction and not an integer. This number is justified on the binary point which is to the right of the sign bit (Bit Ø) of the 24 or 6Ø-bit (EPM) mantissa. The magnitude of this fraction is maintained in the exponent. It is easier to understand binary floating point if one understands scientific notation in the decimal system. For example, the decimal number of 3,ØØØ can be represented in an unlimited number of ways.

DECIMAL $3.X10^3 = 30.X10^2 = 300.X10^1 = 3{,}000.X10^0 = 30{,}000.X10^{-1}$

With binary floating point format, the similar relationship illustrated below exists with the value of six.

BINARY $.11X2^3 = 1.1X2^2 = 11.X2^1 = 110.X2^0 = 1100.X2^{-1}$

There are two operations (Float and Fix) used frequently in the FPP that convert integers to fractions and fractions to integers. They are discussed next.

7.2.1 FLOAT

When a number is floated, it is converted for its integer form into the fractional floating point format. This is done by placing the number of bits of the word length (mantissa is (27)$_8$) into the exponent and then shift the mantissa left until all the nonsignificant ones or zeros (See Paragraph 7.3) are eliminated.

1. The exponent is decremented by one for each shift until the fraction is normalized.
2. Let's take the integer one and float it.

EXPONENT$_{(8)}$	MANTISSA		
	MSW	LSW	
ØØØØ	000,000,000,000,	000,000,000,001	START
0027	0.00,000,000,000,	000,000,000,001	SHIFTS$_{(8)}$
0026	0.00,000,000,000,	000,000,000,010	1
0025	0.00,000,000,000,	000,000,000,100	2
	↓	↓	
0002	0.01,000,000,000,	000,000,000,000	25
0001	0.10,000,000,000,	000,000,000,000	26

The number one is now in correct floating point format. To better understand this fraction we will look at its fractional value.

	S	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	
0001	0.	1	0,	0	0	0,	0	0	0,	--- ETC.

Note that the above number is equal to one.

$0.10\ X2^{1} = 01.0\ X2^{0}$ or 1

7.2.2 FIX OR INTEGERIZE

When a fraction is fixed, it is converted to an integer. This is the reverse process of FLOATING an integer. To integerize a floating point number the exponent is adjusted to $(27)_8$ and the fraction is shifted right depending on the difference of its exponent and $(27)_8$. This difference is loaded into the shift counter and decremented until the shift counter is equal to zero. If the exponent is greater than $(27)_8$, the floating point number is impossible to fix (Too large an integer for 23 bits). The JAL instruction tests to see if fixing is possible. Observe the fixing of the floating point fraction below. (Notice that the value of the fraction is equal to seven: $0.111X2^{3}$ = 7 or $0111.X2^{0}$).

EXPONENT$_{(8)}$	SHIFT CNTR$_{(8)}$	MANTISSA		
		MSW	LSW	
0003	ØØØØ	0.11,100,000,000,	000,000,000,000	START
0027	0024	0.11,100,000,000,	000,000,000,000	exponent difference (24
0027	0023	0.01,110,000,000,	000,000,000,000	Shift 1
0027	0022	0.00,111,000,000,	000,000,000,000	2
0027	0001	0.00,000,000,000,	000,000,001,110	23
0027	0000	0.00,000,000,000,	000,000,000,111	24

At the end of the 24th shift the fraction has been integerized and is equal to seven.

7.3 NORMALIZE

Normalize eliminates nonsignificant leading zeros or ones. To accomplis this, the number (mantissa) is shifted to the left until one of the following conditions is true.

a. Bits Ø and 1 are different (0.1 or 1.0)

b. Only bits Ø and 1 of the entire fraction are equal to ones.

For each shift the exponent is decreased by one. Note that the value of the number is not affected because the exponent keeps track of the direction and number of shifts. Notice that the normalized number can be shifted to the right to regain the exact starting value. Also remember the MSB is the sign bit and the binary point is always to the right of the sign bit.

Observe the examples below based on a six bit mantissa and a twelve bit exponent.

	EXPONENT*	MANTISSA		START .	1 .	2 .	3 .
a)	↲SIGN BIT	↲SIGN BIT					
	000,000,000,000	0.00,101		X			
	111,111,111,111	0.01,010			X		
	111,111,111,11Ø	0.10,100				X	
b)	000,000,011,001	0.00,011	Pos. nos eliminate leading Ø's	X			
	000,000,011,000	0.00,110			X		
	000,000,010,111	0.01,100				X	
	000,000,010,11Ø	0.11,000					X
c)	000,000,000,010	1.11,101		X			
	000,000,000,001	1.11,010			X		
	000,000,000,000	1.10,100				X	
	111,111,111,111	1.01,000					X
d)	111,111,111,101	1.11,000	Neg. nos eliminate leading 1's	X			
	111,111,111,100	1.10,000			X		

*Each step (shift) is actually counted in the shift counter (CAR 6 PRINT) and when the number is normalized, the contents of the shift counter is added to the exponent.

The example program below illustrates the usefulness of having normalized numbers. The registers used, are only three bits long with a four bit exponent.

Program sequence

1. A X B = PRODUCT 1
2. A X PRODUCT 1 = PRODUCT 2
3. A X PRODUCT 2 = PRODUCT 3

Example 1 with no normalization of results

```
                                                     Sign    Value of each bit
                                                     1/2     in Fraction with
                                                     1/4     zero exponent
                                                     1/8
    EXPONENT (4 Bits)
      Sign
1.  0,000                             B = 0.10               1/2 X 1/2 = 1/4
    0,000                             A = 0.10
2.  0,000                   PRODUCT   1 = 0.01               1/4 X 1/2 = 1/8
                                      A = 0.10
    0,000                   PRODUCT   2 = 0.001
                                              significant digit is lost
```

Notice that the significant bit of Product 2 has fallen into oblivion and that our answer has been eliminated with nonsignificant zeros. Two ways of preventing this loss, are to make the registers longer or to normalize the product after each multiply. Example 2 illustrates the use of normalizing the product.

Example 2

```
                                                     Sign
                                                     1/2
    EXPONENT (4 Bits)                                1/4
      Sign                                           1/8
1.  0,000                             B = 0.10               1/2 X 1/2 = 1/4
    0,000                             A = 0.10
    0,000                   PRODUCT   1 = 0.01
2.  1,111  PRODUCT 1 Normalized         = 0.10               1/4 X 1/2 = 1/8
                                      A = 0.10
    1,111                   PRODUCT   2 = 0.01
3.  1,11Ø  PRODUCT 2 Normalized         = 0.10               1/8 X 1/2 = 1/16
                                      A = 0.10
    1,11Ø                   PRODUCT   3 = 0.01
    1,1Ø1  PRODUCT 3 Normalized         = 0.10               Final PRODUCT
```

If the exponent of product 3 was adjusted to zero, the actual fractional value would look something like this:

```
                                                     Sign
                                                     1/2
                                                     1/4
EXPONENT  MANTISSA                                   1/8
                                                     1/16
    1,1Ø1   0.10   =   0,000   0.0001
```

Normalize is done at the completion of every floating point arithmetic operation in DEPOSIT state 11.

7.4 ALIGN

Align is used to shift the FAC mantissa right or left. This is usually performed for purposes of fixing, aligning exponents (Make the exponents equal) and for general shifting. In fixed-point mode the direction and number of shifts of the mantissa, depends on the contents of the specified index register. In floating-point mode, the FAC is shifted until the FAC exponent equals the contents of the specified index register. If the specified index register is index register zero, the FAC is shifted so that its exponent is equal to $(27)_8$. This will intergenize or fix a fraction. (Paragraph 7.2.2) Examples of the ALIGN instruction are shown below.

Examples in floating-point

	EXPONENT	C (INDEX)	MANTISSA	
1.	0000	0003	0.11,010	No. to be aligned
	0003		0.00,011,1	Aligned
			↑ LOST	
2.	0015	0013	0.00,101	No. to be aligned
	0013		0.10,100	Aligned
3.	0025	Align on Index Ø	0.10,100	No. to be aligned
	0027		0.00,101	Aligned

In fixed point mode the sign bit of the contents of the index register determines the direction of the shifting.

Examples in fixed-point

	C(INDEX)	MANTISSA	
1.	0003	0.10,100	Shift 3 left
		0.00,010	
2.	7776	0.00,110	Shift 2 right
		0.11,000	

Align is also done during each add and subtract instruction if the exponents are different, as you cannot add or substract numbers with unlike exponents. This is accomplished by aligning the number with the smaller exponent to that of the larger exponent. An example is shown below.

Example of an Add instruction.

EXPONENTS	OPERANDS	
A = ØØØ2	Ø.1Ø,1ØØ	
B = ØØØØ	Ø.11,1ØØ	--- Align on the larger A
B = ØØØ2	Ø.ØØ,111	Aligned on A
A = 00Ø2	0.1Ø,1ØØ	2 1/2
B = 0002	+ 0.ØØ,111	+ 7/8
ANS = 0002	Ø.11,Ø11	3 3/8

7.5 UNDERSTANDING ADDRESSING

A thorough understanding of the addressing schemes is a must if one is to be effective in diagnosing hardware bugs. If instruction bits 3 or 4 or both are equal to a one, then one of these address schemes is selected. These are the Double-word, Single-word and Single-word indirect. This Chapter will illustrate and describe the operation of the addressing schemes by examples using the FLDA (Load the FAC from memory) instruction in the standard floating-point mode.

7.5.1 DOUBLE WORD example

Y = C(bits 9-23) + M* [C(X+XØ) + C(bit 5)] & (X)

General Form	OP CODE	1	Ø	+	X	ADDRESS	, ADDRESS
	0 - 2	3	4	5	6 8	9 11	12 23
FLDA Example	Ø Ø Ø	1	Ø	1	Ø 1 1	Ø Ø 1	1 1 1 ,ØØØ,ØØØ,ØØØ

Parameters are:
XØ is assigned to 1ØØØ
X3 points to 1ØØ3
1ØØ3/ØØØ1 1 before indexing
1ØØ3/ØØØ2 2 after indexing

17ØØ6/ ØØØ1	FAC EXP	Data to be loaded into FAC
17ØØ7/ 2ØØØ	FAC MSW	
1701Ø/ ØØØØ	FAC LSW	

Steps for calculating address

1. Bits 3 and 4 decode as a double-word instruction.
2. The address of XØ has previously been set to field Ø location 1ØØØ for this example. Bits 6-8 are selecting index register three. So we are concerned with the contents of 1ØØ3 which will later modify the address in bits 9-23. However, if bits 6-8 were equal to zero, bits 9-23 point to the operand directly. (FAC EXP)
3. The contents of X3 are to be indexed because bit 5 is set. The FPP now breaks to 1ØØ3 and does an MB Increment, so location 1ØØ3 now contains ØØØ2.
4. A break to the specified index register (X3) will now pick up its contents and multiply this by 2, 3 or 6 depending on the mode; in fixed-point by 2, floating-point by 3 and in extended-precision by 6. In this example 2 is multiplied by 3.
5. The result, 6, is then added to bits 9-23 which was equal to field 1 location 7ØØØ. The operand address is now equal to 1 7006. At this point the FPP would go from Fetch to Execute and break at location 17006 for the exponent, 17ØØ7 for the FAC-MSW and 17Ø1Ø for the FAC-LSW and then return to fetch.

7.5.2 SINGLE WORD example

Y = C(base register) + 3*(offset)

General Form

OP Code	0	1	,	Offset,
0 - 2	3	4	5	11

FLDA Example

Parameters are:

0 0 0	0	1	0, 0 0 0 , 1 0 0
0 2	3	4	5 11

Base register has been assigned to 6ØØØ

6Ø14 / ØØØ1	FAC EXP	Data to be loaded into FAC
6015 / 2ØØØ	FAC MSW	
6016 / ØØØ4	FAC LSW	

Steps for calculating address

1. Bits 3 and 4 decode as a single-word direct instruction.
2. Bits 5-11 are used as an offset with single-word direct instructions. Bits 5-11 are simply multiplied by 3. In this example, the offset 4 is multiplied by 3 with the result of $(14)_8$.
3. Fourteen is then added to the base register (Ø6ØØØ) and the result Ø6Ø14 is the operand address.
4. At this point, the FPP would go from Fetch to Execute and break at locations 6014, 6015 and 6016 to pick up the exponent, FAC-MSW and FAC-LSW.

This form of addressing may be the easiest to use when dealing with toggle in type programs.

7.5.3 SINGLE WORD INDRIECT Example

Y= C[Bits 21-35 of C (Base reg.) +3*offset]
    + (M)* [C (X + XØ) + C(Bit 5)]S(X)

General Form

Op Code	1	1	+	X	Offset
Ø 2	3	4	5	6 8	9 11

LDA Example

Ø Ø Ø	1	1	1	Ø Ø Ø	Ø Ø 1

Parameters are:

XØ assigned to 2ØØØ
X1 points to 2ØØ1
Base register has been assigned to 7ØØØ
2ØØ1 / ØØØ6 / before indexing
2ØØ1 / ØØØ7 / after indexing
7ØØ4 / ØØØ1 } these are locations in the base table
7ØØ5 / 5ØØØ } that contain the address of data
1 5Ø25 / ØØØ1 FAC EXP }
1 5026 / 2ØØØ FAC MSW } Data to be loaded to FAC
1 5Ø27 / ØØØØ FAC LSW }

Steps for calculating address

1. Bits 3 and 4 decode a single-word indirect instruction.

2. Bits 9-11 contains an offset which is to be multiplied by 3. This will give us 3 in this example.

3. This offset of 3 is then added to the base register pointer which is set to Ø 7ØØØ giving us 7ØØ3. (It is worthy of noting the base register plus its offset always points to every third location following its initial setting. For example, if the offset was 2, the base register plus its offset would point to Ø 7Ø06. This is also true with the single-word direct.) Locations 7ØØ3, 7ØØ4 and 7ØØ5 together contain a three-word quantity. Of this 36-bit quantity, the least significant 15 bits Loc. 7004 and 7005 will represent the indirect address. Therefore the base register pointer must be incremented from 7003 to 7004.

4. The FPP will now break to locations 7ØØ4 (Bits 12-23) and 7ØØ5 (bits 24-35). The contents of these locations will be saved in the MQ. Note that bits 21-35 are the only ones of interest as this is a 15 bit address. From this point the single-word indirect performs identically to the double-word with one exception. The MQ serves the same purpose as bits 9-23 of the double-word instruction. This example must now find the X register, index it, multiply its contents by 3, and add it to the address in the MQ.

5. The instruction (bits 6-8) is calling for index register 1. Since XØ is pointing to 2ØØØ, then X1 will point to 2ØØ1.

6. Bit 5 says to index the contents of X1. So a MB Increment break is done on location 2ØØ1, changing its contents from 6 to 7.

7. Another break is done to location 2ØØ1 to grab its contents (7). Seven is now multiplied by 3 (floating-point mode) which gives the result of $(25)_8$.

8. Twenty-five is now added to the MQ which contained 1 5000. Now we have the final address of our data at location 1 5025, 1, 5Ø26 and 1 5Ø27. Control would now go from Fetch to Execute to actually pick up the words from core.

7.6 UNDERSTANDING TIMING AND FLOWS

Before one can efficiently correct hardware problems in the FPP, it is necessary to understand two concepts: timing and flow diagrams. If you do not understand the timing, you cannot effectively follow the flows. And if you cannot follow the flows, your effectiveness will be greatly impaired. For a detailed description of the timing and flows, see Chapters 5 and 6. The following paragraphs are implemented to describe timing and flows and to express their importance from a maintenance point of view.

7.6.1 TIMING

All timing is generated on the STG print. However, if the EPM logic is implemented, the TMSC print will contain additional time states. Perhaps the easiest way of conveying the timing picture is to list those items of general significance.

1. There are 16 time states on the STG print.
2. There are 4 time states on the TMSC print (EPM only).
3. For each time state there are 4 mini states, originating in the STG print.
4. These mini states are activated by pin S1 of the AND gate, M133, in slot C2Ø going low. This low is shifted into the 4-Bit shift register at slot EØ9U2 to provide STG MINI STATE 1 L at the next STG 1ØMHZ Clock H pulse. (Really 5 MHZ) The following clock pulse will shift this low into STG MINI STATE 2 L as STG MINI STATE 1 L goes high. STG STATE CHANGE prevents additional lows from being shifted in at pin EØ9U2.

5. A great number of individual inputs can qualify the starting of the mini states as seen at pin R1 of C2Ø. However, the basic signal that causes the mini states is, indirectly, the current time state. This is done in one of four ways:

   a. By completion of a break cycle (DBC1 DONE (1) L)

   b. By the current time state immediately.

   c. By the current time state when the shift counter becomes equal to zero, as in MUL, DIV, ALIGN, ATX, XTA, and normalize, etc. It is important to note that the mini states are not active when doing multiple shifts and when the actual multiply or divide cycles are active. Timing is controlled by the shift counter not equalling zero.

   d. By the IOT that advances timing from a pause instruction. (F PAUSE).

6. At the end of mini state 4 (the last mini state) the current time state is disabled by the flop called STG D (1) H. So all work stops at this point except the logic that determines the next time state.

7. The next time state is always 1 greater than the last time state, except when:

   a. The STG ZERO flop has been set. The STG ZERO flop causes the time state generator to go to time state zero. This always accompanies a major state change with one exception, when the major state is Fetch going to Fetch (Single state instruction).

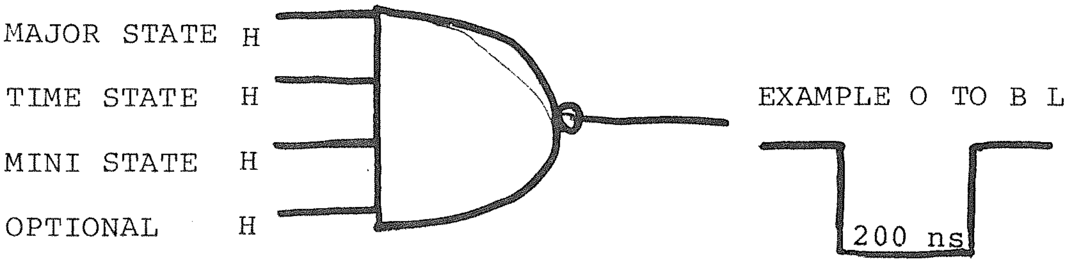
b. The STG STROBE flop has been set. The STG STROBE flop causes the next time state to be more than one state greater than the last. The new time state will be strobed into a holding register (M238 slot C25) and is available at pins P2, R2, V1 and V2. For an example of a time state jump, look at the FEtch flow, time state Ø, for double word instructions (FIR 3=1 + FIR4=Ø) which go to time state four from zero.

8. Note that each mini state is 200ns in duration and the rising edge of the STG 10MHZ clock appears approximately 140-150 ns into a mini state. Also note that the time states and mini states change on the trailing edge of the STG 1ØMHZ clock.

9. To illustrate how timing governs a particular operation, observe the following example.

Example:

Function will be to take the O register to the B register.



The signal EXAMPLE O to B L does two things:

a. Enablesthe O register on the multiplexer that feeds the B register.

b. Provides the enable that clocks the B register.

7.6.2 FLOWS

Without the flow diagrams, it just about impossible to repair a FPP12. This is how important flows will be in the correction of FPP failures. For a detailed description of the flows, see Chapter 6. For a general guide, read the list below.

1. It is not always necessary to know why the FPP does a particular sequence to be able to repair that sequence.

   The most demanding task will be to determine the failure; not necessarily why it is failing.

2. The flow diagrams relate to major states and instructions. The flows indicate step by step, how the FPP12 major statesand instructions are performed.

3. Generally, each operation on any particular flow will be identified by a time state, mini state and the signal causing the decision or action. The print where the logic is located is identified by the prefix of the sign and there may also be notes to clarify certain operations. The above ingredients make the flows an extremely powerful tool for aiding one through the FPP12 logic.

4. Here is an example of a failure which we will try to fix using the flows with the aid of an oscillscope.

   a. First, (Usually the hardest part) we determine from the diagnositc print-out, etc. that the FLDA instructions in floating point mode is failing.

   b. Further interrogation indicates that the LSW of the FAC did not get loaded properly. For additional information we single stepped through the breaks of the instruction to determine if addressing and the data were correct. They were.

   c. Now we have two very important facts which are: 1. The FLDA instruction is failing and; 2. The LSW of the FAC is not being loaded properly during the instruction.

d. We decide to put the diagnostic in a self made scope loop to allow us the best picture possible.

e. Then we look at the FLDA flow (sheet 4) to determine the sequence of the instruction. It is determined from the notes and signal names that the FAC LSW is delt with in time state 2.

f. With an oscilloscope, we then referenced channel 1 on the signal SPI3 XCT3 LDA L which causes the break request. With channel 2, we followed the data from memory to the FAC (MB → ALSW, A → O and finally O → FAC FRAC). It turned out that the gate which supplied RG3 load FAC FRAC LSW L was broken, therefore not loading the FAC LSW.

7.7 DO IT YOURSELF FPP12 PROGRAM

It turns out that many FPP12 problems are not the ambiguous, obscure type. It may be in some cases that just by toggling in a few instructions, you will be able to determine the FPP12's problem without getting bogged down in high-powered diagnostics. This can be done in the following manner.

PDP-8 CODE

```
BEGIN

20/7300   clear AC and link

21/1100   Tad loc 100     /Contains the FPC (Floating point program counter)

22/3501   DCA I 101       /Puts the FPC in the FPC location of the
                           APT table.

23/6553   FPCOM           /Loads the command register with zero and the
                           field bits of the APT with zero.

24/1102   TAD 102         /Load the AC with the starting address of
                           the APT.
```

```
25/6555     FPST          /Start the FPP at the APT address in the AC.

26/7402     HLT           /Should skip - could be I/O bus problems
                           if it halts.

27/6557     FPIST         /Skip when done, read status and clear
                           flag.

30/5027     JUMP-1        /Look for done flag.

31/5020     JUMP BEGIN    /Do program over again.


1ØØ/Ø2ØØ                  /Contains the FPC.

101/0401                  /Contains the address of the FPC in the
                           table.

102/0400                  /Contains the starting address of the APT.


400/ØØØØ                  /Field bits for the operand address,
                           base register, index register location
                           and FPC.

401/Ø2ØØ                  /Lower 12 bits of the FPC.

402/1ØØØ                  /Lower 12 bits of index register Ø location.

4Ø3/7000                  /Lower 12 bits of base register

404/ØØØ-Ø                 /Lower 12 bits of the operand address.

405/ØØØØ                  /Exponent of the FAC.

4Ø6/ØØØØ                  /MSW of the FAC

407/ØØØØ                  /LSW of the FAC


200/                      /First FPP12 instruction

1000/                     /Index register Ø

7000/                     /Base register
```

All you have to do is put the FPP instruction you wish to do in location ∅2∅∅. In location 2∅1 put either an exit instruction, another instruction or a JA instruction (1030, 0200) back to the beginning at location ∅2∅∅.

7.8 BREAK SEQUENCE FOR DATA REFERENCING INSTRUCTIONS

At times it can be very confusing to follow the break sequence to and from memory when debugging programs. To aid the user, the following Table 7-1 is available for the data referencing instructions which will apply only to the data fetching and storing the data after the Fetch major state.

7.9 MAINTENANCE LOGIC

Maintenance logic, built into the FPP12, permits the CPU to examine, in detail, the operation of the FPP12. For instance, the CPU can issue IOTs that force the FPP12 to cease operation after every major time state. Other IOTs permit the CPU to examine internal registers in the FPP12. Using these tools, a diagnostic program can pinpoint the exact step in the flow charts in which the FPP12 fails. This should isolate the failure to within one or two gates. Diagnostic instructions can sometimes be used to debug programs. For instance, there is a maintenance instruction that reads the 12 least significant bits of the APT pointer. If a program has more than one APT it can be desirable to determine which APT is currently in use. This can be done by issuing the maintenance IOT 6565 with the AC clear.

A complete list of maintenance IOTs and their functions follows. Data from the FPP12 is inclusively ORed into the AC when the maintenance mode IOTs are used.

OCTAL CODE	MNEMONIC	FUNCTION
6561	Enter Maintenance Mode or Maintenance Step	a. This IOT is typically issued prior to FPST to begin maintenance mode. b. 6561 is issued when halted at the end of a major time state to cause the advance to the next time state. c. Maintenance mode is cleared whenever the FPP Interrupt Request flag is cleared.
6562	Read States	The current major time state and enable state are ORed into the AC, according to Table 3-3.
6563	Read OMSW	OR the OMSW register into the AC.
6564	Read OLSW	OR the OLSW into the AC.
6565	Read APT	OR the least significant 12 bits of the APT pointer into the AC.
6566	Read MQLSW	OR MQLSW into the AC.
6567	Load Shift Counter	Load the shift counter with the least significant 6 bits of the AC or select the extended-precision mode or read the least significant three words of the O register to the AC according to Table 7-3. Also clears the AC.

TABLE 7-1

BREAK SEQUENCE

See Note	INSTRUCTION	DIRECTION	EXP	MSW	LSW	LSW1	LSW2	LSW3
1	FLDA			1	2			
2	FLDA	Out	1	2	3			
3	FLDA	Out	1	2	6	3	4	5
1	FSTR	In		1	2			
2	FSTR	In	1	2	3			
3	FSTR	In	1	2	3	4	5	6
1	FADD,FSUB	Out		1	2			
2	FADD FSUB	Out	1	2	3			
3	FADD, FSUB	Out	1	2	6	3	4	5
1	FMUL, FDIV	Out		1	2			
2	FMUL, FDIV	Out	3	1	2			
3	FMUL,FDIV	Out	6	1	5	2	3	4
1	FADDM	Out	1	2				
2	FADDM (DEPOSIT)	In	2	1				
	FADDM	Out	1	2	3			
	FADDM (DEPOSIT)	In	3	2	1			
3	FADDM	Out	1	2	6	3	4	5
	FADDM (DEPOSIT)	In	6	5	1	4	3	2
1	FMULM	Out		1	2			
	FMULM (DEPOSIT)	In		2	1			
2	FMULM	Out	3	1	2			
	FMULM (DEPOSIT)	In	3	2	1			
3	FMULM	Out	6	1	5	2	3	4
	FMULM (DEPOSIT)	In	6	5	1	4	3	2

NOTE: 1. Fixed - point 2. Floating Point 3. Extended precision

TABLE 7-2

Definition of AC Bits After IOT 6562 Read States

AC BIT	FUNCTION
00	Most significant bit of major time state counter
01	Bit 1 of major time state counter
02	Bit 2 of major time state counter
03	Bit 3 of major time state counter
04	CRN deposit flop (1) H
05	CNR fetch flop (1) H
06	CRN execute flop (1) H
07	CNR exit flop (1) H
08	CNR initiate flop (1) H
09	CNR process flop (1) H
10	Special st (1) H *
11	TMSC execute *

* These signals are only active when the EPM logic is implemented.

TABLE 7 - 3

Definition of AC Bits Before and After IOT 6567

AC BIT BEFORE	FUNCTION
ØØ	Selects the extended precision mode
Ø1	N/A
Ø2	N/A
Ø3	Reads the OLSW1 to the AC
Ø4	Reads the OLSW2 to the AC
Ø5	Reads the OLSW3 to the AC
Ø6	Set shift counter MSB
Ø7	Set shift counter ↑
Ø8	Set shift counter
Ø9	Set shift counter
1Ø	Set shift counter ↓
11	Set shift counter LSB
AC BIT AFTER	
ØØ	0 register bit 24 or 36 or 48
Ø1	0 register bit 25 or 37 or 49
Ø2	0 register bit 25 or 38 or 50
Ø3	0 register bit 27 or 39 or 51
Ø4	0 register bit 28 or 40 or 52
Ø5	0 register bit 29 or 41 or 53
Ø6	0 register bit 30 or 42 or 54
Ø7	0 register bit 31 or 43 or 55
Ø8	0 register bit 32 or 44 or 56
Ø9	0 register bit 33 or 45 or 57
1Ø	0 register bit 34 or 46 or 58
11	0 register bit 35 or 47 or 59

Maintenance Instructions are detailed on print D-BS-FPP12-0-C12.

CHAPTER 8 - FPP12 INSTALLATION AND ACCEPTANCE

8.1 DESCRIPTION

The Floating Point Processor is a standard PDP-8 type, data break I/O bus peripheral. The FPP12 is attached to positive bus computers with BC08B cables and to negative bus computers with BC08D cables, according to drawing D-IOC-FPP12-0-0 or D-IOC-FPP12-A-Ø. Standard FPP12 logics are wired for PDP-12 computers. Slight wiring alterations for PDP-8/I, PDP-8/L, PDP-8/E, PDP-8, and LINC-8 Computers are normally made as the units are checked out in the factory. The wiring changes for field conversion are shown in Tables 8-1 and 8-2. It is also necessary to exchange nine modules when converting the FPP12 from a positive to negative I/O bus. These modules and their locations are shown in Table 8-3.

Before commencing with the installation, make sure the CPU is up to the proper ECO level.

TABLE 8-1
PDP-8/L, PDP-8/I Positive Bus and PDP-8/E

Name	Run	Add	Delete
CI1 ADD ACC (1) L	DØ4D1 - A11V2		X
	D04D1 - D30N1		X
EXT ENAB INT PAUSE H	F05U2 - B03V2		X
C11 BTS 05 (1) H	D30N1 - A11T2	X	
CI1 BREAK (Ø) H	DØ4D1 - FØ3V2	X	

TABLE 8-2
PDP-8, LINC-8, and PDP-8/I with Negative Bus

Name	Run	Add	Delete
EXT ENAB INT PAUSE H	F05U2 - B03V2		X
C11 IOP 1 H	C01M2 - E01M2		X
	A1ØH1 - C01M2		X
	C01M2 - E01M2	X	
	C01M2 - A10F1	X	
C11 IOP 2 H	C01N2 - E01N2		X
	C01N2 - A10P1		X
	C01N2 - E01N2	X	
	C01N2 - A10N1	X	
C11 IOP 4 H	C01P2 - E01P2		X
	C01P2 - A10S1		X
	C01P2 - E01P2	X	
	C01P2 - A10R1	X	
C11 INIT L	B31P2 - A21D1		X
CI1 INIT H	B31P2 - A21E1	X	
AI1 ADD ACC (1) L	A11V2 - D04D1		X
	D04D1 - D30N1		X
CI1 BTS 05 (1) L	D30N1 - A11S2	X	
CI1 BREAK (Ø) H	DØ4D1 - FØ3V2	X	

TABLE 8-3
Module Changes for Negative Bus Computers

Slot	Positive Bus Modules	Negative Bus Modules
B08	M101	M100
C03	M623	M633
C04	M623	M633
C05	M623	M633
D05	M623	M633
E03	M101	M100
E04	M101	M100
E08	M623	M633
F05	M623	M633

8.2 INSPECTION

After removing the equipment packing material, inspect the equipment and report any damages to the local DEC sales office. Inspection procedures are as follows:

STEP	PROCEDURE
1.	Inspect external surfaces of the cabinet and related equipment for surface damages, etc.
2.	Remove the shipping bolts from the rear door, and internally inspect the cabinet for processor and interconnecting cable damage (if any); inspect for loose or broken modules, blower or fan damage, any loose nuts, bolts, screws, etc.
3.	Inspect the wiring side of the logic panels for bent pins, cut wires, loose external components and foreign material. Remedy any defects found.
4.	Inspect the power supplies for proper seating of

fuses and power connecting plugs.

8.3 CABINET INSTALLATION

The FPP12 cabinet is equipped with roll-around casters and adjustable leveling feet. Cabinet installation procedures are as follows:

STEP	PROCEDURE
1.	With the cabinet in the desired position (See Paragraph 8.6), lower the leveling feet so that the cabinet is supported on the leveling feet, not on the roll-around casters.
2.	Use a spirit level to level all cabinets and be certain that all feet are firmly against the floor.
3.	If necessary, tighten the bolts that secure the cabinets groups together, then recheck cabinet level. Again, make certain that all leveling feet are seated firmly on the floor.

8.4 AC POWER HOOK-UP DESCRIPTION

In the FPP12-A cabinet there is an H854 AC power control (H854B for 5Ø HZ) which supplies the AC to the fans and H740D power supplies. This control is generally operated by remote control. This is accomplished by the turn-on source being connected to terminals 2 and 5 on the "Jones Strip" of the H854 power control. This power control is connected to the AC power with a line cord terminated in a Hubbell 30-amp twist-lock male plug. Follow the procedure below for proper AC installation.

STEP	PROCEDURE
1.	If unit is to be operated remotely, connect terminals 2 and 5 on the H854 to the proper AC source energized with the turn-on of the computer key. This varies with the type of computer used.

STEP	PROCEDURE
2.	Check power supplies, fans and power control for proper 110 or 240 volt wiring. Note that 110-volt source must be found from adjacent cabs or step down transformers to operate the fans on 240-volt systems.
3.	Measure the source AC voltage and insure that the proper voltage is present. Also check AC source terminals (wall outlet) for proper line, neutral and ground relationships. See TABLE 8.4 for the line cord relationships.
4.	Set the primary power circuit breaker (on back of H854) to the OFF position, then plug the FPP12 cabinet primary power line cord into the wall outlet. The red lamp on the power control panel should illuminate indicating primary AC power is applied.
5.	Set the local-remote switch to remote and turn the primary power circuit breaker to the ON position.

TABLE 8-4
POWER LINE CORD IDENTIFICATION

Pigtail Information		Voltage Relationships	Plug
Line	Wire Color		
Green	Frame Ground	Ø Volts	W
White	Netural/Line 2	110V or 240V	X
Black	Line 1	110V or 240V	Y

a. The green wire is the cabinet frame ground and does not carry load current Ø, however, it must be connected for safety reasons. This wire must be securely connected from the FPP12-A cabinet to the grounding point on the primary power source.

b. The white or light gray wire is the netural, common, AC return, or cold lead and must never be used for purposes of grounding.

8.5 DC CONTINUITY CHECK

Before the application of power to the system and FPP logic, a continuity check should be performed at the following check points with an Ohm meter selected on the Rx1 scale.

a. Between A1ØA2 and A1ØC2 - not less than 2 ohms.

b. Between H1ØA2 and H1ØC2 - not less than 2 ohms.

c. Between FØ5B2 and FØ5C2 - not less than 2 ohms.

8.6 CABLING

There is one general rule for cabling and that is to keep the I/O and Data Break bus as short as possible. The I/O bus slots are paralleled as follows:

AØ1 - BØ1
AØ2 - BØ2 } I/O BUS
AØ3 - BØ3

The Data Break bus is wired only to slots:

$$\left.\begin{array}{l}\text{BØ4}\\ \text{BØ5}\end{array}\right\}\text{ DATA BREAK}$$

Slot B1Ø carries the extended memory field bits (EAØ-EA2) and is used only on PDP-8^s and Linc-8's that have no DM01 data multiplexer. This is the equivalent to the PDP-8^s eleventh cable.

8.7 WIRE CHANGE FOR SERIAL MODE

On PDP-12 positive bus systems only, a wire is deleted in the central processor to allow this feature. Delete N16V2-N19T1 (EXT ENAB INT PAUSE H)

8.8 DC POWER CHECK

The system is now ready to be "powered-up". After this has been accomplished check the following points with an oscilloscopes or voltmeter:

a. Between A1ØA2 - A1ØC2 - to read plus 5 volts ±.2 volts.

b. Between H1ØA2 - H1ØC2 - to read plus 5 volts ± .2 volts.

c. Between FØ5B2 - FØ5C2 - to read minus 15 volts ± .5 volts.

8.9 FPP 12 CHECKOUT

Before running the diagnostics, it should be varified that the M4Ø1 oscillator, in slot C15-pin D2 is correctly adjusted at 5MHZ or one clock pulse every 2ØØ nano seconds. There is a potentiometer on the module for this purpose.

The following diagnostics should be run in the sequence shown in accordance with the documentation supplied with each program listing. (Use latest revision of diagnostic.)

TIME IN MINUTES		
1Ø	FPP-12 INSTRUCTION TEST 2A	MAINDEC-12-DØMC
1Ø	FPP-12 INSTRUCTION TEST 2B	MAINDEC-12-DØNB
1Ø	FPP-12 INSTRUCTION TEST 2C	MAINDEC-12-DØOB
1Ø	FPP-12 ADDRESS TEST	MAINDEC-12-DØPC
3Ø	FPP-12 EXERCISER	MAINDEC-12-DØQD
3Ø	TRACE	MAINDEC-12-DØLC
1Ø	FPP-12 INSTRUCTION TEST 3	MAINDEC-12-DØUA
3Ø	TRACE (EPM)	MAINDEC-12-DØTA

If installing the FPP on a PDP-12 system, also run PDP-12 system exerciser, Maindec-12-D7CC or later, for at least 3Ø minutes.

Note that there are two trace programs, however only one is to be run on any particular FPP 12 system. The trace marked (EPM) is run if the extended precision logic is implemented.