PDP8

more than 30,000 installed worldwide

FPP8-A MAINTENANCE MANUAL

digital

EK-FPP8A-MM-001

FPP8-A
maintenance manual

digital equipment corporation • maynard, massachusetts

First Edition, September 1976

Printed in U.S.A.

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

9/76-14

CONTENTS

CONTENTS (Cont)

ILLUSTRATIONS

ILLUSTRATIONS (Cont)

TABLES

TABLES (Cont)

LOCKOUT
1 5776 / 410

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The FPP8-A is a processor that performs arithmetic calculations with floating-point numbers. It is compatible with the FPP12-A instruction set and will run OS8 FORTRAN IV without program modification; with minor program changes, the FPP8-A will run FORTRAN IV at higher speeds.

The FPP8-A consists of two interconnected, hex-size, printed-circuit modules that plug into the Omnibus of a PDP-8/A computer (hereafter, the terms "FPP" and "PDP-8" will be used instead of the full designations). There are no connections from the FPP to external devices, and the FPP derives all of its power from the Omnibus. When the PDP-8 is turned on, the FPP remains inactive until started by IOT instructions issued by the Central Processing Unit (CPU). Once started, the FPP retrieves instructions and operands from the PDP-8 memory by data breaks; many data manipulations and arithmetic calculations are then carried out independently of the CPU and at a higher speed than is possible with CPU timing. The FPP continues to run until halted by an IOT instruction or an FPP instruction, until it encounters numbers that are too large or too small to handle, or until the PDP-8 is halted.

1.2 FPP/CPU INTERACTION

The FPP and the CPU operate in parallel in the data break system. Two modes of parallel operation are possible. In the Interleaved mode, which is automatically entered when power is turned on, the FPP can use a maximum of every other memory cycle; this permits the PDP-8 to run at no less than 50 percent of normal speed. In the Lockout mode, which is selected by an IOT instruction, the FPP can use consecutive memory cycles, as long as no interrupt requests are made by peripheral devices. If such a request is made, the FPP automatically goes to the Interleaved mode; when the interrupt request has been serviced, the FPP returns to the Lockout mode.

1.3 FPP CALCULATING MODES

The FPP can perform calculations in any one of three modes, namely, Floating Point (FP), Double Precision (DP), and Extended Precision (EP). The format of the data used in each of the calculating modes is also unique; both the modes and their respective formats are listed and explained in Table 1-1.

Table 1-1 FPP8-A Calculating Modes and Data Formats

Calculating Mode	Description
Floating Point (FP)	Floating-point calculations are carried out with numbers having a 12-bit, signed, 2's-complement exponent and a 24-bit, signed, 2's-complement fraction. Fraction calculations are made on a 36-bit word, and the result is rounded off to 24 bits at the end of the arithmetic operation. The FPP automatically enters this mode when power is turned on, when either the CAF or FPICL IOT instruction is issued, or when the INIT key is pushed.

Table 1-1 FPP8-A Calculating Modes and Data Formats (Cont)

Calculating Mode	Description
Floating Point (FP) (Cont)	Data used in FP calculations is stored in the PDP-8 memory in this way: The exponent is stored in the memory location pointed to by the FPP instruction; the most-significant word (MSW) of the fraction is stored in the memory location immediately following the exponent; the least-significant word (LSW) of the fraction is stored in the memory location immediately following the MSW.
Double Precision (DP)	Fixed-point calculations are carried out with numbers having a 24-bit, signed, 2's-complement fraction. This mode is the same as the FP mode, except that the exponent is ignored and treated as if it were zero. Data used in DP calculations is stored in the PDP-8 memory in one of two ways, depending on the addressing mode used. For base page (12-bit direct) addressing the MSW of the fraction is stored in the memory location immediately following the location pointed to by the instruction; the LSW is stored in the next consecutive memory location. For other modes of addressing, the MSW is stored in the memory location pointed to by the instruction; the LSW is stored in the next consecutive memory location.
Extended Precision (EP)	Floating-point calculations are carried out with numbers having a 12-bit, signed, 2's-complement exponent and a 60-bit, signed, 2's-complement fraction. Calculations are carried to 60 bits with no round-off. Data used in EP calculations is stored similarly to that of the FP mode; however, three additional locations are needed, with the LSW being stored in the fifth location following the exponent.

1.4 FLOATING POINT CONCEPTS

Various manipulations relating to floating-point arithmetic can be performed by the FPP logic. These are briefly described to familiarize the reader with basic concepts. The following descriptions are based on the FP calculating mode.

1.4.1 Float

When a number is floated, it is converted from its integer form to a fractional floating-point format. To float an integer, one places the number of significant bits of the calculating mode in the exponent (27_8 significant bits plus the sign bit), moves the binary point to reflect the value of the exponent, and then shifts the fraction left until the leading zeros are eliminated, decrementing the exponent with each shift. For example: To float the integer 12_8, write down the whole number

1010.0;

then, write down 27_8 as the exponent

000 000 010 111;

move the binary point to reflect the exponent

0.00 000 000 000   000 000 001 010;

now shift the fraction left until the leading zeros are eliminated, decrementing the exponent with each shift

000 000 000 100   0.10 100 000 000   000 000 000 000.

The floating-point number is $+.101 \times 2^4$, the equivalent of 1010 (12_8).

1.4.2 Fix (or Integerize)

Fixing a number is the reverse process of floating. To fix a number one changes the exponent to 27_8 and then shifts the fraction right a number of times equal to the difference between 27_8 and the original exponent. Thus, to fix the number arrived at by the previous float, which was (in octal notation)

0004 2400 0000;

make the exponent 27_8 and shift the fraction right 23_8 places. The result is

000 000 010 111   0.00 000 000 000   000 000 001 010.

To obtain the integer, move the binary point to reflect the exponent, thereby obtaining

1010.0.

If the exponent is greater than 27_8, the floating-point number is too large to fix; the FPP uses the JAL instruction to check the possibility of fixing fractions.

1.4.3 Normalize

A non-zero floating-point number is normalized by shifting the fraction to the left until non-significant leading zeros are eliminated; each shift is accompanied by a subtraction from the exponent. The number is normalized when the first two bits are different (i.e., 0.1 or 1.0) or when only the first two bits of the fraction are ones (i.e., the number is 6000_8). In DP mode, numbers are not normalized.

1.4.4 Align

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

1.5 REFERENCES

Normalization and alignment are discussed more fully and details concerning floating-point arithmetic are presented in the publication *8/A Series Minicomputer Handbook*, 1976-1977, available from DIGITAL. Other publications that contain instructive information about the FPP and its relationship to the PDP-8 are:

a. FPP8-A Diagnostic, MAINDEC-08-DJFPA
b. FPP8-A Instruction Test and Data Exerciser, MAINDEC-08-DJFPB
c. PDP-8/E, 8/F, and 8/M Maintenance Manual
d. PDP-8/A Miniprocessor User's Manual.

CHAPTER 2
INSTRUCTION SET AND ADDRESSING

2.1 FPP REGISTERS

The FPP logic includes many data registers. Some registers are separate entities, while others occupy space in two high-speed, multiport RAMs that are part of the FPP's Data Path logic. Moreover, some registers are involved only with actual data calculations, while others are also instrumental in setting up and maintaining communication between the FPP and the PDP-8 CPU. These latter registers, which are mentioned frequently in the IOT and FPP instruction lists, are listed and described in Table 2-1.

Table 2-1 FPP Registers

Register	Function	
APTP (Active Parameter Table Pointer)	The 15-bit APTP register in the Data Path logic is loaded with the PDP-8 memory address of the Active Parameter Table (APT) by IOT instructions (FPCOM and FPST). The APT consists of a block of 2, 8, or 11 consecutive memory locations that contain the following information (if the FPCOM instruction has directed a fast start (FS), only locations 1 and 2 are loaded into the FPP hardware; if a normal start is programmed, the information in either the first 8 locations (DP or FP mode) or all 11 locations (EP mode) are obtained by the FPP).	
	Sequence of Memory Locations	**Contents of Location**
	1	**Field Bits** Bits 0-2 = operand address field Bits 3-5 = Base register field Bits 6-8 = Index register address field Bits 9-11 = FPC field
	2	FPC (Floating Program Counter) 12 low-order bits
	3	Index register address – 12 low-order bits
	4	Base register – 12 low-order bits

Table 2-1 FPP Registers (Cont)

Register	Function	
APTP (Cont)	**Sequence of Memory Locations**	**Contents of Location**
	5	Operand address – 12 low-order bits (this word is ignored upon FPP start-up, but is filled upon FPP exit).
	6	FAC exponent
	7	FAC bits 0-11
	8	FAC bits 12-23
	9	FAC bits 24-35*
	10	FAC bits 36-47*
	11	FAC bits 48-59*

*EP mode only

Register	Function		
FPC (Floating Program Counter)	The 15-bit FPC register in the Data Path logic keeps track of the memory location of the FPP instructions. The register is initially loaded from the APT with the address of the MSW of the first instruction to be fetched. Each time an instruction word is fetched, the contents of the FPC are incremented (strictly speaking, only the MSW of a 24-bit instruction is fetched; the LSW is picked up by a memory-read operation).		
Command	The 12-bit Command register in the Control logic is loaded from the PDP-8 Accumulator (AC) register by the FPCOM instruction. The register holds the following information.		
	Bit Position	**Logic Level (1=high)**	**Information**
	0	0	Select FP mode
		1	Select DP mode
	1	0	If exponent underflow occurs, make result of calculation = 0 and continue.
		1	If exponent underflow occurs, exit.
	2	0	Normal addressing

Table 2-1 FPP Registers (Cont)

Register	Bit Position	Logic Level (1=high)	Information
Command (Cont)		1	Forbid access to 4K memory fields other than the one occupied by the last location of the APT. If bits (4:7) =1111 (FS), the field bits will remain equal to the APT field bits when the FPC was obtained. Otherwise, the field bits will remain equal to the APT field when FAC bits 12-23 were obtained. In actual practice the APT is located where it does not cross a field boundary; hence, setting bit 2 forces all FPP operands and instructions to be in the same field as the APT. Attempts to cross field boundaries will then produce wrap-around within the APT field.
	3	0	Disable FPP interrupt.
		1	Enable FPP interrupt.
	(4:7)	=1111	Obtain and restore only the FPC on entry and exit. All other FPP registers retain their previous values. The 9 most-significant field bits of the APT are ignored on entry and cleared upon exit. This mode of operation provides an extremely fast (2 cycle) start and exit, but sacrifices generality.
		≠1111	Obtain entire APT upon startup except for operand address. Restore entire APT upon exit, including operand address.
	8	0	Interleaved operation
		1	Lockout operation
	(9:11)		Most-significant 3 bits of APT pointer.

NOTE

Upon application of power, the APT pointer is undefined.

Table 2-1 FPP Registers (Cont)

Register	Function		
Status	The 12-bit Status register in the Control logic monitors some significant aspects of FPP operation; the contents of the register, which can be transferred to the PDP-8 AC register by the FPRST or FPIST instruction, represent the following information.		
	Bit Position	**Logic Level (1=high)**	**Information**
	0	0	FP or EP mode
		1	DP mode
	1	1	Trap instruction caused exit
	2	1	FPHLT instruction caused exit
	3	1	Attempted divide by zero caused exit. FAC not altered
	4	1	Fraction overflow in DP mode caused exit
	5	1	Exponent overflow caused exit
	6	1	Exponent underflow has occurred. Exit on underflow is optional
	7	1	FADDM or FMULM instruction
	8	0	Interleaved operation
		1	Lockout operation
	9	1	EP mode
	10	1	FPP is currently paused.
	11	1	FPP is currently in run state
Field	This is a 15-bit register located in the Data Path logic that is used only during initialization for temporary storage of the APTP field address. Do not confuse this register with the field bits contained in location 1 of the APT.		

Table 2-1 FPP Registers (Cont)

Register	Function
FAC (Floating Accumulator)	The FAC register has a function similar to the PDP-8 AC register; it can be loaded, stored, and tested, and arithmetic can be performed on its contents (FPP calculations take place in a scratchpad area and the results are stored in the FAC). The FAC occupies space in one of the Data Path random access memories (RAMs) and can comprise 2 (DP mode), 3 (FP mode), or 6 (EP mode) data locations.
OPADD (Operand Address)	The 15-bit OPADD register in the Data Path logic holds the PDP-8 address of the instruction operand. At startup, the register is loaded with the contents of the FPC; thereafter, it is loaded during all data reference and trap instructions. At the conclusion of address decoding of a data reference instruction, OPADD contains the address of bits 12–23 of the operand fraction.
BR (Base)	The 15-bit BR register in the Data Path logic is loaded from the APT during initialization. The address loaded into the register represents the base address, i.e., the origin for relative address calculations, and can be changed at any time with the SETB instruction.
XO (Index)	The 15-bit XO register in the Data Path logic is loaded from the APT during initialization. The address loaded into the register, which may be changed at any time with the SETX instruction, defines the location of the first of eight index registers. These registers may be loaded, retrieved, and used in address calculations and are located in PDP-8 memory.

2.2 FPP8-A IOT INSTRUCTIONS

The PDP-8 IOT instructions relating to the FPP8-A are listed and described in Table 2-2. Table 2-3 lists and describes IOT instructions that are available for maintenance. All IOT instructions require one memory cycle. The FPP8-A uses device codes 55 and 56.

Table 2-2 FPP8-A IOT Instructions

Octal Code	Mnemonic	Description
6551	FPINT	Skip if the FPP8-A flag is set.
6552	FPICL	Produces same results as issuing initialize on the Omnibus. Initialize the FPP – clear flag, enable interleaved operation, stop the FPP, enable FP mode, clear all Status register flags. The APT pointer is not changed.

Table 2-2 FPP8-A IOT Instructions (Cont)

Octal Code	Mnemonic	Description
6553	FPCOM	If the FPP is not in a run state and the flag is not set, the FPP Command register is loaded with the contents of the PDP-8 AC. The AC is not changed by this IOT. If the FPP is in a run state or if the FPP flag is set, the FPCOM instruction is ignored.
6554	FPHLT	Force the FPP to exit, dump its status in the APT, set the forced-exit bit in the Status register, and set the FPP flag at the end of the current instruction. The following special features apply: 1. If FPHLT is issued prior to FPST, the FPP will single-step. FPHLT must be issued after FPIST (or FPICL) and before FPST for each instruction the FPP is to single-step. 2. If the FPP is currently in pause (result of FPAUSE instruction), the FPC will be decremented at exit. 3. If FPHLT and FEXIT occur at virtually the same time (causing a common exit), the status bit indicating forced exit (bit 2) will be cleared.
6555	FPST	If the FPP is not running and the FPP flag is not set, the contents of the AC are loaded into the 12 least-significant bits (LSBs) of the APTP register and the FPP is started. If the FPP is in the run state but paused, the FPST instruction will cause the FPP to continue. If the FPST instruction causes the FPP to start or continue, the next PDP-8 instruction is skipped. Unless the above conditions are met, the FPST instruction has no effect on the FPP and the PDP-8 skip does not occur.
6556	FPRST	Read (jam transfer) the FPP Status register into the PDP-8 AC. The FPRST IOT may be issued at any time.
6557	FPIST	Skip if the FPP flag is set. If the skip occurs, read the FPP Status register into the PDP-8 AC, clear the status bits and the FPP flag.
6567	FPEP	Select EP mode if AC0 = 1 and the FPP is not in the run state. Then clear the AC. This command must be issued after the FPCOM (6553) IOT if EP mode is desired.

Table 2-3 FPP8-A Maintenance IOT Instructions

Octal Code	Mnemonic	Description
6550	FFST	Start maintenance firmware. Forces a jump to μPC address 17. This address, in turn, contains an unconditional jump to μPC address 1700, the actual beginning of the maintenance firmware.
6560		Not used.
6561	FMODE	Enter Maintenance mode. This enabling instruction must be issued to cause the FPP to disable its internal free-running clock, and to cause the FPP to respond to IOT 6565. Maintenance mode is cleared by FPICL, FPIST (if the skip occurs), CAF and the initialize key. Entering Maintenance mode and issuing FPP instructions causes the FPP to function in an internal single-step mode. If the μPC is below 1000, the FPP will execute a data break for every FMDO instruction issued. Because of the way data breaks are synchronized on the Omnibus, this data break occurs after the memory cycle following the IOT, i.e., there is a one-memory-cycle delay between FMDO and the FPP data break. For μPC addresses of 1000 or higher, the FPP executes one microstep for every FMDO IOT. The FPP is clocked at the trailing edge of TP3 of the FMDO IOT. Instructions that require possible modification of index registers will not work properly in maintenance mode (JNX, LDX, ADDX, and indexed addressing).
6562		Not used.
6563	FMRB	Read Data buffer into AC (JAM transfer). This instruction may be executed in either Maintenance or Normal mode, but will result in erroneous information if the μPC is above 1000 and the FPP is in Normal mode.
6564	FMRP	Read μPC into AC (JAM transfer). This instruction may be issued in either mode, but will cause erroneous information to be read into bits 4–11 if the μPC is above 1000 and the FPP is in Normal mode.
6565	FMDO	Execute one step if in Maintenance mode. This instruction is ignored if not in Maintenance mode. See FMODE IOT above.
6566		Not used.
6567	FPEP	See description in Table 2-2.

2.3 FPP8-A OPERATING INSTRUCTIONS

There are two basic classes of floating-point instructions: data reference instructions and special instructions. Data reference instructions perform arithmetic operations on specified data and transfer data between memory and the FAC. Special instructions cause jumps, branches, Index register modifications, pointer moves, manipulations of the FAC, and various housekeeping movements (e.g., alignment and normalization).

2.3.1 Data Reference Instructions and Formats

The 12 data reference instructions are listed in Table 2-4, along with a description of each. The operation carried out by each instruction is noted in the Operation column. For example, the FLDA instruction causes the operand (i.e., the contents, "C," of the effective address, "Y") to be loaded into the FAC. Each of the instructions, except LEA and LEAI, can use any one of three modes to specify the effective address. The format of these modes is illustrated in Figure 2-1. Bits 0, 1, and 2 (which represent the op code) identify the instruction, while bits 3 and 4 identify the mode of addressing. The remaining bits of each instruction determine the operand address, as described by the equations below each format. For example, the operand address for the single word, direct reference format is derived by multiplying bits 5 through 11 by 3 and adding the result to the 7 (or 8, since the product might overflow) LSBs of the base address.

Table 2-4 FPP8-A Data Reference Instructions

Mnemonics	Op Code	Operation	Description
FLDA	0	C(Y)→FAC	The contents of the effective address are loaded into the FAC. If the mode is DP or FP, bits 24–59 of the FAC fraction are not used.
FADD FSUB	1 2	C(Y)+C(FAC)→FAC C(FAC)−C(Y)→FAC	The contents of the effective address are added to or subtracted from the contents of the FAC. In DP mode, no alignment or normalization occurs. The 24 bits from memory are combined with bits 0–23 of the FAC. In FP or EP mode, the two words are aligned by right-shifting the fraction with the lesser exponent until the two exponents are the same. In FP mode bits shifted out of bit 23 are shifted into bits 24–35. Bits shifted out of bit 35 (FP) or bit 59 (EP) are lost. The two fractions are then added or subtracted, using either the 24 MSB (FP) or all 60 bits (EP). The result is normalized. In FP mode the result is then rounded. If either argument is zero, or if its exponent is of such a value that alignment will shift the fraction completely out of its register, no shifting occurs. Under these circumstances, the FAC is either cleared or loaded with the contents of the effective address.
FDIV FMUL	3 4	C(FAC)/C(Y)→FAC C(FAC)*C(Y)→FAC	The old FAC is the multiplier or dividend: the contents of the effective location are the multiplicand or divisor. For multiply, the 36 (FP or DP) or 72 (EP) MSB of the product are computed. For divide, the division is carried to 26 or 61 bits. Lesser bits of product, or the division remainder are lost. In DP mode, the result is rounded to 24 bits. In FP mode, the result is normalized and then rounded to 24 bits. In EP mode, the result is normalized and truncated to 60 bits. For division in FP and EP modes, a preliminary test is made to ensure that the divisor is a normalized number. If the divisor is not normalized, it is first normalized before proceeding with the divide. This operation eliminates the possibility of divide overflow.
FADDM FMULM	5 7	C(Y)+C(FAC)→Y C(FAC)*C(Y)→Y	The calculation described above under FADD or FMUL occurs, except that the FAC is not changed. The result of the computation is stored at the effective address.
FSTA	6	C(FAC)→Y	The contents of the FAC are stored at the effective address. The FAC is not changed.
IMUL IMULI	6 7	C(FAC)*C(Y)→FAC	Available in DP mode only. The contents of the effective address are multiplied by the contents of the FAC, using the rules for integer arithmetic. (The binary point is to the right of bit 23.) The result is loaded into the FAC. A continuous test of overflow is maintained. If overflow occurs, the 24 bits in the FAC are the 24 LSB of the answer, but an unknown number of MSB have been lost.
LEA LEAI	6 7	Y→FAC	Available in FP and EP modes only. The effective address (not its contents) is loaded into bits 9–23 of the FAC. FAC bits 0–8 are cleared. The mode is then changed to DP.

OP CODE			0	1	OFFSET						
0	1	2	3	4	5	6	7	8	9	10	11

Y = BASE ADDRESS + 3* OFFSET

A. SINGLE-WORD, DIRECT REFERENCE

OP CODE			1	0	+	XR			FIELD BITS OF ADDRESS		
0	1	2	3	4	5	6	7	8	9	10	11

12 LSBs OF ADDRESS											
12	13	14	15	16	17	18	19	20	21	22	23

IF XR=0, Y=15-BIT ADDRESS AS GIVEN. THE CONTENTS OF INDEX REGISTER 0 ARE INCREMENTED IF BIT 5 IS LOGIC 1, BUT ARE NOT USED AS PART OF THE ADDRESS CALCULATION.

IF XR≠0, Y=ADDRESS + M*C (XR), WHERE M=2 (DP), 3 (FP), OR 6 (EP). IF BIT 5 IS LOGIC 1, THE CONTENTS OF THE ADDRESSED INDEX REGISTER ARE INCREMENTED BEFORE USE.

B. DOUBLE-WORD, DIRECT REFERENCE

OP CODE			1	1	+	XR			OFFSET		
0	1	2	3	4	5	6	7	8	9	10	11

IF XR=0, Y=BITS 21–35 OF THE TRIPLE WORD LOCATED AT THE BASE ADDRESS + 3* OFFSET. IF BIT 5 IS LOGIC 1, THE CONTENTS OF INDEX REGISTER 0 ARE INCREMENTED, BUT ARE NOT USED AS PART OF THE ADDRESS CALCULATION.

IF XR≠0, Y=BITS 21–35 OF THE TRIPLE WORD LOCATED AT THE BASE ADDRESS + 3* OFFSET + M*C (XR), WHERE M=2, 3, OR 6. IF BIT 5 IS LOGIC 1, THE CONTENTS OF THE ADDRESSED INDEX REGISTER ARE INCREMENTED BEFORE USE.

C. SINGLE-WORD, INDIRECT REFERENCE

Figure 2-1 Data Reference Formats

2.3.1.1 Single Word Direct Reference – The single word, direct reference format is employed when the operand is stored on the base page, which consists of a block of 384 12-bit locations. The origin of the base page is determined by the base address, which can be changed at any time; thus, the base page can encompass a block of locations anywhere in memory. The base address is contained in the BR, which is initially set from the APT and which can be changed with the SETB (Set Base Register) instruction. Data on the base page is stored in the FP format; i.e., the operand consists of three 12-bit words, namely, the 12-bit exponent followed by the 24-bit fraction. Consequently, 128 operands are available on the base page. The relative address of any operand exponent can be specified by multiplying the seven offset bits by 3. When this quantity is added to the base address, the location of the operand exponent is completely identified.

2.3.1.2 Double Word Direct Reference – The double word, direct reference format allows one to specify the 15-bit absolute address of an operand. In addition, this format permits address indexing, which simplifies programming techniques like loop counting and manipulation of push-down stacks.

Address indexing is accomplished by using the contents of an Index register to modify the 15-bit absolute address specified by the data reference instruction. There are eight consecutive 12-bit locations in the PDP-8 memory that are designated as FPP Index registers. The address, XO, of the first of these registers is provided initially by the APT, but can be changed by the special SETX instruction whenever necessary.

Bits 6, 7, and 8 (XR bits) of the double word, direct reference instruction identify Index registers 0 through 7 in octal notation. If Index register 0 is designated, no indexing is to be performed. Instead, the operand absolute address is given by bits 9–23 of the instruction, and the contents of Index register 0 may or may not be incremented. However, if an Index register other than 0 is specified, the 15-bit absolute address is modified by the contents of the selected Index register; note that the contents of the register may or may not be incremented before the operand address is calculated.

2.3.1.3 Single Word Indirect Reference – Indexing is also permitted by the single word, indirect reference instruction. Once again, bits 6, 7, and 8 identify the Index register that will be used in an address modification. However, in this case, the address is specified indirectly, using the base address as the point of reference. As before, bit 5 of the instruction determines if the contents of the Index register are incremented.

2.3.2 Special Instructions and Formats

The FPP special instructions are listed in Table 2-5, along with a description of each function.

Table 2-5 FPP8-A Special Instructions

Mnemonic	OP Code											
LTR	0	1	2	3	4	5	6	7	8	9	10	11
	1	0	1	0	0	0		COND		X	X	X

Function

Load Truth – If the condition is met, +1 (2000 0000 in DP mode) is loaded into the FAC. If the condition is not met, the FAC is cleared.

Conditions:

Octal	Meaning
0	FAC fraction = 0
1	FAC fraction greater than or equal 0
2	FAC fraction less than or equal 0
3	Always
4	FAC fraction not equal 0
5	FAC fraction less than 0
6	FAC fraction greater than 0
7	FAC exponent greater than 27 (octal)

Mnemonic	OP Code											
TRAP 3,	0	1	2	3	4	5	6	7	8	9	10	11
TRAP 4	(3 or 4)			0	0	X	X	X	X		MSB	
	12	13	14	15	16	17	18	19	20	21	22	23
	LSB of Address											

Trapped Instructions – The instruction trap status bit is set, and the FPP exits. The 15-bit address is placed in the APT.

Mnemonic	OP Code											
JNX	0	1	2	3	4	5	6	7	8	9	10	11
	0	1	0	0	0	+		XR			MSB	
	12	13	14	15	16	17	18	19	20	21	22	23
	LSB of Address											

Function

Jump if Index Register is non-zero – The specified Index register is incremented if bit 5 = 1. If the (incremented) Index register is not 0, bits 9–23 are loaded into the FPC, causing a jump.

Mnemonic	OP Code											
JSR	0	1	2	3	4	5	6	7	8	9	10	11
	0	0	1	0	0	1	0	1	1		MSB	
	12	13	14	15	16	17	18	19	20	21	22	23
	LSB of Address											

Function

Jump and Save Return – A 'JA' to the current value of the FPC is constructed and stored in core memory locations BR+1 and BR+2. (1030+FPC field is stored in BR+1; 12 LSB of FPC is stored in BR+2.) Instruction bits 9–23 are then loaded into the FPC. This instruction is one of two ways to call a subroutine. Return from the subroutine is made by either doing a JA to BR+1, or by doing an FLDA base 0 followed by a JAC. The latter method is slightly slower, but much more general.

Table 2-5 FPP8-A Special Instructions (Cont)

Mnemonic	OP Code											
JSA	0	1	2	3	4	5	6	7	8	9	10	11
	0	0	1	0	0	1	0	1	0		MSB	
	12	13	14	15	16	17	18	19	20	21	22	23
					LSB of Address							

Function
Jump and Save at Address – A 'JA' to the current value of the FPC is constructed and stored in core memory at the address specified by bits 9–23 of the instruction. The FPC is then changed to equal 2+bits 9–23 of the instruction. This is the second method for calling a subroutine, and stores the return in two memory locations at the top of the subroutine. Return is accomplished by an unconditional jump to the subroutine entry point.

Mnemonic	OP Code											
SETB	0	1	2	3	4	5	6	7	8	9	10	11
	0	0	1	0	0	1	0	0	1		MSB	
	12	13	14	15	16	17	18	19	20	21	22	23
					LSB of Address							

Function
Set Base Register – Bits 9–23 are loaded into the BR.

Mnemonic	OP Code											
SETX	0	1	2	3	4	5	6	7	8	9	10	11
	0	0	1	0	0	1	0	0	0		MSB	
	12	13	14	15	16	17	18	19	20	21	22	23
					LSB of Address							

Function
Set Index Register Pointer – Bits 9–23 are loaded into X0.

Mnemonic	OP Code											
BRANCH	0	1	2	3	4	5	6	7	8	9	10	11
INSTRUCTIONS	0	0	1	0	0	0		COND			MSB	
	12	13	14	15	16	17	18	19	20	21	22	23
					LSB of Address							

Function
Branch Instructions – If condition is met, bits 9–23 are loaded into the FPC, causing a jump to that address.

Conditions:

	Octal	Meaning
JEQ	0	If FAC fraction = 0
JGE	1	If FAC fraction greater than or equal 0
JLE	2	If FAC fraction less than or equal 0
JA	3	Always
JNE	4	If FAC fraction not equal 0
JLT	5	If FAC fraction less than 0
JGT	6	If FAC fraction greater than 0
JAL	7	If FAC exponent greater than 27 (octal). This condition signifies that the FAC contains a number too large to be fixed in 24 bits.

Table 2-5 FPP8-A Special Instructions (Cont)

Mnemonic	OP Code											
ADDX	0	1	2	3	4	5	6	7	8	9	10	11
	0	0	0	0	0	1	0	0	1	XR		
	12	13	14	15	16	17	18	19	20	21	22	23
	Data											

Function

Add to Index Register – Bits 12–23 are added to the contents of the Index register specified by bits 9–11.

Mnemonic	OP Code											
LDX	0	1	2	3	4	5	6	7	8	9	10	11
	0	0	0	0	0	1	0	0	0	XR		
	12	13	14	15	16	17	18	19	20	21	22	23
	Data											

Function

Load Index Register – Bits 12–23 are loaded into the Index register specified by bits 9–11.

Mnemonic	OP Code											
ALN	0	1	2	3	4	5	6	7	8	9	10	11
	0	0	0	0	0	0	0	0	1	XR		

Function

In FP and EP mode, the fraction 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 of the instruction are zero, the fraction of the FAC is shifted until the FAC exponent equals 27 octal (23 decimal).

In DP mode, an arithmetic shift is performed on the FAC. The number of shifts is equal to the value of the contents of the Index register specified by bits 9–11. The sign of the Index register indicates the direction of shift; a positive sign causes a shift toward the LSB. If the shift is toward the least significant bit, vacated bits are filled with FACO. If the shift is toward the most significant bit, vacated bits are filled with zeros. If bits 9–11 of the instruction are zero, a 23-bit right shift of the FAC is performed.

Mnemonic	OP Code											
ATX	0	1	2	3	4	5	6	7	8	9	10	11
	0	0	0	0	0	0	0	1	0	XR		

Function

FAC to Index Register – If the mode is FP or EP, the contents of the FAC are fixed (i.e., shifted until the exponent = 27 octal) ATX does not test to see if fixing is possible. If the mode is DP, the contents of the FAC are already fixed, so this portion is omitted. Bits 12–23 of the result are then loaded into the Index register specified by bits 9–11. The FAC is not changed by the ATX instruction.

Table 2-5 FPP8-A Special Instructions (Cont)

Mnemonic	OP Code											
XTA	0	1	2	3	4	5	6	7	8	9	10	11
	0	0	0	0	0	0	0	1	1		XR	

Function
Index Register to FAC – The contents of the Index register specified by bits 9–11 are loaded into FAC 12–23. FAC 0–11 is loaded with the contents of FAC 12.

FAC 24–59 are cleared, 27 octal is then loaded into the FAC exponent. If the mode is FP or EP, the FAC is then normalized. (The normalizing operation is omitted in DP mode.)

Mnemonic	OP Code	Function
NOP	004X	No Operation – None, other than a 1-cycle delay in the program. This is the only instruction which will always remain a NOP despite future expansion.
STARTE	005X	Start Extended-Precision Mode – The FPP enters EP mode. If the FPP was previously in a mode other than EP, FAC 24–59 are cleared.
FEXIT	0000	Exit Floating-Point – The contents of the FPP registers are dumped onto the active parameter table, the FPP is stopped, and the FPP flag is set.
FPAUSE	0001	Pause – Suspend FPP operations without updating the APT. IOT FPST will cause the FPP to continue.
FCLA	0002	Clear the FPP Accumulator – Make the FAC fraction zero. If the calculating mode is FP or EP, make the FAC exponent zero, also.
FNEG	0003	Complement the FPP Accumulator – The FAC fraction is replaced by its 2's complement.
FNORM	0004	Normalize – If the FAC fraction is non-zero, and if the FPP mode is FP or EP, the FAC fraction is shifted toward the MSB until the two MSBs are different from each other or until the FAC fraction equals 6000 0000. The FAC exponent is decremented by one for each position shifted. If the FAC fraction is 0, or if the mode is DP, no operation is performed.
STARTF	0005	Enter 24-Bit Floating Point Mode – The FPP enters FP mode. If issued in EP mode, the FAC is rounded to 24 bits.
STARTD	0006	Enter Double-Precision Mode – The FPP enters DP mode. If issued in EP mode, the FAC is chopped to 24 bits. The FAC exponent is ignored, but not modified.
JAC	0007	Jump Per FAC – FAC bits 9–23 are loaded into the FPC.

CHAPTER 3
FPP8-A FIRMWARE

3.1 GENERAL

The FPP8-A is a processor that performs arithmetic operations using floating-point arithmetic. Any logic operations that the FPP may carry out are performed as a secondary role to the primary function of arithmetic calculation. Whatever the operation may be, it is initiated by IOT instructions issued by the PDP-8 CPU. The IOT instructions specify what the FPP is to do, how it is to do it, what it should do when finished, and, most importantly, where it can find the data that it is to work with.

When all the necessary initializing information has been provided, the FPP is started. It begins by fetching its first instruction from PDP-8 memory via the PDP-8 data break system. This instruction could be one that merely directs the FPP to load one of its registers, for example, an operation that can be carried out in a few steps, or it could be one that directs the FPP to perform a division calculation, an operation that requires many steps before a result is obtained. In either case, the FPP logic proceeds through a sequence of actions that depends on the fetched instruction. This sequence is programmed by an internal (to the FPP) read-only memory (ROM) called the Control ROM.

Every location in the Control ROM contains information that causes a particular operation to occur in the FPP logic; instructions are carried out by stringing together these locations in a specific sequence. The starting point for the sequence is always determined during initialization by the IOT instructions, which cause the address of the starting Control ROM location to be loaded into a Micro Program Counter (μPC). The μPC then accesses that location, and the information stored therein causes some FPP operation to take place. The stored information includes the address of the next Control ROM location in the sequence. This next address is loaded into the μPC and new information, including the next address in the sequence, is accessed and acted on. Thus, the sequence is self-perpetuating. When all the steps required by the instruction being performed have been completed, the FPP makes an exit test; that is, should it end the sequence or continue it? If the exit test directs that the sequence end, the FPP stops with a specific address in the μPC and raises its interrupt request flag. The sequence can be restarted only by an IOT instruction. On the other hand, if the exit test does not call for a halt, the FPP continues to access locations, fetching the next FPP instruction from the PDP-8 memory and following a sequence determined by that instruction.

The sequence of FPP operations can be characterized by firmware, that body of information that is neither software nor hardware, but which provides firm, formalized descriptions of the FPP operations. This firmware consists of a general flow diagram, a Control ROM pattern specification, and a Control ROM source code. All of these firmware examples will be discussed in detail in the following sections.

3.2 FLOW DIAGRAM

Figure 3-1 shows a flow diagram that illustrates the general sequence of Control ROM locations that occurs for each FPP instruction. A sequence begins when address 0020 is loaded into the μPC register, i.e., when an instruction is fetched. Fetch can occur in one of three ways: If the FPP has just been started by the FPCOM and FPST instructions, the APT contents will be loaded into the appropriate registers, and the first FPP instruction will be fetched from the memory location specified by the FPC register contents; if the FPP is paused in the RUN state, the FPST instruction will cause it to continue by fetching a new instruction; or, if an instruction has just been concluded, the fetch can result after an exit test is made.

When the instruction has been fetched it is applied to decoding circuits in the Control logic. μPC address 0023 causes a characteristic address to be generated; this address is then loaded into the μPC and the logic operations peculiar to the decoded instruction are started. For example, if the special instruction SETX is decoded, μPC address 0023 is replaced by 0034. A sequence of steps, beginning with 0034, is carried out, and when the SETX operations have concluded an exit test is made. If an exit is not directed, a new instruction is fetched and decoded, and address 0023 dispatches a new address to the μPC.

If a Data Reference instruction is fetched, an address calculation must be performed to determine the PDP-8 memory address of the data. Instruction Dispatch 1 causes the appropriate Control ROM location to be accessed. When the operand memory address has been calculated, Instruction Dispatch 2 transfers control to a data-handling routine. For example, if an FLDA instruction is decoded, control is transferred to Control ROM location 0200, which begins an operation that loads the data into the FAC register. However, if the Data Reference instruction is one that requires an arithmetic calculation, as opposed to one that merely transfers data (like FLDA or LEA), control passes to one of the two preliminary arithmetic routines, GETN and GETARG. Then, the final instruction dispatch is performed and the μPC is loaded with the first address of the appropriate arithmetic routine.

All instruction operations conclude with an exit test. If an exit has been directed by some logic condition, the APT is stored in memory. Then, the μPC is loaded with address 0001, the interrupt request flag is raised, and the FPP halts until serviced by the PDP-8.

3.3 CONTROL ROM PATTERN SPECIFICATION AND SOURCE CODE

The Control ROM is comprised of 31 1024-bit PROMs that are arranged to provide a total of 1400 (768_{10}) 44-bit word locations. Each 44-bit word consists of a number of individual control words, ranging in length from 1 bit to 10 bits. These control words determine how the FPP logic is manipulated to carry out the operations specified by the FPP instructions.

A complete pattern specification for the Control ROM is contained in the FPP8-A print set. This specification lists each μPC address ("μPC Address" and "Control ROM location" are synonomous), the names of the individual control signals, and the state of all signals for each address. Figure 3-2 represents a portion of the specification for information. The functions of each Control ROM address are listed in the Control ROM source code, which is also included in the print set. A portion of this document is reproduced in Figure 3-3. This example describes the Data Path and Control operations for the μPC addresses shown in Figure 3-2. (The pattern specification and the source code are hereinafter referred to, jointly, as "the firmware.")

One can solve the intricacies of the FPP logic by applying the information given in the pattern specification and the source code to the logic diagrams. However, one cannot make use of the source code without first understanding its somewhat complex language syntax. This syntax is described fully in Appendix A; study it before continuing with this description.

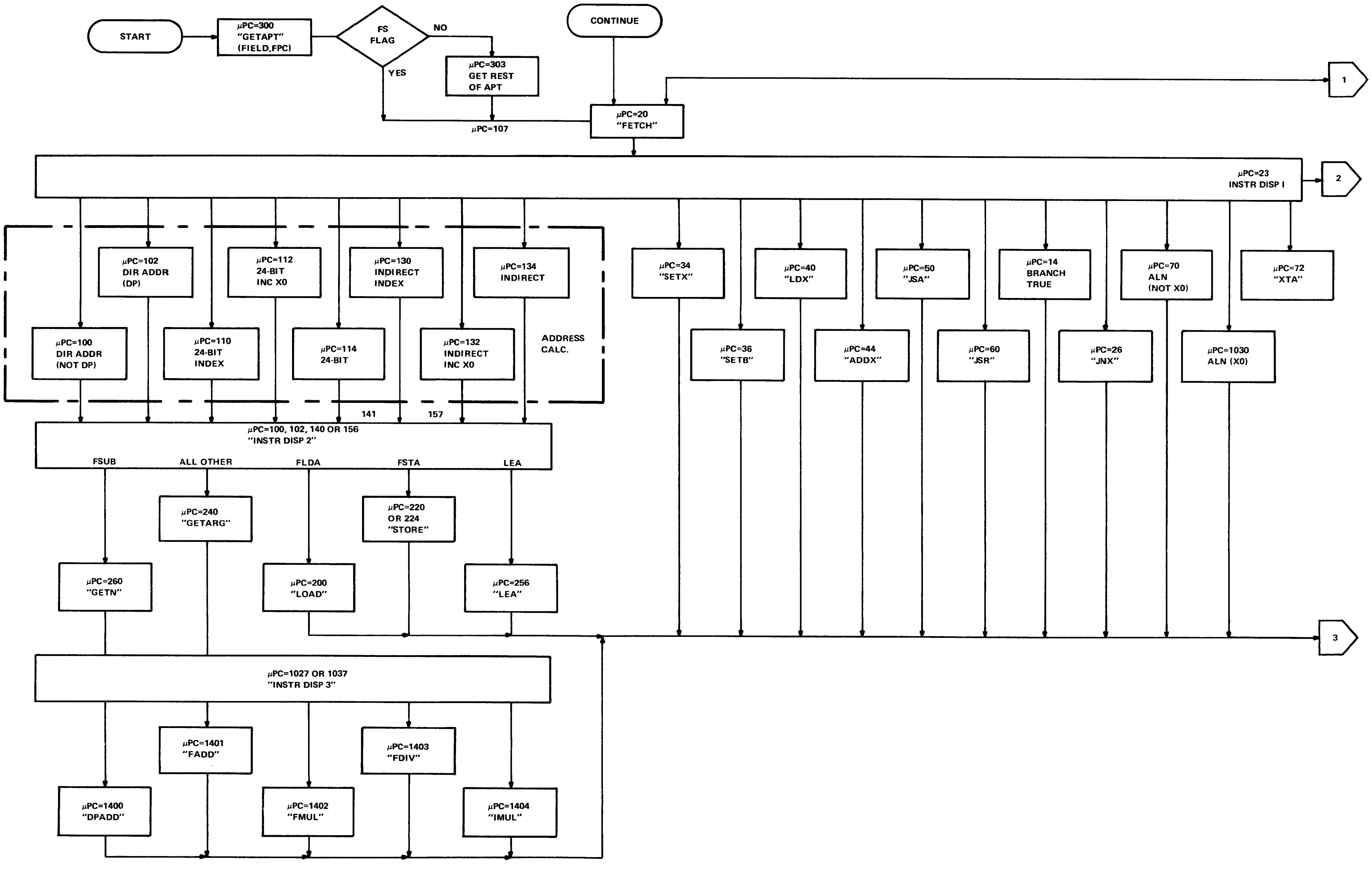


Figure 3-1 FPP8-A Instruction Flow Diagram (Sheet 1 of 2)

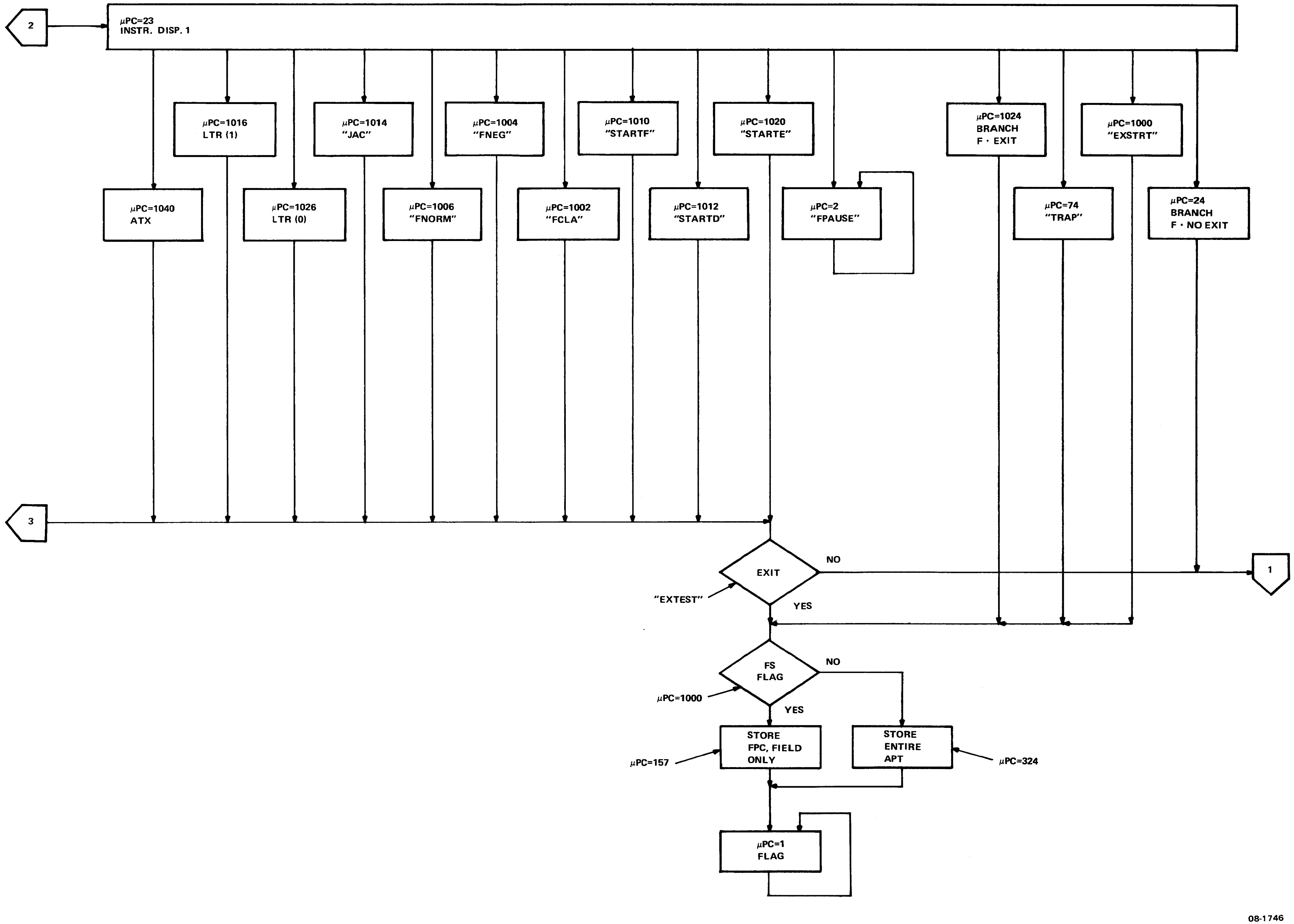


Figure 3-1 FPP8-A Instruction Flow Diagram (Sheet 2 of 2)

UPC ADDR	AAAA	BBBB	BCCC	CCDD	DEFH	JJKL	MMMM	MNPP	PPPP	PPPP	RRRR
0,	HHHH	XXXX	XHHH	HHXX	XHHH	HHHH	HHHH	LHHH	HHHH	HHLL	HHHH
1,	HHHH	XXXX	XHHH	HHXX	XHHH	HHHH	HHHH	LHHH	HHHH	HHHL	HHHH
2,	HHHH	XXXX	XHHH	HHXX	XHHH	HHHH	HHHH	LHHH	HHHH	HHLH	HHHH
3,	HHHH	HHLL	HHHH	HHHH	HHHL	LLHL	HHHH	LHHH	HHHH	HHHH	HLLL
6,	HHHH	XXXX	XHHH	HHXX	XHHH	HHHH	HHHH	LHHH	HHHL	HHHH	HLLL
7,	HHLH	HHLL	HLLH	HLXX	XHLH	HHHH	HHHH	LHHH	HHHH	HHHH	HLLL
13,	HHHH	HHLH	HLLH	HHXX	XHLH	HHHH	HHHH	LHHH	LLHH	HHHH	HLLL
16,	HHHH	HHHH	HHHL	HHXX	XHLH	HHHL	HHHH	LHLH	HHHH	HHHH	HLLL
17,	HHHH	XXXX	XHHH	HHXX	XHHH	HHHH	HHHH	LHLL	LLHH	HHHH	HLLL
4,	HHHH	HHHH	HHHH	HLXX	XHLH	HHHL	HHHH	HHHH	HHHH	HHHH	HLLL
5,	HHHH	XXXX	XHHH	HHXX	XHHH	LLHH	HHLL	LHHH	HHHH	HHHH	HLHH
10,	HHHH	HHHL	LHHH	HLXX	XHLH	HHHL	HHHH	HHHH	HHHH	HHHH	LLLH
11,	HHHH	XXXX	XHHH	HHXX	XHHH	HHHH	HHHH	HHHH	HHHH	HHHH	HLLL
12,	HHHH	XXXX	XHHH	HHXX	XHHH	LLHH	HHLL	LHHH	HHHH	HHHH	HLHH

H=HIGH, L=LOW, X="DON'T CARE"
(X IS ENCODED IN ROM AS L)

COLUMN SIGNALS DRIVEN (TO J1)

- A — CB5 ARITH 0 H – CB5 ARITH 3 H (INVERTED)
- B — CB5 ALOC 0 L – CB5 ALOC 4 L
- C — CB5 BRLOC 0 L – CB5 BRLOC 4 L
- D — CB5 BWLOC 0 L – CB5 BWLOC 2 L
- E — CB5 CARRY BIT L
- F — CB5 WRITE A H (INVERTED)
- H — CB5 WRITE B H (INVERTED)
- J — CB6 DB CTRL 1 L – CB6 DB CTRL 2 L
- K — CB6 EXTEND H (INVERTED)
- L — CB6 READ A H (INVERTED)

COLUMN SIGNALS DRIVEN (USED INTERNALLY)

- M — CB6 UPCTRL 0 L – CB6 UPCTRL 4 L
- N — CB6 B SEL L
- P — CB6 UP2 IN L – CB6 UP11 IN L
- R — (USED ON DWG. CB4 TO CONTROL PULSE GATING AND TO GENERATE CB4 BRK RQ H

Figure 3-2 Example, Control ROM Pattern Specification

```
        /ADDRS     NEXT DATA PATH OPERATION      TIME    CTRL FUNCTION

        *0
0       HALTED,    NO OPERATION                  TS3     GO TO, HLTD1 (3)
1       FLAG,      NO OPERATION                  TS3     GO TO, FLAG (1)
2       PAUSED,    NO OPERATION                  TS3     GO TO, PAUSED (2)

        *3
        /"DB:=MD" IN THE NEXT LINE IS A KLUDGE.  THE DB IS REALLY LOADED
        /FROM THE DATA LINES OF THE OMNIBUS (THIS STEP ONLY)
3       HLTD1,     DB:=MD; TEMP:=FIELD           T4      GO TO, HALTED (0)

        /////////////////////// IOT AREA ///////////////////////////
        /FPST AND CONTINUE CONDITION

        *6
6       FCONT,     NO OPERATION                  T4      GO TO, FETCH (20)

        /FPCOM
        *7
7       FPCOM,     FIELD:=[R3R] DB               T4      GO TO, HALTED (0)

        /FPST AND START CONDITION
        *13
13      FPST,      APTP:=TEMP(1:3), DB           T4      GO TO, GETAPT (300)

        /FPHLT IOT GIVEN WHILE FPP IS PAUSED.  BACK UP FPC, EXIT.

        *16
16      FPHLT,     FPC:=FPC[+]M1                 T4      GO TO, EXSTRT (1000)

        /JUMP TO MAINTENANCE PROGRAM
        *17
17      MAINT,     NO OPERATION                  T4      GO TO, MAINT1 (1700)

        /////////////////////// DATA BREAK AREA //////////////////////

        /SUBROUTINE – GET SECOND HALF OF 24-BIT INSTRUCTION
        *4
4       INST24,    FPC:=FPC[+]K1                 T4      BKCMD:=0
5                  DB:=MD                        BT1     RETURN

        /SUBROUTINE – GET WORD AT NEXT OPADD, BUMP OPADD

        *10
10      NEXTOP,    BKMA, OPADD:=OPADD[+]K1       T3
11      NXTOP1,    NO OPERATION                  T4      BKCMD:=0
12                 DB:=MD                        BT1     RETURN
```

Figure 3-3 Example, Control ROM Source Code

3.3.1 'GETAPT' Firmware

The FPP flow diagram, illustrated in Figure 3-1, begins at START and then passes to a block entitled GETAPT. This block represents the transfer of the APT from PDP-8 memory to the appropriate FPP registers. The firmware describing this transfer operation will be explained in detail so as to provide some insight not only to the firmware but also to the FPP logic. The firmware for the GETAPT routine is shown in Figure 3-4 (this is not the entire APT firmware; only the part dealing with a Fast Start is shown). Certain Data Path pattern specification signals are called out in the figure; the significance of these call-outs will be explained.

The GETAPT routine is started when address 300 is clocked into the μPC register (Paragraph 4.2.7, Clock Logic, describes the initialization procedure that precedes GETAPT). Signals asserted by the Control ROM during address 300 both direct the Data Path to read the contents of the APTP register and send them to the BKMA register, and cause the break control logic in the Data Path to begin data break operations. The Control ROM signals (except those pertaining to the data break control logic) are loaded into Pipe-line registers in the Data Path at T3 time, and the μPC address is changed to 301. The Pipe-line registers and a number of PROMs driven by the register outputs, produce signals that gate the APTP register contents to the BKMA/BKEMA register (these contents are the address of the first word of the APT, which contains field address information). Meanwhile, the data break control logic is both carrying out a priority test to determine if an FPP data break can be started, and preparing to assert Omnibus data break signals that can cause a memory read to take place (the control statement 'BKCMD:=4' means that signals BKCMD (0:2) L are asserted in a combination that will produce a memory-read data break transfer). At T4 time the μPC address is changed to 302 and, if the FPP has priority, which we assume it has, the APT pointer address is loaded into the BKMA/BKEMA register and the Omnibus data break control signals are asserted. Thus, a PDP-8 data break takes place with a memory read at the address specified by the contents of the APTP register. Note that a one-step delay has occurred between the Data Path statement and its fulfillment – an important point to understand and remember. In the rest of the discussion, this delay will be implied by describing the Data Path statement in this manner: the APTP is sent to the BKMA. When reading such a statement, keep in mind the fact that the destination, the BKMA in this example, is not loaded until one step later in the firmware.

The Control ROM is now in location 302, during which the MD is sent to the DB. The Control tests its Command register to see if a Fast Start (FS) has been programmed. We assume that it has; hence, Control loads address 317 into the μPC register at BT1 time. Meanwhile, the information in the addressed memory location has been read from memory and strobed onto the MD lines; the information settles on the bus between BT1 time and T2 time. Consequently, the information read from memory during the data break is loaded into the DB at T2 time, while the Control is obtaining the contents of μPC address 317.

During address 317, the DB is sent to TEMA, a 15-bit register in the Data Path's A-file (TEMA is now temporarily holding field address information). At T2 time, Control jumps to the μPC address of the subroutine APT1. Location 321 adds 1 (K1) to the contents of the APTP register and sends the incremented address back to the APTP, to TEMP1, a B-file register, and to the BKMA/BKEMA. This address is the memory address of the second word of the APT. This second word contains the low-order address bits of the first FPP instruction to be fetched, and will be loaded into the Data Path FPC register. The field address of the first FPP instruction was located in bits 9–11 of the first word in the APT. Hence, this field address is now located in TEMA.

During μPC address 322, the TEMA register is rotated right three places and sent to the TEMP register and back to TEMA; the field address of the first FPP instruction is now located in bits (1:3) of both TEMA and TEMP. The data break operations cause word two of the APT to be gated onto the MD lines (BKCMD:=4). The MD is sent to the DB during address 323, and TEMP1 is sent to the OPADD register (the latter operation has significance only if the entire APT is being picked up and the operating mode is EP).

```
        /GET ACTIVE PARAMETER TABLE
        *300
300     GETAPT,   BKMA:=APTP                           T3
301               NO OPERATION                         T4    BKCMD:=4
302               DB:=MD                               BT1   IF FS, APT2 (317)

        /FAST START – FS=1.  GET FPC ONLY, THEN GO TO FETCH
317     APT2,     TEMA:=DB                             T2    SUB, APT1 (321)

321     APT1,     BKMA, TEMP1, APTP:=APTP[+]K1         T3
322     APT1B,    TEMP, TEMA:=[R3R]TEMA                T4    BKCMD:=4
323               DB:=MD; OPADD:=TEMP1                 BT1   RETURN

320               TEMP7, FPC:=TEMP(1:3), DB            T2    GO TO, FETCH2 (107)

107     FETCH2,   BKMA, OPADD:=TEMP7                   T3    GO TO, FETCH1 (21)

21      FETCH1,   :=FACE[EXPSIZE]M30                   T4    BKCMD:=7
22                FPC:=FPC[+]K1; DB:=MD                BT1
23                TEMP:=FIR(9:11)                      T2    INSTR DISP 1
```

UPC ADDR	AAAA	BBBB	BCCC	CCDD	DEFH	JJKL	MMMM	MNPP	PPPP	PPPP	RRRR
300,	HHHH	HHLH	HHHH	HHXX	XHHH	HHHL	HHHH	HHHH	HHHH	HHHH	LLLH
301,	HHHH	XXXX	XHHH	HHXX	XHHH	HHHH	HHHH	HHHH	HHHH	HLHH	HLLL
302,	HHHH	XXXX	XHHH	HHXX	XHHH	LLHH	HLHL	LHHH	LLHH	LLLL	HLHH
317,	HHHH	HHLH	LLLH	HLXX	XHLH	HHHH	HHLH	LHHH	LLHL	HHHL	HLHL
321,	HHHH	HHLH	HHHH	HLHH	LHLL	HHHL	HHHH	HHHH	HHHH	HHHH	LLLH
322,	HHLH	HHLH	LHHH	HHHH	HHLL	HHHL	HHHH	HHHH	HHHH	HLHH	HLLL
323,	HHHH	HHHL	LLHH	HLXX	XHLH	LLHH	HHLL	LHHH	HHHH	HHHH	HLHH
320,	HHHH	HHHH	HLLH	HHLL	LHLL	HHHH	HHHH	LHHH	HLHH	HLLL	HLHL
107,	HHHH	HHHL	LLHL	LLXX	XHLH	HHHH	HHHH	LHHH	HHHL	HHHL	LLLH
21,	LHLH	HLHH	HHLL	LLXX	XHHH	HHHL	HHHH	HHHH	HHHH	HLLL	HLLL
22,	HHHH	HHHH	HHHH	HLXX	XHLH	LLHL	HHHH	HHHH	HHHH	HHHH	HLHH
23,	HHHH	XXXX	XLLH	LHHH	HHHL	HHHH	HHHL	HHHH	HHHH	HHHH	HLHL

ARITH (0:3) H | ALOC (0:4) L | BRLOC (0:4) L | BWLOC (0:2) L | WRITE A L | WRITE B L | READ A L | BKCMD (0:2) L

Figure 3-4 'GETAPT' Firmware

At BT1 time Control returns to address 320. TEMP bits (1:3) (the field address) and the DB contents (the low-order address) are sent to the FPC register and to TEMP7. Now the complete memory address of the first FPP instruction is in the FPC register (as well as in TEMP7), and Control jumps to μPC address 107, to begin the fetch of the first FPP instruction. During address 107 the memory address of the instruction is sent to the BKMA and data break operations begin.

The Data Path operations during μPC address 21 are concerned with a test that can be made on floating-point numbers by the JAL instruction. These operations are described in detail in Paragraph 4.2.8, Instruction Dispatch Logic. Meanwhile, the Data Break control logic is preparing to assert break control signals (BKCMD:=7 and BKCMD:=4 are identical with respect to data break control signals), which it does at T4 time (we again assume that the FPP has priority). The FPP instruction in the addressed memory location is placed on the MD lines and sent to the MB as well as to the instruction decoding logic. During μPC address 22, the address in the FPC register is incremented. The register now contains the address of either the next instruction to be fetched or the operand that is to be retrieved.

If a Fast Start had not been programmed, the entire APT would have been picked up just as were the first two locations. For example, location 3 of the APT contains the base address 12 LSBs. The field bits of the base address are located in bit positions 9, 10, and 11 of TEMA after TEMA is rotated during the pick-up of the FPC field bits. Thus, another rotation of TEMA permits the base address field bits to be sent to TEMP (1:3). Then, the field bits in TEMP (1:3) and the 12 LSBs in the DB can be sent to the BR register.

During an exit from FPP operation, the APT is updated by an exit routine. If a Fast Start was carried out at the beginning of operations, a fast exit (FASTX) is effected, during which the FPC and its field bits are stored in locations 1 and 2 of the APT.

The pattern specification signals that are called out in Figure 3-4 manipulate the Data Path. For example, during μPC address 300, the APTP register must be read. This register is located in the A-file, which is controlled by Control ROM signals ALOC (0:4) L. The state of these signals during address 300 is HHLHH. If we refer to Table 4-1, we can see that the APTP is indeed selected for reading (READ A L is also asserted during address 300). Table 4-1, along with Tables 4-2, 4-3, and 4-4, is discussed in relation to the FPP block diagram description. If one reads the block diagram description and then returns to Figure 3-4 and attempts to relate the Control ROM signals to the Data Path operations, one can acquire some understanding of how the FPP works.

For another example, consider μPC address 321, during which the APTP is incremented. Constants are located in the Constant generator, which is controlled by the BRLOC (0:4) L signals. Table 4-2 shows that the constant 1 is applied to the B input of the ALU during address 321. Table 4-1 shows that the APTP register is applied to the A input of the ALU. The ARITH signals determine what happens to the two ALU inputs. The pattern specification indicates the state of the Control ROM signals at the output of the ROM. Some of these signals, including the ARITH signals, are inverted before being applied to the Data Path; hence, the levels indicated in Figure 3-4 should be inverted. That is, the ARITH bits for μPC address 321 are shown as HHHH in Figure 3-4. Invert these to LLLL and you have the correct condition at the interconnecting cable. Thus, Table 4-4 indicates that the A and B inputs of the ALU (plus a carry in, if such is applicable) are added and the result is placed on the OBUS. The addition result is sent back to the APTP and to TEMP1. Table 4-1 shows that the APTP will be written (WRITE A L is asserted), while Table 4-3 shows that TEMP1 will also be written.

One can also see something of what is happening in the Control by examining Figure 3-4. During address 302, for example, the Control tests the Command register to see if a Fast Start was programmed. If it was, a jump address is loaded into the μPC register. This address is provided by the Control ROM, itself, and is represented by the 'P' bits in address 302, viz., H HLL HHL LLL (0317).

3.3.2 FPADD Firmware

Other examples that illustrate the relation of firmware and logic appear in this and the following two sections. These examples describe the operations involved in arithmetic calculations performed by the FPP.

Figure 3-5 lists the firmware pertaining to an addition carried out in the FP mode. The FADD instruction has been fetched and decoded, and the effective address of the operand has been determined (it is assumed that at some earlier time the FAC has been loaded with one of the numbers in question). The firmware in Figure 3-5 begins at Instruction Dispatch 2. The second number, the operand, will be retrieved from memory and placed in TEMP registers. Then, both the operand and the number in the FAC are tested for zero fractions (if either number has a zero fraction, the addition is shortened considerably). Following this test, the logic determines which exponent is smaller and checks to ensure that the difference in exponents is not so large as to make alignment impossible (if alignment is impossible, the smaller number is treated as though it were zero). Alignment is carried out by right-shifting the fraction of the number having the smaller exponent. Then, the two fractions are added, and the result is normalized and rounded off. Finally, the result is placed in the FAC and an exit test is made. The steps detailed by the firmware will be explained by a running commentary; this commentary lists the μPC addresses appearing in the firmware and, where necessary, amplifies the Data Path and Control statements relating to each address. The commentary follows.

```
        /GET ARGUMENT, PLACE FRACTION IN TEMP1-TEMP5, AND EXPONENT
        /(IF USED) IN TEMP6.  TEMP1, BKMA ALREADY CONTAIN ADDRESS OF
        /ARGUMENT AT ENTRY.  USED BY FADD, FADDM, FMUL, FMULM, IMUL AND FDIV.
        *240
240     GETARG, OPADD:=TEMP1                          T4      BKCMD:=0
241             DB:=MD; TEMP3:=[0]                    BT1     IF DP, GET1 (243)
242             TEMP6:=DB                             T2      SUB, NEXTOP (10)

        /SUBROUTINE--GET WORD AT NEXT OPADD, BUMP OPADD
        *10
10      NEXTOP, BKMA, OPADD:=OPADD[+]K1               T3
11      NXTOP1, NO OPERATION                          T4      BKCMD:=0
12              DB:=MD                                BT1     RETURN

243     GET1,   TEMP1:=DB                             T2      SUB, NEXTOP (10)
244             TEMP2:=DB                             T2      IF NOT EP, ARITH (1037)

        /ARITHMETIC DISPATCH
1037    ARITH,  NO OPERATION                          FREE    INSTR DISP 3

1401    FADD,   DB, TEMP7:=FACE                       FREE    GO TO, FPLUS (1422)

        ////////FLOATING POINT ADD///////////
        /FIRST TEST FOR ZERO ARGUMENT.
1422    FPLUS,  DB, SC:=TEMP7                         FREE*   IF TEMPZERO, FADD11 (1453)
1423            DB, SCRATCHE:=TEMP7                   FREE*   SUB, FTOS (1333)

        /SUBROUTINE--MOVE FAC FRACTION TO SCRATCH.
1333    FTOS,   DB, TEMP:=FACM                        FREE*
1334    FTOS1,  DB, SCRATCHM:=TEMP                    FREE*   IF NOT EP, FTOS3 (1346)

1346    FTOS3,  DB, SCRATCHP:=[0]                     FREE*   GO TO, FTOS2 (1344)

1344    FTOS2,  DB, TEMP:=FACN                        FREE*
1345            DB, SCRATCHN:=TEMP                    FREE*   RETURN

1424            DB, TEMP7, SC:=SC[MINUS]TEMP6         FREE*   IF FACZERO, FADD0 (1455)
        /NOW FIND SMALLER EXPONENT.  TEST FOR OVERSHIFT.
1425            NO OPERATION                          FREE*
1426            NO OPERATION                          FREE*   IF OVFLO, FADD1 (1457)
1427            NO OPERATION                          FREE*   IF EXPFL, FPLUS1 (1462)
1430            DB, SC:=[MINUS]TEMP7                  FREE*
1431            DB:=SC[MINUS]M30                      FREE*   IF NOT EP, FPLUS2 (1433)

1433    FPLUS2, NO OPERATION                          FREE*
1434            NO OPERATION                          FREE*   IF SGN, FADD1A (1460)
1435            DB, TEMP6:=FACE                       FREE*   SUB, EST (1314)

        /SUBROUTINE--EXCHANGE SCRATCH AND TEMP FRACTIONS.
1314    EST,    DB, TEMP:=SCRATCHM                    FREE*
1315            DB, SCRATCHM:=TEMP1                   FREE*
1316            DB, TEMP1:=TEMP                       FREE*   IF NOT EP, EST1 (1330)

1330    EST1,   DB, TEMP:=SCRATCHN                    FREE*
1331            DB, SCRATCHN:=TEMP2                   FREE*
1332            DB, TEMP2:=TEMP                       FREE*   RETURN
```

Figure 3-5 FPADD Firmware (Sheet 1 of 2)

```
        /ALIGN NUMBERS.  SMALLER NUMBER IS IN SCRATCH; SC CONTAINS EXP DIFF.
        /DIFFERENCE IN EXPONENTS IS SMALL ENOUGH THAT A NON-ZERO
        /NUMBER WILL BE IN SCRATCH AFTER THE SHIFT.
1436    FADD4,  NO OPERATION                            FREE*   SUB, SHR (1260)

        /SUBROUTINE--SHIFT SCRATCH RIGHT PER SC.  USE WORD MOVE IF POSSIBLE.
        /SC CONTAINS 2'S COMPLEMENT OF NUMBER OF SHIFTS ON ENTRY, 0 AT EXIT
1260    SHR,    DB, SC:=SC                              FREE*
1261    SHR1B,  DB, SC:=SC[12BIT]K14                    FREE*
1262            NO OPERATION                            FREE*   IF EXPFL, SHR1A (1264)

1264    SHR1A,  NO OPERATION                            FREE*   IF EXPFL, RWM (1300)

1300    RWM,    DB, TEMP:=SCRATCHN                      FREE*   IF NOT EP, RWM1 (1310)

1310    RWM1,   DB, SCRATCHP:=TEMP                      FREE*
1311            DB, TEMP:=SCRATCHM                      FREE*
1312            DB, SCRATCHN:=TEMP                      FREE*
1313            DB, SCRATCHM:=[SIGN]SCRATCHM            FREE*   GO TO, SHR1B (1261)

1261    SHR1B,  DB, SC:=SC[12BIT]K14                    FREE*
1262            NO OPERATION                            FREE*   IF EXPFL, SHR1A (1264)

1264    SHR1A,  NO OPERATION                            FREE*   IF EXPFL, RWM (1300)
1265            DB, SC:=SC[12BIT]M14                    FREE*
1266    SHR1,   DB, SC:=SC[12BIT]K1                     FREE*
1267            DB, SCRATCHM:=[SHR]SCRATCHM             FREE*
1270            DB, SCRATCHN:=[SHR][EXT]SCRATCHN        FREE*   IF EP, SHR2 (1273)
1271            DB, SCRATCHP:=[SHR][EXT]SCRATCHP        FREE*   IF EXPFL, SHR1 (1266)
1272            NO OPERATION                            FREE*   RETURN

1437    FA,     DB, SCRATCHS:=SCRATCHS[12BIT]TEMP5      FREE*   IF NOT EP, FB (1467)

        /START FP ADD.
1467    FB,     DB, SCRATCHN:=SCRATCHN[12BIT]TEMP2      FREE*   GO TO, FADD7 (1443)

1443    FADD7,  DB, SCRATCHM:=SCRATCHM[12BIT][EXT]TEMP1 FREE*
1444            DB, SC:=TEMP6                           FREE*
1445            NO OPERATION                            FREE*   IF OVFLO, FADD2 (1470)
        /NORMALIZE RESULT.
1446    FADD8,  NO OPERATION                            FREE*   SUB, NMI (1177)


        /SUBROUTINE--NORMALIZE SCRATCH.  DECREMENT SC ONCE FOR EACH SHIFT.
        /USE WORD MOVE, WHEN POSSIBLE, TO SAVE TIME.
        /ROUND OFF IF NOT IN EP MODE.  DB IS LOADED AT FIRST FIVE STEPS FOR
        /BETTER VISIBILITY OF UN-NORMALIZED ANSWER.

1177    NMI,    DB:=SCRATCHM                            FREE*   IF DP, RND (1240)
1200            DB:=SCRATCHN                            FREE*   IF TEMPZERO, RND (1240)
1201    NMI1,   DB:=SCRATCHP                            FREE*   IF MOVE OK, NMI4 (1215)

1215    NMI4,   DB, SC:=SC[12BIT]M14                    FREE*
1216            DB, TEMP:=SCRATCHN                      FREE*
1217            DB, SCRATCHM:=TEMP                      FREE*   TEST OVFLO
1220            DB, TEMP:=SCRATCHP                      FREE*
1221            DB, SCRATCHN:=TEMP                      FREE*   IF NOT EP, NMI5 (1231)

1231    NMI5,   DB, SCRATCHP:=[0]                       FREE*   GO TO, NMI1 (1201)

1201    NMI1,   DB:=SCRATCHP                            FREE*   IF MOVE OK, NMI4 (1215)
1202            DB:=SCRATCHR                            FREE*   IF NORMED, NMI6 (1237)
1203            DB:=SCRATCHS                            FREE*   IF NOT EP, NMI3 (1232)

1232    NMI3,   DB, SCRATCHP:=[SHL]SCRATCHP             FREE*   IF NORMED, NMI3A (1236)
1233            DB, SCRATCHN:=[SHL][EXT]SCRATCHN        FREE*   TEST OVFLO
1234            DB, SCRATCHM:=[SHL][EXT]SCRATCHM        FREE*
1235            DB, SC:=SC[12BIT]M1                     FREE*   GO TO, NMI3 (1232)

1232    NMI3,   DB, SCRATCHP:=[SHL]SCRATCHP             FREE*   IF NORMED, NMI3A (1236)

1236    NMI3A,  DB, SCRATCHP:=[SHR][EXT]SCRATCHP        FREE*   TEST OVFLO
1237    NMI6,   NO OPERATION                            FREE*   IF FORBIDDEN, NMI8 (1250)
1240    RND,    NO OPERATION                            FREE*   IF EP, NMI7 (1247)
1241            NO OPERATION                            FREE*   IF TEMPSGN, RND1 (1254)
1242            DB:=SCRATCHP[12BIT]K3777+1              FREE*
1243    RND2,   DB, SCRATCHN:=SCRATCHN[12BIT][EXT]      FREE*
1244            DB, SCRATCHM:=SCRATCHM[12BIT][EXT]      FREE*   IF TEMPZERO, RND4 (1255)
1245            DB, SCRATCHP:=[0]                       FREE*   IF DP, NMI7 (1247)
1246            NO OPERATION                            FREE*   IF FORBIDDEN, OVREC (1405)
1247    NMI7,   NO OPERATION                            FREE*   RETURN

1447    FADD9,  DB, TEMP7:=SC                           FREE*   IF NZSET, FADD10 (1451)

        /STORE IN EITHER MEMORY OR FAC, DEPENDING ON OP CODE.
1451    FADD10, DB, SCRATCHE:=TEMP7                     FREE*   IF MEM, DEPOS (353)
1452            DB, FACE:=TEMP7                         FREE*   GO TO, STOF (1347)

        ///////////////////////MOVE SCRATCH TO FAC AND EXIT//////////////
1347    STOF,   DB, TEMP:=SCRATCHM                      FREE*
1350    STOF1,  DB, FACM:=TEMP                          FREE*   IF NOT EP, STOF2 (1357)

1357    STOF2,  DB, TEMP:=SCRATCHN                      FREE*   IF TEMPZERO, STOF3 (1361)
1360            DB, FACN:=TEMP                          FREE*   EXTEST

        /FLOATING-POINT INSTRUCTION FETCH

        *20
20      FETCH,  BKMA:=FPC                               T3
```

Figure 3-5 FPADD Firmware (Sheet 2 of 2)

μPC Address	Comment
240	The address of the operand exponent is sent to the File A OPADD register; the address is placed on the Omnibus MA lines and a Read data break is started.
241	The operand exponent is sent to the DB register; the File B TEMP3 register is cleared in preparation for fraction alignment.
242	The operand exponent is sent to TEMP6; control is transferred to the NEXTOP subroutine.
10	The address in the OPADD register is incremented and sent to both the BKMA register and the OPADD register. A Read data break is started.
11	The address of the 12 most-significant bits (MSBs) of the operand fraction is placed on the MA lines.
12	The MSW of the operand fraction is sent to the DB; control is returned to GET1.
243	The MSW of the fraction is sent to TEMP1, which is examined to determine if the MSW is zero; this information is sent to the Register Flags logic for subsequent testing. Control is transferred to NEXTOP.
10, 11, 12	The OPADD register is bumped; the 12 LSBs of the operand fraction (the LSW) are sent to the DB; control is returned to address 244.
244	The LSW of the fraction is sent to TEMP2, which is examined to determine if the LSW is zero; this information is sent to the Register Flags logic for subsequent testing. If both the MSW and the LSW are zero, the Register Flags logic asserts the TEMP ZERO H signal at clock time of address 1037. Control is transferred to ARITH, address 1037 and, in turn, to FADD, address 1401; free-running clock timing begins; register flags reflect the state of TEMP1 through TEMP5. At this point the operand is stored in the TEMP registers thusly: Exponent stored in TEMP6 Fraction MSW stored in TEMP1 Fraction LSW stored in TEMP2
1401	The exponent of the number in the FAC is sent to TEMP7 and to the DB (the DB is loaded merely for visibility during single-stepping; since this is the case throughout the remaining firmware, the DB will be ignored in the rest of the commentary). Control is transferred to 1422.
1422	The FAC exponent is sent to the SC register; the μPC Gating Control logic tests the TEMP ZERO flag, which reflects the state of TEMP1 and TEMP2, i.e., the state of the operand fraction. Remember that there is a one-step delay in the fulfillment of a Data Path statement. Similarly, registers in the Register Flags logic are loaded after a 1-step delay. Consequently, register flags can be tested no earlier than 2 steps after the Data Path statement that involves the register.

µPC Address	Comments
1422 (Cont)	This is why address 1422 is testing TEMP registers, rather than SCRATCH registers as the asterisk in the Timing statement would seem to indicate. If the TEMP ZERO H signal is high, indicating a zero fraction in the operand, the answer is simply the number in the FAC. In the example being considered, TEMP ZERO H is not asserted (the Register Flags logic is discussed in Paragraph 4.2.6).
1423	The FAC exponent is sent to SCRATCHE; control is transferred to the FTOS subroutine.
1333, 1334	The MSW of the FAC fraction is sent to SCRATCHM; control passes to 1346.
1346	SCRATCHP is zeroed in preparation for fraction alignment; control jumps to 1344.
1344, 1345	The LSW of the FAC fraction is sent to SCRATCHN; control returns to 1424. At this point the number originally loaded into the FAC (referred to hereafter as "the FAC number," or "the FAC exponent," etc...) is stored in the SCRATCH registers thusly: Exponent stored in SCRATCHE (and SC) Fraction MSW stored in SCRATCHM Fraction LSW stored in SCRATCHN
1424	The FAC ZERO flag is tested. When the FLDA instruction loaded the FAC prior to issuance of the FADD instruction, the state of FACM and FACN was recorded in the Register Flags logic; this information is retained until new data is written into the FAC. If the FAC ZERO L signal is asserted, indicating a zero fraction in the FAC, the answer is simply the number in the SCRATCH. In this example, FAC ZERO L is not asserted. At this point, numbers are assigned to the operand and the FAC so as to more easily illustrate the procedures that follow. Hence, the following normalized octal numbers are assigned:

	Exponent	MSW	MSW
Operand	0003	5001	0003
FAC	0020	2010	2111

Now, the operand exponent is subtracted from the FAC exponent (2's complement subtract); the result is sent to the SC register and to TEMP7.

SC (FACE)	000 000 010 000
TEMP6 (2's complement of OPE)	111 111 111 101
Exponent Difference (FAC is larger)	000 000 001 101

μPC Address	Comments
1425	The difference in the exponents, 15_8, is loaded into both SC and TEMP7. The state of the SC sign bit is loaded into the Register Flags logic; the positive sign bit causes the EXPFL H signal (which will be tested two steps hence) to be negated.
1426	If the two numbers being added were grossly different, i.e., if one had a large positive exponent and the other had a large negative exponent, the subtraction in 1424 could produce an overflow (the OVFLO H signal would be asserted). In that case, the very small number is discarded, the remaining number is normalized, if necessary, and stored in the FAC. The exponents in this example do not produce an overflow.
1427	The negated EXPFL H signal indicates that the FAC exponent is larger than the operand exponent. This means that, because all shifting operations are carried out in the SCRATCH, the operand fraction, which must be shifted during alignment, is to be transferred to the SCRATCH. If the operand had been larger, EXPFL H would have been asserted by the Register Flags logic, indicating that the smaller number was already in the SCRATCH.
1430	The 2's complement of the difference in the exponents is sent to the SC to control the shifting operation during alignment.
1431	The difference in the exponents is tested for overshift. Overshift is the condition that exists when the fraction of the smaller number must be right-shifted more than 24 times to achieve alignment. If the fraction is shifted exactly 24 times, the MSB of the fraction ends up in the MSB of the guard-bit word (SCRATCHP for an FP addition or subtraction) and can affect the result when round-off is carried out. However, more than 24 shifts produces a situation wherein the fraction of the smaller number has no effect at all on the calculation result. Consequently, the smaller number is discarded when an overshift condition exists (overshift exists in the EP mode when the fraction must be shifted more than 59 times). If the overshift condition were present, the subtraction in this step would cause SIGN H to be asserted by Register Flags (the signal would be asserted when the subtraction result is placed on the OBUS). When this signal is tested 2 steps hence, control would jump to 1460. There, steps would be taken to normalize the fraction of the larger exponent (FACE). In the present example the subtraction produces the following: SC (2's complement of exponent difference) 111 111 111 011 2's complement of M30 000 000 011 000 —————— 000 000 001 011
1434	SIGN H is tested; control passes to 1435.
1435	FACE is sent to TEMP6. Control is transferred to subroutine EST.

<table>
<tr><th>μPC Address</th><th>Comments</th></tr>
<tr><td>1314–
1332</td><td>The smaller fraction is the operand. It must be right-shifted. However, the operand fraction is in TEMP1 and TEMP2 and must be placed in the SCRATCH in order to be shifted. This subroutine swaps TEMP1/2 and SCRATCHM/N.</td></tr>
<tr><td>1436</td><td>Control jumps to 1260.</td></tr>
<tr><td>1260</td><td>The SC (containing the 2's complement of the exponent difference) is sent to the SC, so as to test the sign bit. This test, which occurs two steps hence, checks to see if the SC is zero, indicating no difference between the exponents. If there is any difference in the two exponents, the 2's complement of the difference must have logic 1 in the MSB. Only no difference can cause the EXPFL H signal to be low. If this were the case, control would return to 1437 for the FPADD operation.</td></tr>
<tr><td>1261</td><td>14_8 (12_{10}) is added to the SC. Two steps from now the EXPFL flag will be tested. If the EXPFL H signal is then high, indicating more than 12 shifts are to be made, control will jump to 1300 (word move). If the exponent difference is exactly 12, 12 separate shifts are carried out. (Had there been zero difference in the exponents, the 14_8 added in this step would be subtracted in step 1263 – not shown – before control returned to 1437.)</td></tr>
<tr><td>1262</td><td>There is a difference in the exponents, so EXPFL H is high; go to 1264.</td></tr>
<tr><td>1264</td><td>More than 12 shifts must be made, so EXPFL H is high; go to 1300 for word move.</td></tr>
<tr><td>1300–
1312</td><td>SCRATCHN is sent to SCRATCHP, and SCRATCHM is sent to SCRATCHN.</td></tr>
<tr><td>1313</td><td>The sign bit of the fraction is examined. If the sign is 0, the ALU output is 0000_8; if the sign is 1, the ALU output is 7777_8. The SCRATCH contents before and after word move are:
<table>
<tr><th></th><th>SCRATCHM</th><th>SCRATCHN</th><th>SCRATCHP</th></tr>
<tr><td>Before 1300:</td><td>101 000 000 001</td><td>000 000 000 011</td><td>000 000 000 000</td></tr>
<tr><td>After 1313:</td><td>111 111 111 111</td><td>101 000 000 001</td><td>000 000 000 011</td></tr>
</table>
The SC contains 7777_1.</td></tr>
<tr><td>1261</td><td>14_8 is again added to the SC, producing 0013_8; this is a check to see if another word move can be made (this would be possible only in the EP mode).</td></tr>
<tr><td>1262</td><td>EXPFL H is still high from the previous 1264 operation.</td></tr>
<tr><td>1264</td><td>The addition in 1261 causes EXPFL H to go low.</td></tr>
<tr><td>1265</td><td>The 14_8 added to the SC in 1261 to check for a possible word move must be subtracted; thus, 7777_8 is returned to the SC.</td></tr>
</table>

μPC Address	Comments
1266	One is added to the SC, producing 0000_8.
1267	SCRATCHM is shifted right once. The Shift logic causes the MSB of SCRATCHM to be returned to the same position (i.e., the sign bit is retained), while loading the LSB into the SLINK flip-flop.
1270	SCRATCHN is shifted right once. The asserted EXTEND H signal causes the content of the SLINK flip-flop to be shifted into the MSB of SCRATCHN; the LSB of SCRATCHN is loaded into SLINK.
1271	SCRATCHP is shifted right once. The asserted EXTEND H signal causes the content of the SLINK flip-flop to be shifted into the MSB of SCRATCHP; the LSB of SCRATCHP is loaded into SLINK. The addition of step 1266 causes EXPFL H to go low, indicating that the required number of shifts has been carried out. Control returns to 1437 with the SCRATCH and TEMP1/2 containing the following numbers: **SCRATCHM** 111 111 111 111 **SCRATCHN** 110 100 000 000 **SCRATCHP** 100 000 000 001 **TEMP1** 010 000 001 000 **TEMP2** 010 001 001 001
1437	Go to 1467 (the Data Path operation is of significance only in EP mode).

1467	Add SCRATCHN and TEMP2.
	
	  110 100 000 000
	  010 001 001 001
	1 000 101 001 001
	
	The carry from this addition is loaded into the CLINK flip-flop in the Shift logic.

1443	Add SCRATCHM and TEMP1. The asserted EXTEND H signal causes the content of the CLINK flip-flop to be applied as a carry-in.
	
	  111 111 111 111
	  010 000 001 000
	  1
	1 010 000 001 000

1444	The exponent of the answer is sent to the SC.

μPC Address	Comments
1445	If the addition produced an overflow, control jumps to an overflow recovery subroutine. In this example, overflow does not occur and the result is a normalized number, viz: **SCRATCHM** **SCRATCHN** **SCRATCHP** 010 000 001 000 000 101 001 001 100 000 000 001 Control passes to the normalization subroutine.

The numbers chosen to illustrate the FP addition produced a normalized result. The normalizing subroutine can be more fully described if we assume that the addition produced an unnormalized answer. Therefore, suppose that the number in the SCRATCH after the operations in 1467 and 1443 is:

SCRATCHM	SCRATCHN	SCRATCHP
000 000 000 000	001 100 100 001	101 001 111 000

The FAC exponent in the SC is still 0020.

1200	If the results of the 12-bit additions carried out in 1467 and 1443 were zero, the TEMP ZERO H signal would have been asserted by the Register Flags logic. Zero is considered to be a normalized number; thus, control would pass to the round-off process. This is not the case, so continue.
1201	Each of the 13 MSBs of the SCRATCH is zero; hence, the Register Flags logic asserted the MOVE OK H signal when SCRATCHM and SCRATCHN were loaded (the same signal would be asserted if each of the 13 MSBs was one). This means that a word move can be carried out. Checking 13 bits, rather than only the 12 MSBs, ensures that the fraction sign bit remains unchanged after the word move. Go to 1215 for the start of the word move.
1215	Since the fraction is to be shifted left 12 places (14_8), the value of the exponent must be reduced by 12_{10}. If the exponent is a large negative value, subtracting 12 from it could result in a number too small to be represented in 12 bits, i.e., an underflow could result. Such an event must be recorded; thus, an overflow test is made in 1217. In the present example no such problem arises, as shown in the subtraction. SC (FACE) 000 000 010 000 M14 111 111 110 100 000 000 000 100

μPC Address	Comments
1216–1231	The word move is carried out, leaving this result: **SCRATCHM** **SCRATCHN** **SCRATCHP** 001 100 100 001 101 001 111 000 000 000 000 000 Go to 1201.
1201	Another word move is not possible. This step is significant only in the EP mode.
1202	If the two MSBs in SCRATCHM were different (the definition of a normalized number), the Register Flags logic would have asserted the NORMED H signal. Thus, the normalization process would have been completed. This is not the case, so proceed.
1203	Go to 1232.
1232	SCRATCHP is shifted left once (the firmware does not know that SCRATCHP was zeroed in 1231; one can arrive here without going to 1231). The Shift logic causes a zero to be shifted into bit 15 of SCRATCHP; the MSB of SCRATCHP is loaded into the SLINK register. The test for NORMED H is inapplicable at this time.
1233	SCRATCHN is shifted left once. Because the EXTEND H signal is asserted, the content of SLINK is shifted into bit 15 of SCRATCHN, while bit 4 of SCRATCHN is loaded into SLINK. The overflow test is inapplicable at this time.
1234	SCRATCHM is shifted left once. EXTEND H is asserted; therefore, bit 4 of SCRATCHN, which was in SLINK, is shifted into bit 15 of SCRATCHM. Bit 4 of SCRATCHM is loaded into SLINK, but this has no significance. The SCRATCH is now in this form: **SCRATCHM** **SCRATCHN** **SCRATCHP** 011 001 000 011 010 011 110 000 000 000 000 000
1235	The value of the exponent is reduced by one to reflect the left shift just completed. Since this subtraction could cause an exponent underflow, an overflow test will be made two steps hence. The new exponent value, 0003, is sent to the SC. Go to 1232.
1232	SCRATCHP is shifted left once. SCRATCHM is examined and found to be normalized (NORMED H is now asserted). Control goes to 1236.

μPC Address	Comments
1236	SCRATCHP is shifted right to restore its content to that which existed before 1232. The subtraction of 1235 is tested for overflow, which has not occurred.
1237	SCRATCHM and SCRATCHN are examined to determine if the normalized result in 1234 is 4000 0000, which it is not. If such a result was obtained, the Register Flags logic would have asserted the FORBIDDEN H signal. The result would then be converted to 6000 0000 by shifting SCRATCHM right once; the exponent would be increased by one to reflect the shift.
1240	Continue
1241	SCRATCHM is examined to determine the sign of the fraction. Because the sign is positive, TEMP SGN H is not asserted and control passes to 1242.
1242	The first step in the round-off process is taken, i.e., the number 4000 is added to SCRATCHP. If the fractional value of SCRATCHP is 1/2, or more, the addition produces a carry, which is propagated to SCRATCHN (step 1243) and, perhaps, to SCRATCHM (step 1244). (In this example there is no carry.) If the fraction before round-off had been negative, the TEMP SGN H signal would have been asserted and the test in 1241 would have caused control to jump to 1254. This step adds 3777 to SCRATCHP. If the fractional value of SCRATCHP is greater than 1/2, the addition produces a carry, which, again, is propagated to SCRATCHN and perhaps to SCRATCHM; this causes the negative fraction to be rounded down, as opposed to the rounding-up of a positive fraction.
1243	Propagate the carry from 1242, if appropriate.
1244	Propagate the carry from 1243, if appropriate. TEMP ZERO test is inapplicable here.
1245	Zero SCRATCHP.
1246	If the normalized result, before round-off, had been 3777 7777, and step 1242 has produced a carry, the round-off process would have generated 4000 0000. Hence, FORBIDDEN H would have been asserted, and the test in this step would transfer control to 1405. There, 4000 0000 would be converted to 2000 0000 and the exponent would be increased by one. Continue.
1447	The exponent of the answer is sent to TEMP7. The NZ SET H signal is tested. When an exponent underflow occurs, the action taken by the FPP at the end of the calculation is programmed by the FPCOM instruction. The FPP can be directed to exit or to continue after setting the calculation result to zero. If the NZ SET H signal is low, a non-trapped underflow has occurred; control passes to 1450, which causes the calculation result to be set to zero and stored in the FAC. However, if the NZ SET H signal is high, either there was no underflow, as in this example, or a trap of the underflow was directed by the FPCOM instruction. Control jumps to 1451.

μPC Address	Comments
1451	The exponent of the answer is sent to SCRATCHE. If the TO MEM H signal is high, indicating the result is to be transferred to memory (used by FMULM and FADDM), control jumps to 353. The answer in this example is to be placed in the FAC, so control passes to 1452.
1452	The exponent of the answer is sent to FACE. Control jumps to 1347.
1347–1360	The fraction is sent to FACM and FACN; an exit test is made. Since neither underflow nor overflow has occurred, control jumps to 20, and a new instruction is fetched.

3.3.3 FPMUL FIRMWARE

Figure 3-6 lists the firmware pertaining to a multiply operation carried out in the FP mode. The FMUL instruction has been fetched and decoded, and the effective address of the operand has been determined (it is assumed that at some earlier time the FAC has been loaded with one of the numbers in question). The firmware in Figure 3-6 begins at Instruction Dispatch 2. The second number, the operand, will be retrieved from memory and placed in TEMP registers. Then, both numbers are tested for zero fractions (if either has a zero fraction, a zero result is stored in the FAC). After this test, the multiplication begins. When the result has been obtained, it is normalized, rounded off, and placed in the FAC. Finally, the exit test is made. The steps detailed by the firmware are explained in the following commentary. Operations that have already been described in the FPADD firmware, e.g., the pickup of the operand fraction, are considered only to whatever extent they differ.

```
240     GETARG, OPADD:=TEMP1                           T4      BKCMD:=0
241             DB:=MD; TEMP3:=[0]                     BT1     IF DP, GET1 (243)
242             TEMP6:=DB                              T2      SUB, NEXTOP (10)

10      NEXTOP, BKMA, OPADD:=OPADD[+]K1                T3
11      NXTOP1, NO OPERATION                           T4      BKCMD:=0
12              DB:=MD                                 BT1     RETURN

243     GET1,   TEMP1:=DB                              T2      SUB, NEXTOP (10)

10      NEXTOP, BKMA, OPADD:=OPADD[+]K1                T3
11      NXTOP1, NO OPERATION                           T4      BKCMD:=0
12              DB:=MD                                 BT1     RETURN

244             TEMP2:=DB                              T2      IF NOT EP, ARITH (1037)

        /ARITHMETIC DISPATCH
1037    ARITH,  NO OPERATION                           FREE    INSTR DISP 3

1402    FMUL,   DB, TEMP7:=FACE                        FREE*   GO TO, FTIMES (1472)

        //////////FLOATING AND FIXED POINT FRACTIONAL MULTIPLY//////////
        /MULTIPLY IS DIRECT MULT OF SIGNED 2'S COMPLEMENT NUMBERS, WITH
        /A CORRECTION FOR NEGATIVE MULTIPLIER.  ENTER WITH TEMP FLAGS
        /SET, CHECK FOR ZERO FACTOR.  EXTEND SIGN OF TEMP1 INTO TEMP.
1472    FTIMES, DB, TEMA:=TEMP1                        FREE*   IF TEMPZERO, FADD9A (1450)
1473            DB, TEMP:=[SIGN]TEMA                   FREE*   IF FACZERO, FADD9A (1450)
1474            DB, SC:=TEMP7                          FREE*   SUB, CLRS (1111)

        /SUBROUTINE--CLEAR SCRATCH FRACTION.  ALL MODES
1111    CLRS,   DB, SCRATCHM:=[0]                      FREE*
1112    CLRS2,  DB, SCRATCHN:=[0]                      FREE*   IF NOT EP, CLRS1 (1116)

1116    CLRS1,  DB, SCRATCHP:=[0]                      FREE*   RETURN

        /MULTIPLY FRACTIONS
1475            NO OPERATION                           FREE*   IF NOT EP, FMUL4 (1532)

1532    FMUL4,  DB:=FACN                               FREE*   PRESET BIT COUNT
1533            SCRATCHP:=SCRATCHP[MDS]TEMP2           FREE*   CSUB, MUL3A (1556)

1556    MUL3A,  SCRATCHN:=SCRATCHN[MDS][EXT]TEMP1      FREE*
1557            SCRATCHM:=SCRATCHM[MDS][EXT]TEMP       FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2           FREE*   CSUB, MUL3A (1556)

1556    MUL3A,  SCRATCHN:=SCRATCHN[MDS][EXT]TEMP1      FREE*
1557            SCRATCHM:=SCRATCHM[MDS][EXT]TEMP       FREE*   RETURN
```

Figure 3-6 FPMUL Firmware (Sheet 1 of 3)

```
1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1533            SCRATCHP:=SCRATCHP[MDS]TEMP2                   FREE*   CSUB, MUL3A (1556)

1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1534            DB, TEMP7:=SCRATCHN                            FREE*   SUB, R2MA (1125)

1125   R2MA,    DB, SCRATCHR:=TEMP7                            FREE*
1126            DB, TEMP7:=SCRATCHM                            FREE*
1127   R2M,     DB, SCRATCHP:=TEMP7                            FREE*   PRESET BIT COUNT
1130            DB, SCRATCHM:=SCRATCHM[SHR] [EXT]              FREE*
1131            DB, TEMP7, SCRATCHM:=[SIGN]SCRATCHM            FREE*
1132            DB, SCRATCHN:=TEMP7                            FREE*   RETURN

1535            DB:=FACM                                       FREE*
1536            SCRATCHR:=SCRATCHR[MDS]                        FREE*   CSUB, MUL4A (1555)

1555   MUL4A,   SCRATCHP:=SCRATCHP[MDS] [EXT]TEMP2             FREE*
1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1536            SCRATCHR:=SCRATCHR[MDS]                        FREE*   CSUB, MUL4A (1555)

1555   MUL4A,   SCRATCHP:=SCRATCHP[MDS] [EXT]TEMP2             FREE*
1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1536            SCRATCHR:=SCRATCHR[MDS]                        FREE*   CSUB, MUL4A (1555)

1555   MUL4A,   SCRATCHP:=SCRATCHP[MDS] [EXT]TEMP2             FREE*
1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1536            SCRATCHR:=SCRATCHR[MDS]                        FREE*   CSUB, MUL4A (1555)

1555   MUL4A,   SCRATCHP:=SCRATCHP[MDS] [EXT]TEMP2             FREE*
1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1536            SCRATCHR:=SCRATCHR[MDS]                        FREE*   CSUB, MUL4A (1555)

1555   MUL4A,   SCRATCHP:=SCRATCHP[MDS] [EXT]TEMP2             FREE*
1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN

1536            SCRATCHR:=SCRATCHR[MDS]                        FREE*   CSUB, MUL4A (1555)

1555   MUL4A,   SCRATCHP:=SCRATCHP[MDS] [EXT]TEMP2             FREE*
1556   MUL3A,   SCRATCHN:=SCRATCHN[MDS] [EXT]TEMP1             FREE*
1557            SCRATCHM:=SCRATCHM[MDS] [EXT]TEMP              FREE*   RETURN
```

Figure 3-6 FPMUL Firmware (Sheet 2 of 3)

```
1536              SCRATCHR:=SCRATCHR[MDS]                FREE*    CSUB, MUL4A (1555)

1555    MUL4A,    SCRATCHP:=SCRATCHP[MDS][EXT]TEMP2      FREE*
1556    MUL3A,    SCRATCHN:=SCRATCHN[MDS][EXT]TEMP1      FREE*
1557              SCRATCHM:=SCRATCHM[MDS][EXT]TEMP       FREE*    RETURN

1536              SCRATCHR:=SCRATCHR[MDS]                FREE*    CSUB, MUL4A (1555)

1555    MUL4A,    SCRATCHP:=SCRATCHP[MDS][EXT]TEMP2      FREE*
1556    MUL3A,    SCRATCHN:=SCRATCHN[MDS][EXT]TEMP1      FREE*
1557              SCRATCHM:=SCRATCHM[MDS][EXT]TEMP       FREE*    RETURN

1536              SCRATCHR:=SCRATCHR[MDS]                FREE*    CSUB, MUL4A (1555)

1555    MUL4A,    SCRATCHP:=SCRATCHP[MDS][EXT]TEMP2      FREE*
1556    MUL3A,    SCRATCHN:=SCRATCHN[MDS][EXT]TEMP1      FREE*
1557              SCRATCHM:=SCRATCHM[MDS][EXT]TEMP       FREE*    RETURN

1536              SCRATCHR:=SCRATCHR[MDS]                FREE*    CSUB, MUL4A (1555)

1555    MUL4A,    SCRATCHP:=SCRATCHP[MDS][EXT]TEMP2      FREE*
1556    MUL3A,    SCRATCHN:=SCRATCHN[MDS][EXT]TEMP1      FREE*
1557              SCRATCHM:=SCRATCHM[MDS][EXT]TEMP       FREE*    RETURN

1536              SCRATCHR:=SCRATCHR[MDS]                FREE*    CSUB, MUL4A (1555)


1555    MUL4A,    SCRATCHP:=SCRATCHP[MDS][EXT]TEMP2      FREE*
1556    MUL3A,    SCRATCHN:=SCRATCHN[MDS][EXT]TEMP1      FREE*
1557              SCRATCHM:=SCRATCHM[MDS][EXT]TEMP       FREE*    RETURN

1536              SCRATCHR:=SCRATCHR[MDS]                FREE*    CSUB, MUL4A (1555)

1555    MUL4A,    SCRATCHP:=SCRATCHP[MDS][EXT]TEMP2      FREE*
1556    MUL3A,    SCRATCHN:=SCRATCHN[MDS][EXT]TEMP1      FREE*
1557              SCRATCHM:=SCRATCHM[MDS][EXT]TEMP       FREE*    RETURN
1537              NO OPERATION                           FREE*    GO TO, FMUL2 (1514)


        /IF MULTIPLIER IS NEGATIVE, A CORRECTION IS REQUIRED.
1514    FMUL2,    NO OPERATION                           FREE*    IF FACSGN, FMUL6 (1522)

        /CORRECTION FOR NEGATIVE MULTIPLIER--SUBTRACT 2*MULTIPLICAND
1522    FMUL6,    DB, SCRATCHS:=SCRATCHS[MINUS]TEMP5     FREE*    SUB, N (1525)

1525    N,        DB, SCRATCHR:=SCRATCHR[MINUS][EXT]TEMP4 FREE*   IF EP, M (1530)
1526              DB, SCRATCHN:=SCRATCHN[MINUS]TEMP2     FREE*
1527    P,        DB, SCRATCHM:=SCRATCHM[MINUS][EXT]TEMP1 FREE*   RETURN

1523              DB, SCRATCHS:=SCRATCHS[MINUS]TEMP5     FREE*    SUB, N (1525)

1525    N,        DB, SCRATCHR:=SCRATCHR[MINUS][EXT]TEMP4 FREE*   IF EP, M (1530)
1526              DB, SCRATCHN:=SCRATCHN[MINUS]TEMP2     FREE*
1527    P,        DB, SCRATCHM:=SCRATCHM[MINUS][EXT]TEMP1 FREE*   RETURN

1524              NO OPERATION                           FREE*    GO TO, R (1515)

        /NORMALIZE (IF NOT DP), ROUND OFF RESULT IF NOT EP MODE.
1515    R,        NO OPERATION                           FREE*    SUB, NMI (1177)

1177    NMI,      DB:=SCRATCHM                           FREE*    IF DP, RND (1240)
1200              DB:=SCRATCHN                           FREE*    IF TEMPZERO, RND (1240)
1201    NMI1,     DB:=SCRATCHP                           FREE*    IF MOVE OK, NMI4 (1215)
1202              DB:=SCRATCHR                           FREE*    IF NORMED, NMI6 (1237)

1237    NMI6,     NO OPERATION                           FREE*    IF FORBIDDEN, NMI8 (1250)
1240    RND,      NO OPERATION                           FREE*    IF EP, NMI7 (1247)
1241              NO OPERATION                           FREE*    IF TEMPSGN, RND1 (1254)

1254    RND1,     DB:=SCRATCHP[12BIT]K3777               FREE*    GO TO, RND2 (1243)

1243    RND2,     DB, SCRATCHN:=SCRATCHN[12BIT][EXT]     FREE*
1244              DB, SCRATCHM:=SCRATCHM[12BIT][EXT]     FREE*    IF TEMPZERO, RND4 (1255)
1245              DB, SCRATCHP:=[0]                      FREE*    IF DP, NMI7 (1247)
1246              NO OPERATION                           FREE*    IF FORBIDDEN, OVREC (1405)
1247    NMI7,     NO OPERATION                           FREE*    RETURN

        /ADD EXPONENTS, TEST FOR EXPONENT OVERFLOW.
1516              DB, SC:=SC[12BIT]TEMP6                 FREE*
1517    FMUL3,    NO OPERATION                           FREE*    IF DP, DPADD1 (1420)
1520              NO OPERATION                           FREE*    TEST OVFLO
1521              NO OPERATION                           FREE*    GO TO, FADD9 (1447)

1447    FADD9,    DB, TEMP7:=SC                          FREE*    IF NZSET, FADD10 (1451)

        /STORE IN EITHER MEMORY OR FAC, DEPENDING ON OP CODE.
1451    FADD10,   DB, SCRATCHE:=TEMP7                    FREE*    IF MEM, DEPOS (353)
1452              DB, FACE:=TEMP7                        FREE*    GO TO, STOF (1347)

1347    STOF,     DB, TEMP:=SCRATCHM                     FREE*
1350    STOF1,    DB, FACM:=TEMP                         FREE*    IF NOT EP, STOF2 (1357)

1357    STOF2,    DB, TEMP:=SCRATCHN                     FREE*    IF TEMPZERO, STOF3 (1361)
1360              DB, FACN:=TEMP                         FREE*    EXTEST
```

Figure 3-6 FPMUL Firmware (Sheet 3 of 3)

μPC Address	Comment
	(The following floating-point numbers have been chosen to illustrate the multiplication technique: **FACE** 0016 · **FACM** 4001 · **FACN** 6114 **OPE** 0020 · **FRACTION MSW** 3102 · **FRACTION LSW** 1111)
To 1037	At this point the operand is stored in the TEMP register thusly: Exponent stored in TEMP6 Fraction MSW stored in TEMP1 Fraction LSW stored in TEMP2
1402	The FAC exponent is sent to TEMP7. Control jumps to 1472.
1472	The MSW of the operand fraction is sent to TEMA. The operand fraction is tested; if the fraction were zero, control would jump to 1450, which causes zero to be stored as an answer.
1473	The sign of the operand fraction is examined; since the sign is 0, 0000_8 is sent to TEMP. The FAC fraction is tested for zero contents.
1474	The FAC exponent is sent to the SC. Control jumps to CLRS subroutine, 1111.
1111–1116	SCRATCHM, SCRATCHN, and SCRATCHP are cleared in preparation for the multiplication.
1532	Here are the two fractions, aligned as they would be if one were preparing to multiply them by hand: **OP** — MSW: 011 001 000 010 — LSW: 001 001 001 001 **FAC** — MSW: 100 000 000 001 — LSW: 110 001 001 100

μPC Address	Comments
1532 (Cont)	The FAC is the multiplier, the operand is the multiplicand. The entire operand both MSW and LSW, will be multiplied, first, by the LSW of the FAC and, second, by the MSW of the FAC. In effect, the partial product of the second multiplication is shifted left one word position and added to the partial product of the first multiplication. The 12 LSBs of the answer are dropped off and the remaining 36 bits are rounded off to produce the final 24-bit result. The logic implements the multiplication in this way: Each bit of the FAC is examined by the Shift logic; if a bit is logic 1, the multiplicand is added to the existing partial product and the result is shifted left once, but if a bit is logic 0, the existing partial product is merely shifted left once; the 36 bits of the product of the FAC LSW and the operand are shifted right two word positions, dropping off the 12 LSBs, which are irrelevant after carries and shifts have been propagated to the 24 MSBs; the 24 MSBs of this first multiplication are used as the initial partial products for the multiplication of the operand by the FAC MSW; when the second multiplication is finished, the 36-bit result is rounded off to the 24 MSBs. The first step in the multiplication is to load the multipllier LSW (FACN) into the DB register, as the firmware states. The bit counter in the μP Register logic is preset to a count of -12, permitting 12 successive calls to a subroutine (the first call takes place in step 1533).
1533	DB0 (the MSB of the multiplier) is examined by the Shift logic. Since the bit is logic 1, SCRATCHP and TEMP2 (OP LSW) are added; the sum is shifted left once in the shift gates and sent to SCRATCHP. The initial contents of SCRATCHP, i.e., the initial partial product, were 0000_8; hence, this first partial product is merely the multiplicand, itself. The DB is rotated left one place so as to make the second MSB available for examination. Control jumps to subroutine MUL3A, 1556.
1556	Because the EXTEND H signal is asserted, DB11 is examined by the Shift logic (this is the same bit that controlled events in step 1533; here, it is being considered in relation to the operand MSW and, thus, the entire multiplicand has been manipulated as directed by the first bit of the multiplier). The bit is 1; hence, SCRATCHN (initially zero) and TEMP 1 (OP MSW) are added, the result is shifted left once and held in SCRATCHN. The DB is not changed.
1557	Once again, DB11 is inspected. TEMP and SCRATCHM are added, and the result is shifted left once and held in SCRATCHM. This operation must be included to hold the bits shifted left from step 1556, as well as any carries that might have occurred. TEMP is 0000_8 at the beginning because the operand is a positive fraction. Had the fraction been negative, 7777_8 would have been placed in the TEMP so as to extend the sign bit throughout the multiplication process.

μPC Address	Comments
1557 (Cont)	The preceding three steps are tabulated in Figure 3-7. Pass 1 shows the result after the operand has been manipulated in response to the MSB of the multiplier. When RETURN is encountered in step 1557 of Pass 1, control returns to 1533, rather than 1534, as would be the case had the bit counter not been preset in step 1532 (the bit counter is incremented during each pass). Thus, the operations in 1533, 1556, and 1557 are performed again in Pass 2; however, the second MSB of the multiplier now controls the manipulation of the multiplicand. These three steps are followed 12 times in all. At the end of Pass 12, the bit counter has been returned to zero, permitting control to return to step 1534. The octal number in the SCRATCH is: **SCRATCHM** **SCRATCHN** **SCRATCHP** 4636 5263 1530
1534– 1127	SCRATCHM and SCRATCHN are moved two words to the right. Now we have: **SCRATCHM** **SCRATCHN** **SCRATCHP** **SCRATCHR** 4636 5263 4636 5263 The bit counter is again preset to –12.
1130	If the operand fraction had been a negative number, the final left shift in Pass 12 could have shifted logic 1 into SLINK (a positive fraction will always have a zero shifted into SLINK by the last left shift). This step retrieves such a bit by asserting the EXTEND H signal and shifting SCRATCHM right once.
1131	The sign of SCRATCHM is examined. If a logic 1 had been retrieved from SLINK, 7777_8 would be sent to SCRATCHM and TEMP7. In this example 0000_8 is sent to both.
1132	TEMP7 (0000_8) is sent to SCRATCHN. The number in the SCRATCH is: **SCRATCHM** **SCRATCHN** **SCRATCHP** **SCRATCHR** 0000 0000 4636 5263
1535	The multiplier MSW (FACM) is sent to the DB register preparatory to multiplying the operand by FACM.
1536	The bit counter was set to –12 in step 1127; hence, the present step, along with subroutine MUL4A, will be performed 12 times. At the end of 12 passes SCRATCHR will contain 0000_8, all its information having been shifted left into SCRATCHP.

	PASS 1	PASS 2	PASS 3 — — — — — — —	PASS 11	PASS 12	OCTAL RESULT IN SCRATCH
DB CONTENTS (START)	110 001 001 100	100 010 011 001	000 100 110 011	001 100 010 011	011 000 100 110	
AFTER ROTATION	100 010 011 001	000 100 110 011	001 001 100 110	011 000 100 110	110 001 001 100	
SCRATCHP (START)	000 000 000 000	010 010 010 010	110 110 110 110	110 011 010 110	100 110 101 100	
TEMP2	001 001 001 001	001 001 001 001	001 001 001 001	001 001 001 001	001 001 001 001	
ALU OUTPUT	001 001 001 001	011 011 011 011	110 110 110 110	110 011 010 110	100 110 101 100	
CARRY OUT (CLINK)	0	0	0	0	0	
MSB SHIFT IN	0	0	0	0	0	
LSB SHIFT OUT (SLINK)	0	0	1	1	1	
SCRATCHP (END)	010 010 010 010	110 110 110 110	101 101 101 101	100 110 101 100	001 101 011 000	1530
SCRATCHN (START)	000 000 000 000	110 010 000 100	010 110 001 100	101 010 101 100	010 101 011 001	
TEMP1	011 001 000 010	011 001 000 010	011 001 000 010	011 001 000 010	011 001 000 010	
CARRY IN	0	0	0	0	0	
ALU OUTPUT	011 001 000 010	001 011 000 110	010 110 001 100	101 010 101 100	010 101 011 001	
CARRY OUT (CLINK)	0	1	0	0	0	
MSB SHIFT IN	0	0	1	1	1	
LSB SHIFT OUT (SLINK)	0	0	0	1	0	
SCRATCHN (END)	110 010 000 100	010 110 001 100	101 100 011 001	010 101 011 001	101 010 110 011	5263
SCRATCHM (START)	000 000 000 000	000 000 000 000	000 000 000 010	001 001 100 111	010 011 001 111	
TEMP	000 000 000 000	000 000 000 000	000 000 000 000	000 000 000 000	000 000 000 000	
CARRY IN	0	1	0	0	0	
ALU OUTPUT	000 000 000 000	000 000 000 001	000 000 000 010	001 001 100 111	010 011 001 111	
CARRY OUT (CLINK)	0	0	0	0	0	
MSB SHIFT IN	0	0	0	1	0	
LSB SHIFT OUT (SLINK)	0	0	0	0	0	
SCRATCHM (END)	000 000 000 000	000 000 000 010	000 000 000 100	010 011 001 111	100 110 011 110	4636

Figure 3-7 FACN Times Operand Fraction

μPC Address	Comments
1555–1557	SCRATCHM, SCRATCHN, and SCRATCHP are manipulated as directed by each DB bit. Figure 3-8 tabulates Passes 1, 2, 11, and 12 for information. The SCRATCH now contains: **SCRATCHM** 3103 **SCRATCHN** 4153 **SCRATCHP** 7505 **SCRATCHR** 0000
1537	Go to 1514
1514	The FAC is tested to determine its sign. In this example the sign is negative; thus, the FAC SIGN H signal is asserted and control jumps to 1522.
1522	When the multiplier is negative, a correction must be made to the number presently in the SCRATCH. Consider the following multiplication, for example: **A** 0111 **B** 1011 0111 0111 **C** 01110 **D** 1001101 The multiplier, line **B**, is a negative number. If it were a positive number, the partial product in line C would be zero and the answer would be 10101. The difference in the two answers is 2-times the multiplicand (rather than 1-times the multiplicand, which might seem to be the case - refer to *The Logic of Computer Arithmetic* by Ivan Flores, or a similar work, for discussion of the peculiarities of 2's-complement arithmetic); thus, 2-times the multiplicand must be subtracted from the answer in line D to obtain the correct result. The same type of correction must be applied to the number in the SCRATCH; this is done beginning with step 1525. The SCRATCHS operation has no significance in this example. Go to 1525.
1525	SCRATCHR is 0000_8 for the FP multiply. Continue.
1526	Subtract TEMP2 from SCRATCHN. Logic 1 is loaded into CLINK. **SCRATCHN** 100 001 101 011 **TEMP2** (2's Complement) 110 110 110 111 1 011 000 100 010

	PASS 1	PASS 2	— — — — — — — PASS 11	PASS 12	OCTAL RESULT IN SCRATCH
DB CONTENTS (START)	100 000 000 001	000 000 000 011	011 000 000 000	110 000 000 000	
AFTER ROTATION	000 000 000 011	000 000 000 110	110 000 000 000	100 000 000 001	
SCRATCHR (START)	101 010 110 011	010 101 100 110	110 000 000 000	100 000 000 000	
ALU OUTPUT	101 010 110 011	010 101 100 110	110 000 000 000	100 000 000 000	
MSB SHIFT IN	0	0	0	0	
LSB SHIFT OUT (SLINK)	1	0	1	1	
SCRATCHR (END)	010 101 100 110	101 011 001 100	100 000 000 000	000 000 000 000	0000
SCRATCHP (START)	100 110 011 110	011 111 001 111	111 010 101 100	110 101 011 001	
TEMP2	001 001 001 001	001 001 001 001	001 001 001 001	001 001 001 001	
ALU OUTPUT	101 111 100 111	011 111 001 111	111 010 101 100	111 110 100 010	
CARRY OUT (CLINK)	0	0	0	0	
MSB SHIFT IN	1	0	1	1	
LSB SHIFT OUT (SLINK)	1	0	1	1	
SCRATCHP (END)	011 111 001 111	111 110 011 110	110 101 011 001	111 101 000 101	7505
SCRATCHN (START)	000 000 000 000	110 010 000 101	101 011 111 001	010 111 110 011	
TEMP1	011 001 000 010	011 001 000 010	011 001 000 010	011 001 000 010	
CARRY IN	0	0	0	0	
ALU OUTPUT	011 001 000 010	110 010 000 101	101 011 111 001	110 000 110 101	
CARRY OUT (CLINK)	0	0	0	0	
MSB SHIFT IN	1	0	1	1	
LSB SHIFT OUT (SLINK)	0	1	1	1	
SCRATCHN (END)	110 010 000 101	100 100 001 010	010 111 110 011	100 001 101 011	4153
SCRATCHM (START)	000 000 000 000	000 000 000 000	000 110 010 000	001 100 100 001	
TEMP	000 000 000 000	000 000 000 000	000 000 000 000	000 000 000 000	
CARRY IN	0	0	0	0	
ALU OUTPUT	000 000 000 000	000 000 000 000	000 110 010 000	001 100 100 001	
CARRY OUT (CLINK)	0	0	0	0	
MSB SHIFT IN	0	1	1	1	
LSB SHIFT OUT (SLINK)	0	0	0	0	
SCRATCHM (END)	000 000 000 000	000 000 000 001	001 100 100 001	011 001 000 011	3103

Figure 3-8 FACM Times Operand Fraction

µPC Address	Comments
1527	Subtract TEMP1 from SCRATCHM. Since EXTEND H is asserted, the contents of CLINK are applied to the carry input of the ALU. Note that only the 1's complement of TEMP1 is taken; the 2's complement has already been applied to the 12 LSBs. **SCRATCHM** 011 001 000 011 **TEMP1** (1's Complement) 100 110 111 101 **Carry In** 1 1 000 000 000 001
1523	Not applicable, go to 1525.
1525	0000_8 to SCRATCHR.
1526	**SCRATCHN** 011 000 100 010 **TEMP2** (2's Complement) 110 110 110 111 1 001 111 011 001
1527	**SCRATCHM** 000 000 000 001 **TEMP2** (1's Complement) 100 110 111 101 **Carry In** 1 100 110 111 111
1524	The number now in the SCRATCH is: **SCRATCHM** 4677, **SCRATCHN** 1731, **SCRATCHP** 7505, **SCRATCHR** 0000 Go to 1515.
1515	Go to 1177.
1177–1241	The SCRATCH is already normalized, so the round-off process begins at step 1254.
1254–1247	The number is negative, so 3777_8 is added to SCRATCHP. The resulting carry is added to SCRATCHN, and SCRATCHP is zeroed. The number now in the SCRATCH is: **SCRATCHM** 4677, **SCRATCHN** 1732, **SCRATCHP** 0000, **SCRATCHR** 0000

μPC Address	Comments
1516	The two exponents (FACE in SC, OPE in TEMP6) are added to test for overflow. None occurs in this example. The sum, 0036_8, is held in the SC.
1447	The SC is sent to TEMP7. Because there was no overflow, control jumps to 1451.
1451–1360	The result of the multiplication is stored in the FAC. Thus, we have: FACE FACM FACN 0036 4677 1732 An exit test causes control to jump to FETCH, location 0020.

3.3.4 FPDIV Firmware

While multiplication involves a sequence of additions and shifts, or shifts alone, division entails repeated subtraction and shifts. Implementation of division requires an examination of the divisor in relation to the dividend or partial remainder. A quotient bit is assumed and verified by reduction, i.e., a subtraction of the divisor from the dividend or partial remainder. If the reduction produces a result having the same sign as the partial remainder, the assumed quotient bit is correct; however, if a sign change occurs, the quotient bit is incorrect. If incorrect, the bit is discarded, the partial remainder is restored to its pre-reduction condition, a new assumption is made, and another reduction is attempted.

This process of bit assumption, reduction, and possible restoration is time-consuming. Several methods are available for increasing the speed of division. One method involves non-restoration of the partial remainder; this is the method that is implemented by the FPP logic; specifically, the FPP performs a non-restoring divide of a signed divisor and a positive dividend. The logic compares the signs of both the divisor and the quotient bit determined by the previous reduction; the comparison determines whether the divisor is subtracted from the dividend or added to the dividend. The complement of the sign bit of the reduction is then retained as the quotient bit.

Figure 3-9 shows part of the firmware of a division carried out in the FP mode. The firmware begins at the FDIV pointer address, 1403, and includes preliminary steps leading up to the reduction and shifting operations. The firmware proceeds through the generation of the quotient MSW and LSW. The remainder of the division process, much of which has been detailed in preceding examples, is left to the reader's ingenuity. The portion of the firmware that is illustrated is described briefly in the following commentary.

```
1403    FDIV,    DB:=[SHL]FACM                                 FREE    GO TO, FQUO (1562)

        //////////FLOATING AND FIXED POINT DIVIDE//////////
        /SHIFT LINK HOLDS FAC SIGN AT ENTRY.  CHECK FIRST FOR ZERO DIVISOR
        /(SET DIVZERO FLAG AND EXIT); IF IN FP OR EP MODE, MAKE SURE
        /DIVISOR NORMED--IF NOT, DO IT; THEN CHECK FOR ZERO DIVIDEND (ANS
        /ALREADY IN FAC).  XOR FRACTION SIGNS AND MAKE SIGN OF TEMA
        /EQUAL TO SIGN OF RESULT.  SETTING DB=1 AT "FQUO1"
        /FORCES CORRECT FIRST DIVIDE OPERATION, SINCE HARDWARE
        /EXAMINES DB11 TO DETERMINE WHAT TO DO.  THE DIVIDE IS A NON-
        /RESTORING DIVIDE OF A SIGNED DIVISOR AND A POSITIVE DIVIDEND.

1562    FQUO,    DB, TEMA:=[SHR][EXT]                          FREE    IF TEMPZERO, FDIV0 (1630)
1563             DB, SCRATCHP:=[0]                             FREE    IF DP, FQUO2 (1570)
1564             DB, SCRATCHT:=[0]                             FREE    IF NORMED, FQUO2 (1570)

1570    FQUO2,   DB, TEMP:=FACE                                FREE*   IF FACZERO, CLRFAC (1050)
1571             DB, SC:=TEMP                                  FREE*   SUB, FTOS (1333)

        /SUBROUTINE--MOVE FAC FRACTION TO SCRATCH.
1333    FTOS,    DB, TEMP:=FACM                                FREE*
1334    FTOS1,   DB, SCRATCHM:=TEMP                            FREE*   IF NOT EP, FTOS3 (1346)

1346    FTOS3,   DB, SCRATCHP:=[0]                             FREE*   GO TO, FTOS2 (1344)

1344    FTOS2,   DB, TEMP:=FACN                                FREE*
1345             DB, SCRATCHN:=TEMP                            FREE*   RETURN

1572             DB, TEMP7:=TEMP1                              FREE*   IF FACSGN, FDIV1 (1624)
1573    FQUO1,   DB, TEMA:=TEMA[12BIT]TEMP1                    FREE*   PRESET BIT COUNT
1574             DB:=K1                                        FREE*   IF NOT EP, FDIV10 (1632)

        /DO 24-BIT DIVIDE
1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN


1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1632    FDIV10, SCRATCHP:=SCRATCHP[MDS]TEMP3                   FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN

1633             MQM:=DB                                       FREE*   PRESET BIT COUNT
1634             SCRATCHP:=SCRATCHP[MDS]TEMP3                  FREE*   CSUB, DIV3A (1664)

1664    DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2             FREE*
1665             SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1           FREE*   RETURN
```

Figure 3-9 FPDIV Firmware (Sheet 1 of 2)

```
1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1634            SCRATCHP:=SCRATCHP[MDS]TEMP3              FREE*   CSUB, DIV3A (1664)

1664   DIV3A,   SCRATCHN:=SCRATCHN[MDS][EXT]TEMP2         FREE*
1665            SCRATCHM:=SCRATCHM[MDLST][EXT]TEMP1       FREE*   RETURN

1635            MQN:=DB                                   FREE*
```

Figure 3-9 FPDIV Firmware (Sheet 2 of 2)

μPC Address	Comment
1403	Shift FACM left, send to the DB. FAC sign is loaded into SLINK. Go to 1562.
1562	Shift right content of SLINK into bit 4, send to TEMA (put FAC sign into bit 4 of TEMA). Check operand fraction; if 0, go to FDIV0, which sets DIV0 flag in Exit Test logic and exits.
1563, 1564	0 to SCRATCHP and SCRATCHT.
1564	Assume divisor is normalized; go to 1570.
1570	FAC exponent to TEMP. Check to see if FAC fraction is 0; if so, the answer (0) is already in the FAC. (In that case, clear FAC and exit test.)
1571	FAC exponent to SC. Go to sub 1333.
1333, 1334	FACM to SCRATCHM; go to 1346.
1346	0 to SCRATCHP; go to 1344.
1344, 1345	FACN to SCRATCHN; return to 1572.
1572	Operand MSW to TEMP7. Check FAC sign, if negative, go to 1624 (1624 complements the SCRATCH so that the dividend is always positive).
1573	Add TEMA (FAC sign is in bit 4, 0s in bits 5–15) and operand MSW; send result to TEMA and DB (effectively XORing fraction signs; carry, if any, is lost). Preset bit counter.
1574	0001 to DB. Forces a correct first divide operation.
1632–1665	First 12 reduction/shifting operations.
1633	Move quotient MSW to MQM.
1634–1665	Second 12 reduction/shifting operations.
1635	Move quotient LSW to MQN

CHAPTER 4
FPP8-A LOGIC

4.1 FPP8-A BLOCK DIAGRAM

The FPP logic is contained on two printed circuit boards – the Control logic board (M8410) and the Data Path logic board (M8411). A block diagram of the Control logic is shown in Figure 4-1, while a similar diagram for the Data Path logic can be seen in Figure 4-2.

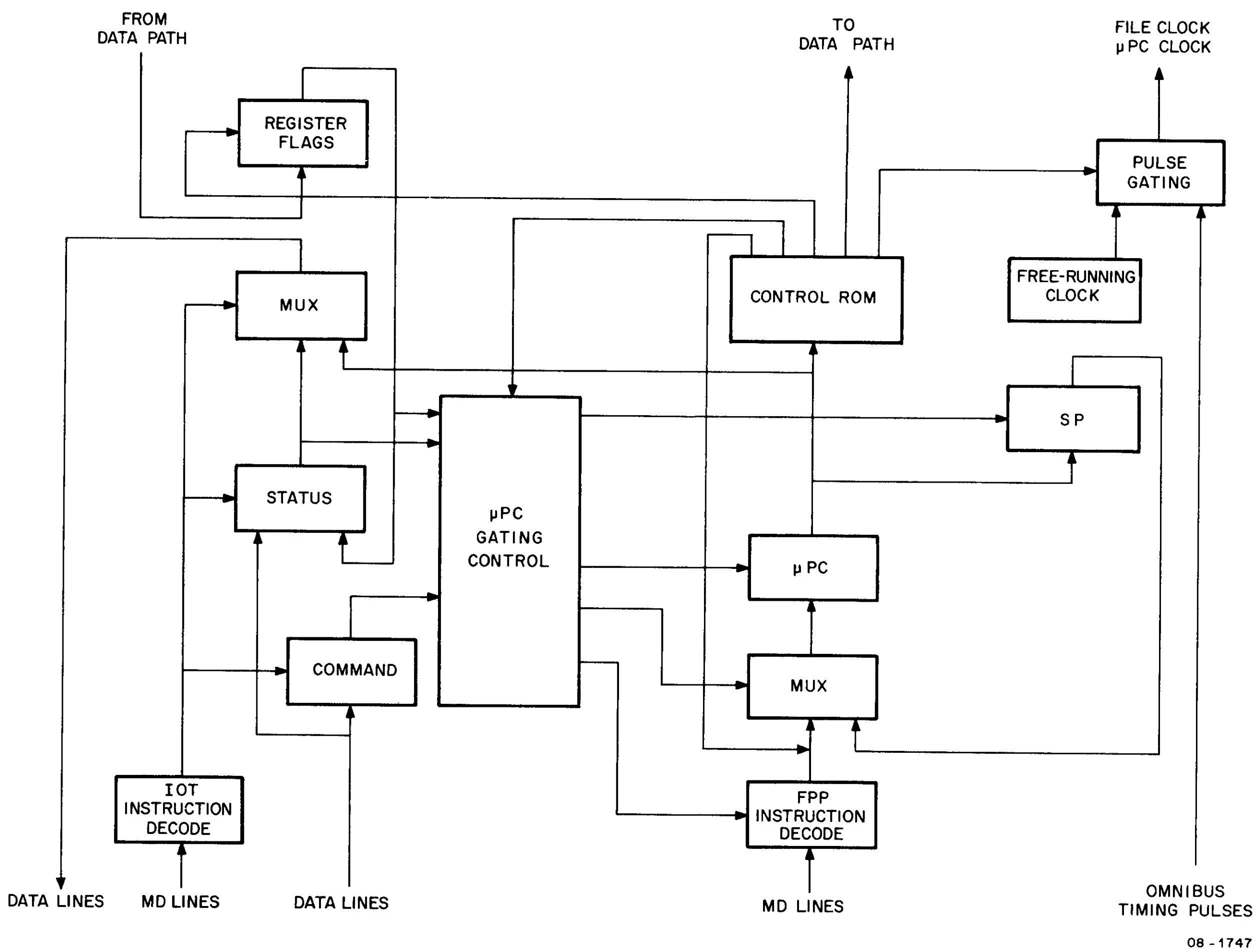


Figure 4-1 Block Diagram, Control Logic

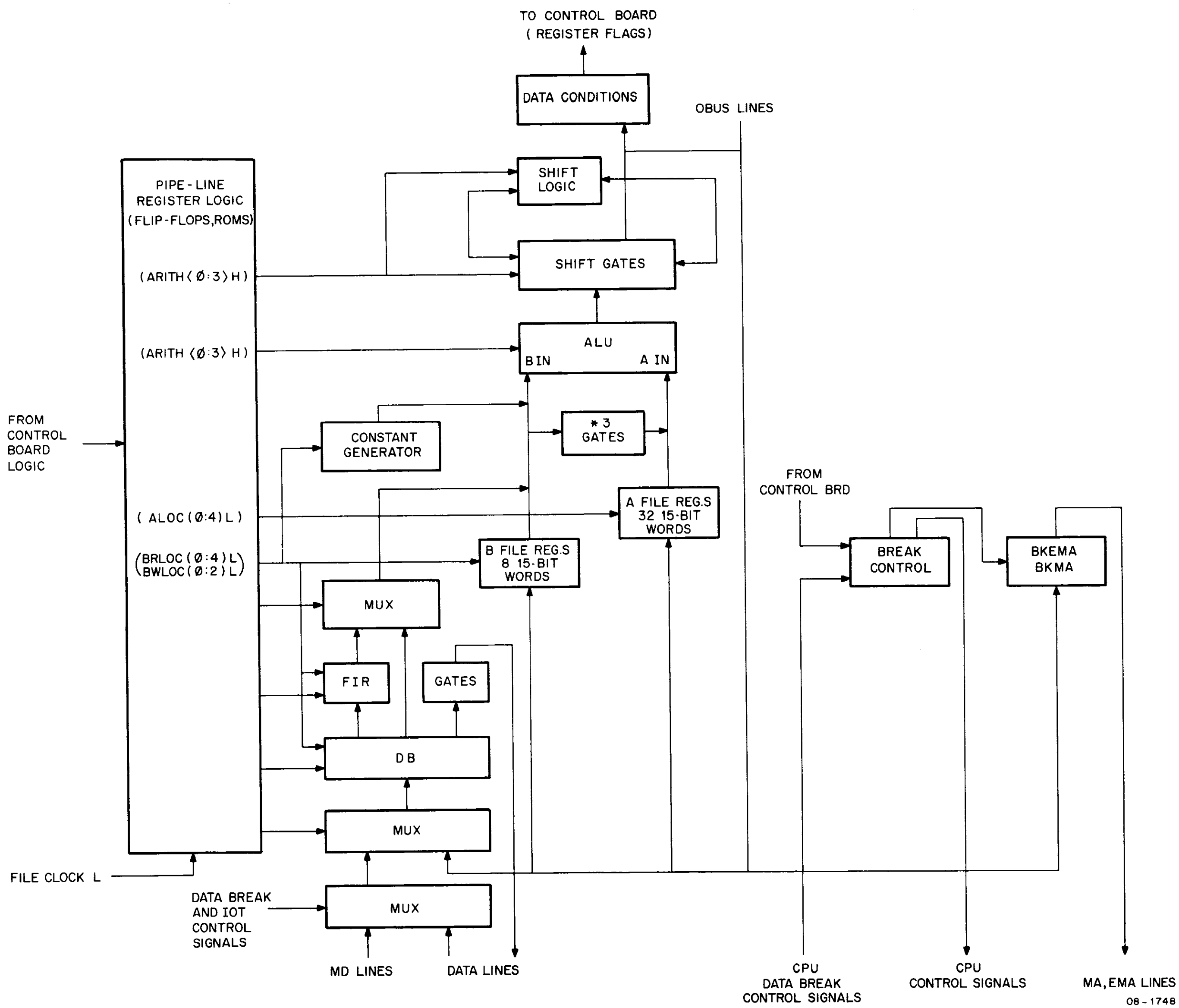


Figure 4-2 Block Diagram, Data Path Logic

One of the Control logic functions is IOT decoding. The decoding process uses and produces the familiar PDP-8 I/O control signals (I/O PAUSE L, TP3 H, C0 L, SKIP L, etc....). The FPCOM IOT instruction loads the Command register and part of the Status register with information that selects various FPP operating features. The Status register also monitors some significant operating characteristics, and the stored information concerning these characteristics can be tested by both the μPC Gating Control logic and the FPRST IOT instruction.

The Control board includes logic that provides a clock for the entire FPP. The μPC CLOCK signal is utilized by the Control logic; it generates, in turn, the FILE CLOCK signal that is used by the Data Path logic.

The central element in the Control logic is the Control ROM. This element generates signals that direct operations in both the Data Path logic and the Control logic. Control ROM locations are accessed by the μPC register, which is supplied with address information from a number of sources. One of these sources is the Control ROM itself, which provides a jump address when control is to be transferred to a subroutine (if a jump address is not provided by the ROM or by one of the other sources, the μPC address is merely incremented to the next consecutive address). Another source is the SP register; when control is to be transferred to a subroutine and then returned, the return address is saved in the SP and gated to the μPC at the end of subroutine operations. Finally, when FPP instructions are decoded, the decoding logic forces the μPC address to the routine dictated by the instruction.

The source that is selected to provide the address of the next Control ROM location depends on the operation being directed by the present Control ROM location. The present location generates signals that regulate the μPC Gating Control; this logic then examines its inputs and gates the μPC register sources in such a way that the appropriate address is supplied to the μPC. For example, if the present location directs a jump to a subroutine, with return, the μPC Gating Control causes the return address to be saved in the SP, places the μPC in the parallel-load mode, and multiplexes the jump address contained in the present location to the μPC. Or, consider the Register Flags logic for a moment. The FPP logical sequence periodically checks the data being manipulated and follows a course of action that reflects some data condition, which has been temporarily stored in the Register Flags logic. The present location might direct a conditional address jump based on the state of a selected register flag. The μPC Gating Control logic looks at the signal representing the register flag; if the stated condition is positive, the μPC receives the jump address contained in the present location, but if the condition is negative, the μPC is kept in the counting mode and the μPC address is merely incremented.

Conditional jumps are also carried out based on the contents of the Status and Command registers. For instance, the FPP logical sequence depends heavily on what calculating mode was programmed – DP, FP, or EP. Thus, the DP and EP status flags are often checked during operations to determine whether or not an address jump should be carried out.

Each Control ROM location generates signals that are applied to the logic on the Data Path board (Figure 4-2). Except for three signals that are put to use in the Break Control logic, all the Control board signals are applied to the Pipe-line register logic, which consists of a number of flip-flops and ROMs. This logic provides gating and control signals necessary for the Data Path manipulations. The signals shown in the Pipe-line register logic block are generated by the Control ROM. It is these signals, primarily, that are responsible for manipulating the Data Path elements with which they are associated. The ARITH (0:3) H signals are responsible for controlling the ALU and gating the Shift Gates; they do so by generating, in turn, a number of signals in the Pipe-line register logic (these latter signals are not indicated in the block diagram). The ALOC (0:4) L signals are applied, indirectly, to the A-file and select registers for reading and writing* (the signals shown are inputs to the Pipe-line register;their names change at the output). The BRLOC (0:4) L and BWLOC (0:2) L signals select B-file registers for reading and writing; the BRLOC (0:4) L signals select the DB register, the FIR register, or

*'Reading' is defined as gating the contents of the selected source to the appropriate ALU input; 'writing' is defined as gating the OBUS contents to the selected register.

the Constant generator for reading. Tables 4-1, 4-2, and 4-3 relate the ALOC (0:4) L, BRLOC (0:4) L, and BWLOC (0:2) L signals, respectively, to the sources selected for reading and writing. Table 4-4 lists the ARITH (0:3) H signals and describes the operation carried out by the ALU and the Shift Gates for each set of signals. (More information concerning Data Path signals can be found in drawing D-CS-M8411-0-1 of the FPP8-A Print Set.)

Table 4-1 ALOC (0:4) L Functions

ALOC (0:4) L					Sources Selected for Reading or Writing*
0	1	2	3	4	(Source gated to A input of ALU or loaded from OBUS)
H	H	H	H	H	FPC
H	H	H	H	L	X0
H	H	H	L	H	BR
H	H	H	L	L	OPADD
H	H	L	H	H	APTP
H	H	L	H	L	TEMA
H	H	L	L	H	FIELD
H	H	L	L	L	NOT USED
H	L	H	H	H	FACE (EXPONENT)
H	L	H	H	L	FACM [FRACTION (0:11)]
H	L	H	L	H	FACN [(12:23)]
H	L	H	L	L	FACP [(24:35)]
H	L	L	H	H	FACR [(36:47)]
H	L	L	H	L	FACS [(48:59)]
H	L	L	L	H	NOT USED
H	L	L	L	L	SC
L	H	H	H	H	SCRATCHE
L	H	H	H	L	SCRATCHM
L	H	H	L	H	SCRATCHN
L	H	H	L	L	SCRATCHP
L	H	L	H	H	SCRATCHR
L	H	L	H	L	SCRATCHS
L	H	L	L	H	SCRATCHT
L	H	L	L	L	NOT USED
L	L	H	H	H	MQE
L	L	H	H	L	MQM
L	L	H	L	H	MQN
L	L	H	L	L	MQP
L	L	L	H	H	MQR
L	L	L	H	L	MQS
L	L	L	L	H	NOT USED
L	L	L	L	L	NOT USED

*READ A H must be asserted for reading.
WRITE A H must be asserted for writing.

Table 4-2 BRLOC (0:4) L Functions

BRLOC (0:4) L					Sources Selected for Reading
0	1	2	3	4	(Contents gated to B input of ALU-0 goes to (1:3) unless otherwise noted)
L	H	H	H	H	TEMP
L	H	H	H	L	TEMP1
L	H	H	L	H	TEMP2
L	H	H	L	L	TEMP3
L	H	L	H	H	TEMP4
L	H	L	H	L	TEMP5
L	H	L	L	H	TEMP6
L	H	L	L	L	TEMP7
L	L	H	H	H	Bits (1:3) of TEMP to (1:3), DB to (4:15)
L	L	H	H	L	0 to (1:3), DB to (4:15)
L	L	H	L	H	If FIR 4=0, Bits (9:11) of FIR to (13:15), 0 to (4:12) If FIR 4=1, Bits (5:11) of FIR to (9:15), 0 to (4:12)
L	L	H	L	L	Bits (6:8) of FIR to (13:15), 0 to (4:12)
L	L	L	H	H	NOT USED
L	L	L	H	L	NOT USED
L	L	L	L	H	Bits (1:3) of A input to (13:15)
L	L	L	L	L	Bits (1:3) of A input to (13:15), 103_8 to (4:12)
H	H	H	H	H	CONSTANT (0)
H	H	H	H	L	CONSTANT (1)
H	H	H	L	H	CONSTANT (2)
H	H	H	L	L	CONSTANT (3)
H	H	L	H	H	CONSTANT (–1)
H	H	L	H	L	CONSTANT (–2)
H	H	L	L	H	CONSTANT (–27)
H	H	L	L	L	CONSTANT (–73)
H	L	H	H	H	Bits (1:3) of TEMP to (1:3), 0 to (4:15)
H	L	H	H	L	CONSTANT (14)
H	L	H	L	H	CONSTANT (–14)
H	L	H	L	L	CONSTANT (–5)
H	L	L	H	H	CONSTANT (–6)
H	L	L	H	L	CONSTANT (2000)
H	L	L	L	H	CONSTANT (4000)
H	L	L	L	L	CONSTANT (–30)

Table 4-3 BWLOC (0:2) L Functions

BWLOC (0:2) L			Sources Selected For Writing* (OBUS contents loaded into selected register)
0	1	2	
H	H	H	TEMP
H	H	L	TEMP1
H	L	H	TEMP2
H	L	L	TEMP3
L	H	H	TEMP4
L	H	L	TEMP5
L	L	H	TEMP6
L	L	L	TEMP7

*WRITE B H must be asserted.

Table 4-4 ARITH (0:3) H Functions

ARITH0 H	ARITH1 H	ARITH2 H	ARITH3 H	Function Carried Out in Data Path Logic
L	L	L	L	A + B + CARRY (15 bits) to OBUS.
L	L	L	H	(A + B + CARRY) * 2 to OBUS (2*B).
L	L	H	L	(A + B + CARRY) Logically right-rotated 3 places to OBUS.
L	L	H	H	(3 * B + CARRY) (15 bits) to OBUS (3*B).
L	H	L	L	(3 * B + CARRY) * 2 to OBUS (6*B).
L	H	L	H	A + B + CARRY (12 bits) to OBUS.
L	H	H	L	0 to OBUS (15 bits).
L	H	H	H	A sign to OBUS (0000 or 7777).
H	L	L	L	B to OBUS (12 bits).
H	L	L	H	A + $\overline{B}$ + CARRY (12 bits) to OBUS (A–B).
H	L	H	L	Exponent Size (12 bits).
H	L	H	H	Overflow recovery (Complement of sign to SGN L, shift right)
H	H	L	L	(A + B + CARRY) * 2 + Shift Bit (12 bits) to OBUS.
H	H	L	H	(A + B + CARRY) ÷ 2 + Shift Bit (12 bits) to OBUS.
H	H	H	L	Divide Final.
H	H	H	H	MUL/DIV Step.

Since the purpose of the Data Path logic is data manipulation, the ALU and the Shift Gates are of primary importance, for it is these elements that carry out the maneuvers required for both data calculation and data circulation. However, the data, itself, must first be supplied to the logic before any other operations can be started. The task of transferring data to the FPP from the PDP-8 CPU, and in the reverse direction as well, is carried out by the DB register. The DB can receive data from any one of three sources. Two of these sources (Omnibus DATA and MD lines) are involved with data input (to the FPP), while the third source (FPP OBUS lines) is related to data output.

The Omnibus DATA lines provide input information for the DB during initialization, when the APT pointer address is loaded into the DB for transfer to the APTP register. During all other input transfers, the information is taken from the Omnibus MD lines. This information can be strictly data that is to be used in calculations or it can be an FPP instruction. If the information is data, i.e., an operand, it is loaded into the DB register and multiplexed to the B inputs of the ALU. The ALU and the Shift Gates then place the data on the OBUS, and it is loaded into the selected file register. If the information is an FPP instruction, the procedure is somewhat different. Certain bits of the first 12-bit word of the instruction are loaded into the FIR register (the second word of the instruction, if applicable, is loaded into only the DB). This operation permits either the field bits of a double-word instruction to be saved, or the offset of a single-word instruction to be added to the base address. The FIR output and the DB output, which together specify a 15-bit operand address, can then be placed on the OBUS and gated to the A-file OPADD register and to the BKMA/BKEMA registers. The address is placed on the Omnibus MA and EMA lines, and a data break is requested. When the request is granted, the operand is placed on the MD lines by the CPU and loaded into the MB register.

Since the DB register contents can be gated onto the Omnibus DATA lines, the data in any file register or the results of any calculation carried out in the FPP can be sent to the PDP-8 CPU. The reason might be storage, as in the FMULM instruction, for example, or it might be for visibility during maintenance operations.

When the data required by an FPP instruction has been specified, the ALU and the Shift Gates can be put to work to carry out the necessary operations. The ALU performs direct addition and subtraction (2's complement) with the quantities on its A and B inputs; multiplication and division are effected with the aid of the Shift logic. See Table 4-4 for a list of the available operations.

The *3 Gates shown at the ALU inputs are used during address decoding to account for the fact that operands are of various word lengths, depending on the operating mode. For example, in the FP mode, operands comprise three data words; hence, an operand address must be multiplied by 3 to obtain the next operand address. This multiplication is accomplished as follows: The *3 Gates gate the address on the ALU B inputs to the ALU A inputs, while at the same time shifting the address left one place, i.e., bit 15 of the B input, for example, is gated to bit 14 of the A input, and so on (this operation multiplies the binary number by 2); then, the A and B inputs are added, i.e., the address is added to 2-times the address. If the mode is EP, the operand address must be multiplied by 6. This multiplication begins as just described. Then, the ALU output, which is 3-times the B-input address, is shifted left one place in the Shift Gates, resulting in the desired multiplication.

4.2 CONTROL LOGIC

Detailed descriptions of the significant portions of the logic on the Control board appear in Paragraphs 4.2.1 through 4.2.9.

4.2.1 IOT Decoding Logic

The IOT Decoding logic is shown in Figures 4-3 and 4-4. Figure 4-3 illustrates that part of the logic that decodes the Omnibus MD bits. The comparator, E12, decodes MD bits (0:8) [I/O PAUSE L is asserted if MD (0:2) L is 6_8] and generates the FPIOT L signal when device codes 55_8 or 56_8 are detected. PROMs E55 and E56 decode the MD (9:11) L signals to generate the specific IOT instruction signals. The PROM inputs are related to the IOT instructions by Table 4-5.

Figure 4-4 shows the PROM output signals that are generated in response to the IOT instructions. Table 4-5 relates the PROM input/output signals (in the "INPUT SIGNAL LOW" column, the logic levels of the MD bits are listed; these must be inverted to get the levels actually applied to the PROM inputs). PROM E56 can be enabled when the FPP is running. Among the operations that can be initiated by E56 are maintenance and exit. Unlike E56, E55 is enabled only when the FPP is not busy, i.e., when BUSY H is negated. In such a situation, the μPC register contains address 0, 1, 2, or 3.

When the FPP is turned on but has not yet been started, the Control ROM location alternates between 0 and 3 (see Paragraph 4.2.7 for details concerning FPP initialization). Each location contains the jump address of the other. That is, when address 0 is in the μPC, the Control ROM location points to address 3. At clock pulse time, address 3 is loaded into the μPC. Now the Control ROM location points to address 0, which is loaded into the μPC by the next clock pulse.

IOT instructions are decoded during the time that address 0 is held in the μPC. Assume, for example, that FPST is issued. E55 asserts μP8IN L and μP9IN L. The Control ROM points to address 3, i.e., the ROM asserts μP10IN L and μP11IN L. Thus, at clock-pulse time, address 17_8 is loaded into the μPC register and the FPP firmware jumps to the maintenance program. If FPCOM had been issued, instead, address 7 would be loaded at clock time, the APT pointer field bits would be sent to the FIELD register, and the μPC would return to address 0. FPCOM is followed by the FPST instruction when initialization is being carried out. Address 13_8 is forced into the μPC, the APT pointer address is sent to the APT register, and the FPP jumps to the GETAPT routine.

If the FPP had been operating and was returned to the "paused" condition (μPC address 2), it could be restarted with the FPST instruction, which forces address 6 into the μPC register. Should the FPHLT instruction be issued while the FPP is paused, address 16_8 is forced into the μPC, the FPC is decremented, and an exit is carried out.

4.2.2 Status and Command Register Logic

Figure 4-5 illustrates the Status and Command Register logic. The Command register, E8, is loaded from the DATA (1:8) L lines when the FPCOM instruction is decoded. Bit 0 of the Status register, E3 and E4, is also loaded by FPCOM, this bit representing the FPP's calculating mode (FP or DP).

The Status register holds not only FP/DP information, but also information provided by the Exit Test logic (refer to Paragraph 4.2.9), the EP mode bit (loaded by the FPEP instruction), and the Trap instruction bit. The Status information can be read by both the FPIST and FPRST instructions, the former clearing the Status register after gating the information onto the DATA lines.

If the FMRP instruction is issued, the μPC register address is gated onto the DATA lines. The information is valid in maintenance mode and in normal mode provided the FPP is not free-running.

4.2.3 Control ROM Logic

The Control ROM was referred to in Chapter 3 in relation to the FPP firmware. The logic is illustrated in Figure 4-6. The complete Control ROM array consists of 31 1024-bit PROMs arranged to provide a total of 768 (1400_8) 44-bit word locations. Word locations are addressed by the outputs of the μPC register, represented by the μPC (2:11) H signals. For clarity, Figure 4-6 shows only a portion of the array; i.e., 8 PROMs are missing from each row.

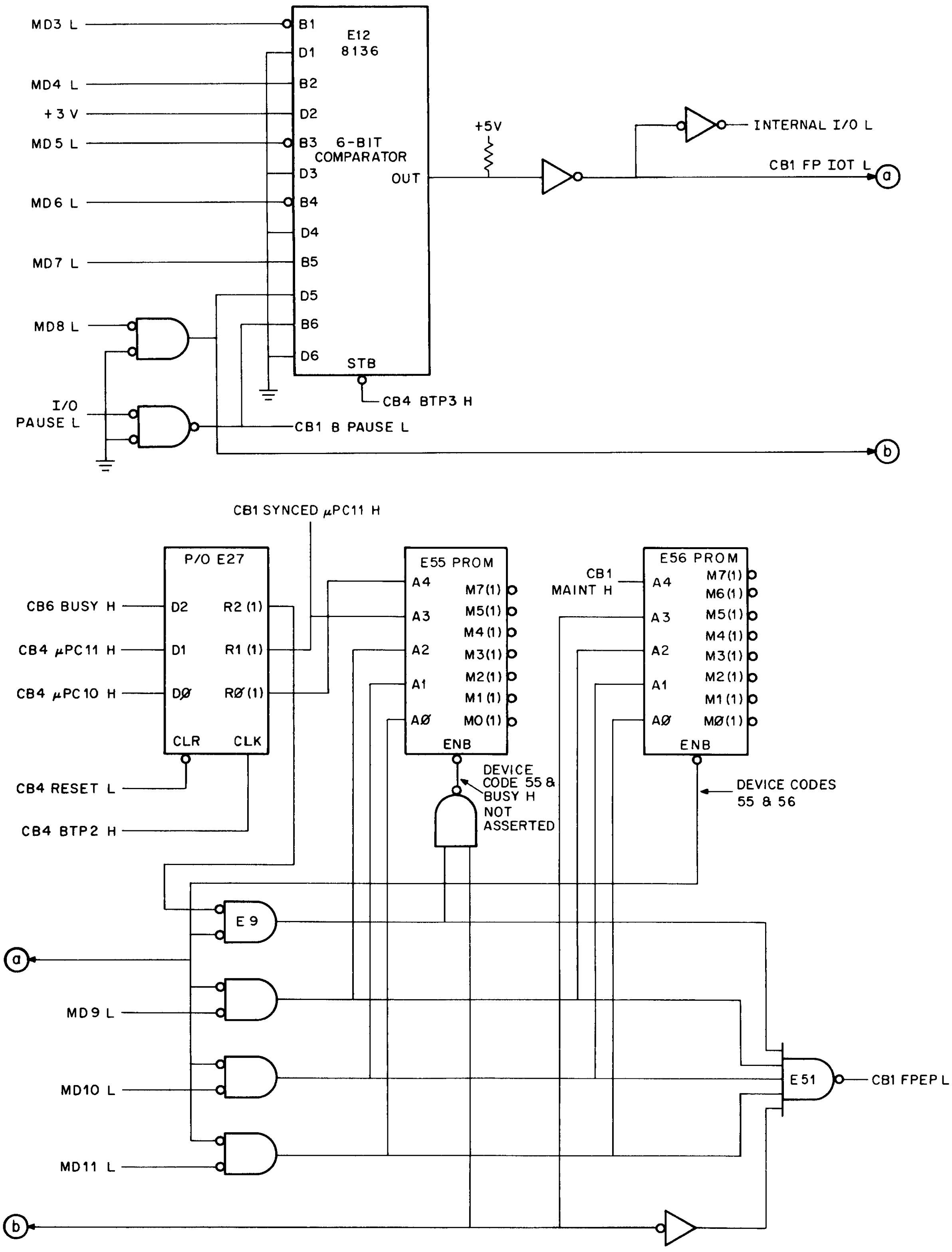


Figure 4-3 IOT Decoding Logic (A)

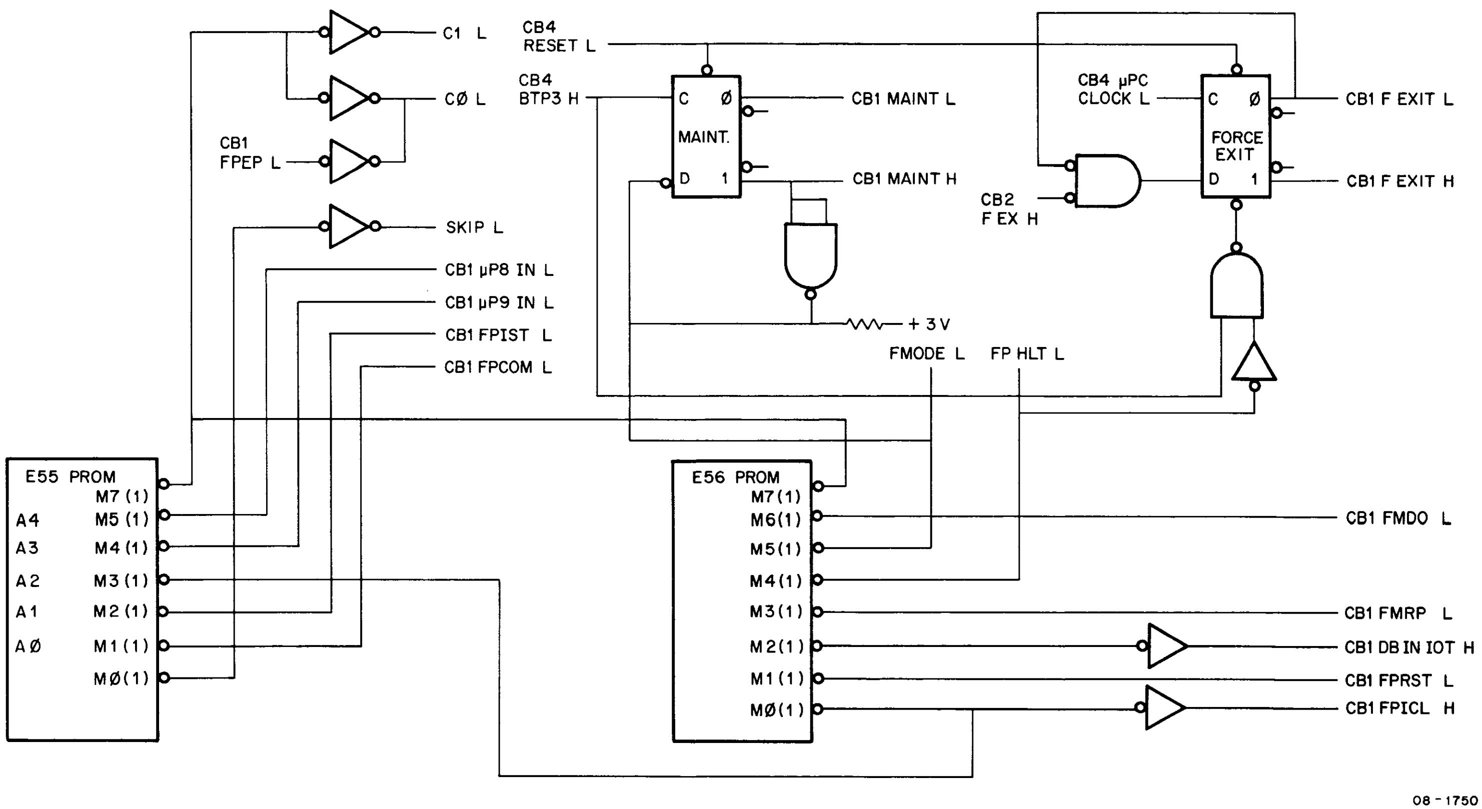


Figure 4-4 IOT Decoding Logic (B)

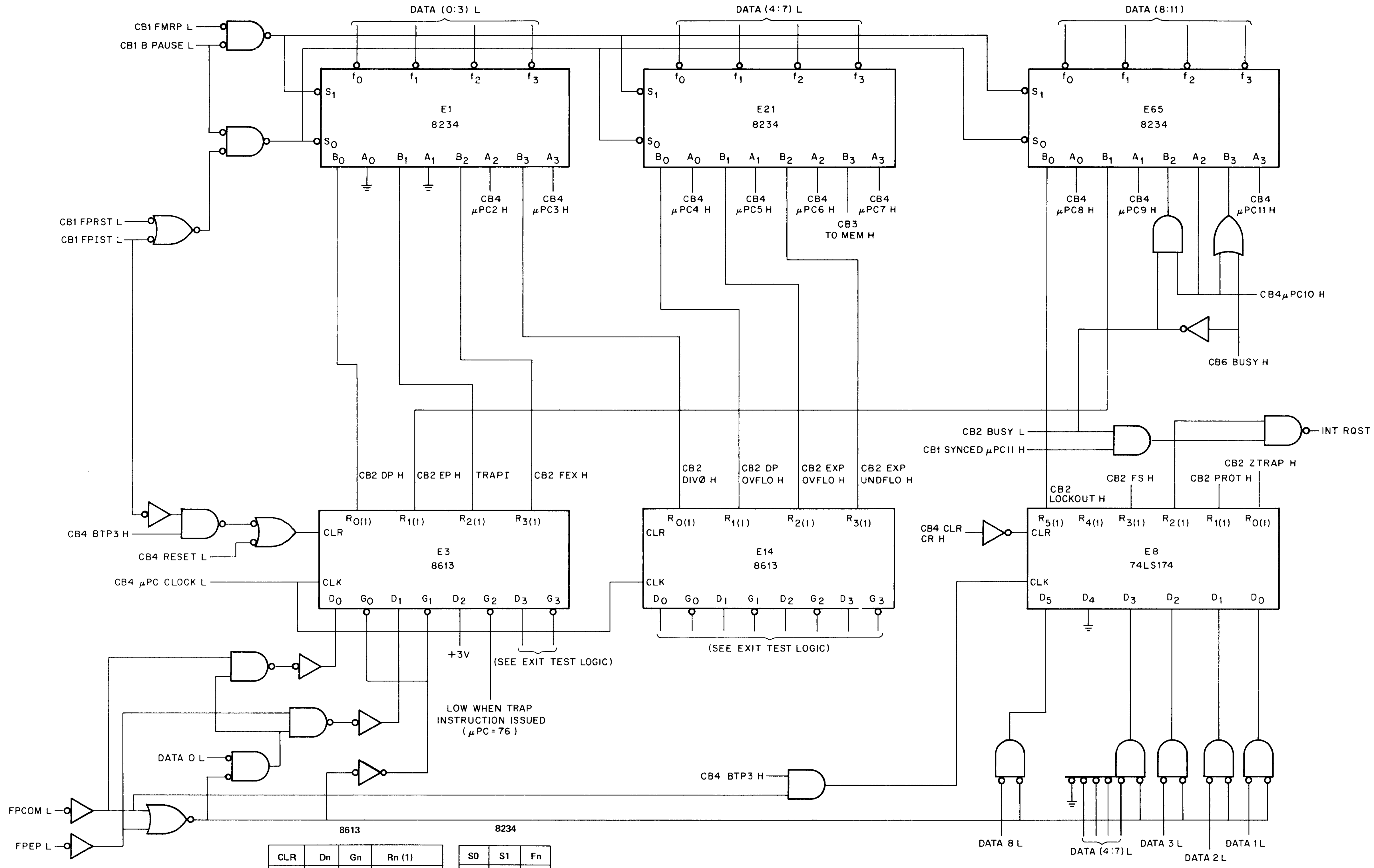
Table 4-5 PROMs E55/E56 Input/Output Tables

E55 (Enabled if 655X IOT is issued when FPP is not Busy)

Input	Input Signal Low					Output Signal Asserted							
Code	μPC10 H	μPC11 H	MD9 L	MD10 L	MD11L	C0 L/C1 L	μP8 IN L	μP9 IN L	FPICL H	FPIST L	FPCOM L	SKIP L	IOT Instruction Decoded
0	X	X					X	X					FFST (μPC=17)
3	X	X		X	X			X			X		FPCOM (μPC=7)
5	X	X	X		X		X					X	FPST (START, μPC=13)
11	X				X							X	FPINT
17	X		X	X	X	X			X	X		X	FPIST
24		X	X				X	X					FPHLT (μPC=16)
25		X	X		X			X				X	FPST (CONTINUE, μPC=6)

E56 (Enabled when FPP IOT is Decoded)

Input	Input Signal Low					Output Signal Asserted								IOT Instruction
Code	MAINT H	MD8 L	MD9 L	MD10L	MD11L	C0 L/C1 L	FMDO L	FMODE L	FPHLT L	FMRP L	DB IN IOT H	FPRST L	FPICL H	Decoded
1					X			X						FMODE
3				X	X	X						X		FMRB
4			X			X				X				FMRP
5			X		X		X							–
12		X		X									X	FPICL
14		X	X						X					FPHLT
16		X	X	X		X						X		FPRST
23	X			X	X	X					X			FMRB (MAINT)
24	X		X			X				X				FMRP (MAINT)
25	X		X		X		X							FMDO (MAINT)
32	X	X		X									X	FPICL (MAINT)
34	X	X	X						X					FPHLT (MAINT)
36	X	X	X	X		X						X		FPRST (MAINT)



8613

CLR	Dn	Gn	Rn (1)
LO	HI	LO	HI
LO	LO	LO	LO
LO	X	HI	NO CHANGE
HI	X	X	LO*

*ASYNCHRONOUS TRANSITION

8234

S0	S1	Fn
LO	LO	$\overline{Bn}$
HI	LO	$\overline{An}$
LO	HI	$\overline{Bn}$
HI	HI	1

Figure 4-5 Status and Command Register Logic

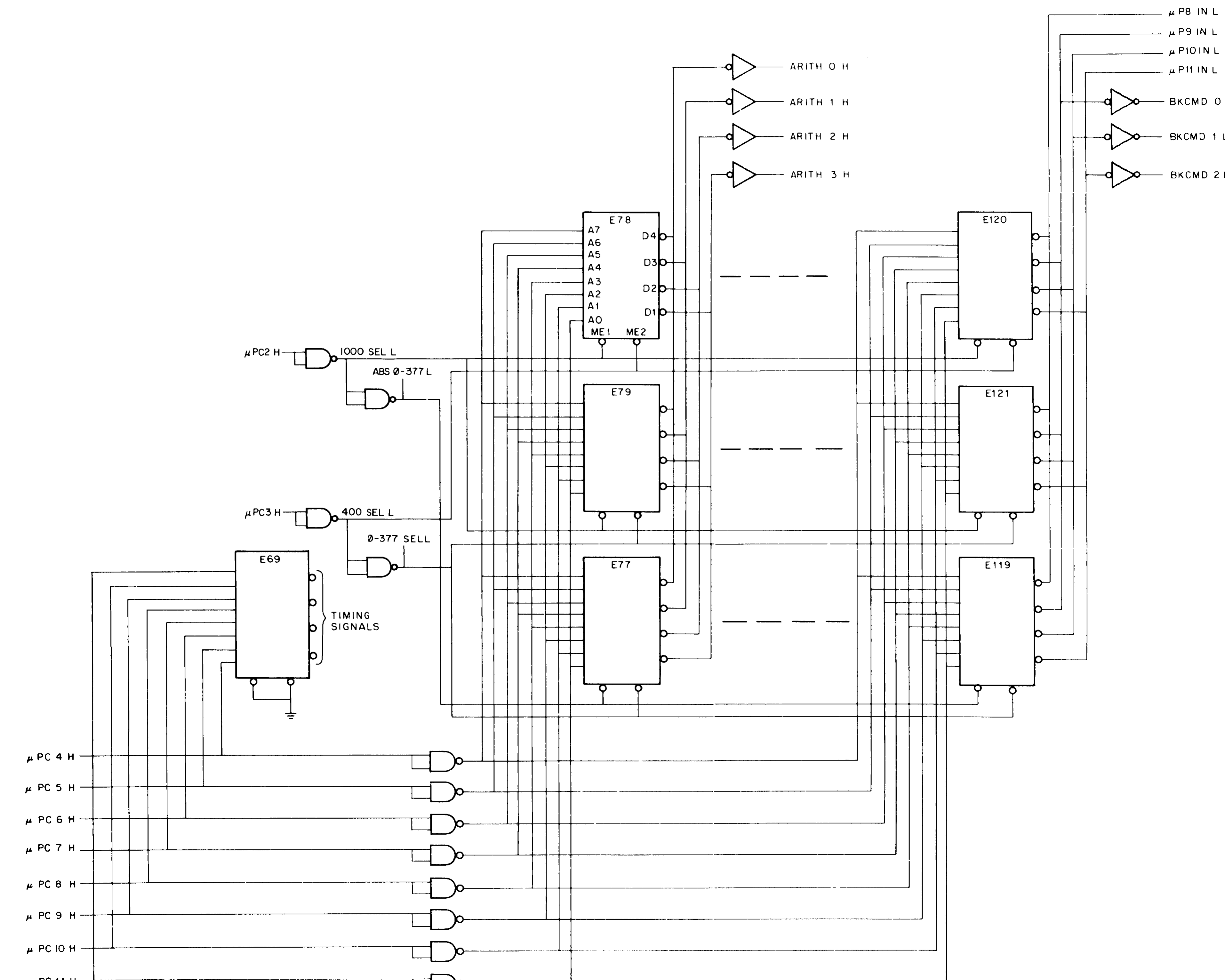


Figure 4-6 Control ROM Logic

Each row of PROMs furnishes 256 word locations. The rows are addressed by the μPC (2:3) H signals. Locations within each row are selected by the μPC (4:11) H signals. The relationship between μPC addresses and the enabled PROMs is given below.

μPC Addresses	μPC2 H	μPC3 H	PROMs Enabled
$0–377_8$	LO	LO	Bottom row (E77–E119) and E69
$1000–1377_8$	HI	LO	Middle row (E79–E121) and E69
$1400–1777_8$	HI	HI	Top row (E78–E120) and E69

Note that PROM E69 is enabled for all addresses. This PROM supplies timing signals that are used in the Clock logic; the timing signals are utilized only for μPC address 0-377. In the free-running area (addresses of 1000 and above), E69 is enabled but its outputs are irrelevant.

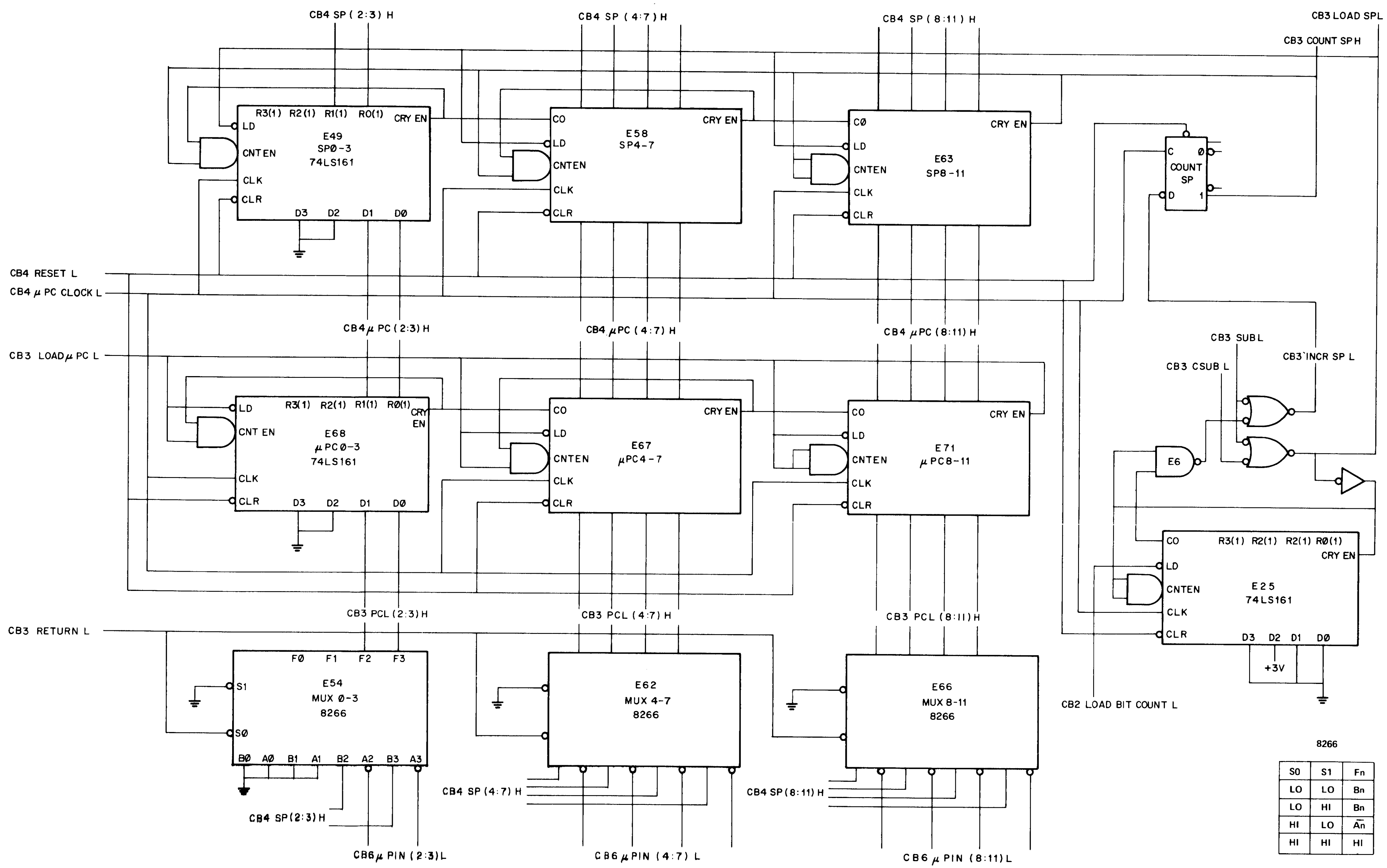
4.2.4 μPC/SP Register Logic

Figure 4-7 shows the μPC/SP Register logic. The μPC register (E68, E67, and E71) addresses the Control ROM locations. The register is supplied with address information from the SP register or from the μPIN (2:11) L lines; the latter source carries a jump address from the Control ROM or from the Instruction Dispatch logic.

The μPC register has two operating modes, namely, count and parallel-load. If the LOAD μPC L signal has been asserted by the μPC Gating Control logic, the register is parallel-loaded with the address information on the PCL lines. If LOAD μPC L is not asserted, the register is in the count mode and its contents are incremented at clock-pulse time.

The address information that appears on the PCL lines is either a jump address or a return address. A jump address is placed on the μPIN lines when an FPP instruction is decoded, when one of a number of possible tests on the data being manipulated proves to be true, when a new FPP instruction is to be fetched, or when control is to pass to a subroutine, with or without return to the departure point. In each of these instances, the multiplexers pass the jump address information to the μPC register and the asserted LOAD μPC L signal enables the address to be loaded by μPC CLOCK L.

When the information in the Control ROM present location directs that control jump from the present address to a subroutine and then return, the return address is saved in the SP register. When the subroutine has been completed, the μPC Gating Control logic asserts the RETURN L signal, thereby gating the return address from the SP register to the μPC register inputs. The return-address-save is accomplished by the logic that includes the COUNT SP flip-flop. For example, assume that the present location is μPC address 0050_8, and it calls for a jump to address 0060_8, with return. The signals [μPCTRL (0:4) L] asserted by the Control ROM cause the μPC Gating Control logic to assert the SUB L signal, which, in turn, asserts INCR SP L and LOAD SP L. The μPC CLOCK L pulse of address 0050 both loads address 0050 into the SP register and sets the COUNT SP flip-flop. The same pulse loads address 0060 into the μPC register. Then, the μPC CLOCK L pulse of address 0060 both increments the SP register and, because SUB L is now negated, clears the COUNT SP flip-flop. The last address in the subroutine will generate signals that cause RETURN L to be asserted; thus, address 0051 will be loaded into the μPC register and control will resume the main routine at that point.



8266

S0	S1	Fn
LO	LO	Bn
LO	HI	Bn
HI	LO	$\overline{An}$
HI	HI	HI

Figure 4-7 μPC/SP Register Logic

During multiply and divide operations, it is necessary to have certain functions carried out repetitively; that is, a subroutine is called 12 times in succession before the main routine is allowed to resume. In this circumstance counter E25 is activated. At some time prior to the subroutine call, the μPC Gating Control logic asserts the LOAD BIT COUNT L signal; the next clock pulse loads E25 with a starting count of 0100_2. When the subroutine is called, by address 0070, for example, the CSUB L signal is asserted and the μPC CLOCK L signal both loads the SP register with 0070 and increments counter E25. At the end of the subroutine, control returns to 0070; again, CSUB L is asserted, the SP is loaded, and E25 is incremented. This happens 12 times in succession. On the 12th assertion of CSUB L, E25 is incremented to a count of 0000 and generates a carry out that enables NAND gate E6. Hence, INCR SP L is asserted and the SP register is incremented by the clock pulse that occurs during the first address of the subroutine (the 12th occurrence of this address). When the subroutine finishes its 12th pass, control returns to address 0071 and the main routine resumes.

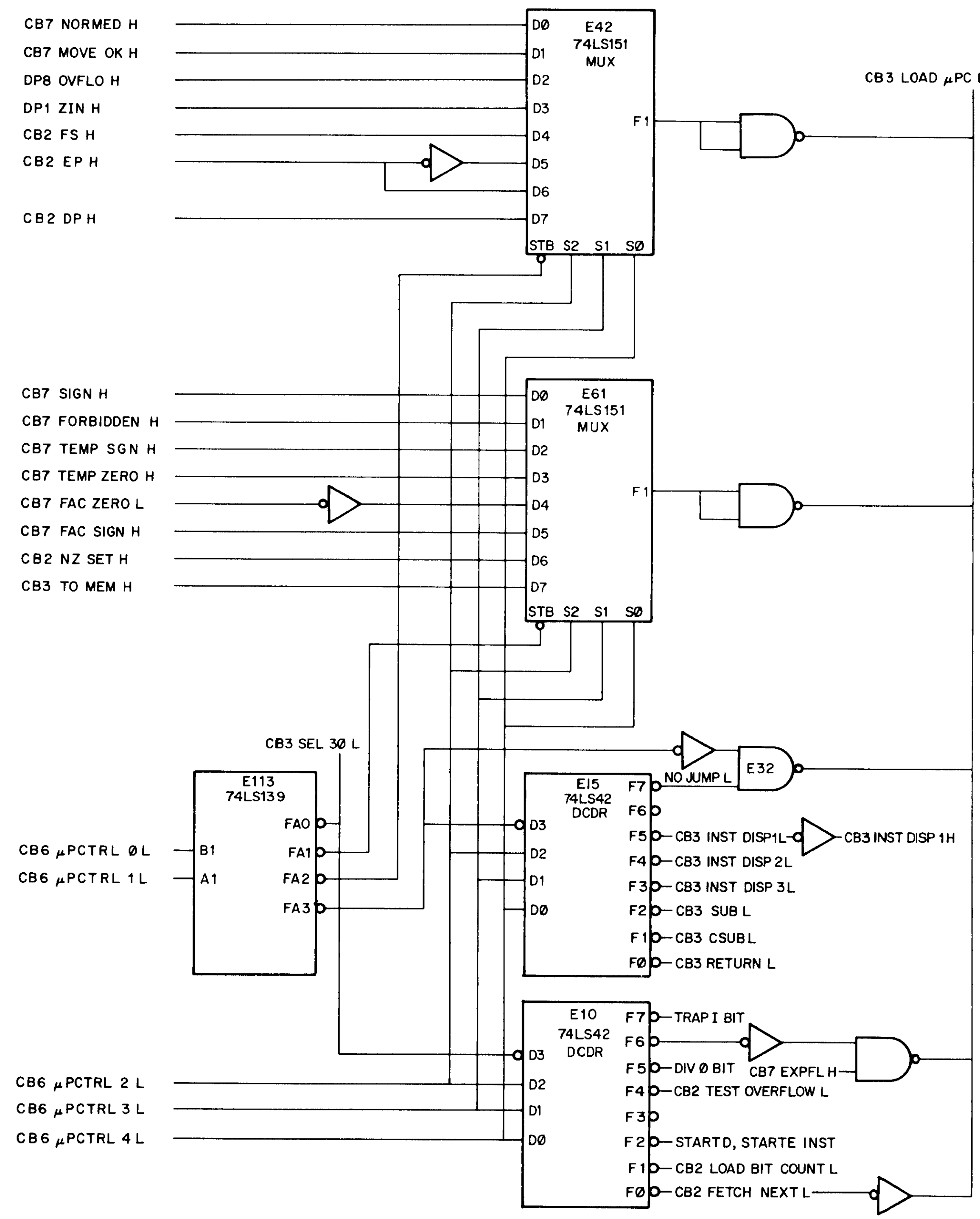
4.2.5 μPC Gating Control Logic

The μPC Gating Control logic is shown in Figure 4-8; its primary function is to control the loading of the μPC register. The logic includes two multiplexers, E42 and E61, and two decoders, E15 and E10, all of which are controlled by the μPCTRL (0:4) L signals. The two multiplexers cause LOAD μPC L to be asserted if a selected test condition is true. The two decoders also assert LOAD μPC L, but, with one exception, no conditions are attached to the assertion.

During an FPP operation various courses of action can be carried out by the logic, depending on the condition of the data at a selected moment. For example, after the addition of two floating-point numbers, the result must be normalized. However, if a test of the resulting data shows that it is already normalized, the normalization routine can be skipped, saving a great deal of time. To carry out such a test, the Control ROM asserts the signals necessary to load the data into a specific register. The condition of the data is reflected by the state of a flag, NORMED H in the case of a normalization test. To check the state of the flag, a subsequent Control ROM location asserts the μPCTRL (0:4) L signals so as to select input D0 of multiplexer E42 for testing; in addition, this location provides an address to which control jumps should the data be normalized. If the data is normalized, NORMED H is asserted; thus, LOAD μPC L is asserted, and the jump address is loaded into the μPC register.

All the signals that can be selected by the two multiplexers are listed following this paragraph; included is a brief description of the meaning of each signal. More detailed information concerning most of these signals can be found in the Register Flags logic.

Signal	Meaning (When Asserted)
NORMED H	The tested data is normalized.
MOVE OK H	The tested data is such that an entire 12-bit word can be shifted (the 13 MSBs of the word are all 0 or all 1).
OVFLO H	An overflow condition has occurred during a calculation.
ZIN H	A JNX instruction is being carried out; the contents of the addressed index register are 0.
FS H	A Fast Start has been programmed.
EP H	The EP calculating mode has been programmed (or, the EP mode has not been programmed).
DP H	The DP mode has been programmed.



74LS139

B1	A1	OUT LO
LO	LO	FA0
LO	HI	FA1
HI	LO	FA2
HI	HI	FA3

74LS42

D3	D2	D1	D0	OUT LO
LO	LO	LO	LO	F0
LO	LO	LO	HI	F1
LO	LO	HI	LO	F2
LO	LO	HI	HI	F3
LO	HI	LO	LO	F4
LO	HI	LO	HI	F5
LO	HI	HI	LO	F6
LO	HI	HI	HI	F7

74LS151

S2	S1	S0	STB	F1
X	X	X	HI	LO
LO	LO	LO	LO	D0
LO	LO	HI	LO	D1
LO	HI	LO	LO	D2
LO	HI	HI	LO	D3
HI	LO	LO	LO	D4
HI	LO	HI	LO	D5
HI	HI	LO	LO	D6
HI	HI	HI	LO	D7

08-1754

Figure 4-8 µPC Gating Control Logic

Signal	Meaning (When Asserted)
SIGN H	The sign of the tested data is negative.
FORBIDDEN H	A calculation has produced the result 4000 0000 (DP or FP mode; if EP, 36 additional 0s).
TEMP SGN H	The sign of the tested data is negative.
TEMP ZERO H	The tested data is 0.
FAC ZERO L	The data in the FAC fraction is 0.
FAC SIGN H	The sign of the data in the FAC is negative.
NZ SET H	Exponent underflow has not occurred, or a trap of a possible underflow has been programmed.
TO MEM H	The calculation result is to be transferred to memory.

Decoders E15 and E10 can assert the LOAD μPC L signal, thereby causing an unconditional jump to an address provided by the Control ROM, by the Instruction Dispatch logic, by the SP register, or by the Exit Test logic. Decoder E15 generates three output signals (at f0, f1, and f2) that are used in the μPC/SP register logic; three other output signals (at f3, f4, and f5) are used in the Instruction Dispatch logic. When any of these six signals is asserted, output f7 is high; hence, NAND gate E32 is enabled and LOAD μPC L is asserted. For example; assume that the Control ROM location directs E15 to assert INST DISP 1 L. This means that the Instruction Dispatch logic will decode an FPP instruction and place an appropriate address on the μPIN lines (refer to the μPC/SP register logic). The asserted LOAD μPC signal will enable the address to be loaded into the μPC register, and the first ROM location of the FPP instruction will be addressed.

When the μPC address is to be incremented to the next consecutive address, f7 of decoder E15 is asserted. NO JUMP L keeps LOAD μPC L negated, and the μPC register is placed in the count mode.

Decoder E10 can also assert LOAD μPC L. When the Control ROM location directs an FPP instruction fetch, output f0 is enabled; LOAD μPC L permits an address provided by the Exit Test logic to be loaded into the μPC register. Output f6 permits testing of the EXPFL flag; if the flag is set, indicating that the sign of the SC register is negative (refer to the Register Flags logic), the jump address provided by the ROM location is loaded in the μPC. The remaining outputs of E10 are applied to the Status register, the Exit Test logic, or the μPC/SP register logic, and do not directly affect LOAD μPC L.

4.2.6 Register Flags Logic

The FPP logic periodically checks the data being manipulated and follows a course of action that reflects the condition of the data. For example, during floating-point addition, the logic tests the fraction of both numbers. If either fraction is zero, many steps normally performed during an addition can be dispensed with.

The data is checked as it is placed on the OBUS (4:15) L lines. The logic shown in Figure 4-9 continually monitors the OBUS lines and generates six output signals that identify certain characteristics of the data word. These signals, except OVFLO H, are applied to the Register Flags logic, shown in Figures 4-10 and 4-11. The Register Flags logic records the characteristics of any data word that is being written into registers TEMP1 through TEMP5, SCRATCHM through SCRATCHS, FACM through FACS, or the SC, and generates an output signal that can be tested later.

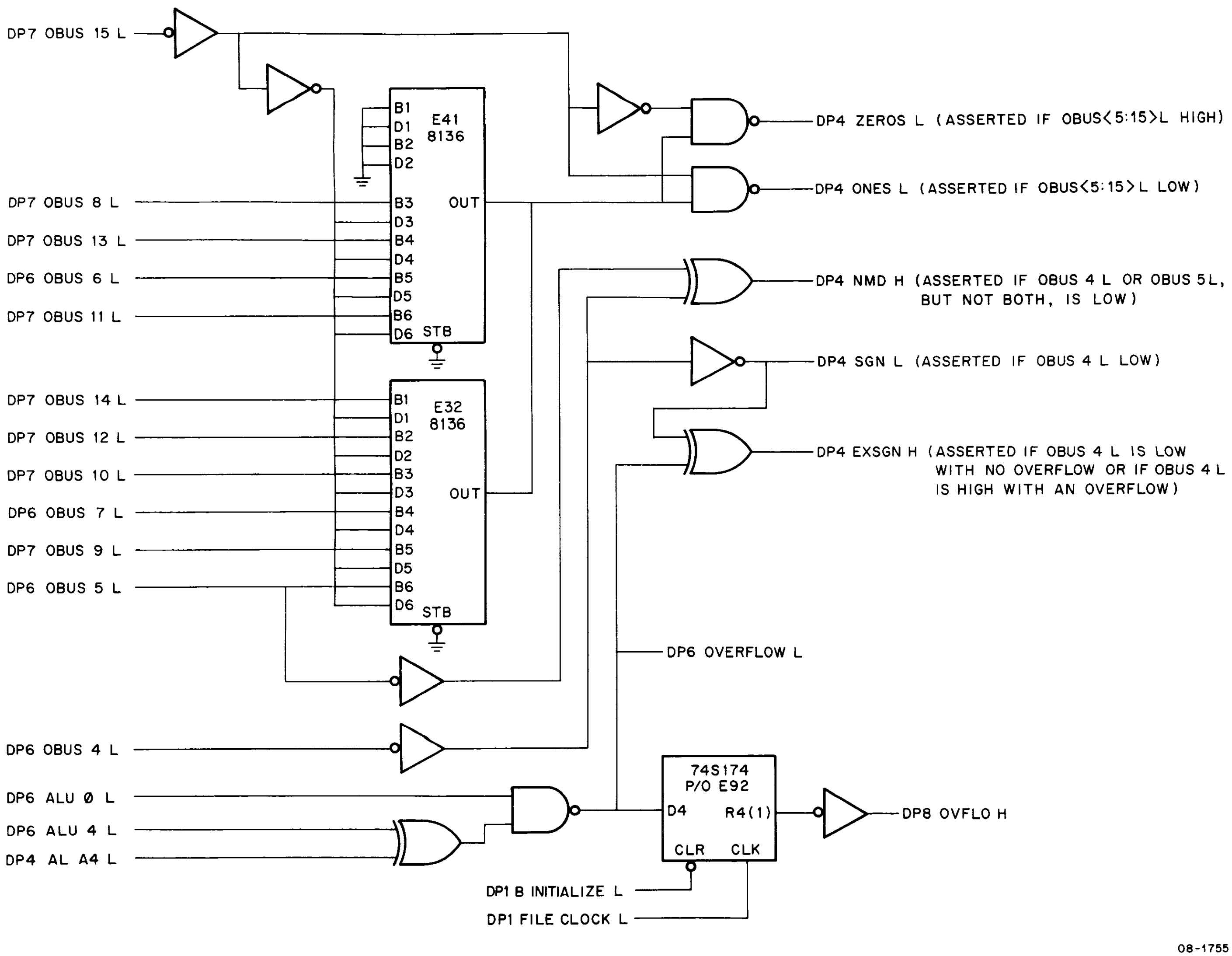
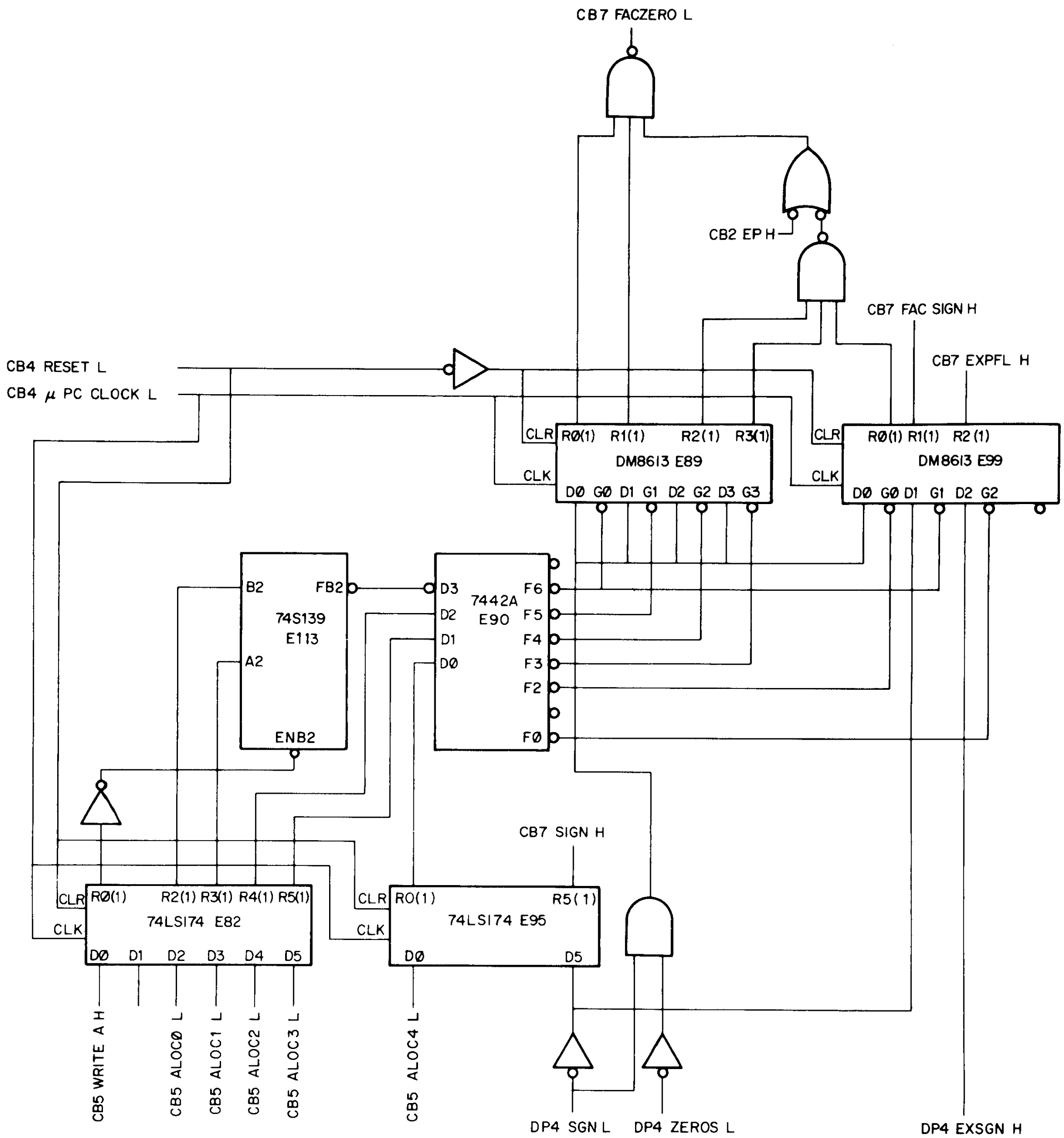


Figure 4-9 OBUS Flags



74S139

ENB2	B2	A2	OUT LO
H	–	–	–
L	L	L	FB0
L	L	H	FB1
L	H	L	FB2
L	H	H	FB3

7442A

D3	D2	D1	D0	OUT LO
L	L	L	L	F0
L	L	L	H	F1
L	L	H	L	F2
L	L	H	H	F3
L	H	L	L	F4
L	H	L	H	F5
L	H	H	L	F6
L	H	H	H	F7

08-1756

Figure 4-10 Register Flags (SC, FAC)

Consider Figure 4-10. The logic represented here records certain features of the data words that are written into the SC register or into the FAC register. The desired register is identified by the ALOC (0:4) L signals when WRITE A H is asserted. Decoder E90 then asserts an output signal that enables the particular characteristic of the data word to be retained in gated flip-flop E89 or E99. Table 4-6 relates the ALOC (0:4) L signals, the selected register and the meaning of the output signals generated. For example, when a data word is being written into the SC register, the ALOC (0:4) L signals cause decoder E90 to assert its f0 output. This enables the state of the EXSGN H signal (which characterizes the sign of the SC data word) to be loaded into flip-flop E99. The resulting EXPFL H signal can then be tested by the μPC Gating Control logic and, if true, causes a jump address to be loaded into the μPC. Or, if data is written into FACM, for instance, decoder E90 asserts its f6 output. The state of the sign bit, represented by SGN L, causes E99 to assert or negate FAC SIGN H. Thus, the sign of the FAC can be checked whenever necessary. One can also check to see if the FAC is all zeros by writing into FACM and FACN (for DP or EP) or FACM through FACS (for EP); the FAC ZERO L signal will be asserted if the FAC is, indeed, zero.

Now look at Figure 4-11. This logic records features of the data written into TEMP and SCRATCH registers. Table 4-7 relates the selected registers and the meaning of the output signals generated. Shown below is a portion of the firmware that includes a number of tests made on the flags listed in Table 4-7.

```
1227          DB, SCRATCHS : = TEMP                   FREE*
1230          DB, SCRATCHT : = [0]                    FREE*  GO TO, NMI1 (1201)
1231 NMI5,    DB, SCRATCHP : = [0]                    FREE*  GO TO, NMI1 (1201)
1232 NMI3,    DB, SCRATCHP : = [SHL]SCRATCHP          FREE*  IF NORMED, NMI3A (1236)
1233          DB, SCRATCHN: = [SHL][EXT]SCRATCHN      FREE*  TEST OVFLO
1234          DB, SCRATCHM: = [SHL][EXT]SCRATCHM      FREE*
1235          DB, SC: = SC[12BIT]M1                   FREE*  GO TO, NMI3 (1232)
1236 NMI3A,   DB, SCRATCHP : = [SHR][EXT]SCRATCHP     FREE*  TEST OVFLO
1237 NMI6,    NO OPERATION                            FREE*  IF FORBIDDEN, NMI8 (1250)
1240 RND,     NO OPERATION                            FREE*  IF EP, NMI7 (1247)
1241          NO OPERATION                            FREE*  IF TEMPSGN, RND1 (1254)
1242          DB: = SCRATCHP[12BIT]K3777+1            FREE*
1243 RND2,    DB, SCRATCHN: = SCRATCHN[12BIT][EXT]    FREE*
1244          DB, SCRATCHM: = SCRATCHM[12BIT][EXT]    FREE*  IF TEMPZERO, RND4 (1255)
1245          DB, SCRATCHP : = [0]                    FREE*  IF DP, NMI7 (1247)
1246          NO OPERATION                            FREE*  IF FORBIDDEN, OVREC (1405)
1247 NMI7,    NO OPERATION                            FREE*  RETURN
1250 NMI8,    DB, SC: = SC[12BIT]K1                   FREE*
1251          DB, SCRATCHM: = [SHR]SCRATCHM           FREE*
1252          NO OPERATION                            FREE*  TEST OVFLO
1253          NO OPERATION                            FREE*  RETURN
1254 RND1,    DB: = SCRATCHP[12BIT]K3777              FREE*  GO TO, RND2 (1243)
1255 RND4,    DB, SCRATCHP : = [0]                    FREE*  IF DP, NMI7 (1247)
1256          NO OPERATION                            FREE*  IF TEMPZERO, NMI7 (1247)
1257          NO OPERATION                            FREE*  GO TO, NM11 (1201)
```

Table 4-6 Registers Flags (SC, FAC)

WRITE A H	ALOC0 L	ALOC1 L	ALOC2 L	ALOC3 L	ALOC4 L	E90 OUT LOW	FILE A ADDRESS	ADDRESS ASSIGNMENT	Output Signals Possible
H	H	L	L	L	L	f0	17	SC	EXPFL H is asserted if the sign of the SC is negative and no overflow occurs, or if the sign is positive and an overflow occurs.
			L	H	L	f2	15	FACS [FAC (48:59)]	FACZERO L is asserted if ZEROS L is true and SGN L is false for each register (FACM and FACN for DP and FP modes, FACM through FACS for EP mode).
			L	H	H	f3	14	FACR [FAC (36:47)]	
			H	L	L	f4	13	FACP [FAC (24:35)]	
			H	L	H	f5	12	FACN [FAC (12:23)]	FACSIGN H is asserted if the sign of FACM is true (SGN L is low for FACM).
			H	H	L	f6	11	FACM [FAC (0:11)]	

Table 4-7 Register Flags (TEMP, SCRATCH)

WRITE A H	WRITE B H	B SEL L	ALOC (0:4) L	BWLOC (0:2) L	E117 OUT LO	File A Address	File B Address	Output Signals Possible	
X	H	H	X	HHL	f6	X	1 (TEMP1)	TEMP ZERO H	Used to check for zero fraction in TEMP1 and TEMP2 or SCRATCHM and SCRATCHN (DP and FP modes), and TEMP1 through TEMP5 or SCRATCHM through SCRATCHS (EP mode); asserted if SGN L is negated and ZEROS L is asserted.
			X	HLH	f5	X	2 (TEMP2)		
			X	HLL	f4	X	3 (TEMP3)		
			X	LHH	f3	X	4 (TEMP4)		
			X	LHL	f2	X	5 (TEMP5)		
H	X	L	LHHHL	X	f6	21 (SCRATCHM)	X		
			LHHLH	X	f5	22 (SCRATCHN)	X		
			LHHLL	X	f4	23 (SCRATCHP)	X	TEMP SGN H	Used to check the sign of the fraction in TEMP1 or SCRATCHM; asserted if SGN L is asserted.
			LHLHH	X	f3	24 (SCRATCHR)	X		
			LHLHL	X	f2	25 (SCRATCHS)	X		
								FORBIDDEN H	Used to check for the forbidden result 4000 0000 in TEMP1 and TEMP2 or SCRATCHM and SCRATCHN (DP and FP modes), and TEMP1 through TEMP5 or SCRATCHM through SCRATCHS (EP mode); asserted if SGN L is asserted for TEMP1/SCRATCHM, SGN L is negated for remaining TEMP/SCRATCH, and ZEROS L is asserted for all TEMP/SCRATCH.
								NORMED H	Used to check for normalized number; asserted if TEMP ZERO H is asserted (0 is a normalized number), or if NMD H is asserted for TEMP1/SCRATCHM.
								MOVE OK H	Used to check for possibility that an entire word can be shifted during normalization or alignment; asserted if ZEROS L asserted for TEMP1/SCRATCHM, and SGN L negated for TEMP1/SCRATCHM, and SGN L negated for TEMP2/SCRATCHN or ONES L asserted for TEMP1/SCRATCHM, and SGN L asserted for TEMP1/SCRATCHM, and SGN L asserted for TEMP2/SCRATCHN.

X = Don't Care

Remember that a test comes at least two steps after the data is loaded into the register in question. For example, the FORBIDDEN test in step 1246 is testing the data loaded into SCRATCHM and SCRATCHN in steps 1244 and 1243, respectively (the asterisk in the timing statement indicates that SCRATCH registers, rather than TEMP registers, are being checked).

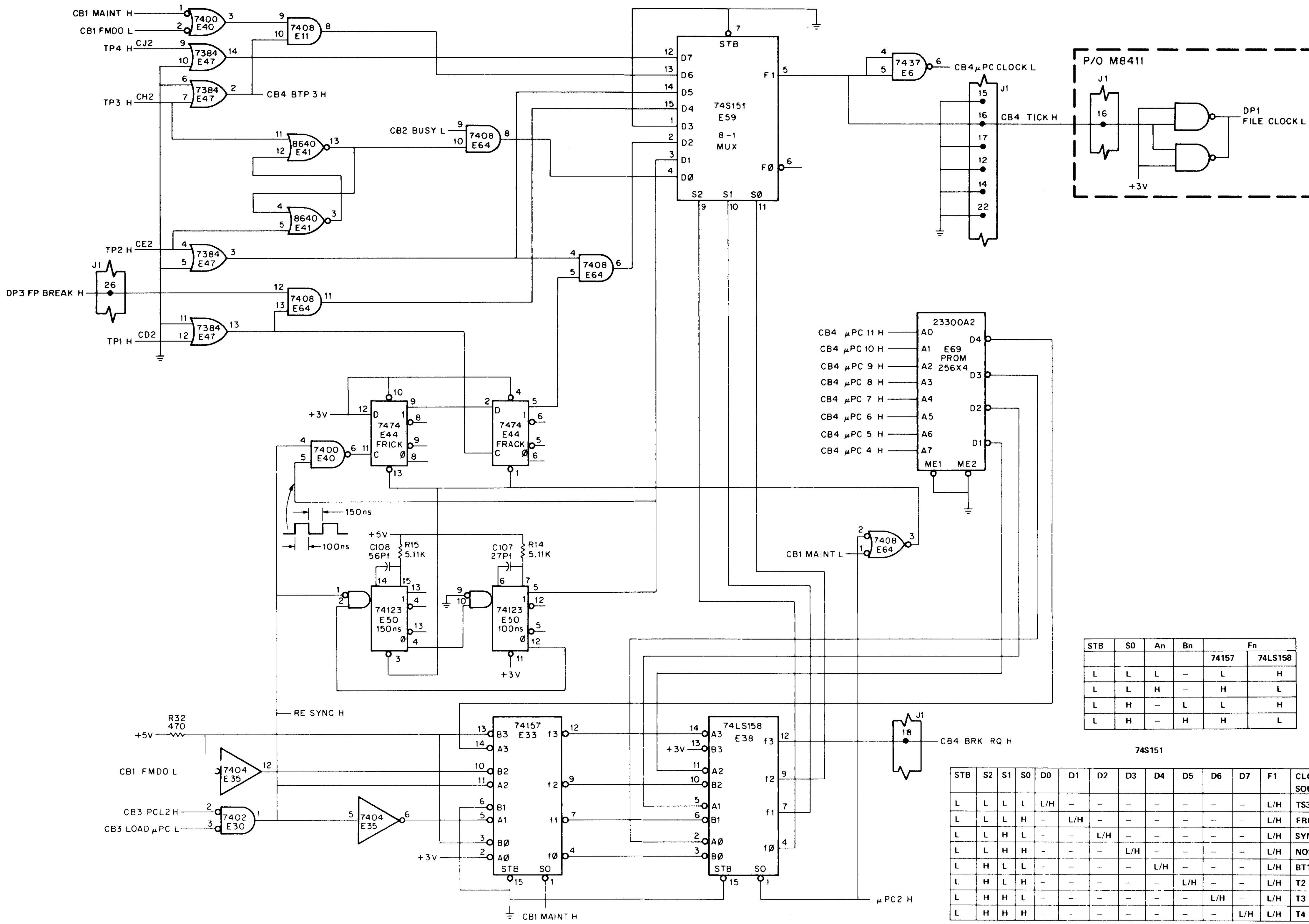
4.2.7 Clock Logic

The FPP8-A operates within the PDP-8 I/O transfer scheme; i.e., it uses programmed-I/O data transfers, program interrupts, and data break transfers to accomplish its tasks. Consequently, FPP timing must be synchronized with PDP-8 timing. On the other hand, the FPP is capable of faster, independent operation, as when it is performing a series of calculations in response to an initial FPP instruction. Thus, two methods of timing FPP operations are used: PDP-8 timing pulses (TP1 H through TP4 H) control operations for all Control ROM addresses below 1000_8 (i.e., in the IOT and data break area); and, an FPP free-running clock controls operations for addresses above, and including, 1000_8 (the terms "Control ROM address" and "μPC address" are synonomous and both are used throughout).

The Clock logic, shown in Figure 4-12, generates the μPC CLOCK L timing signal, which is used in the Control logic, and the FILE CLOCK L timing signal, which is used in the Data Path logic. Each of these signals is derived from TICK H, which is the output of the 8-to-1 multiplexer, E59. The outputs of another multiplexer, E38, control the source of TICK H. Basically, there are two sources, viz., the free-running clock and the PDP-8 timing pulses. The free-running clock is used as the source when the Control ROM address is 1000_8 or above, i.e., when μPC2 H is high. When this signal is asserted, E38 gates its Bn inputs (except B3) to the control inputs of E59. The Bn inputs, except B3, are taken, in turn, from another multiplexer, E33. If the FPP is *not* in the maintenance mode (i.e., if MAINT H is low), E33 gates its An inputs (except A3) to the Bn inputs of E38. Assume, for the moment, that the RE SYNC H signal is low; thus, the Bn inputs of E38 exhibit the following logic levels: B0 is high; B1 is high; B2 is low; and, B3 is high. The first three levels are inverted and applied to control inputs S2, S1, and S0, respectively, of E59 (output f3 of E38 has significance only in the data break area of addresses); input D1, which is taken from monostable-multivibrator (MV) E50, the free-running clock, is selected as the source of TICK H.

On the other hand, if the ROM address is below 1000_8, a timing pulse is selected as the source of TICK H. Because the specific pulse selected depends on the particular ROM address, the ROM takes part in the selection process. Thus, when μPC2 H is low E38 gates its An inputs (except A3) to the control inputs of E59; the An inputs (except A3) are taken directly from PROM E69, which is part of the Control ROM and which provides outputs in response to address $0–377_8$. If, for example, the address is one that directs Control operations to take place at TP2 time, the An inputs of E38 exhibit the following logic levels: A0 is low; A1 is high; and A2 is low (ignore A3 at present). These levels are inverted and applied to control inputs S2, S1, and S0, respectively, of E59; input D5, which reflects the state of the TP2 H signal, is selected as the source of TICK H. Any Control ROM address that directs an operation to take place at TP2 H time, has T2 listed in its timing statement (i.e., the entry under "Time") in the firmware. Similarly, operations taking place at TP1 H, TP3 H, or TP4 H time have BT1, T3, or T4, respectively, listed in the firmware (BT1 merely states that an FPP data break must be in progress). Table 4-8 relates Control ROM addresses to the selected input of multiplexer E59, describing the addresses below 1000_8 (except 0, 1, and 2) in terms of the timing statement.

As Table 4-8 indicates, addresses 0, 1, and 2 use TP2 H and TP3 H to generate a TICK H pulse that is somewhat different from that generated for other addresses. Figure 4-13 illustrates some Control signals during the FPP initialization procedure and shows how TICK H is generated during address 0. Assume that the μPC has recently been cleared (either by the FPICL IOT instruction or by the Omnibus INITIALIZE H signal). Until the initializing instructions, FPCOM and FPST, are issued, the μPC address will alternate between 0 and 3. The TICK H signal will be generated by TP2 H and TP3 H during address 0, and by TP4 H during address 3.



STB	S0	An	Bn	Fn	
				74157	74LS158
L	L	L	–	L	H
L	L	H	–	H	L
L	H	–	L	L	H
L	H	–	H	H	L

74S151

STB	S2	S1	S0	D0	D1	D2	D3	D4	D5	D6	D7	F1	CLOCK SOURCE
L	L	L	L	L/H	–	–	–	–	–	–	–	L/H	TS3
L	L	L	H	–	L/H	–	–	–	–	–	–	L/H	FREE
L	L	H	L	–	–	L/H	–	–	–	–	–	L/H	SYNCED T2
L	L	H	H	–	–	–	L/H	–	–	–	–	L/H	NONE
L	H	L	L	–	–	–	–	L/H	–	–	–	L/H	BT1
L	H	L	H	–	–	–	–	–	L/H	–	–	L/H	T2
L	H	H	L	–	–	–	–	–	–	L/H	–	L/H	T3
L	H	H	H	–	–	–	–	–	–	–	L/H	L/H	T4

08-1758

Figure 4-12 Clock Logic

Table 4-8 Clock Timing Sources

Control ROM Address	Selected E59 Input	Remarks
Below 1000_8		
0, 1, and 2	D0	Starting address is set by IOT Decoding logic at TP2 time; loaded into μPC at TP3 time.
Addresses having BT1 in timing statement	D4	Using TP1 H.
Addresses having T2 in timing statement	D5	Using TP2 H.
Addresses having T3 in timing statement	D6	Using TP3 H. Used in both normal mode and when single-stepping in maintenance mode (MAINT H asserted).
Addresses having T4 in timing statement	D7	Using TP4 H.
1000_8 and above	D1	Using free-running clock. (Also used when carrying out maintenance firmware as long as MAINT H is not asserted.)
1000_8 and above	D6	Using TP3 H. Used when single-stepping in maintenance mode (MAINT H is asserted).
Selected 1000_8-and-above addresses when going back below 1000_8.	D2	Using TP2 H to generate TICK H.

The initializing procedure begins when a PDP-8 TAD instruction loads the AC register with the information listed under the FPCOM instruction (bits 9–11 represent the field address of the APT pointer). The FPCOM instruction follows the TAD instruction (if not, the AC register must not change until FPCOM is issued), causing the AC contents to be loaded into the FPP Command register (bits 0–8) and the FPP DB register (all 12 DB bits are loaded, although only bits (9:11) will later be used). When FPCOM is issued the IOT decoding logic asserts μP9 IN L. Since address 0 has already asserted both μP10 IN L and μP11 IN L, address 7 is loaded into the μPC by the same TICK H pulse that loads the DB register. At the next TP4 H pulse, the μPC reverts to address 0 and the field bits are transferred from the DB register to the FIELD register. Another TAD instruction follows (not necessarily immediately after FPCOM) and loads the AC with the relative address bits of the APT pointer. This address is transferred to the DB register, from there, along with the field address, to the APTP register, and, finally, to the BKMA register. Figure 4-13 describes graphically how this is accomplished.

Note that when the μPC address becomes 300_8 the BRK RQ H signal is asserted. Each address below 1000_8 that directs the Data Path logic to load its BKMA register, also causes the BRK RQ H signal to be asserted (address 300_8 directs that the BKMA register be loaded, address 301_8 provides the pulse that actually loads the register). Then, the data break system acknowledges the request at TP3 H time, negating BRK RQ H, and begins a priority check; if the FPP has priority, the BKMA is loaded at TP4 H time of the following address. When address 300_8 (or any address having T3 in its timing statement) is loaded into the μPC, output D4 of PROM E69 goes low (Figure 4-12). When the FPP is *not* in the maintenance mode (i.e., MAINT H is low), the input at A3 of E33 causes BRK RQ H to be asserted at output f3 of E38. At the following TP3 H time the break request is acknowledged and TICK H is generated. Certain addresses below 1000_8 can be used in the maintenance mode. In such circumstances multiplexer E33 selects its Bn, rather than An, inputs, and the BRK RQ H signal is asserted as a result of a low input at Bn of E33. This low is provided by the FMDO L signal, which is asserted when the FMDO instruction is decoded.

When the FPP has retrieved the APT it begins executing FPP instructions at the address specified by the FPC. Figure 4-14 shows the Clock logic timing as it appears during an assumed sequence of operations that begins with the fetch of the FCLA instruction (Clear the FAC). A portion of the firmware is shown below. This relates to the μPC addresses in Figure 4-14 and is included for reference.

```
20,     HHHH HHHH HHHH HHXX XHHH HHHL HHHH HHHH HHHH HHHH LLLH
21,     LHLH HLHH HHLL LLXX XHHH HHHL HHHH HHHH HHHH HLLL HLLL
22,     HHHH HHHH HHHH HLXX XHLH LLHL HHHH HHHH HHHH HHHH HLHH
23,     HHHH XXXX XLLH LHHH HHHL HHHH HHHL HHHH HHHH HHHH HLHL
1002,   HHHH XXXX XHHH HHXX XHHH HHHH HHHH LLLH HHLH LHHH
1050,   HLLH HLHH LHHH HHXX XHLH LHHH HLHH HLLH HHLH LLHL
1055,   HLLH HLHL HHHH HHXX XHLH LHHH LLLL LLHH HHHH HHHH
72,     HHHH HHHH LLLH LHHH HHHL HHHL HHLH LHHH LHHL LLHL LLLH
```

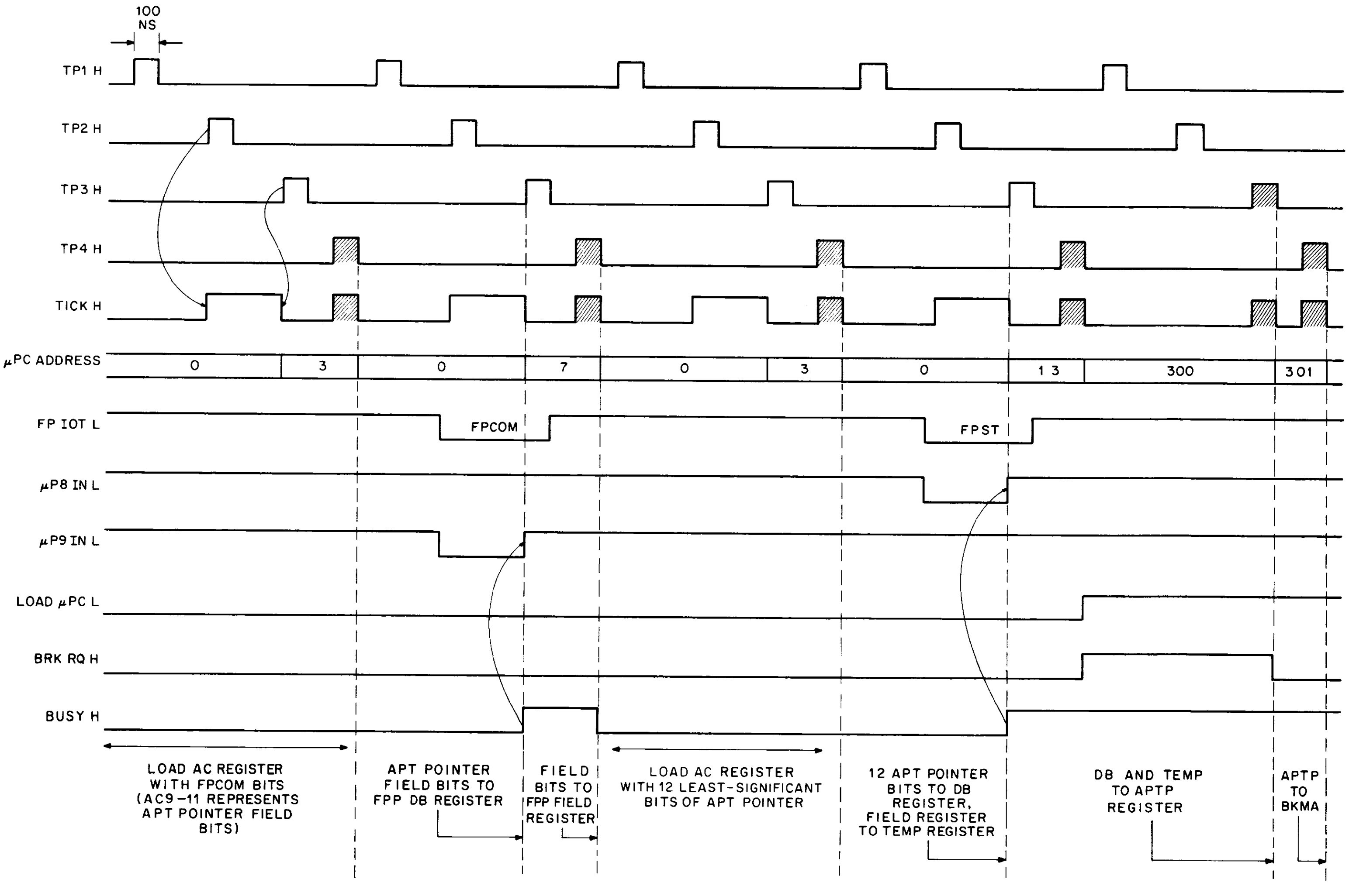


Figure 4-13 Start-Up Timing

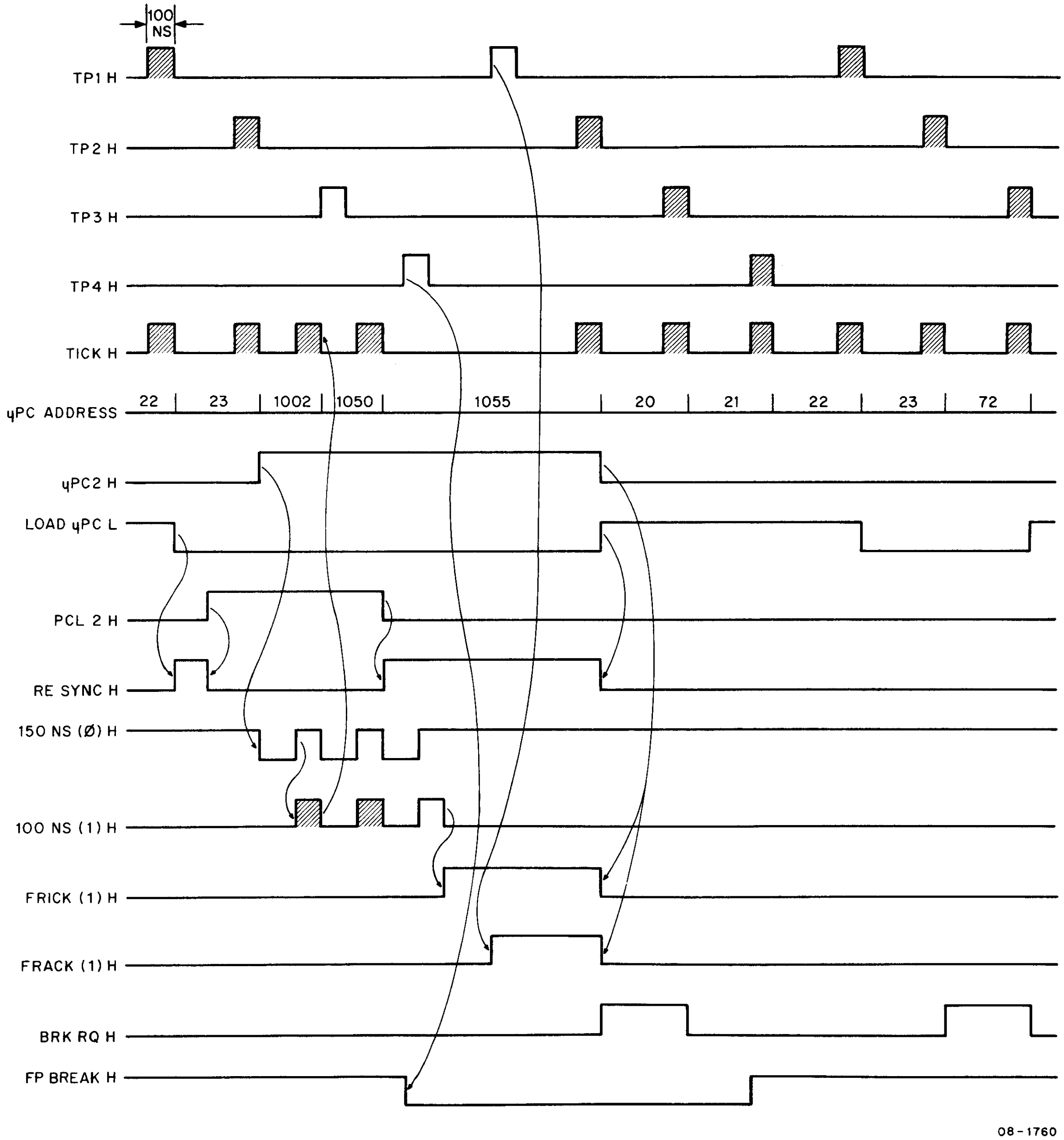


Figure 4-14 FCLA Timing

```
        *20
20      FETCH,    BKMA: = FPC                    T3
21      FETCH1,   : = FACE[EXPSIZE] M30          T4      BKCMD: = 7
22                FPC: = FPC[+] K1; DB: = MD     BT1
23                TEMP: = FIR(9:11)              T2      INSTR DISP 1

        *1002
        /CLEAR FAC
1002    FCLA,     NO OPERATION                   FREE*   GO TO, CLRFAC (1050)

        *1050
1050    CLRFAC,   DB, FACM: = [0]                FREE*   IF DP, CLRF1 (1055)
1055    CLRF1,    DB, FACN: = [0]                FREE*   EXTEST
        /XTA
        *72
72      XTA,      BKMA, TEMP: = X0[+] FIR(9:11)  T3      SUB, GETXR (235)
```

The timing shows that a previously requested data break has been granted (FP BREAK H is high). The FCLA instruction is placed on the MD lines during TS2 L of the FPP's data break cycle. The instruction is decoded by the Control's PLA and address 1002_8 is loaded into the μPC by the TICK H pulse generated at TP2 H time. The same TICK H pulse causes the μPC2 H signal to go high, removing the ground from the "clear" input (pin 3) of MV E50 (Figure 4-12), and from the clear input of both E44 flip-flops, FRICK and FRACK. The RE SYNC H signal is low at this time; consequently, E50 begins to run free. Furthermore, the low RE SYNC H signal results in multiplexer E59 selecting the free-running clock output as the source of TICK H. This source remains selected as long as RE SYNC H remains low.

During address 1050_8, the FPP tests to determine the operating mode; the timing assumes DP. In address 1055_8, an exit test is made. The next μPC address will be either 1000_8 (if an exit is to be made) or 0020_8 (if a new instruction is to be fetched). The example assumes that a new instruction is to be fetched; hence, the next address to be placed in the μPC is 0020_8, which uses Omnibus timing pulses to generate TICK H pulses. Whenever control of the timing of FPP operations is to pass from the free-running clock to the Omnibus timing pulses, a resynchronization takes place. This procedure begins when the PCL 2 H signal goes low, an event that occurs each time an address below 1000_8 is about to be loaded into the μPC, as is the case illustrated by the timing (if an exit were to be made in this example, rather than a new fetch, the PCL 2 H signal would return high in a matter of nanoseconds; thus, the third 100NS (1) H pulse would generate a TICK H pulse that would cause 1000_8 to be loaded into the μPC). When PCL 2 H goes low, the RE SYNC H signal goes high. This action, first, changes the control inputs of multiplexer E59, so that the free-running clock is removed as the source of TICK H pulses and, second, ensures that the clock will stop running after one more cycle of operation.Multiplexer E59 now selects input D2 to be the source. This source generates a TICK H pulse at TP2 H time if the FRACK flip-flop is set. As the timing shows, FRICK is set when MV E50 stops and FRACK is set by the following TP1 H pulse. At the next TP2 H pulse, the FPP operations are resynchronized with the Omnibus timing and a new instruction fetch operation begins.

4.2.8 Instruction Dispatch Logic

When FPP instructions are fetched they are decoded by the Instruction Dispatch logic. The logic generates μPC input signals that force the appropriate Control ROM location to be addressed.

Figure 4-15 shows the Instruction Dispatch 1 logic. Instruction Dispatch 1 is the primary level of instruction decoding; it decodes the Special instructions and points to the address calculation addresses for the Data Reference instructions. The part of the firmware that deals with instruction fetch and Instruction Dispatch 1 is included below for reference.

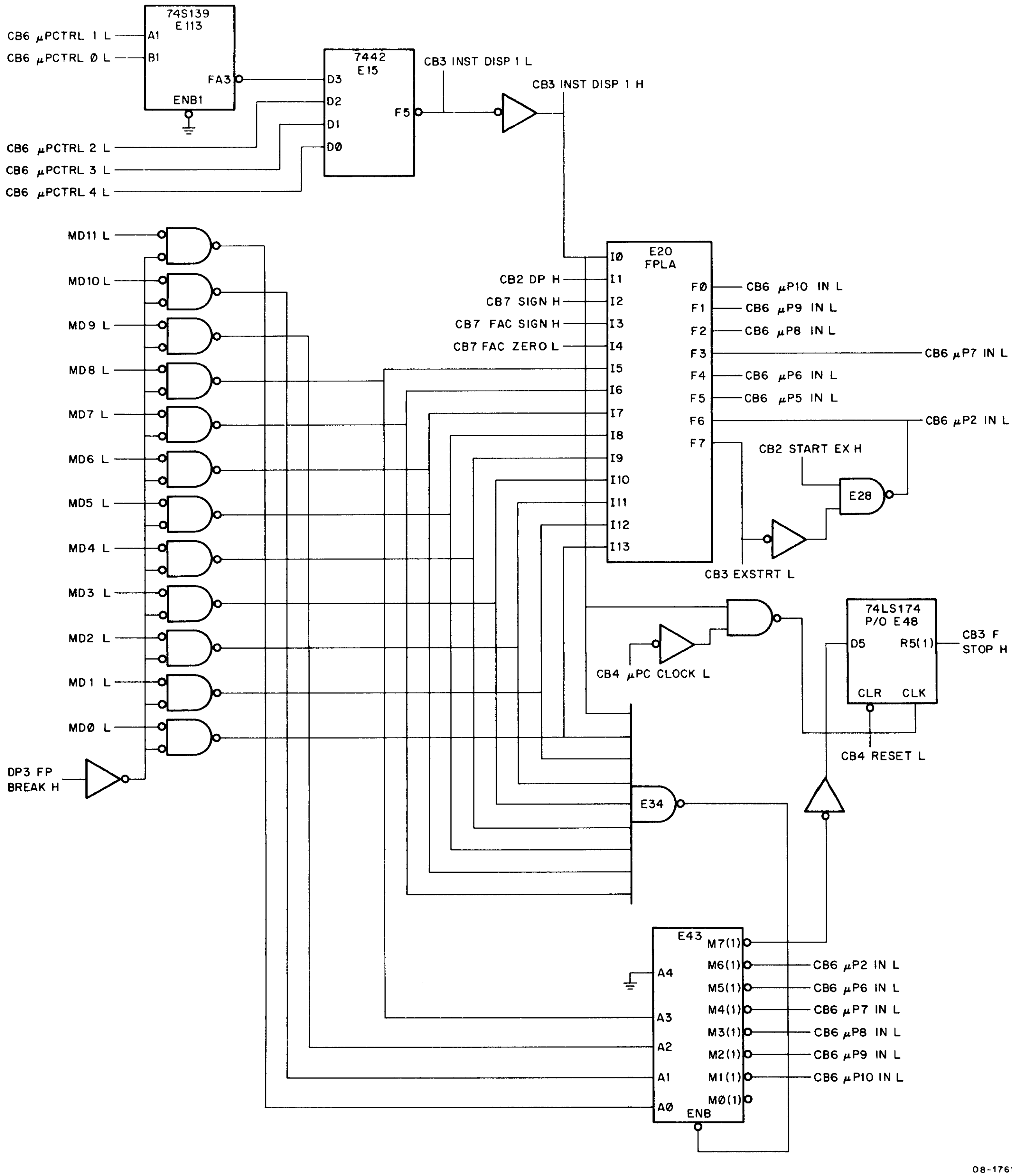


Figure 4-15 Instruction Dispatch 1 Logic

```
    *20
20  FETCH,  BKMA: = FPC                                 T3
21  FETCH1, : = FACE[EXPSIZE] M30                       T4          BKCMD: = 7
22          FPC: = FPC[+] K1;  DB:=MD                   BT1
23          TEMP: = FIR(9:11)                           T2          INSTR DISP 1
    /INSTRUCTION DISPATCH 1 DISPATCHES MICRO PC AS FOLLOWS:
    /       INSTRUCTION        ADDRESS    INSTRUCTION        ADDRESS
    /       SETX               34         SETB               36
    /       LDX                40         ADDX               44
    /       JSA                50         JSR                60
    /       BRANCH (TRUE)      14         BRANCH (FALSE)     24 AND EXTEST
    /       TRAP               74         JNX                26
    /       ALN (NOT XR0)      70         ALN (XR0)          1030
    /       XTA                72         ATX                1040
    /       LTR(0)             1026       LTR(1)             1016
    /       JAC                1014       FNORM              1006
    /       FNEG               1004       FCLA               1002
    /       FPAUSE             2          FEXIT              1000
    /       STARTF             1010       STARTD             1012
    /       STARTE             1020
    /       ALL UNDEFINED      EXTEST
    /ALL DATA REFERENCE INSTRUCTIONS (LEA, LEAI, FLDA, FADD, FSUB, FDIV,
    /FMUL, FADDM, FSTA, AND FMULM) DO ONE OF THE FOLLOWING ADDRESS CALC:
    /       ADDRESS MODE                                LABEL       ADDRESS
    /       12 BIT DIRECT (NOT DP)                      DIRFP       100
    /       12 BIT DIRECT (DP)                          DIRDP       102
    /       24 BIT, NO INCR, NO INDEX                   NINC24      114
    /       24 BIT, INCR, NO INDEX                      INC24       112
    /       24 BIT, INDEXED                             X24         110
    /       12 BIT INDIRECT, NO INCR, NO INDEX          INDIR       134
    /       12 BIT INDIRECT, INCR, NO INDEX             INCIND      132
    /       12 BIT INDIRECT, INDEXED                    XIND        130
    ////IN ADDITION, GATING IN MAJOR REGISTERS CAUSES THE FOLLOWING:
    ////////INSTRUCTION                                 OPERATION
    ////////DIRECT 12-BIT ADDRESSING                    TEMP: = 3*FIR(5:11)
    ////////INDIRECT ADDRESSING (ALSO LEAI)             TEMP: = 3*FIR(9:11)
    ////////ALL OTHER INSTRUCTIONS, 24-BIT
    ////////    ADDRESSING MODE                         TEMP: = [R3R] FIR(9:11)
```

At BT1 time, when Control ROM location 23_8 is addressed, control signals μPCTRL (0:4) L cause INST DISP 1 L to be asserted by decoder E15 (Figure 4-15). When the FPP instruction is placed on the MD lines early in TS2, selected MD (0:11) L signals are gated to the decoding elements, E20 and E43. If the FPP instruction has MD (0:7) L negated, NAND gate E34 enables PROM E43 to decode MD (8:11) L. The μPC is dispatched by E43 as indicated in Table 4-9.

When the FEXIT instruction is dispatched, Control ROM address 1000_8 begins the exit sequence. The APT is stored and the FPP halts in address 1 with the interrupt flag raised. The FSTOP H signal, which is asserted when FEXIT is dispatched, ensures that the FEX H flag in the Exit Test logic is cleared; thus, FEXIT is recorded as being the reason for the exit operation.

Table 4-9 PROM E43 Input/Output Signals

PROM Code	Input Signal Low					Output Signal Asserted Low							Control ROM Address	FPP Instruction
	A4 (GND)	A3 (MD8 L)	A2 (MD9 L)	A1 (MD10 L)	A0 (MD11 L)	μP_ IN L 2	6	7	8	9	10	M7(1)		
0	X	X	X	X	X		X	X	X				0070	ALN (XR=7)
1	X	X	X	X			X	X	X				0070	ALN (XR=6)
2	X	X	X		X		X	X	X				0070	ALN (XR=5)
3	X	X	X				X	X	X				0070	ALN (XR=4)
4	X	X		X	X		X	X	X				0070	ALN (XR=3)
5	X	X		X			X	X	X				0070	ALN (XR=2)
6	X	X			X		X	X	X				0070	ALN (XR=1)
7	X	X				X		X	X				1030	ALN (XR=0)
10	X		X	X	X	X			X	X			1014	JAC
11	X		X	X		X			X		X		1012	STARTD
12	X		X		X	X			X				1010	STARTF
13	X		X			X				X	X		1006	FNORM
14	X			X	X	X				X			1004	FNEG
15	X			X		X					X		1002	FCLA
16	X				X						X		0002	FPAUSE
17	X					X						X	1000	FEXIT

All other Instruction Dispatch 1 operations are initiated by E20, a Field Programmable Logic Array (FPLA). Table 4-10 relates the input and output signals of the FPLA and indicates how each instruction dispatches the μPC.

Each of the Branch instructions causes a jump to a designated μPC address if the condition specified in the instruction is met. The stated condition always involves the contents of the FAC; the three signals FAC ZERO L, FAC SIGN H, and SIGN H allow the FPLA to test the FAC contents to determine if the condition is met. If the condition is satisfied, μPC address 14_8 is dispatched; the table entry in the right column states that the branch condition is true and notes what the condition is. For example, in the first entry of the branch instructions, the jump is made because the tested condition (FAC must be zero) is true. If the condition is not met, a different address is dispatched; then the right-column entry states that the branch condition is false and why. For example, in the first branch-false entry, the jump is not made because the tested condition (FAC must be zero) is false, the FAC being not equal to zero.

A branch-false condition can dispatch one of two μPC addresses. During branch-false, the FPLA asserts the EXSTRT L signal along with μP7 IN L and μP9 IN L. During normal operation, the EXSTRT L signal is ignored, μPC address 24_8 is dispatched, and a new FPP instruction is fetched. However, if the single-cycle mode of operation has been programmed (FPHLT is issued prior to FPST), the FPHLT instruction has caused the Exit Test logic to assert START EX H. Consequently, μPC address 1024_8 is dispatched and an exit operation is started at the end of the branch instruction. If EXSTRT L were not asserted, the exit would occur at the end of the instruction following the branch instruction; thus, two FPP instructions, rather than just one, would have been performed.

Two of the table entries test the SIGN H signal. These entries deal with the JAL instruction, which tests a floating-point number to determine if the fraction can be fixed, i.e., converted to an integer. Should the fraction exponent be greater than 27_8, the number cannot be fixed. During the fetch of any FPP instruction, the FAC exponent is examined by the ALU. μPC address 21_8 causes the FAC exponent and -30_8 to be gated to the ALU. If the sign of the exponent is positive, the exponent and -30_8 are added in the ALU. Should the exponent be greater than 27_8, the addition produces a result that leaves OBUS 4 L high. Thus, SGN L is high and SIGN H is low. If the instruction that has been fetched is JAL, the negated SIGN H signal causes a branch true operation to be carried out. But, when the exponent is less than 27_8, or is negative [in this case 7777_8 is placed on the OBUS (4:15) L lines], SIGN H is asserted, causing a branch-false condition.

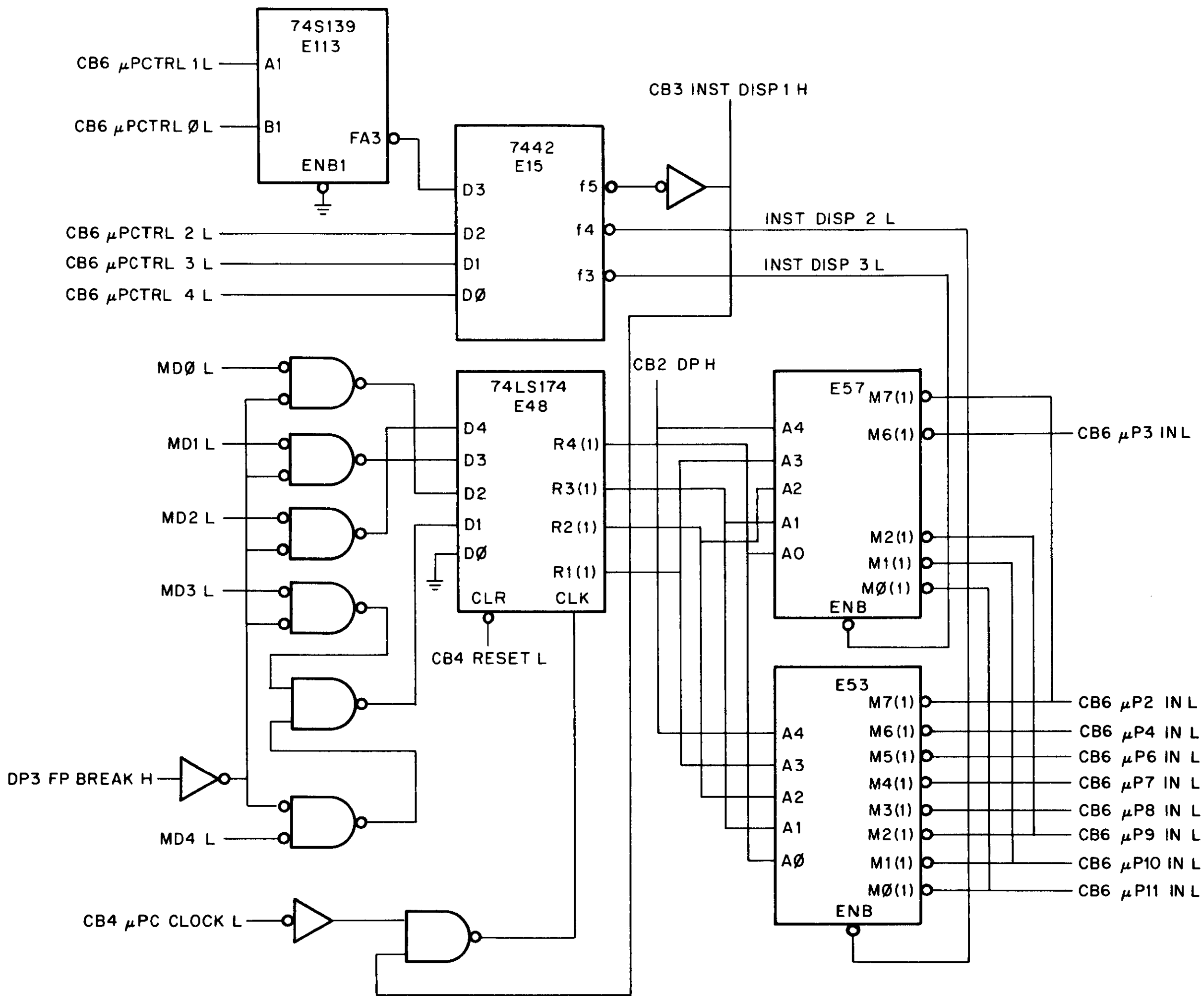
Figure 4-16 shows the Instruction Dispatch 2 and 3 logic. Instruction Dispatch 2 (related firmware shown below) decodes the FLDA, FSTA, and LEA Data Reference instructions, and points to preliminary arithmetic routines that must be carried out prior to Instruction Dispatch 3, which dispatches the purely arithmetic Data Reference instructions.

```
        /DIRECT ADDRESS CALCULATION
        *100
100     DIRFP,   BKMA, TEMP1: = TEMP[+]BR;  DB: = 0          T3     INSTR DISP 2

        /DP CALCULATION ADDS 1 BECAUSE BASE PAGE ALWAYS CONTAINS 3-WORD ARG.
        *102
102     DIRDP,   BKMA, TEMP1: = TEMP[+]BR+1;  DB: = 0        T3     INSTR DISP 2
```

Table 4-10 FPLA I/O Signals

Input Signal Logic Level (no entry implies 'don't care')														Outputs Asserted Low (no entry indicates the output is disconnected for the related input conditions)								FPP Instruction Represented by Input Signals
I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	
H	L	H	H	H	-	-	-	-	-	-	-	-	H	-	-	-	-	L	-	L	L	JNX
H	H	L	H	H	L	H	H	H	-	-	-	-	H	-	-	-	-	L	L	L	-	SETX
H	H	L	H	H	L	H	H	L	-	-	-	-	H	-	-	-	-	L	L	L	L	SETB
H	H	H	H	H	L	H	H	H	-	-	-	-	H	-	-	-	L	-	-	-	-	LDX
H	H	H	H	H	L	H	H	L	-	-	-	-	H	-	-	-	L	-	-	L	-	ADDX
H	H	L	H	H	L	H	L	H	-	-	-	-	H	-	-	-	L	-	L	-	-	JSA
H	H	L	H	H	L	H	L	L	-	-	-	-	H	-	-	-	L	L	-	-	-	JSR
H	H	H	H	H	H	H	L	H	-	-	-	-	H	-	L	-	L	-	-	-	-	ATX
H	H	H	H	H	H	H	L	L	-	-	-	-	H	-	-	-	L	L	L	-	L	XTA
H	L	L	H	H	-	-	-	-	-	-	-	-	H	-	-	-	L	L	L	L	-	TRAP
L	H	H	H	H	-	-	-	-	-	-	-	-	H	-	-	-	L	L	L	L	-	TRAP
-	-	-	H	L	-	-	-	-	-	-	-	L	H	-	-	L	-	-	-	-	-	DIRFP–Direct Base Page ADDR (not DP)
-	-	-	H	L	-	-	-	-	-	-	-	H	H	-	-	L	-	-	-	-	L	DIRDP–Direct Base Page ADDR (DP)
-	-	-	L	H	-	-	-	-	-	-	-	-	H	-	-	L	-	-	L	-	-	24-Bit ADDR
L	L	H	H	H	-	-	-	-	-	-	-	-	H	-	-	L	-	-	L	-	-	24-Bit ADDR (LEA or IMUL)
-	-	-	L	-	L	H	H	H	-	-	-	-	H	-	-	L	-	-	L	-	L	Increment XR0 (24-Bit or Indirect ADDR)
L	L	-	H	H	L	H	H	H	-	-	-	-	H	-	-	L	-	-	L	-	L	Increment XR0 (24-Bit or Indirect, LEA or IMUL)
-	-	-	L	-	H	H	H	H	-	-	-	-	H	-	-	L	-	-	L	L	-	Do Not INCR (24-Bit or Indirect ADDR)
L	L	-	H	H	H	H	H	H	-	-	-	-	H	-	-	L	-	-	L	L	-	Do Not INCR (24-Bit or Indirect ADDR–LEA, IMUL)
-	-	-	L	L	-	-	-	-	-	-	-	-	H	-	-	L	-	L	L	-	-	Indirect ADDR
L	L	L	H	H	-	-	-	-	-	-	-	-	H	-	-	L	-	L	L	-	-	Indirect ADDR (LEAI or IMULI)
L	H	L	H	H	H	-	-	-	-	-	-	-	H	-	L	-	-	-	-	-	L	LTR ("OR" ED with Branch)
-	H	L	H	H	H	H	-	-	L	-	-	-	H	-	-	-	-	-	L	L	-	BRANCH TRUE–FAC=0
-	H	L	H	H	H	H	H	L	-	L	-	-	H	-	-	-	-	-	L	L	-	BRANCH TRUE–FAC>=0
-	H	L	H	H	H	H	L	H	-	H	-	-	H	-	-	-	-	-	L	L	-	BRANCH TRUE–FAC<=0
-	H	L	H	H	H	H	L	L	-	-	-	-	H	-	-	-	-	-	L	L	-	BRANCH TRUE–ALWAYS
-	H	L	H	H	H	L	H	H	H	-	-	-	H	-	-	-	-	-	L	L	-	BRANCH TRUE–FAC<>0
-	H	L	H	H	H	L	H	L	-	H	-	-	H	-	-	-	-	-	L	L	-	BRANCH TRUE–FAC<0
-	H	L	H	H	H	L	L	H	H	L	-	-	H	-	-	-	-	-	L	L	-	BRANCH TRUE–FAC>0
-	H	L	H	H	H	L	L	L	-	-	L	-	H	-	-	-	-	-	L	L	-	BRANCH TRUE–SIGN H Negated (JAL Test)
-	H	L	H	H	H	H	H	H	H	-	-	-	H	L	-	-	-	L	-	L	-	BRANCH FALSE–FAC<>0
-	H	L	H	H	H	H	H	L	-	H	-	-	H	L	-	-	-	L	-	L	-	BRANCH FALSE–FAC<0
-	H	L	H	H	H	H	L	H	H	L	-	-	H	L	-	-	-	L	-	L	-	BRANCH FALSE–FAC>0
-	H	L	H	H	H	L	-	H	L	-	-	-	H	L	-	-	-	L	-	L	-	BRANCH FALSE–FAC=0
-	H	L	H	H	H	L	H	L	-	L	-	-	H	L	-	-	-	L	-	L	-	BRANCH FALSE–FAC>=0
-	H	L	H	H	H	L	L	H	-	H	-	-	H	L	-	-	-	L	-	L	-	BRANCH FALSE–FAC<0
-	H	L	H	H	H	L	L	L	-	-	H	-	H	L	-	-	-	L	-	L	-	BRANCH FALSE–SIGN H Asserted (JAL Test)



74S139

B1	A1	OUT LO
LO	LO	FA0
LO	HI	FA1
HI	LO	FA2
HI	HI	FA3

7442

D3	D2	D1	D0	OUT LO
LO	LO	LO	LO	F0
LO	LO	LO	HI	F1
LO	LO	HI	LO	F2
LO	LO	HI	HI	F3
LO	HI	LO	LO	F4
LO	HI	LO	HI	F5
LO	HI	HI	LO	F6
LO	HI	HI	HI	F7

08-1762

Figure 4-16 Instruction Dispatch 2 Logic

```
/          *      *      *      *      *      *      *      /
/INSTRUCTION DISPATCH 2 DISPATCHES MICRO PC AS FOLLOWS:      /
/     INSTRUCTION                          LABEL    ADDRESS  /
/     LEA, LEAI (FP AND EP MODES)          LEAB     256      /
/     FLDA                                 LOAD     200      /
/     FSTA (NOT DP)                        STOREF   220      /
/     FSTA (DP)                            STORED   224      /
/     FSUB                                 GETN     260      /
/     FADD, FADDM, FMUL, FMULM, FDIV       GETARG   240      /
/     IMUL (SAME OP CODE AS LEA, LEAI                        /
/                      BUT DP MODE)        GETARG   240      /
/     NO OTHER INSTRUCTIONS USE THIS DISPATCH                /
```

Data Reference instructions specify both the address of data and the operation to be performed on the data. The Instruction Dispatch 1 logic decodes the instruction and causes the FPP to calculate the address of the data. During this primary level of instruction decoding, that part of the instruction that specifies the operation to be performed on the data must be retained, since the instruction is fetched only once. Then, during either Instruction Dispatch 2 or Instruction Dispatch 3 the appropriate operation can be carried out.

Figure 4-16 includes flip-flop E48. During the primary decoding operation the INST DISP 1 H signal enables E48 to be clocked. The information represented by the MD (0:4) L signals, which identifies the operation that will ultimately be performed on the data, is loaded into E48.

After the data address has been calculated, the INST DISP 2 L signal is asserted. This signal enables PROM E53 to decode the 5 MSBs of the instruction and dispatch the μPC to the appropriate address. Table 4-11 relates the input and output signals for E53 and includes the FPP instruction associated with each PROM code.

If the Data Reference instruction is FLDA, FSTA, or LEA, an exit test is made after Instruction Dispatch 2. The remaining Data Reference instructions go through the final decoding level initiated by Instruction Dispatch 3 (firmware shown below). The INST DISP 3 L signal enables PROM E57 to decode the 5 MSBs of the instruction and dispatch the proper μPC address. Table 4-12 relates the input and output signals for E57 and includes the FPP instruction associated with each PROM code.

```
      /ARITHMETIC DISPATCH
1037  ARITH.  NO OPERATION                    FREE     INSTR DISP 3

      /     *     *     *     *     *     *     *     *
      /INSTRUCTION DISPATCH 3 DISPATCHES ARITHMETIC
      /                               INSTRUCTIONS AS FOLLOWS:
      /     INSTRUCTION                       LABEL    ADDRESS
      /     FADD, FADDM (DP MODE)             DPADD    1400
      /     FADD, FADDM (NOT DP)              FADD     1401
      /     FMUL, FMULM                       FMUL     1402
      /     FDIV                              FDIV     1403
      /     IMUL                              IMUL     1404
```

Table 4-11 PROM E53 Input/Output Signals
E53 Enabled for INST DISP 2 L

PROM Code	Input Signal Low					Output Signal Asserted μP_ IN L								Control ROM Address	FPP Instruction
	A4 (DP H)	A3 (MD3 L & MD4 L HIGH)	A2 (MD0 L)	A1 (MD1 L)	A0 (MD2 L)	2	4	6	7	8	9	10	11		
0	X	X	X	X	X		X	X		X	X	X		0256	LEAI
1	X	X	X	X			X	X		X	X	X		0256	LEA
10	X		X	X	X		X	X						0240	FMULM
11	X		X	X			X		X					0220	FSTA
12	X		X		X		X	X						0240	FADDM
13	X		X				X	X						0240	FMUL
14	X			X	X		X	X						0240	FDIV
15	X			X			X	X	X					0260	FSUB
16	X				X		X	X						0240	FADD
17	X						X							0200	FLDA
20		X	X	X	X		X	X						0240	IMULI
21		X	X	X			X	X						0240	IMUL
30			X	X	X		X	X						0240	FMULM
31			X	X			X		X		X			0224	FSTA
32			X		X		X	X						0240	FADDM
33			X				X	X						0240	FMUL
34				X	X		X	X						0240	FDIV
35				X			X	X	X					0260	FSUB
36					X		X	X						0240	FADD
37							X							0200	FLDA

Table 4-12 PROM E57 Input/Output Signals
E57 Enabled for INST DISP 3 L

PROM Code	Input Signal Low					Output Signal Asserted μP_ IN L					Control ROM Address	FPP Instruction
	A4 (DP H)	A3 (MD3 L & MD4 L HIGH)	A2 (MD0 L)	A1 (MD1 L)	A0 (MD2 L)	2	3	9	10	11		
10	X		X	X	X	X	X		X		1402	FMULM
12	X		X		X	X	X			X	1401	FADDM
13	X		X			X	X		X		1402	FMUL
14	X			X	X	X	X		X	X	1403	FDIV
15	X			X		X	X			X	1401	FSUB
16	X				X	X	X			X	1401	FADD
20		X	X	X	X	X	X	X			1404	IMULI
21		X	X	X		X	X	X			1404	IMUL
30			X	X	X	X	X		X		1402	FMULM
32			X		X	X	X				1400	FADDM
33			X			X	X		X		1402	FMUL
34				X	X	X	X		X	X	1403	FDIV
35				X		X	X				1400	FSUB
36					X	X	X				1400	FADD

4.2.9 Exit Test Logic

At the completion of every operation the FPP logic makes a test to determine if a new instruction should be fetched or if an exit should be carried out immediately. If a new instruction is fetched, it might be an FEXIT instruction; thus, the sequence of FPP operations would end, although under controlled conditions. However, if the test calls for an immediate exit, it could be for one of three basic reasons, viz., because the IOT instruction FPHLT was issued by the CPU, because the FPP calculations resulted in overflow or underflow, or because a divide-by-zero operation was detected. These events occur without FPP-instruction control and in a random fashion, and it is important that a means be provided for determining why an exit takes place. The Exit Test logic not only decides whether or not to exit but also records the reason for an exit. The logic is illustrated in Figure 4-17.

At the end of every FPP operation or calculation, the FETCH NEXT L signal is asserted by the μPC Gating Control logic. If an immediate exit is called for, the START EX H signal is asserted at the output of NOR gate E94. Thus, μP2 IN L is asserted, address 1000_8 is loaded into the μPC by the next clock pulse, and the exit sequence is carried out. Should the START EX H signal be low, μP7 IN L is asserted, instead. Hence, address 20_8 is loaded into the μPC and a new instruction (which might be FEXIT) is fetched.

The START EX H signal can be asserted by the outputs of the Status register, gated flip-flops E14 and E3. The output of E3, FEX H, is asserted whenever the PDP-8 CPU issues an FPHLT instruction. At TP3 time of such an instruction, the FORCE EXIT flip-flop is dc set. The flip-flop output signals cause FEX H to be asserted at the next occurring μPC CLOCK L pulse, as illustrated in Figure 4-18. The next μPC CLOCK L pulse clears the FORCE EXIT flip-flop, and the resulting high input at the gate (G3) of E3 keeps FEX H asserted until the exit test is made (NAND gate E40 can be enabled only by the FPP FEXIT instruction). Then, FEX H can be negated by the RESET L signal or by the FPIST IOT instruction.

Another way of negating the FEX H signal is by the FEXIT instruction. When this instruction is fetched, the Instruction Dispatch 1 logic (Figure 4-15) asserts μP2 IN L and F STOP H. The exit sequence is carried out and the FPP halts with the μPC address equal to 1. Therefore, AND gate E36 is enabled, as is NAND gate E40 (F STOP H stays high until RESET L is asserted or until another FPP instruction is fetched). Now, the gate of E3 is low, allowing the low at D3 to be transferred to R3(1) at clock time. This procedure ensures that, should FPHLT and FEXIT occur at approximately the same time, causing a common exit, the recorded reason will be FEXIT rather than FPHLT.

The outputs from E14 of the Status register likewise assert START EX H. The DIV0 H signal goes high when address 1630_8 is loaded into the μPC. This address indicates that the FPP logic has detected a zero divisor during a divide calculation. The other three outputs of E14 indicate overflow or underflow conditions that might develop during arithmetic calculations. The DP OVFLO H signal is asserted during DP-mode calculations and indicates that the calculation has resulted in either an overflow, i.e., the result is too large or small to be contained in a 24-bit word, or the forbidden number $4000\ 0000_8$. The EXP OVFLO H and EXP UNDFLO H signals are asserted during FP-mode and EP-mode calculations and indicate that the calculation has resulted in a number having an exponent too large or too small to be contained in a 12-bit word.

The three overflow indications are produced in response to outputs from PROM E4, a 1024-bit PROM organized to provide 256 4-bit data locations. Table 4-13 lists the PROM input codes, showing the state of the PROM input signals and relating these inputs to the actions carried out by gated flip-flop E14.

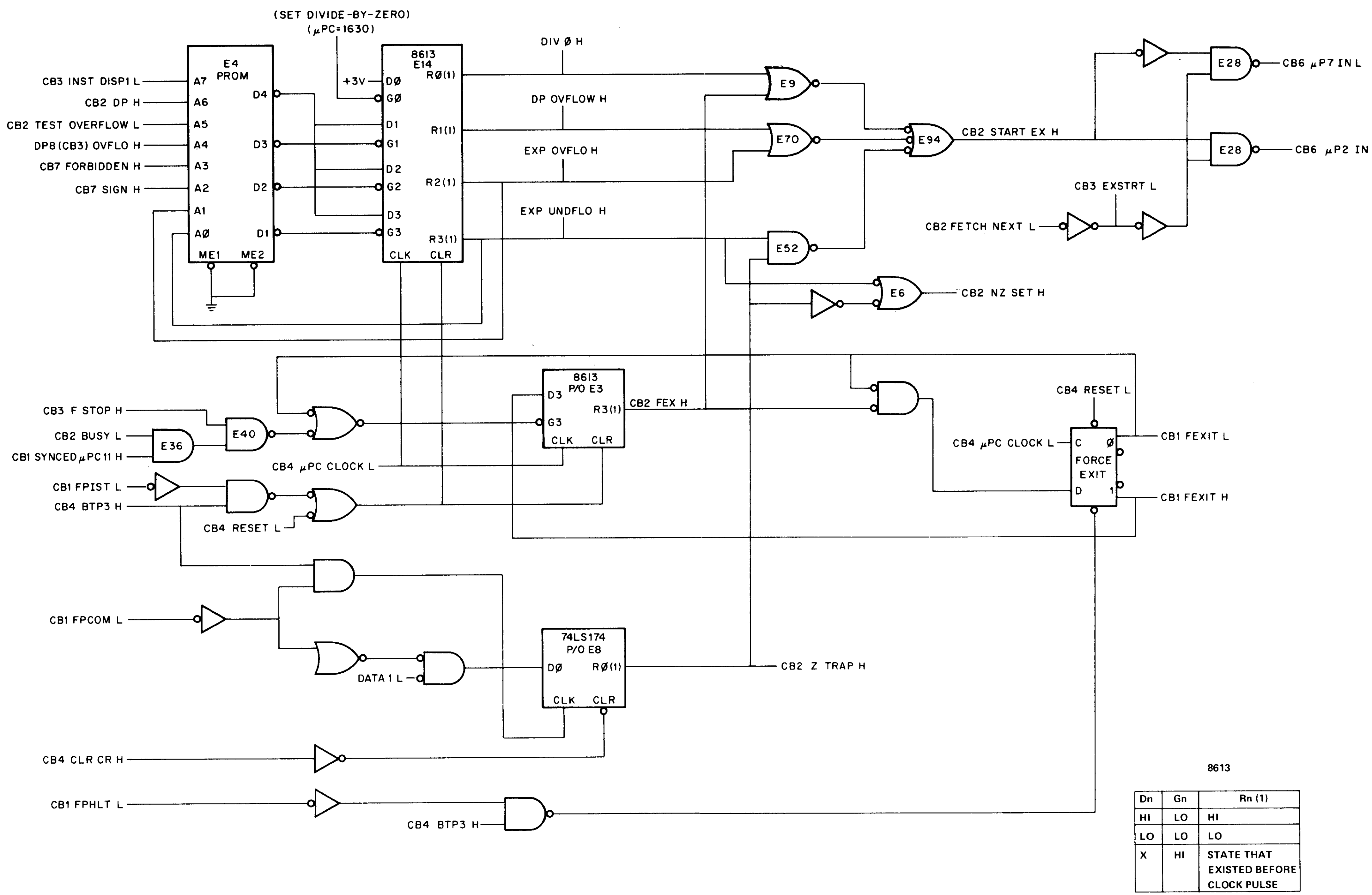


Figure 4-17 Exit Test Logic

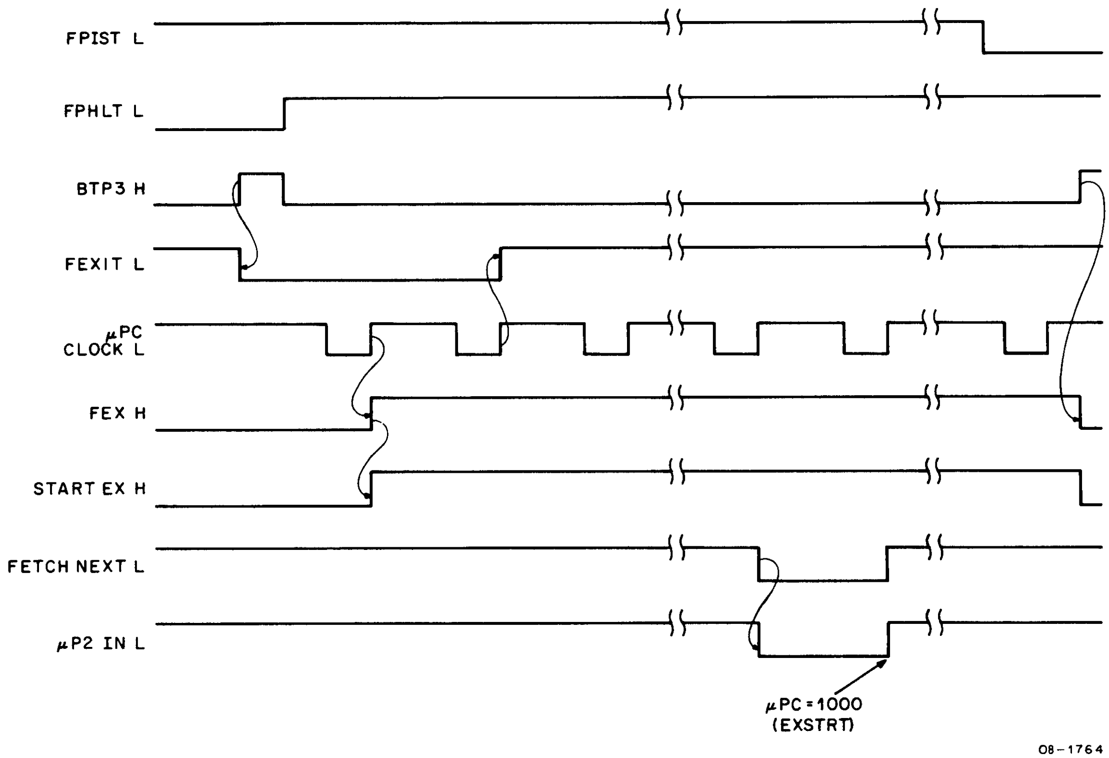


Figure 4-18 Timing, FPHLT EXIT

Table 4-13 PROM E4 Input/Output Signals

PROM Input Code	INST DISP 1 L	DP H	TEST OVFLO L	OVFLO H	FORBIDDEN H	SIGN H	EXP OVFLO H	EXP UNDFLO H	D4 D3 D2 D1	Action Taken by E14
220	HI	LO	LO	HI	LO	LO	LO	LO	HI HI HI LO	SET EXP UNDFLO H
222	HI	LO	LO	HI	LO	LO	HI	LO	LO HI LO HI	CLR EXP OVFLO H
224	HI	LO	LO	HI	LO	HI	LO	LO	HI HI LO HI	SET EXP OVFLO H
225	HI	LO	LO	HI	LO	HI	LO	HI	LO HI HI LO	CLR EXP UNDFLO H
234	HI	LO	LO	HI	HI	HI	LO	LO	HI HI LO HI	SET EXP OVFLO H
235	HI	LO	LO	HI	HI	HI	LO	HI	LO HI HI LO	CLR EXP UNDFLO H
310	HI	HI	LO	LO	HI	LO	LO	LO	HI LO HI HI	SET DP OVFLO H (FORBIDDEN H)
314	HI	HI	LO	LO	HI	HI	LO	LO	HI LO HI HI	SET DP OVFLO H (FORBIDDEN H)
320	HI	HI	LO	HI	LO	LO	LO	LO	HI LO HI HI	SET DP OVFLO H (OVFLO H)
324	HI	HI	LO	HI	LO	HI	LO	LO	HI LO HI HI	SET DP OVFLO H (OVFLO H)
334	HI	HI	LO	HI	HI	HI	LO	LO	HI LO HI HI	SET DP OVFLO H (FORBIDDEN H)
000–177	LO	–	–	–	–	–	–	–	LO LO LO LO	CLR OVFLO & UNDFLO
All Others	–	–	–	–	–	–	–	–	LO HI HI HI	NO CHANGE

Note, in the logic, that EXP UNDFLO H can assert START EX H only if the ZTRAP H signal is asserted. If ZTRAP H is low, a calculation that results in exponent underflow is stored as a zero result. In such a situation no exit is performed, and a new instruction is fetched. However, the EXP UNDFLO H flag, which would be cleared by either the FPIST instruction or the FPICL instruction after an exit operation (the latter generates RESET L), remains set. Hence, the signal INST DISP1 L, asserted when an instruction is fetched, causes the PROM to generate the signals needed to clear all the overflow flags.

4.3 DATA PATH LOGIC

Detailed descriptions of the significant portions of the logic on the Data Path board appear in Paragraphs 4.3.1 through 4.3.7.

4.3.1 ALU B Inputs

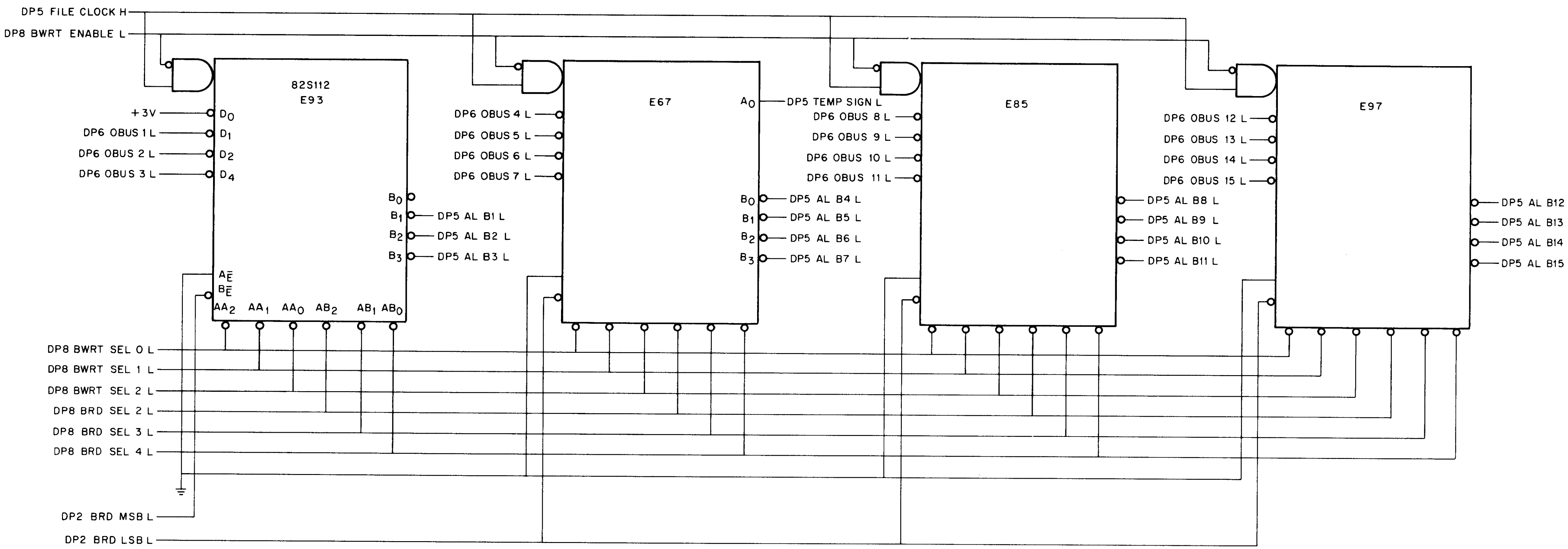
4.3.1.1 B File – The B inputs of the ALU are taken from a variety of sources; one source is the B file, a read-and-write memory that provides storage for eight 15-bit data words. The B file RAM is illustrated in Figure 4-19, while Figure 4-20 shows the logic that generates the RAM control signals. The B file is comprised of four 82S112 ICs (organized in 8 words of 4-bits each); these are arranged to provide eight 15-bit data locations. These locations are assigned to the eight temporary registers, TEMP and TEMP1 through TEMP7, and can be written into and read from; however, if both a read and a write are directed by the same data path statement, the location being read must be different from the location being written.

When a B file is to be used in an operation, it is identified by the B RD (2:4) L signals or the B WRT (0:2) L signals (Table 4-14). If a read operation is directed, the B RD MSB L and B RD LSB L signals are asserted (Table 4-15 shows the input/output signals for PROM E88). The 15-bit data word held in the selected register is applied to the ALU on the AL B (1:15) L lines. The sign bit of the data word is made available as the TEMP SIGN L signal, which is used to manipulate the ALU during multiply and divide calculations. Sometimes only the B RD MSB L signal is asserted during a read operation; then, only the three MSBs of the data word are read from the temporary location. Such is the case, for example, when the field bits of the APT pointer are transferred from the TEMP register to the APTP register during FPP initialization.

Even when only a write operation is directed by the data path statement, the B file goes through the read mode; however, since neither the B RD MSB L signal nor the B RD LSB L signal is asserted, the Bn outputs of the file remain negated. When the mode switches from read to write, the data is written and the outputs remain high.

Both a read and a write can be performed during the same data path statement. For example, the statement for μPC address 147_8 is

TEMP1:=BR[+] TEMP+1



82S112

FILE CLOCK H	B WRT ENABLE L	MODE	BE̅	A0	Bn
LO	X	READ	HI	DATA	HI
LO	X		LO		DATA
HI	HI		HI		HI
HI	HI		LO		DATA
HI	LO	WRITE	HI	DATA BEING WRITTEN	HI
HI	LO		LO		DATA B ADDRESS

08-1765

Figure 4-19 B File RAM Logic

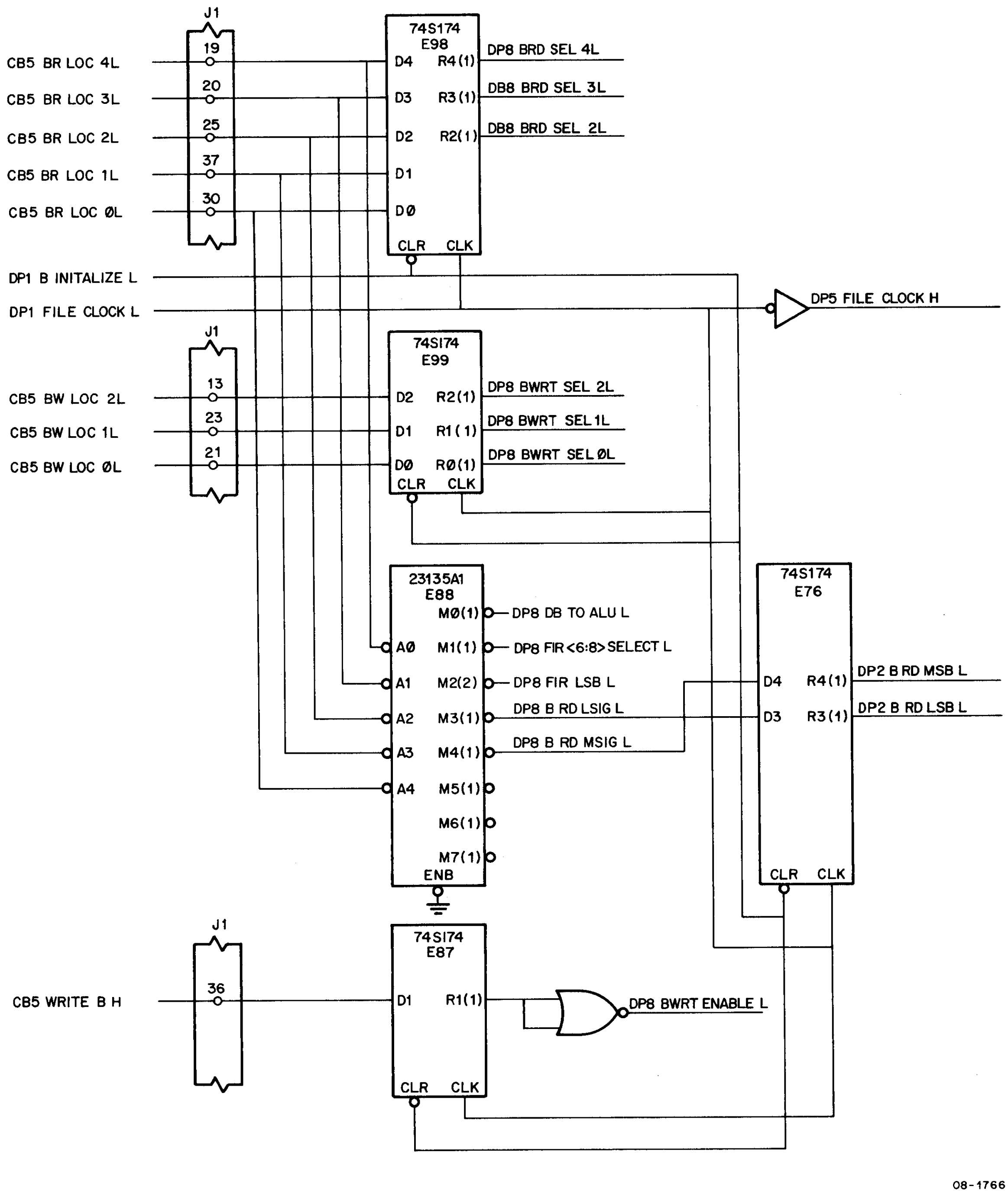


Figure 4-20 B File Control Signals

Table 4-14 TEMP Register Selection

B RD SEL 2 L B WRT SEL 0 L	B RD SEL 3 L B WRT SEL 1 L	B RD SEL 4 L B WRT SEL 2 L	TEMP Register Selected
HI	HI	HI	TEMP
HI	HI	LO	TEMP1
HI	LO	HI	TEMP2
HI	LO	LO	TEMP3
LO	HI	HI	TEMP4
LO	HI	LO	TEMP5
LO	LO	HI	TEMP6
LO	LO	LO	TEMP7

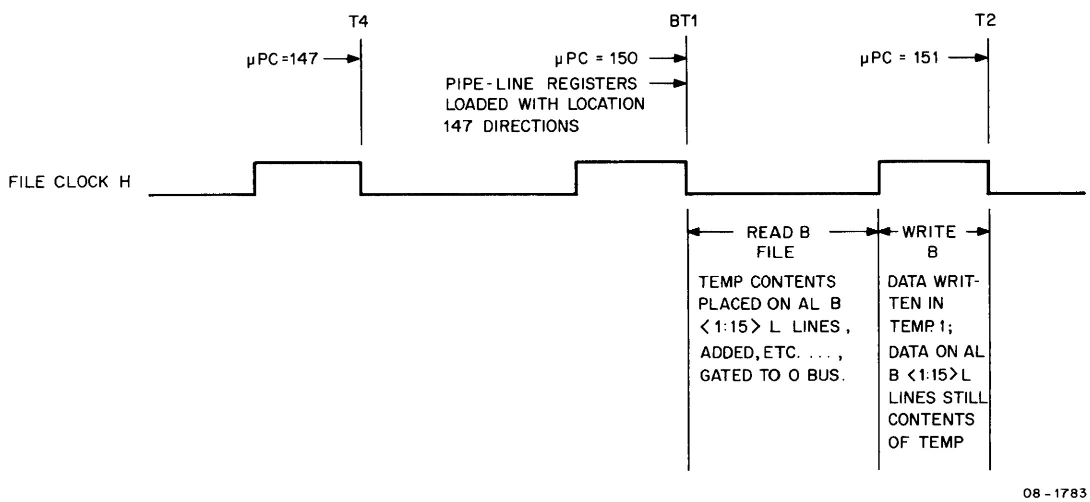
Table 4-15 PROM E88 Input/Output Signals
(PROM Enabled Permanently)

	Input Signal Low					Output Signal Asserted				
PROM Address	BR LOC 0 L	1L	2L	3L	4L	DB TO ALU L	FIR (6:8) SELECT L	FIR LSB L	B RD LSIG L	B RD MSIG L
4	X	X		X	X		X			
5	X	X		X				X		
6	X	X			X	X				
7	X	X				X				X
10	X		X	X	X				X	X
11	X		X	X					X	X
12	X		X		X				X	X
13	X		X						X	X
14	X			X	X				X	X
15	X			X					X	X
16	X				X				X	X
17	X								X	X
27		X								X

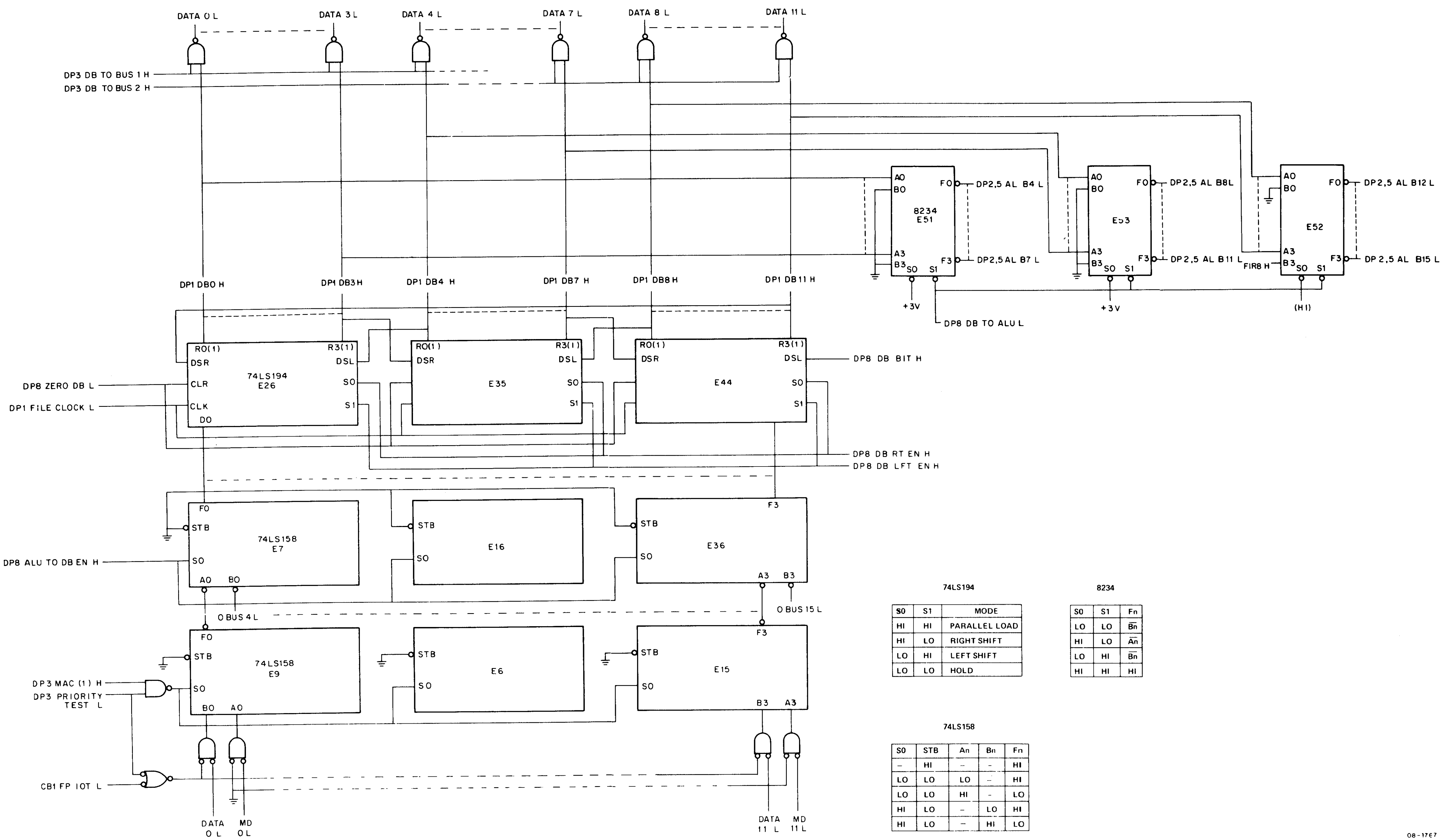
Location 147_8 produces the Control ROM output signal logic levels that select TEMP for reading and TEMP1 for writing, and that cause **B RD MSB L**, **B RD LSB L**, and **B WRT ENABLE L** to be asserted. These logic levels are:

BR LOC	0L	1L	2L	3L	4L
	LO	HI	HI	HI	HI;
BW LOC	0L	1L	2L		
	HI	HI	LO;		
WRITE B	H				
	HI				

The read-and-write operation is carried out as outlined in the following diagram, which uses FILE CLOCK H to delineate intervals.



4.3.1.2 DB Register Logic – The DB register is the data interface between the FPP and the PDP-8 CPU or memory. The register and the logic directly related to it are illustrated in Figure 4-21; Figure 4-22 shows the logic that generates the register control signals.



74LS194

S0	S1	MODE
HI	HI	PARALLEL LOAD
HI	LO	RIGHT SHIFT
LO	HI	LEFT SHIFT
LO	LO	HOLD

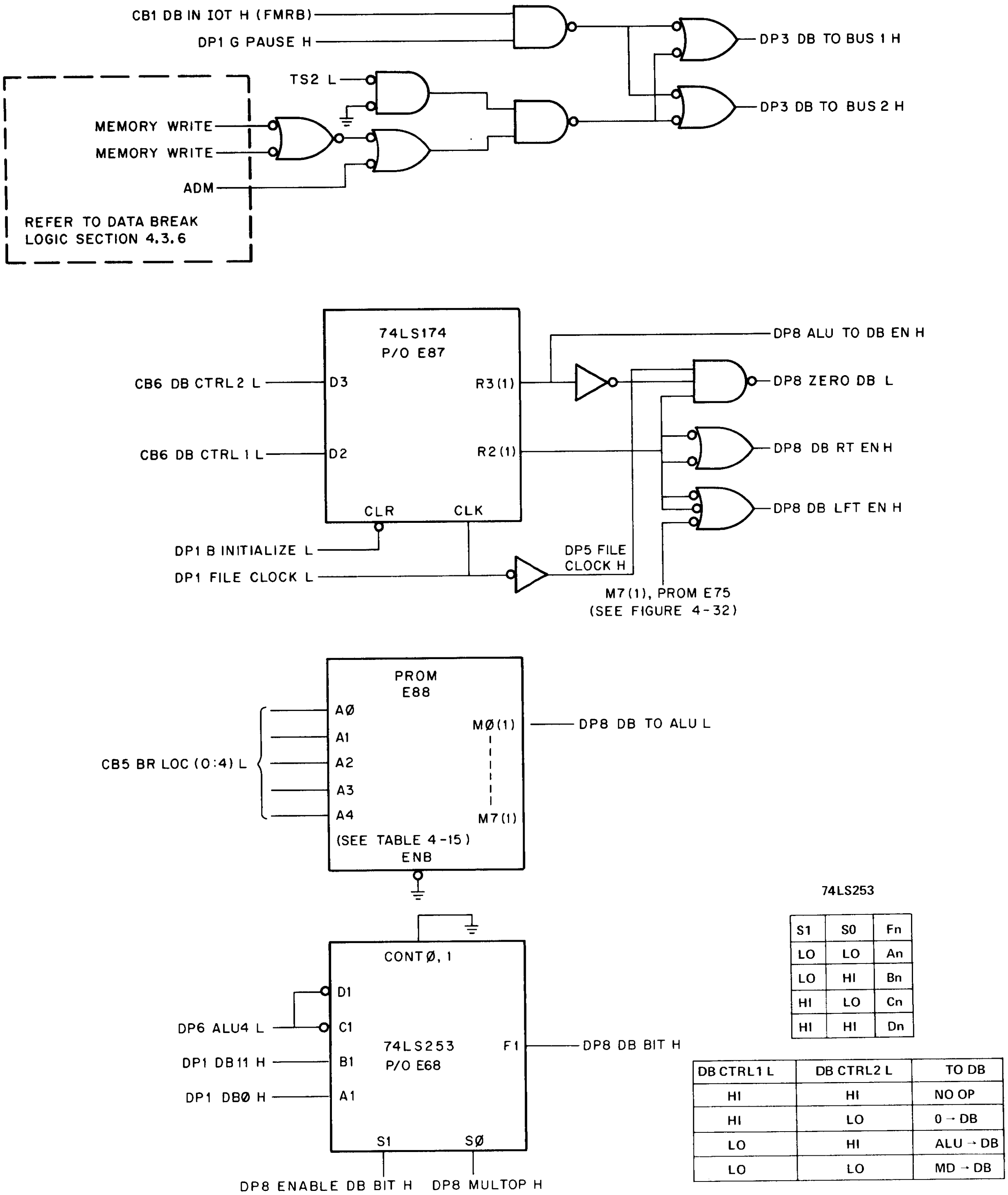
8234

S0	S1	Fn
LO	LO	$\overline{Bn}$
HI	LO	$\overline{An}$
LO	HI	$\overline{Bn}$
HI	HI	HI

74LS158

S0	STB	An	Bn	Fn
–	HI	–	–	HI
LO	LO	LO	–	HI
LO	LO	HI	–	LO
HI	LO	–	LO	HI
HI	LO	–	HI	LO

Figure 4-21 DB Register Logic



74LS253

S1	SO	Fn
LO	LO	An
LO	HI	Bn
HI	LO	Cn
HI	HI	Dn

DB CTRL1 L	DB CTRL2 L	TO DB
HI	HI	NO OP
HI	LO	0 → DB
LO	HI	ALU → DB
LO	LO	MD → DB

Figure 4-22 DB Register Control Signals

The DB register, itself, is a 74LS194 bidirectional shift register. The modes available are notated in the table in Figure 4-21 (the right-shift mode is not used). The DB can be parallel-loaded from a number of sources, depending on the state of the Control ROM's DB CTRL (1:2) L signals (refer to the function table in Figure 4-22 that relates these signals to the DB source). If the ALU is to be loaded into the DB, the register is placed in the parallel-load mode (DB CTRL1 L is asserted, causing both DB RT EN H and DB LFT EN H to go high), and ALU TO DB EN H is asserted; this gates the ALU information from the OBUS (4:15) L lines to the DB inputs for loading at clock time. If, instead of OBUS data, the information on either the MD or DATA lines is to be placed in the DB register, the ALU TO DB EN H signal is negated. The DATA lines are a DB source during FPP initialization when the APT pointer address is to be transferred to the APTP register; at all other times the DATA line information is gated through the first tier of multiplexers (E9, E6, and E15) only during priority checking (the information is not loaded into the DB during this procedure - see Figure 4-33). The MD lines are the DB source when an operand or an FPP instruction is to be placed in the register. If the FPP has priority, a data break cycle is started and the MD information is gated to the DB.

Input data from the MD or DATA lines is gated through three data selectors (E51, E52, and E53) by the DB TO ALU L signal. The data is applied to the B inputs of the ALU via the DP2,5 AL B (4:15) L lines, gated onto the OBUS, and loaded into the applicable file register. Conversely, DB output data is placed on the DATA lines by the DB TO BUS (1:2) H signals, which can be asserted during data break operations or by the FMRB IOT instruction.

The DB register can be left-shifted if DB LFT EN H is asserted and DB RT EN H is negated. This is possible only during multiply and divide operations when PROM E75 causes DB LFT EN H to go high. If the operation is a multiply, DB0 is rotated into DB11. If the operation is a divide, a quotient bit (represented by the ALU4 L signal) is shifted into the DB11.

4.3.1.3 FIR Logic - Another source for the B inputs of the ALU is the FIR logic, which is used during address calculations. The logic is illustrated in Figure 4-23. Shown below is the portion of the FPP firmware that relates to an instruction fetch; refer to this while reading what follows.

```
        *20
20      FETCH,      BKMA:=FPC                  T3
21      FETCH1,     :=FACE[EXPSIZE]M30         T4      BKCMD:=7
22                  FPC:=FPC[+]K1; DB:=MD      BT1
23                  TEMP:=FIR(9:11)            T2      INSTR DISP 1
```

When an FPP instruction is being fetched it is gated to the DB register from the MD lines and loaded at T2 time (T2 of Control ROM address 23_8). Since the ENABLE FIR L signal is low (having been asserted at the preceding BT1 time), the FIR register, E43 and E45 in Figure 4-23, is loaded just after the DB. Not only is the DB register loaded at T2 time, but also hex flip-flop E76 is clocked, causing its R2(1) output to go low (FIR LSB L is asserted by PROM E88 during μPC address 23_8 - refer to Table 4-15). Thus, the An inputs of multiplexer E49 are selected, and FIR bits (9:11) are gated onto the AL B (13:15) L lines (for the moment, ignore output F3).

At this point the data path statement TEMP:=FIR (9:11) is modified by the FPP instruction that has been loaded into the DB. If the instruction is Special or Double-Word Data Reference, the signals on the AL B (13:15) L lines are gated through the ALU to the shift gates. There, the signals are rotated right three places and placed on the OBUS (1:3) lines. At T3 time TEMP is loaded, as described by the statement TEMP:= [R3R] FIR (9:11). This modification of the data path statement by the instruction enables the Data Path logic to retain the field bits of the address contained in the instruction.

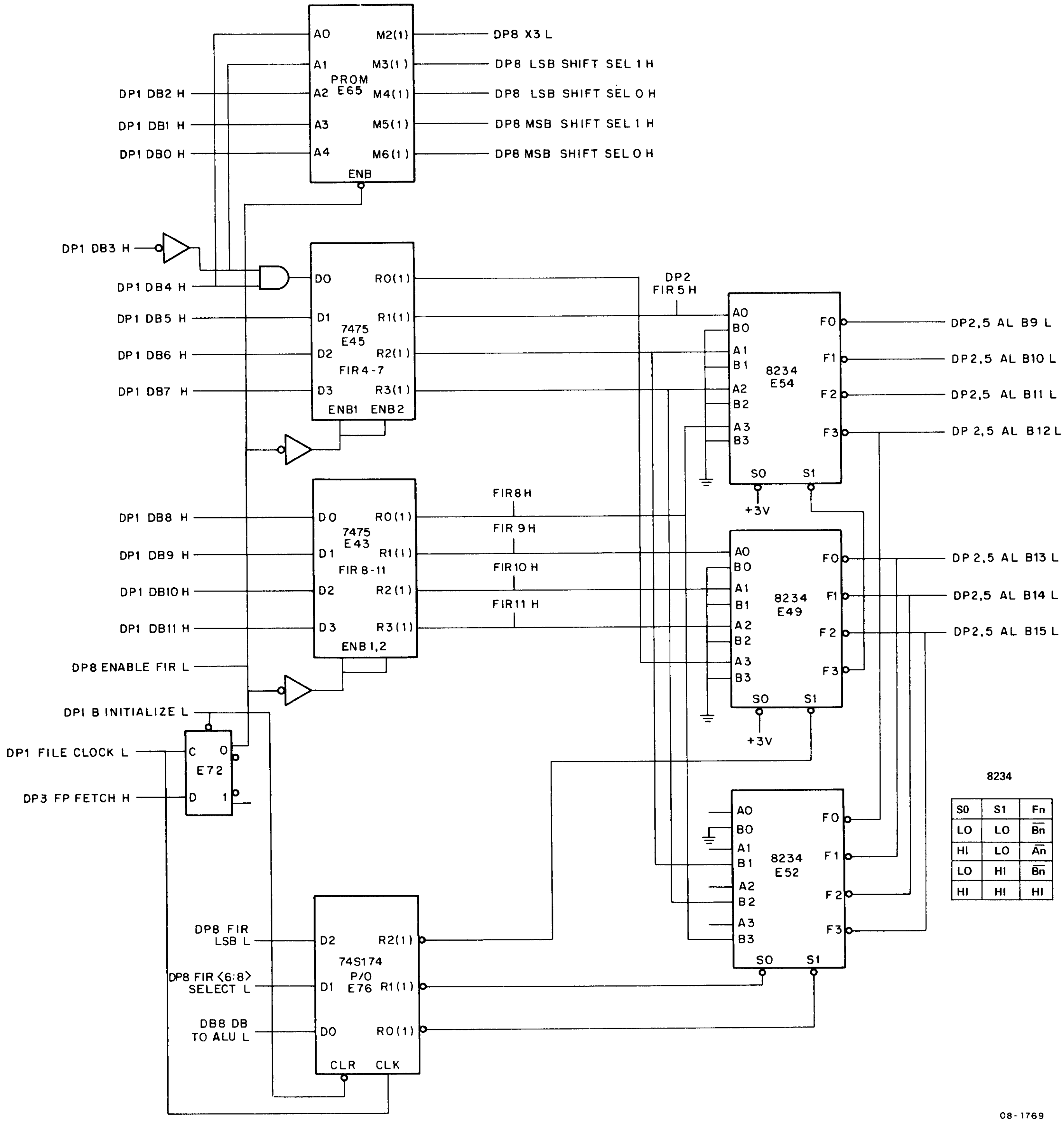


Figure 4-23 FIR Logic

The decision to rotate the FIR bits is made by PROM E65 in the FIR logic. This PROM monitors the five MSBs of the DB register and generates the outputs that modify the address 23_8 data path statement. Table 4-16 shows the input/output signal relationship for the PROM. Note that an R3R operation (rotate right 3 places) is carried out for all Special instructions [except LTR, where bits (9:11) are immaterial] and for all Double-Word Data Reference instructions. For many of the Special instructions the operation is superfluous and is performed only in the interest of limiting the number of decisions that the logic must make. Shown below is that part of the firmware that relates to the JSA instruction, a Special instruction that does require the R3R operation.

```
        /JSA
        *50
50      JSA,        BKMA:=FPC                   T3      SUB, INST24 (4)
51                  OPADD:=TEMP(1:3), DB        T2
52                  BKMA:=OPADD; DB:=0          T3
53                  DB:=1030!FPC(1:3)           T4      BKCMD:=1
54                  DB:=FPC                     BT1
55                  BKMA, OPADD:=OPADD[+]K1     T3
56                  TEMP:=OPADD[+]K1            T4      BKCMD:=1
57                  FPC:=TEMP                   BT1     EXTEST

        /SUBROUTINE–GET SECOND HALF OF 24-BIT INSTRUCTION
        *4
4       INST24,     FPC:=FPC[+]K1               T4      BKCMD:=0
5                   DB:=MD                      BT1     RETURN
```

If the JSA instruction were dispatched, for example, the field bits of the address specified in the instruction would be loaded in TEMP (1:3) at T3 time of address 50_8. Then, after the 12 LSBs of the address had been read from memory and loaded into the DB, both the DB contents and TEMP (1:3) would be loaded into OPADD at T3 time.

If, instead of JSA, a Single-Word, Indirect Reference instruction were fetched, a different modification of the data path statement would be carried out. Once again, the FIR logic would gate FIR bits (9:11) onto the AL B (13:15) L lines. Now, however, PROM E65 asserts the X3 L signal. The resulting operation causes the offset specified by the instruction to be multiplied by 3 and placed in TEMP. TEMP is then added to the base address to specify the indirect address of the instruction operand. The firmware portion shown below relates to the dispatch of a Single-Word, Indirect Reference instruction.

```
        /ENTER HERE FOR NON-INCREMENTED, NON-INDEXED INDIRECT ADDRESS CALC.
        //TEMP CONTAINS 3*FIR(9:11) AT ENTRY
        *134
134     INDIR.      BKMA, TEMP1:=TEMP[+]BR+1    T3
135                 OPADD:=TEMP1                T4      BKCMD:=0
136                 DB:=MD                      BT1
```

Table 4-16 PROM E65 Input/Output Signals
(enabled when 'ENABLE FIR L' is low)

PROM Input Code	Input Signal Low					Output Signal Asserted					Result	Applicable FPP Instruction
	DB0 H	DB1 H	DB2 H	DB3 H	DB4 H	LSB SHFT SEL 1 H	LSB SHFT SEL 0 H	MSB SHFT SEL 1 H	MSB SHFT SEL 0 H	X3 L		
0	X	X	X	X	X						R3R	ADDX, LDX, ALN, ATX, XTA, NOP, STARTE, FEXIT, FPAUSE, FCLA, FNEG, FNORM, STARTF, STARTD, JAC
1	X	X	X	X		X	X	X	X	X	X3	FLDA (Single-Word, Direct Ref)
2	X	X	X		X						R3R	FLDA (Double-Word)
3	X	X	X			X	X	X	X	X	X3	FLDA (Single-Word, Indirect Ref)
4	X	X		X	X						R3R	BRANCH, SETX, SETB, JSA, JSR
5	X	X		X		X	X	X	X	X	X3	FADD (Single-Word, Direct Ref)
6	X	X			X						R3R	FADD (Double-Word)
7	X	X				X	X	X	X	X	X3	FADD (Single-Word, Indirect Ref)
10	X		X	X	X						R3R	JNX
11	X		X	X		X	X	X	X	X	X3	FSUB (Single-Word, Direct Ref)
12	X		X		X						R3R	FSUB (Double-Word)
13	X		X			X	X	X	X	X	X3	FSUB (Single-Word, Indirect Ref)
14	X			X	X						R3R	TRAP Instruction
15	X			X		X	X	X	X	X	X3	FDIV (Single-Word, Direct Ref)
16	X				X						R3R	FDIV (Double-Word)
17	X					X	X	X	X	X	X3	FDIV (Single-Word, Indirect Ref)
20		X	X	X	X						R3R	TRAP Instruction
21		X	X	X		X	X	X	X	X	X3	FMUL (Single-Word, Direct Ref)
22		X	X		X						R3R	FMUL (Double-Word)
23		X	X			X	X	X	X	X	X3	FMUL (Single-Word, Indirect Ref)
24		X		X	X	X	X	X	X			LTR
25		X		X		X	X	X	X	X	X3	FADDM (Single-Word, Direct Ref)
26		X			X						R3R	FADDM (Double-Word)
27		X				X	X	X	X	X	X3	FADDM (Single-Word, Indirect Ref)
30			X	X	X						R3R	LEA
31			X	X		X	X	X	X	X	X3	FSTA (Single-Word, Direct Ref)
32			X		X						R3R	FSTA (Double-Word)
33			X			X	X	X	X	X	X3	FSTA (Single-Word, Indirect Ref)
34				X	X	X	X	X	X	X	X3	LEAI
35				X		X	X	X	X	X	X3	FMULM (Single-Word, Direct Ref)
36					X						R3R	FMULM (Double-Word)
37						X	X	X	X	X	X3	FMULM (Single-Word, Indirect Ref)

A Single-Word, Direct Reference instruction also causes the FIR logic to assert the X3 L signal. Furthermore, such an instruction causes output F3 of multiplexer E49 to go low (only single-word, direct referencing has instruction bit 3 low and instruction bit 4 high). Hence, multiplexer E54 selects its An inputs, and FIR bits (5:11) are gated onto the AL B (9:15) L lines. The resulting operation causes the instruction offset to be multiplied by 3 and placed in TEMP. When TEMP is added to the base address, the operand absolute address is completely identified. The firmware entries that follow relate to the direct address calculation, location 100 applying to FP mode and location 102 applying to DP mode.

```
        /DIRECT ADDRESS CALCULATION
        *100
100     DIRFP,  BKMA, TEMP1:=TEMP[+]BR;  DB:=0          T3      INSTR DISP 2

        /DP CALCULATION ADDS 1 BECAUSE BASE PAGE ALWAYS CONTAINS 3-WORD ARG.
        *102
102     DIRDP,  BKMA, TEMP1:=TEMP[+]BR+1;  DB:=0        T3      INSTR DISP 2
```

If the operand address specified by the FPP instruction is to be modified by the contents of an Index register, bits 6, 7, and 8 of the instruction will contain an octal number from 1 to 7 (Single-Word, Direct Address instructions are not indexed). After FIR bits (9:11) have been manipulated and the result stored in TEMP, FIR bits (6:8) are added to the contents of the X0 register at the start of the address calculation, as illustrated in the firmware entry that follows.

```
        /INDEXED INDIRECT ADDRESS CALCULATION
        //TEMP HOLDS 3*FIR(9:11) AT ENTRY.
        *130
130     XIND.       BKMA:=X0[+]FIR(6:8)                 T3      GO TO, XIND1 (147)
```

Control ROM address 130_8 causes PROM E88 (Table 4-15) to assert the FIR (6:8) SELECT L signal; hence, multiplexer E52 gates FIR (6:8) onto the AL B (13:15) L lines and to the ALU.

4.3.1.4 Constant Generator – The Constant generator, illustrated in Figure 4-24, includes two PROMs, E94 and E96. The PROMs are controlled by the B RD SEL (0:4) L signals and provide inputs for the B lines of the ALU. Table 4-17 relates the input/output signals for the PROMs and lists the applicable constants.

PROM addresses 0 and 1 are used to carry out three specific FPP operations. Address 0 is involved in the firmware entry

DB:=1030!FPC(1:3),

which is part of the JSA and JSR firmware routines. During the decoding of each of these instructions, a JA (unconditional jump) to the current value of the FPC is constructed and stored in core memory. The MSB of this JA instruction is 103X, where X represents the field bits of the current value of the FPC. The FPC is read from the A file and placed on the AL A (1:15) L lines. The constant generator logic shifts the FPC field bits [AL A (1:3) L] onto the AL B (13:15) L lines and places 0103 on the AL B (1:12) L lines. The ALU then gates its B inputs to the shift gates and the constructed MSB of the JA instruction is sent to the DB register for transfer to memory.

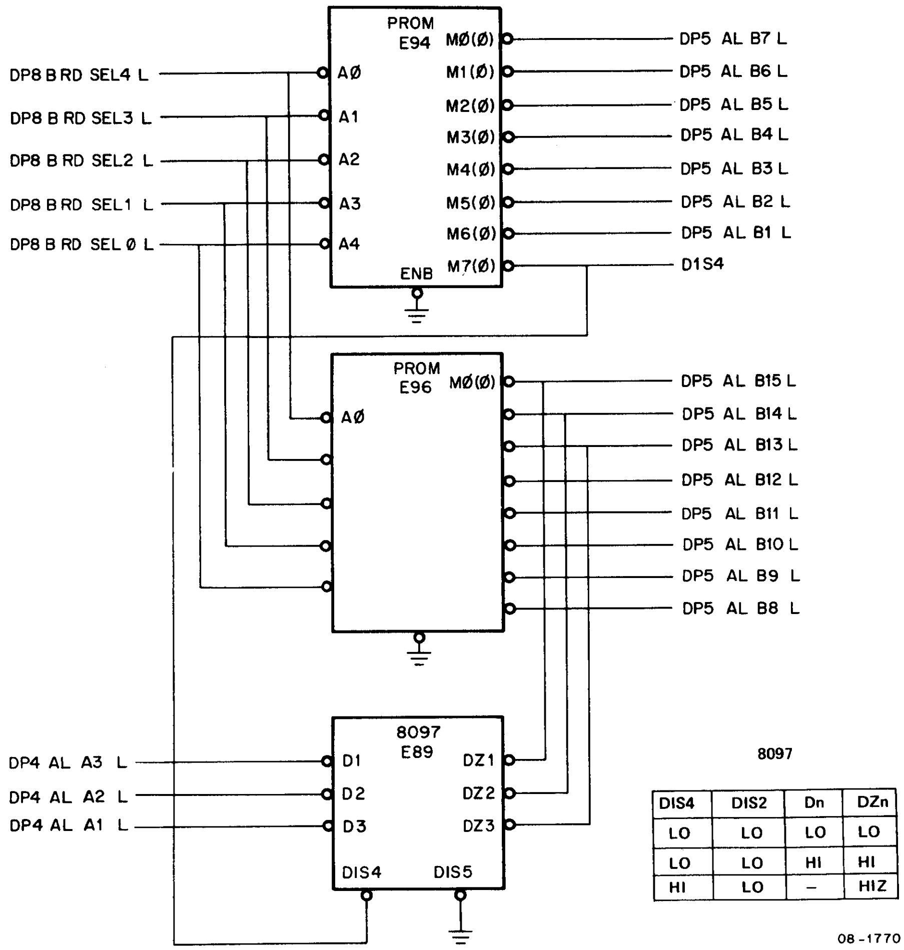


Figure 4-24 Constant Generator

Table 4-17 PROM E94/E96 Input/Output Signals

PROM Address	Input Signal Low (B RD SEL _L)					Output Signal Asserted (DP5 AL B_ L)																Constant
	0	1	2	3	4	DIS4	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	X	X	X	X	X	X						X					X	X				0103X, where X is determined by AL A (1:3) L
1	X	X	X	X		X																0000X, where X is determined by AL A (1:3) L
20		X	X	X	X		X	X	X	X	X	X	X	X	X	X		X				-30
21		X	X	X							X	X	X	X	X	X	X	X	X	X	X	3777
22		X	X		X						X											2000
23		X	X				X	X	X	X	X	X	X	X	X	X	X	X		X		-6
24		X		X	X		X	X	X	X	X	X	X	X	X	X	X	X		X	X	-5
25		X		X			X	X	X	X	X	X	X	X	X	X	X		X			-14
26		X			X													X	X			14
30			X	X	X		X	X	X	X	X	X	X	X	X				X		X	-73
31			X	X			X	X	X	X	X	X	X	X	X	X		X			X	-27
32			X		X		X	X	X	X	X	X	X	X	X	X	X	X	X	X		-2
33			X				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	-1
34				X	X															X	X	3
35				X																X		2
36					X																X	1
37																						0

PROM address 1 is used during either of the following operations:

DB:=FPC(1:3);
DB, FACM:=FACM(1:3).

The first operation occurs during a Fast Exit, when only the FPC and the APT field locations are filled; the second occurs during an LEA instruction, when only the field bits of the effective address must be loaded into the FAC.

4.3.2 ALU A Inputs

The A inputs of the ALU are taken from the A-file, a write-while-read memory that provides storage for 15-bit data words. The A-file is illustrated partially in Figure 4-25; the signals that control the file are generated by the logic shown in Figure 4-26. The file is comprised of eight 82S21 ICs (for clarity, three are not shown). Each IC is organized so as to permit storage of 32 2-bit words; thus, eight units enable 32 15-bit words to be stored by the file (bit M0 of E100 is not used). The 32 file locations are assigned to specific operating registers and can be written into and read from; however, if both a read and a write are directed by the same data path statement, the source and the destination must be the same.

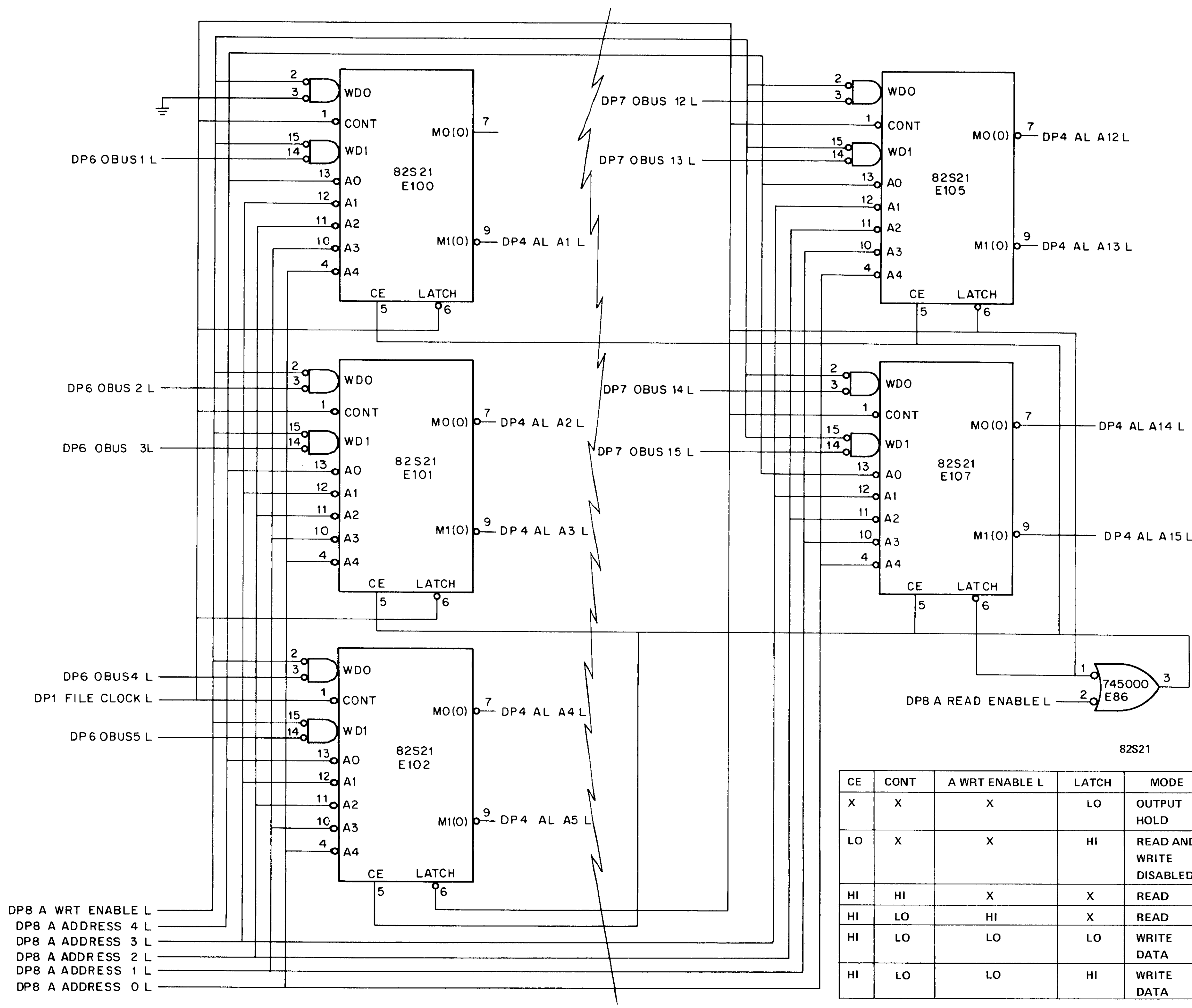
When an a file is to be used in an operation, it is identified by the A ADDRESS (0:4) L signals (Table 4-18 relates these signals to the A-file registers). The 15-bit word stored in the selected register is applied to the ALU on the AL A (1:15) L lines. Conversely, the 15-bit word to be stored in the selected location is applied to the A-file on the OBUS (1:15) L lines.

4.3.3 ALU and Shift Gates

The ALU and the Shift Gates jointly manipulate data as directed by the Control ROM ARITH (0:3) H signals. The ARITH signals are shown below in relation to the function carried out in the ALU/Shift Gates.

	ARITH 0	ARITH 1	ARITH 2	ARITH 3	FUNCTION
ADDRESS CALCULATION (15-BIT ARITHMETIC)	L	L	L	L	A+B+CARRY (15 BITS) TO OBUS
	L	L	L	H	(A+B+CARRY) *2 TO OBUS (2*B)
	L	L	H	L	(A+B+CARRY) LOGICALLY RIGHT ROTATED 3 PL. TO OBUS
	† L	L	H	H	(3*B+CARRY) (15 BITS) TO OBUS (3*B)
	† L	H	L	L	(3*B+CARRY) *2 TO OBUS (6*B)
DATA MANIPULATION (12-BIT ARITHMETIC)	L	H	L	H	A+B+CARRY (12 BITS) TO OBUS
	L	H	H	L	0 TO OBUS (15 BITS)
	L	H	H	H	A SIGN (0000 OR 7777) TO OBUS
	H	L	L	L	B TO OBUS (12 BITS)
	H	L	L	H	A+$\overline{B}$+CARRY (12 BITS) TO OBUS (A–B)
	H	L	H	L	EXP SIZE
	H	L	H	H	OVFLO RECOVERY (COMPLEMENT OF SIGN→SGN, SHIFT RT)
	H	H	L	L	(A+B+CARRY) *2+SHIFT BIT^x (12 BITS) TO OBUS
	H	H	L	H	(A+B+CARRY) ÷ 2+SHIFT BIT^x (12 BITS) TO OBUS
	H	H	H	L	DIV FINAL
	H	H	H	H	MUL/DIV STEP

† A READ MUST BE DISABLED
x SHIFT BIT IF EXTEND IS H; SIGN BIT IF EXTEND IS L AND RIGHT SHIFT; 0 IF EXTEND IS L AND LEFT SHIFT
EXTEND = LOW: CARRY BIT TO ALU, ZERO OR SIGN TO VACATED BIT POSITION
EXTEND = HIGH: CARRY FROM LAST OPERATION TO ALU, SHIFTED BIT FROM LAST OPERATION TO VACATED BIT



82S21

CE	CONT	A WRT ENABLE L	LATCH	MODE	OUTPUTS
X	X	X	LO	OUTPUT HOLD	DATA FROM LAST ADDRESSED WORD WHEN CE = 1 (HI)
LO	X	X	HI	READ AND WRITE DISABLED	DISABLED LOGIC 1
HI	HI	X	X	READ	DATA STORED IN ADDRESSED WORD
HI	LO	HI	X	READ	DATA STORED IN ADDRESSED WORD
HI	LO	LO	LO	WRITE DATA	DATA FROM LAST WORD ADDRESSED WHEN LATCH WENT FROM 1 TO 0
HI	LO	LO	HI	WRITE DATA	DATA BEING WRITTEN INTO MEMORY

Figure 4-25 A File RAM Logic

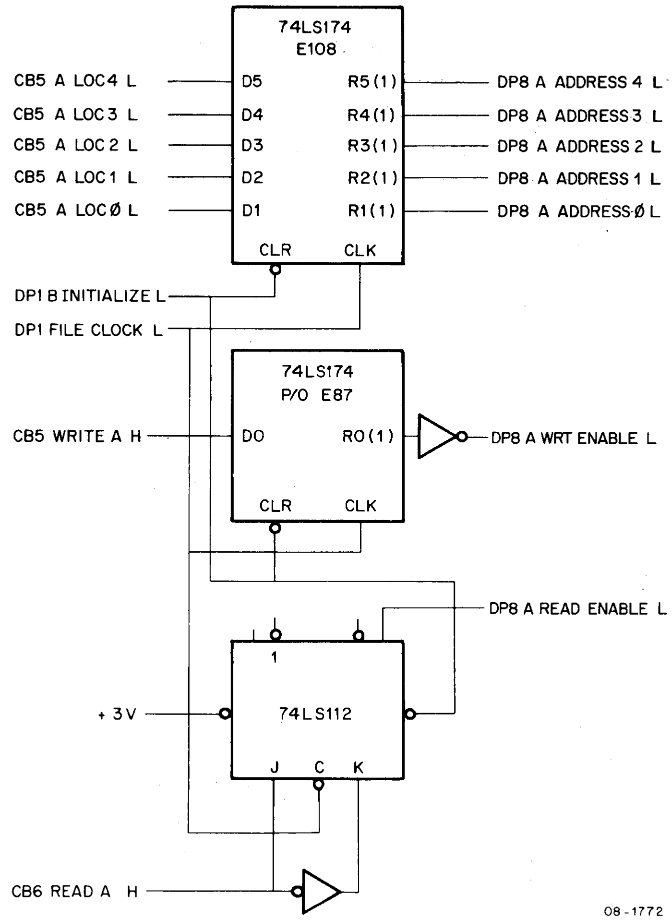


Figure 4-26 A File Control Signals

As noted, both 12- and 15-bit arithmetic can be carried out. 15-bit arithmetic is necessary during address calculation, since instruction and operand memory addresses include the field bits, which are designated as the address MSB to differentiate them from the relative address bits, or LSB. 12-bit arithmetic deals with operands and data derived from operand manipulations; only 12 bits of data are involved and, thus, only an LSB is considered in calculations.

During 15-bit arithmetic, both the MSB and the LSB of the ALU are placed in the same arithmetic mode; likewise, both the MSB and the LSB of the Shift Gates are placed in the same shifting mode. Hence, relevant information is gated onto all 15 OBUS lines. For 12-bit arithmetic, the LSB of the ALU is put in the necessary arithmetic or logic mode, while the MSB is kept in the logic mode. During 12-bit addition (i.e., when the arithmetic function [12 BIT] is included in the Data Path statement), the mode of the MSB permits these bits to be used for overflow detection. Nevertheless, in both this and other 12-bit arithmetic operations, the ALU MSB contents are irrelevant where the OBUS is concerned. This is so because in 12-bit arithmetic the MSB of the Shift Gates is kept in a mode that always gates zeros onto OBUS (1:3) L. Consequently, only the LSB of the Shift Gates places relevant information on the OBUS.

Table 4-18 A Address (0:4) L Functions

A Address (0:4) L					Sources Selected for Reading or Writing* (Source gated to A input of ALU or loaded from OBUS)
0	1	2	3	4	
H	H	H	H	H	FPC
H	H	H	H	L	X0
H	H	H	L	H	BR
H	H	H	L	L	OPADD
H	H	L	H	H	APTP
H	H	L	H	L	TEMA
H	H	L	L	H	FIELD
H	H	L	L	L	NOT USED
H	L	H	H	H	FACE (EXPONENT)
H	L	H	H	L	FACM [FRACTION (0:11)]
H	L	H	L	H	FACN [(12:23)]
H	L	H	L	L	FACP [(24:35)]
H	L	L	H	H	FACR [(36:47)]
H	L	L	H	L	FACS [(48:59)]
H	L	L	L	H	NOT USED
H	L	L	L	L	SC
L	H	H	H	H	SCRATCHE
L	H	H	H	L	SCRATCHM
L	H	H	L	H	SCRATCHN
L	H	H	L	L	SCRATCHP
L	H	L	H	H	SCRATCHR
L	H	L	H	L	SCRATCHS
L	H	L	L	H	SCRATCHT
L	H	L	L	L	NOT USED
L	L	H	H	H	MQE
L	L	H	H	L	MQM
L	L	H	L	H	MQN
L	L	H	L	L	MQP
L	L	L	H	H	MQR
L	L	L	H	L	MQS
L	L	L	L	H	NOT USED
L	L	L	L	L	NOT USED

*READ A H must be asserted for reading.
WRITE A H must be asserted for writing.

Recall that the firmware (source code) describes a 15-bit add by including the arithmetic function [+] in the Data Path statement. The ARITH (0:3) H signals select this operation, all four signals being negated (refer to the first entry in the table that appears earlier in this section). Although a 15-bit add is characterized by this combination of the ARITH signals, the occurrence of this combination does not necessarily identify a 15-bit addition. For example; the Data Path statement of μPC address 1435 reads

DB, TEMP6:= FACE.

The ARITH (0:3) H signals are negated during this address, but this operation is correctly termed a "move," rather than an add (one might consider this an add of FACE and 0). Furthermore, this move involves only 12 relevant bits, i.e., the LSB; at some point in the particular routine that this address is part of, the MSB is made zero so that the OBUS (1:3) L signals are negated.

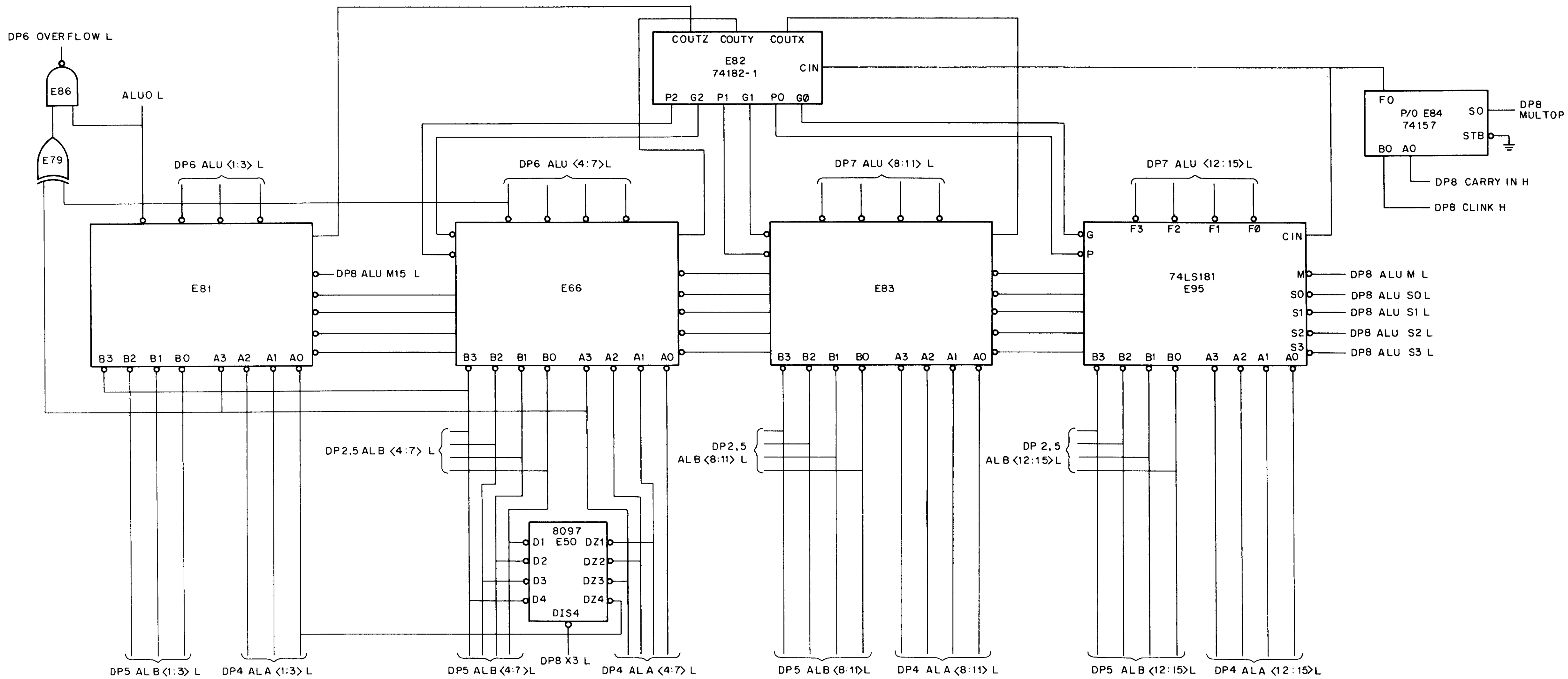
Similar reasoning can apply to other combinations of the ARITH (0:3) H signals. Do not try to generalize Data Path operations from the ALU/Shift Gates control signals.

4.3.3.1 ALU Logic - The ALU logic is illustrated in Figure 4-27. The ALU, itself, is composed of four 74LS181 ALUs and performs binary arithmetic operations on two 15-bit words (data calculations are carried out with the 12 LSBs, while address calculations use all 15 bit positions). The various arithmetic and logic operations are selected by five control signals - ALU M L (ALU M15 L is used for the MSB) and ALU S0 L through ALU S3 L. These control signals are generated by the PROMs shown in Figure 4-28. PROM E75 is used during multiply and divide operations and its use is described in detail in Paragraph 4.3.5. Table 4-19 gives the input/output signal relationship for PROM E77 and states the ALU operation that takes place for each combination of input signals. PROM input codes 0–4 and 20–24 are related to 15-bit arithmetic. When 15-bit arithmetic is considered, the functions carried out in the ALU for codes 0–4 are repeated, respectively, for codes 20–24. Code 0 corresponds to the first entry in the table of Paragraph 4.3.3, code 1 corresponds to the second entry of that table, and so on. Both the MSB and the LSB are in the same mode when these codes are generated (the MSB mode is controlled by ALU M15 L, which is asserted by PROM E78 - refer to Table 4-20).

The rest of the input codes are related to 12-bit arithmetic and correspond to the 12-bit arithmetic entries in the table of Paragraph 4.3.3. During this arithmetic, the MSB is placed in the logic mode by the negated ALU M15 L signal. Generally, codes 5–17 and 25–37 result in the same functions; however, codes 7 and 27 produce two different results, as do codes 12 and 32. Such differences result from tests of the AL A4 L signal; codes 7 and 27 are involved in the [SIGN] operation, while 12 and 32 relate to the [EXPSIZE] operation (refer to Paragraph 4.2.8 for a discussion of the latter).

The ALU logic includes two 8097 hex buffers, although for clarity, only part of one - E50 - is shown in the figure. These buffers form the X3 Gates, which are used during address decoding operations and which are described in Paragraph 4.1.

The ALU logic also includes a Look-Ahead Carry Generator, E82. This unit permits high-speed propagation of carries by anticipating a carry across the four binary ALU ICs. A carry is propagated to both the ALU and the carry generator by multiplexer E84, and originates in either the Control ROM location (if CARRY IN H is asserted) or in the Shift logic (if CLINK H is asserted). That is: The Control ROM location sometimes adds 1 to a quantity during address calculations by asserting the CARRY BIT L signal, which generates CARRY IN H; during subtraction operations, which the ALU performs in 1's-complement arithmetic, the ARITH signals generate CARRY IN H so as to produce 2's complement subtraction of the ALU inputs; during operations when the Control ROM asserts EXTEND H, the contents of the CLINK flip-flop in the Shift logic provide a carry input (refer to Paragraph 4.3.5 for details).



74LS181

S3	S2	S1	S0	M = H LOGIC FUNCTONS	M = L ARITHMETIC FUNCTIONS (NO CARRY)	M = L ARITHMETIC FUNCTIONS (CARRY)
L	L	L	L	F = $\overline{A}$	F = A MINUS 1	F = A
L	L	L	H	F = $\overline{AB}$	F = AB MINUS 1	F = AB
L	L	H	L	F = $\overline{A}$+B	F = A$\overline{B}$ MINUS 1	F = A$\overline{B}$
L	L	H	H	F = 1	F = -1 (2's COMP)	F = 0
L	H	L	L	F = $\overline{A+B}$	F = A PLUS (A+$\overline{B}$)	F = A PLUS (A+$\overline{B}$) PLUS 1
L	H	L	H	F = $\overline{B}$	F = AB PLUS (A+$\overline{B}$)	F = AB PLUS (A+$\overline{B}$) PLUS 1
L	H	H	L	F = $\overline{A \oplus B}$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	F = A+$\overline{B}$	F = A+$\overline{B}$	F = (A+$\overline{B}$) PLUS 1
H	L	L	L	F = AB	F = A PLUS (A+B)	F = A PLUS (A+B) PLUS 1
H	L	L	H	F = A⊕B	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	F = A$\overline{B}$ PLUS (A+B)	F = A$\overline{B}$ PLUS (A+B) PLUS 1
H	L	H	H	F = A+B	F = A+B	F = (A+B) PLUS 1
H	H	L	L	F = 0	F = A PLUS A	F = A PLUS A PLUS 1
H	H	L	H	F = A$\overline{B}$	F = AB PLUS A	F = AB PLUS A PLUS 1
H	H	H	L	F = AB	F = A$\overline{B}$ PLUS A	F = A$\overline{B}$ PLUS A PLUS 1
H	H	H	H	F = A	F = A	F = A PLUS 1

L = LO, + = OR
H = HI, ⊕ = EXCLUSIVE OR

74157

STB	S0	F0
L	L	A0
	H	B0

08-1773

Figure 4-27 ALU Logic

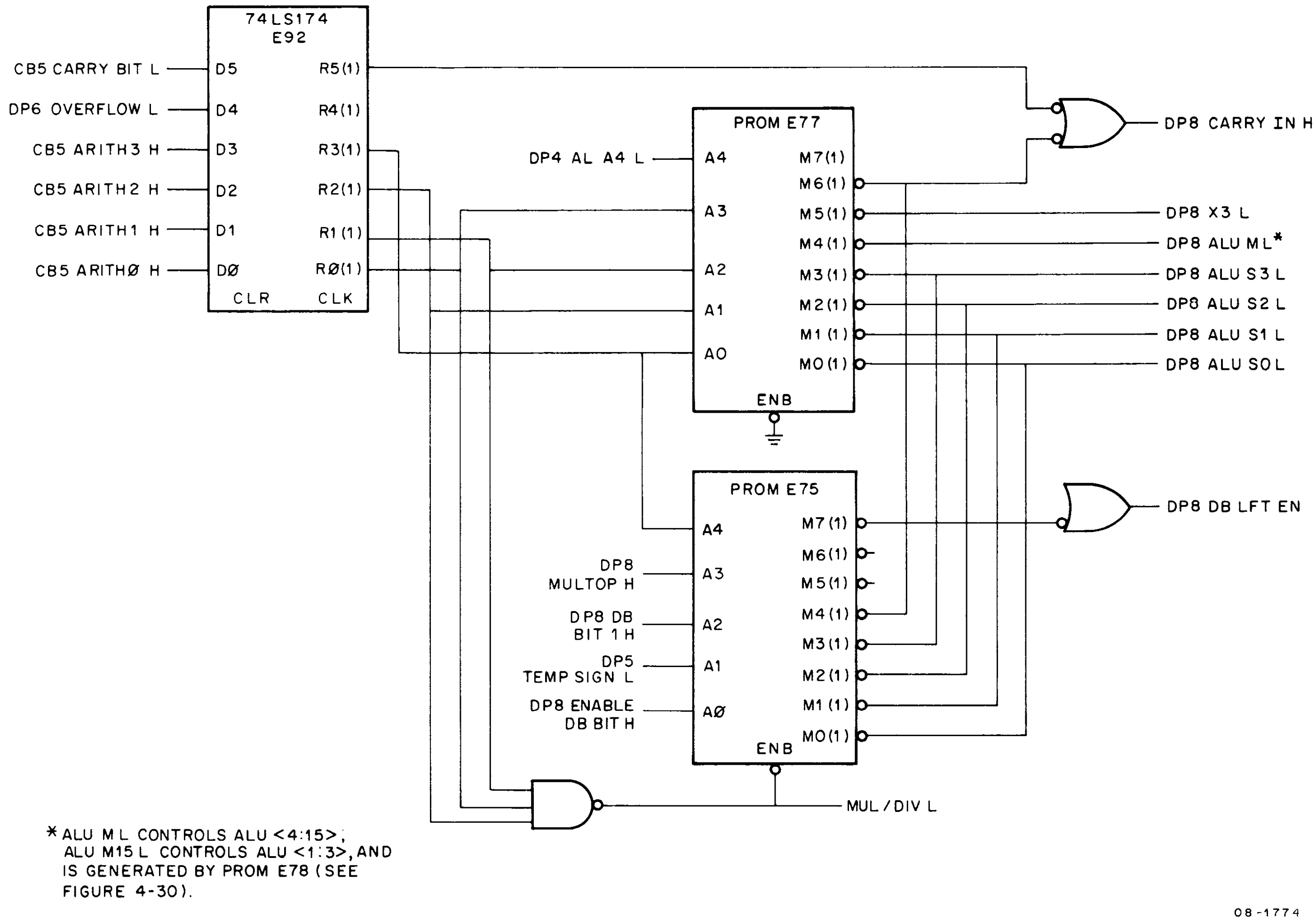


Figure 4-28 ALU Control ROMs

Table 4-19 PROM E77 Input/Output Signals
(PROM Enabled Permanently)

PROM Input Code	Input Signal Low					Output Signals Asserted							ALU Output	
	DP4 AL A4 L	ARITH0 H	ARITH1 H	ARITH2 H	ARITH3 H	ALU M L	ALU S3 L	ALU S2 L	ALU S1 L	ALU S0 L	X3 L	CARRY IN H	Logic Operation	Arithmetic Operation
0	X	X	X	X	X	X		X	X					A+B
1	X	X	X	X		X		X	X					A+B
2	X	X	X		X	X		X	X					A+B
3	X	X	X			X		X	X		X			A+B (B multiplied by 3)
4	X	X		X	X	X		X	X		X			A+B (B multiplied by 3)
5	X	X		X		X		X	X					A+B
6	X	X			X				X	X			0 (HI OUT)	
7	X	X					X	X					1 (LO OUT)	
10	X		X	X	X			X		X			B	
11	X		X	X		X	X			X		X		A–B
12	X		X		X		X	X					1	
13	X		X										A	
14	X			X	X	X		X	X					A+B
15	X			X		X		X	X					A+B
16	X				X	X								A+1
17	X					X								A+1
20		X	X	X	X	X		X	X					A+B
21		X	X	X		X		X	X					A+B
22		X	X		X	X		X	X					A+B
23		X	X			X		X	X		X			A+B (B multiplied by 3)
24		X		X	X	X		X	X		X			A+B (B multiplied by 3)
25		X		X		X		X	X					A+B
26		X			X				X	X			0	
27		X							X	X			0	
30			X	X	X			X		X			B	
31			X	X		X	X			X		X		A–B
32			X		X	X		X	X					A+B
33			X										A	
34				X	X	X		X	X					A+B
35				X		X		X	X					A+B
36					X	X								A+1
37						X								A+1

During the course of an arithmetic calculation, a number can be encountered that is either too small or too large to be represented by a 12-bit word. Either an underflow condition or an overflow condition results; both conditions are detected by XOR gate E79 and NAND gate E86. An overflow results when two positive numbers are added to produce a number greater than 3777_8. Because both numbers are positive, ALU0 L is high (during a 12-bit add the MSB (E81) is placed in an Exclusive-OR mode to facilitate overflow detection; thus, if B3 or A3, but not both, is low, ALU0 L is low); also, since AL A4 L is high, one input of E79 is high; if an overflow has occurred, ALU4 L is asserted, and E79 is enabled. Hence, OVERFLOW L is generated by E86. An underflow occurs when two negative numbers produce a result that exceeds 4000_8 (3777_8, for example). As before ALU0 L is high (1 + 1 = 0); however, ALU4 L is now high, while AL A4 L is low. Once again OVERFLOW L is asserted by E86.

4.3.3.2 Shift Gates - The Shift Gates are shown in Figure 4-29. Each unit is a 74LS253 multiplexer. The multiplexers are controlled by the shift signals so that the shifting operations indicated in the function table can be effected. Each multiplexer has four inputs. Generally, each input is supplied with a signal from one ALU output. Depending on the state of the control signals, any one of four ALU outputs can be gated onto a specific OBUS line. For example, OBUS4 L can carry the output from ALU4 L (no shift), the output from ALU5 L (shift left one), the output from ALU1 L (shift right three), or an output from the Shift logic (shift right one). Multiplexers E57 and E59–E64 are controlled by the shift signals designated LSB SHFT SEL0,1 H. These multiplexers are used for both 12- and 15-bit arithmetic. Multiplexers E74 and E58, controlled by the MSB SHFT SEL0,1 H signals, are used only during 15-bit arithmetic.

The shift select signals are generated by PROM E78, illustrated in Figure 4-30 (PROM E65 is discussed in the FIR logic, Section 4.3.1.3). Table 4-20 gives the input/output signal relationship for E78 and states the shifting operations that result. The first 5 entries in the table apply to 15-bit arithmetic, the rest to 12-bit arithmetic.

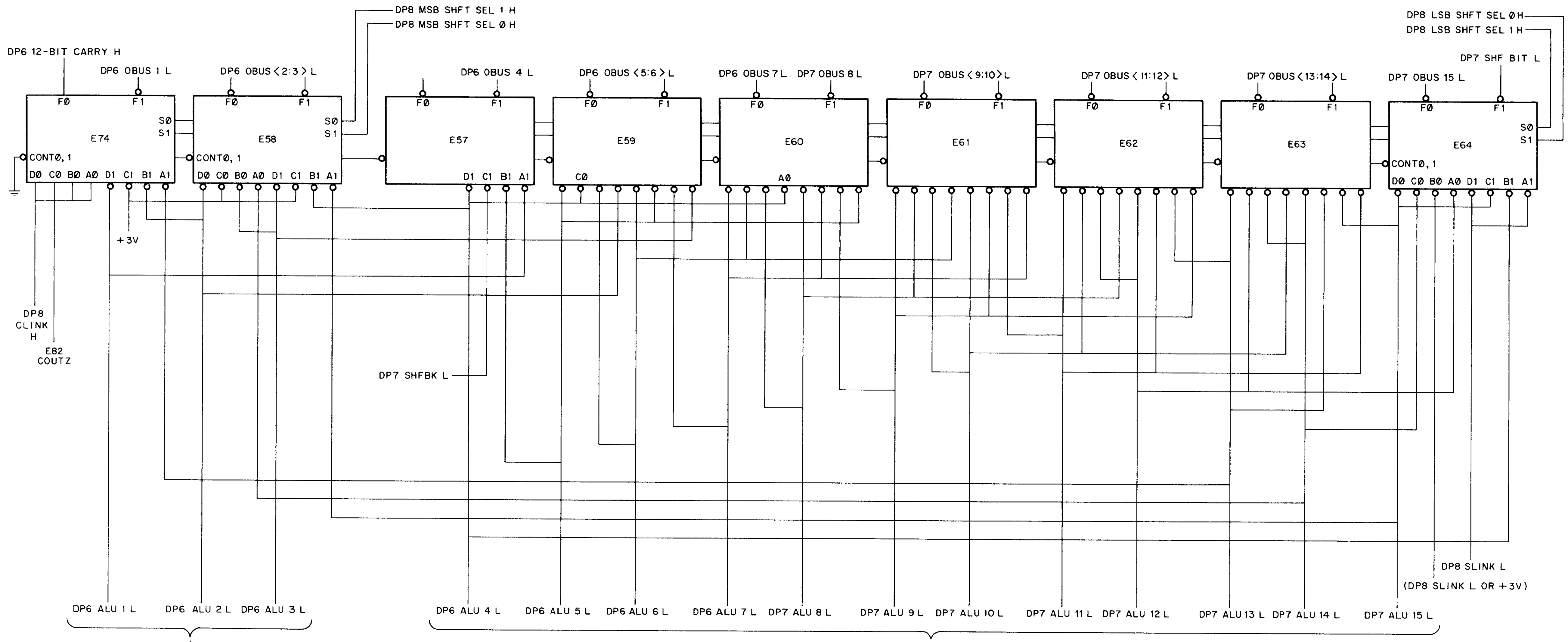
PROM Input Codes 0 and 3 result in no shift of the inputs, i.e., the ALU (1:15) L signals are gated onto the OBUS (1:15) L lines, respectively. This happens for addition and for move operations. The code 3 addition involves only the ALU B inputs, which are multiplied by 3 (X3 L is asserted by PROM E77) during the ALU operation. The shift-left that occurs during codes 1 and 4 produces a multiplication by 2 of the ALU (1:15) L information. During code 4 the X3 Gates are enabled, multiplying the B inputs by 3; thus, 6-times the ALU B inputs are gated to the OBUS lines. During a 15-bit left shift, the ALU1 L bit is lost and 0 is gated onto the OBUS15 L line (+3 V is applied to the B0 input of multiplexer E64 by the Shift logic). The last 15-bit manipulation is the shift-right three that occurs during the R3R operation. This is an end-around shift that is used to move field bits into position during pick-up and storage of the APT.

Input codes 5–17 deal with 12-bit arithmetic. Note that the MSB multiplexers are always in the shift-right one mode. Since the Cn inputs of these multiplexers are tied to +3 V, zeros are gated onto OBUS (1:3) L. Codes 5–12 are used during 12-bit addition and no shifts of the LSB are involved. Code 13 is used to recover the sign bit after an overflow has occurred during a data calculation. The OVF RECOVER H signal causes the complement of the ALU4 L signal to be gated to input C1 of multiplexer E57 and placed on the OBUS4 L line (the complement is identified as the SHFBK L signal, which is generated in the Shift logic). Thus, a FORBIDDEN result, for example, is converted from 4000_8 to 2000_8 by the OVFREC routine.

Codes 14 and 15 deal with left shift (SHL) and right shift (SHR) operations, respectively. During a SHL operation, the Shift logic provides either logic 0 or the content of the SLINK flip-flop at input B0 of E64. During a SHR operation, the Shift logic provides either the sign bit (the state of ALU4 L) or the content of SLINK at C1 of E57. Codes 16 and 17 deal with the MDS and MDLST operations; these manipulations are described in Paragraph 4.3.5

Table 4-20 PROM E78 Input/Output Signals
(PROM Enabled Permanently)

PROM Input Code	Input Signal Low					Output Signal Asserted						Shift Operation
	GND	ARITH0 H	ARITH1 H	ARITH2 H	ARITH3 H	LSB SHFT SEL 1 H	LSB SHFT SEL 0 H	MSB SHFT SEL 1 H	MSB SHFT SEL 0 H	OVF RECOVER H	ALU M15 L	
0	X	X	X	X	X	X	X	X	X		X	NO SHIFT
1	X	X	X	X		X		X			X	SHIFT LEFT 1
2	X	X	X		X						X	SHIFT RIGHT 3
3	X	X	X			X	X	X	X		X	NO SHIFT
4	X	X		X	X	X		X			X	SHIFT LEFT 1
5	X	X		X		X	X		X			LSB - NO SHIFT MSB - SHIFT RIGHT 1
6	X	X			X	X	X		X			SAME AS 5
7	X	X				X	X		X			SAME AS 5
10	X		X	X	X	X	X		X			SAME AS 5
11	X		X	X		X	X		X			SAME AS 5
12	X		X		X	X	X		X			SAME AS 5
13	X		X				X		X	X		SHIFT RIGHT 1
14	X			X	X	X			X			LSB - SHIFT LEFT 1 MSB - SHIFT RIGHT 1
15	X			X			X		X			SHIFT RIGHT 1
16	X				X	X			X			LSB - SHIFT LEFT 1 MSB - SHIFT RIGHT 1
17	X					X			X			SAME AS 16



NOTE: MULTIPLEXERS ARE 74LS253

SHFT SEL0 H	SHFT SEL1 H	Fn	TO OBUS MSB	TO OBUS LSB
LO	LO	An	ALU SHIFTED RIGHT 3	ALU SHIFTED RIGHT 3
LO	HI	Bn	ALU SHIFTED LEFT 1	ALU SHIFTED LEFT 1
HI	LO	Cn	0	ALU SHIFTED RIGHT 1
HI	HI	Dn	ALU, NO SHIFT	ALU, NO SHIFT

08-1775

Figure 4-29 Shift Gates

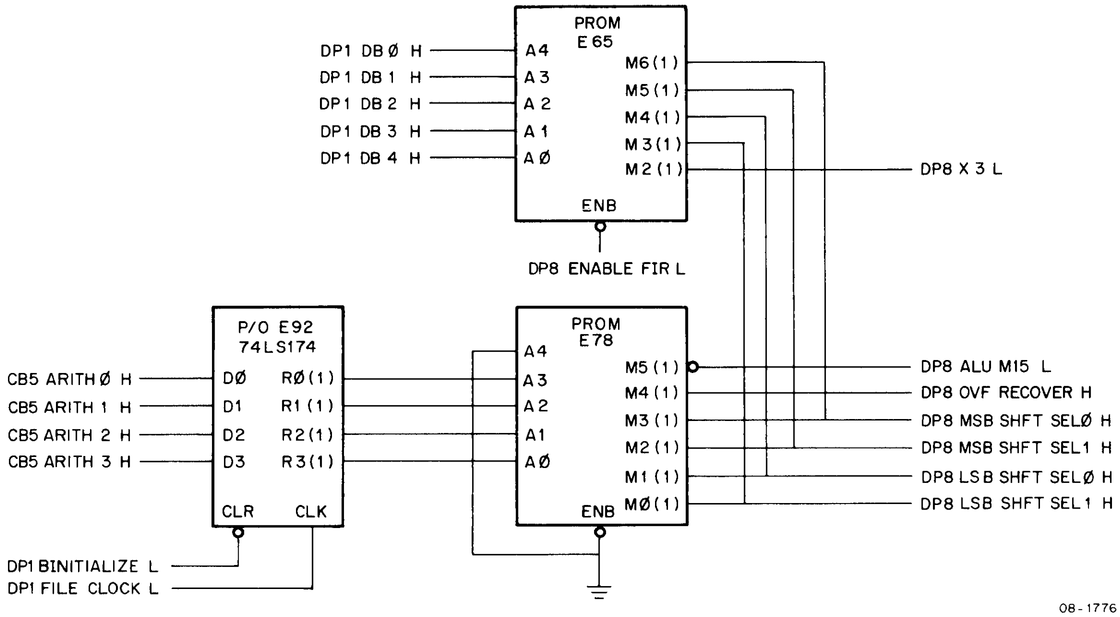


Figure 4-30 Shift Gate Control ROMs

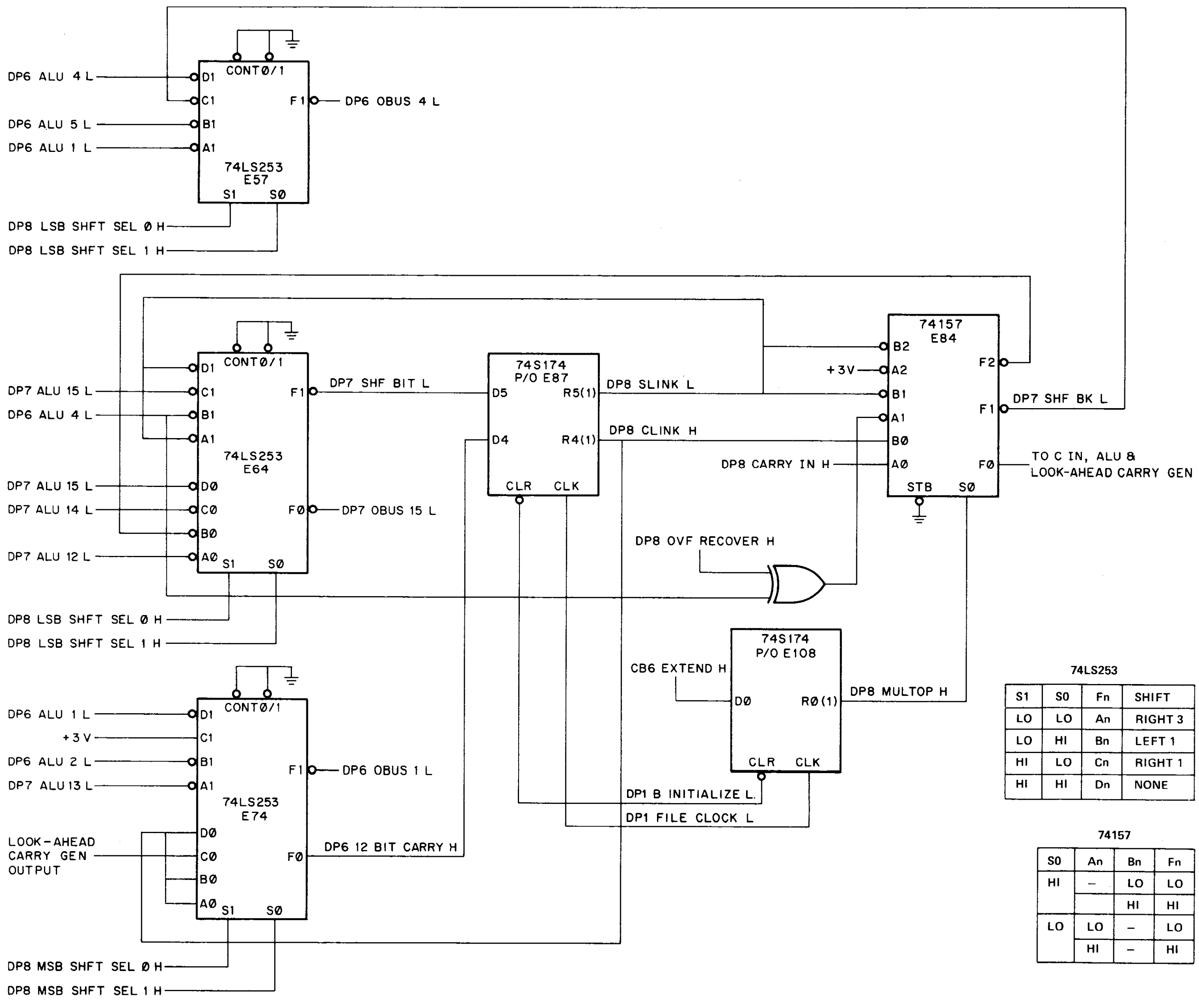
Half of multiplexer E74 is devoted to carry manipulations. In 12-bit arithmetic a carry out of the ALU is represented by the signal from COUTZ of the carry generator. This signal is gated through E74, becoming the 12-BIT CARRY H signal that is stored in the CLINK flip-flop of the Shift logic. This stored carry can then be used as a carry-in during subsequent calculations. During 15-bit arithmetic carry-in signals for the ALU are generated only by the CARRY IN H signal (Figure 4-28); the output of CLINK is gated back to its input, forming a closed loop during 15-bit calculations.

4.3.4 Shift Logic

The logic shown in Figure 4-31 is an essential ingredient during 12-bit shifting operations. The logic includes three of the shift gates, E57, E64, and E74, the SLINK and CLINK flip-flops (E87), and multiplexer E84. Bits shifted out of a word during a left or right shift are temporarily stored in the SLINK flip-flop; the stored bit can then be shifted into the MSB position or the LSB position of the next word to be shifted. Moreover, carries that are generated during 12-bit additions are temporarily stored in the logic's CLINK flip-flop, from where they can be propagated to the LSB position of the next-more-significant word.

During a SHR operation, the sign of the word being shifted is retained, while the LSB of the word is shifted into the MSB position of the next word to the right. For example: The firmware extract shown below directs a right-shift of the three MSWs of the SCRATCH register;

```
1267  DB, SCRATCHM: = [SHR] SCRATCHM              FREE*
1270  DB, SCRATCHN: = [SHR] [EXT] SCRATCHN        FREE*    IF EP, SHR2 (1273)
1271  DB, SCRATCHP: = [SHR] [EXT] SCRATCHP        FREE*    IF EXPFL, SHR1 (1266)
1272  NO OPERATION                                FREE*    RETURN
```



74LS253

S1	S0	Fn	SHIFT
LO	LO	An	RIGHT 3
LO	HI	Bn	LEFT 1
HI	LO	Cn	RIGHT 1
HI	HI	Dn	NONE

74157

S0	An	Bn	Fn
HI	–	LO	LO
		HI	HI
LO	LO	–	LO
	HI	–	HI

Figure 4-31 Shift Logic

First, while SCRATCHM is shifted right one position, its sign bit is retained and its LSB is loaded into the SLINK flip-flop; then, while SCRATCHN is shifted right one position, the bit in SLINK is loaded into the MSB position of SCRATCHN and its LSB is loaded into SLINK; finally, SCRATCHP is shifted, the SLINK bit being loaded into the MSB position and the LSB being loaded into SLINK.

To accomplish these operations, the logic begins by moving SCRATCHM onto the ALU (4:15) L lines. Both ALU4 L and ALU15 L are examined at the input of the Shift logic multiplexer E64. ALU15 L is gated to the data input (D5) of the SLINK flip-flop, while ALU4 L, the sign bit, is applied through an Exclusive-OR gate to the A1 input of multiplexer E84. During address 1267, the MULTOP H signal is low; hence, E84 gates A1, which represents the sign of SCRATCHM, to multiplexer E57; E57 then gates the sign signal onto OBUS4 L. Meanwhile, the rest of the Shift Gates have gated bits 4 through 14 of SCRATCHM onto OBUS (5:15) L, respectively; thus, SCRATCHM has been shifted right once, but its original sign is retained. At the next clock pulse, that of address 1270, the shifted information is loaded back into SCRATCHM and bit 15 is loaded into SLINK.

At the same clock pulse time, SCRATCHN is gated to the ALU (4:15) L lines. Once again ALU15 L is gated to the data input of SLINK, to be loaded eventually by the clock pulse of address 1271. Because the EXTEND H signal is now asserted by the Control ROM, MULTOP H goes high; E84 selects the output of SLINK and gates it to E57, where it is placed on the OBUS4 L line. The rest of the shift gates place bits 4 through 14 of SCRATCHN onto OBUS (5:15) L; hence, SCRATCHN has been shifted right once, its MSB position being filled with the original LSB of SCRATCHM. The SCRATCHP shift is handled the same way as SCRATCHN. Since this is the last step of this FP-mode shifting operation, any bits loaded into SLINK from SCRATCHP are lost.

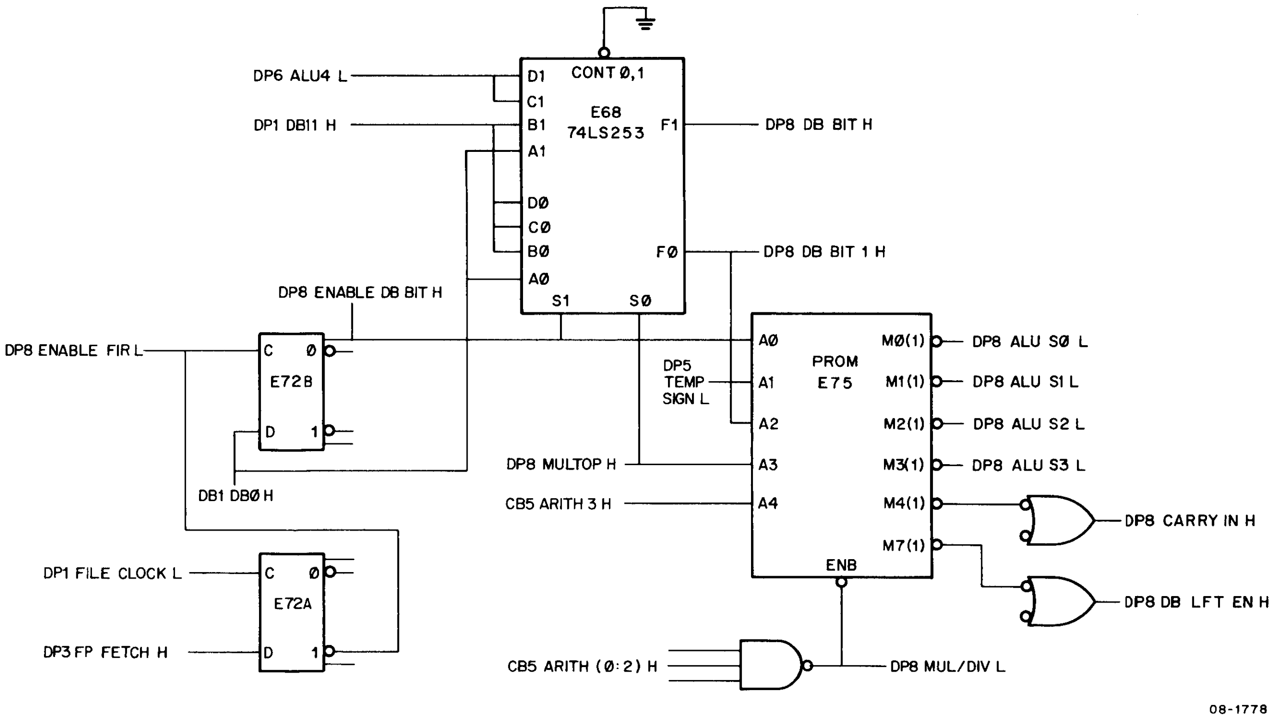
The manipulations carried out while shifting left are similar to those just described, but in the reverse direction, of course. First, the LSW is shifted; i.e., bit 4 is gated to SLINK, bits 5 through 15 are gated to OBUS (4:14) L, respectively, and zero is gated onto OBUS15 L (MULTOP H is low for the LSW shift, so E84 gates +3 V to E64, B0). Then, the next-more-significant word is shifted, its LSB being filled with the SLINK content and its MSB being loaded into SLINK for transfer to the next-more-significant word.

4.3.5 Multiply/Divide Logic

Figure 4-32 illustrates the logic that is used primarily during multiplication and division. The logic monitors a number of signals that characterize multiply and divide operations and manipulates the ALU and the DB register accordingly.

When an FPP instruction is fetched, flip-flop E72B is loaded with a bit (DB0 H) that identifies the instruction as FMUL or FDIV (DB0 H is asserted if the instruction is FMUL, but negated if the instruction is FDIV). The resulting ENABLE DB BIT H signal is applied to both multiplexer E68 and PROM E75. These two elements examine their input signals and manipulate both the ALU and the DB register according to the conditional inputs. Table 4-21 gives the input/output signal relationship for PROM E75, while Table 4-22 relates the various signal conditions to the results achieved in the ALU and in the DB register.

For example: PROM input code 11 represents a divide operation (ENABLE DB BIT H is high) with MDLST specified by the ARITH3 H signal; the PROM examines the sign of both the data word in the selected TEMP register (TEMP SIGN L) and the DB11 bit (DB11 H is gated to DB BIT1 H by E68 during a divide operation); since the two logic levels are different, the B input of the ALU is subtracted from the A input; the complement of the sign of the result (ALU4 L) is gated through E68 and to the DB register shift-left input (DB BIT H); the PROM asserts DB LFT EN H so that the DB BIT H signal is shifted into DB11 at clock pulse time; at the same time the Shift Gates are shifted left once.



74LS253

S1	SO	Fn
LO	LO	An
LO	HI	Bn
HI	LO	Cn
HI	HI	Dn

SIGNAL FUNCTION

DB0 H	ENABLE DB BIT H	MULTOP H	DB BIT 1 H	DB BIT H	FUNCTION
HI	LO	LO	DB0 H	DB0 H	MULT-ROTATE DB
HI	LO	HI	DB11 H	DB11 H	MULT EXT–NO SHIFT OF DB
LO	HI	LO	DB11 H	ALU4 L	DIV
LO	HI	HI	DB11 H	ALU4 L	DIV EXT

Figure 4-32 Multiply/Divide Logic

Table 4-21 PROM E75 Input/Output Signals

Input Code	Input Signal Low					Output Signal Asserted						ALU Arithmetic Operation
	ARITH 3 H	MULTOP H	DB BIT 1 H	TEMP SIGN L	ENABLE DB BIT H	ALU S0 L	ALU S1 L	ALU S2 L	ALU S3 L	CARRY IN H	DB LFT EN H	
11	X		X	X		X			X		X	A–B
13	X		X				X	X			X	A+B
15	X			X			X	X			X	A+B
17	X					X			X		X	A–B
20		X	X	X	X						X	A
21		X	X	X		X			X	X		A–B
22		X	X		X						X	A
23		X	X				X	X				A+B
24		X		X	X		X	X			X	A+B
25		X		X			X	X				A+B
26		X			X		X	X			X	A+B
27		X				X			X	X		A–B
30			X	X	X							A
31			X	X		X			X			A–B
32			X		X							A
33			X				X	X				A+B
34				X	X		X	X				A+B
35				X			X	X				A+B
36					X		X	X				A+B
37						X			X			A–B

Table 4-22 Multiply/Divide Conditional Operations

<table>
<tr><th colspan="4">Multiply (MDS)</th><th colspan="4">Divide (MDS and MDLST*)</th></tr>
<tr><th colspan="2">EXTEND H Asserted</th><th colspan="2">EXTEND H Negated</th><th colspan="4">EXTEND H Asserted or Negated</th></tr>
<tr><th rowspan="3">DB11 H Asserted</th><th rowspan="3">DB11 H Negated</th><th rowspan="3">DB0 H Asserted</th><th rowspan="3">DB0 H Negated</th><th colspan="2">MDS</th><th colspan="2">MDLST</th></tr>
<tr><th colspan="2">TEMP SIGN L and DB11 H</th><th colspan="2">TEMP SIGN L and DB11 H</th></tr>
<tr><th>Same Logic Level</th><th>Different Logic Level</th><th>Same Logic Level</th><th>Different Logic Level</th></tr>
<tr><td>A+B→Shift Gates
Shift Left Once
DB11→DB11
(No Rotation)</td><td>A→Shift Gates
Shift Left Once
DB11→DB11
(No Rotation)</td><td>A+B→Shift Gates
Shift Left Once
DB0→DB11
(Rotate DB Once)</td><td>A→Shift Gates
Shift Left Once
DB0→DB11
(Rotate DB Once)</td><td>A+B→Shift Gates
Shift Left Once</td><td>A–B→Shift Gates
Shift Left Once</td><td>A+B→Shift Gates
Shift Left Once
ALU4 L→DB11</td><td>A–B→Shift Gates
Shift Left Once
ALU4 L→DB11</td></tr>
</table>

* MDS: ARITH (0:3) H ASSERTED
MDLST: ARITH (0:2) H ASSERTED, ARITH 3 H NEGATED

Note that the EXTEND H signal is irrelevant to the logic in Figure 4-32 (MULTOP H and EXTEND H are synonomous – the firmware identifies the condition of the signals being asserted as [EXT]); however, the signal still has relevance when the Shift logic is considered.

4.3.6 Data Break Logic

The FPP uses data breaks to fetch instructions, to read data from memory, and to store data in memory. A data break operation is initiated by many Control ROM locations within the data break area, i.e., the area represented by μPC addresses below 400_8 (except addresses 0–3, 6, 7, 13, and 17).

The Data Break logic is divided into two sections for descriptive purposes. The first, shown in Figure 4-33, generates signals that, more or less, simply enable the FPP to control the PDP-8 CPU; the second, shown in Figure 4-35, generates signals that, generally, describe the type of data break and where in memory it is to take place.

The Omnibus Control logic, Figure 4-33, initiates the data break, assumes control of CPU gating, and performs a priority check; Figure 4034 relates the significant data break control signals. When a data break is required by the operation being carried out, the Control ROM generates an output that causes the Clock logic to assert BRK RQ H. Providing certain conditions are met, BRK RQ H asserts the Omnibus signals that allow the FPP direct access to the PDP-8 memory.

There are, essentially, two questions that the logic considers when the BRK RQ H signal is asserted: If the present timing cycle is a Data Break cycle (being used by the FPP), has the FPP been programmed so that it can use the next timing cycle as well?; is any higher priority device requesting a data break at the same time as the FPP? The first question, if appropriate, is answered at TP3 time, when flip-flop E47 is clocked. If the FPP can use consecutive memory cycles for data break transfers, the ALLOW B-B BREAKS L signal will be asserted (refer to Paragraph 4.3.7) and output R5(1) will go high at TP3 time. If the present cycle is not a Data Break cycle, the MAC flip-flop is in its clear state and the question is irrelevant; thus, R5 (1) goes high at TP3 time after BRK RQ H is asserted. Output R4(1) also goes high at this time, as does R3(1); the latter output generates the NEW BRK signal, while the former output is NANDed with the output from R5(1). This NAND operation begins the priority check procedure that answers question two.

All data break devices place their priorities on the DATA bus (i.e., they assert the DATA line assigned to them) during TS4 L; e.g., the FPP asserts DATA11 L. If a higher priority device is requesting a data break at the same time as the FPP, its DATA line is asserted. For example, if the device assigned priority 10 requests a data break, it asserts DATA10 L during TS4 L. Output F2 of multiplexer E15 goes low, inhibiting NAND gate E8. Thus, the PRIORITY OK L signal remains negated, and the FPP must wait at least one timing cycle before it can begin a break. However, if no higher priority device is present, PRIORITY OK L is asserted and the MAC flip-flop is set at TP4 time.

When the MAC flip-flop is set, MAC(0) H goes low and the logic in Figure 4-35 is enabled. MAC(0) H is applied to decoder E28 and to the BKMA/BKEMA registers. The latter registers are loaded from the OBUS with the data break address at TP4 H time, providing NEW BRK is asserted. The negated MAC(0) H signal then gates the address onto the EMA and MA lines.

Decoder E28 decodes, basically, the BKCMD (0:2) L signals asserted by the Control ROM. Table 4-23 relates the BKCMD (0:2) L signals, the output of decoder E28, and the resulting data break operation.

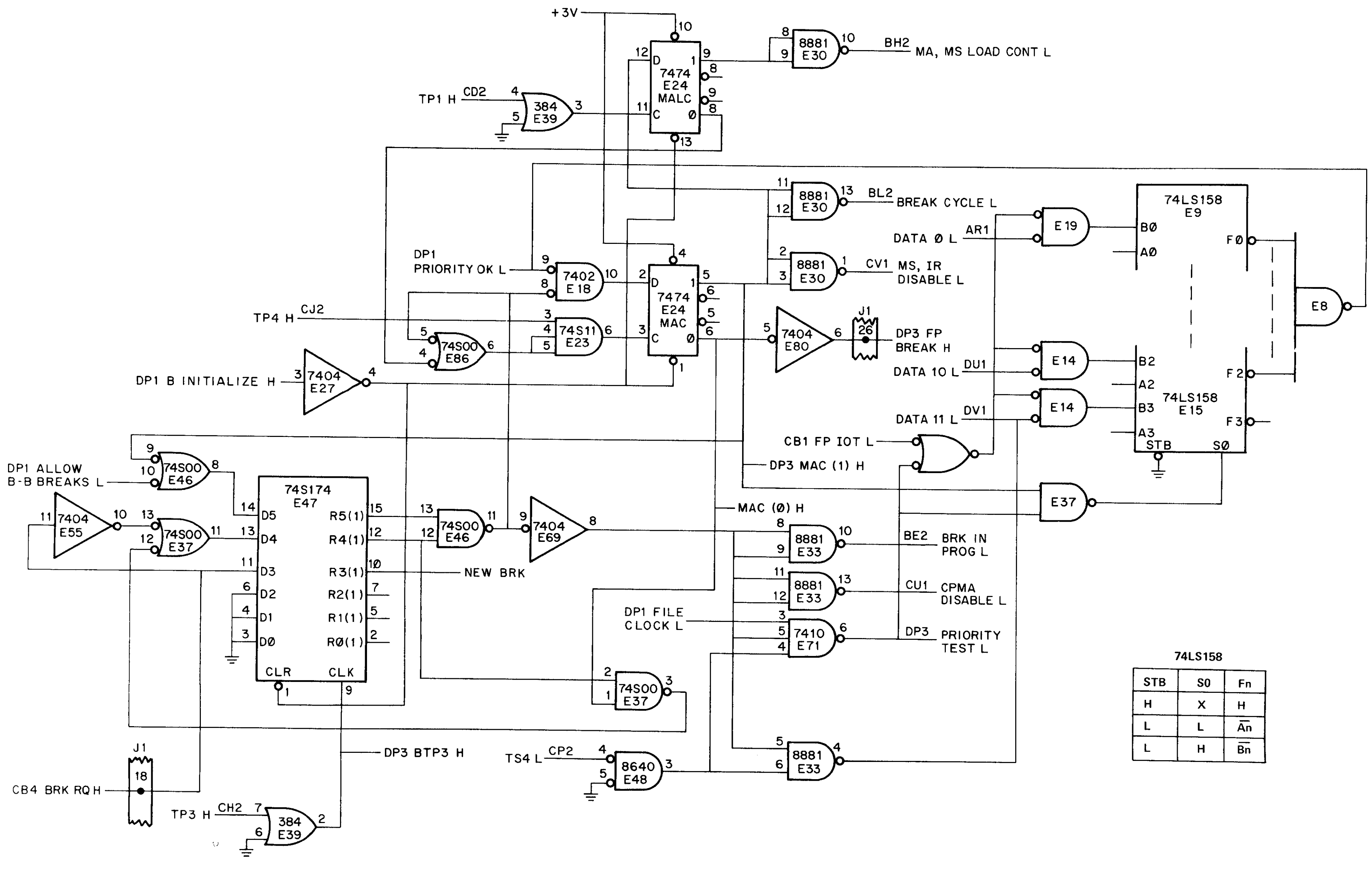


Figure 4-33 Data Break Logic, Omnibus Control

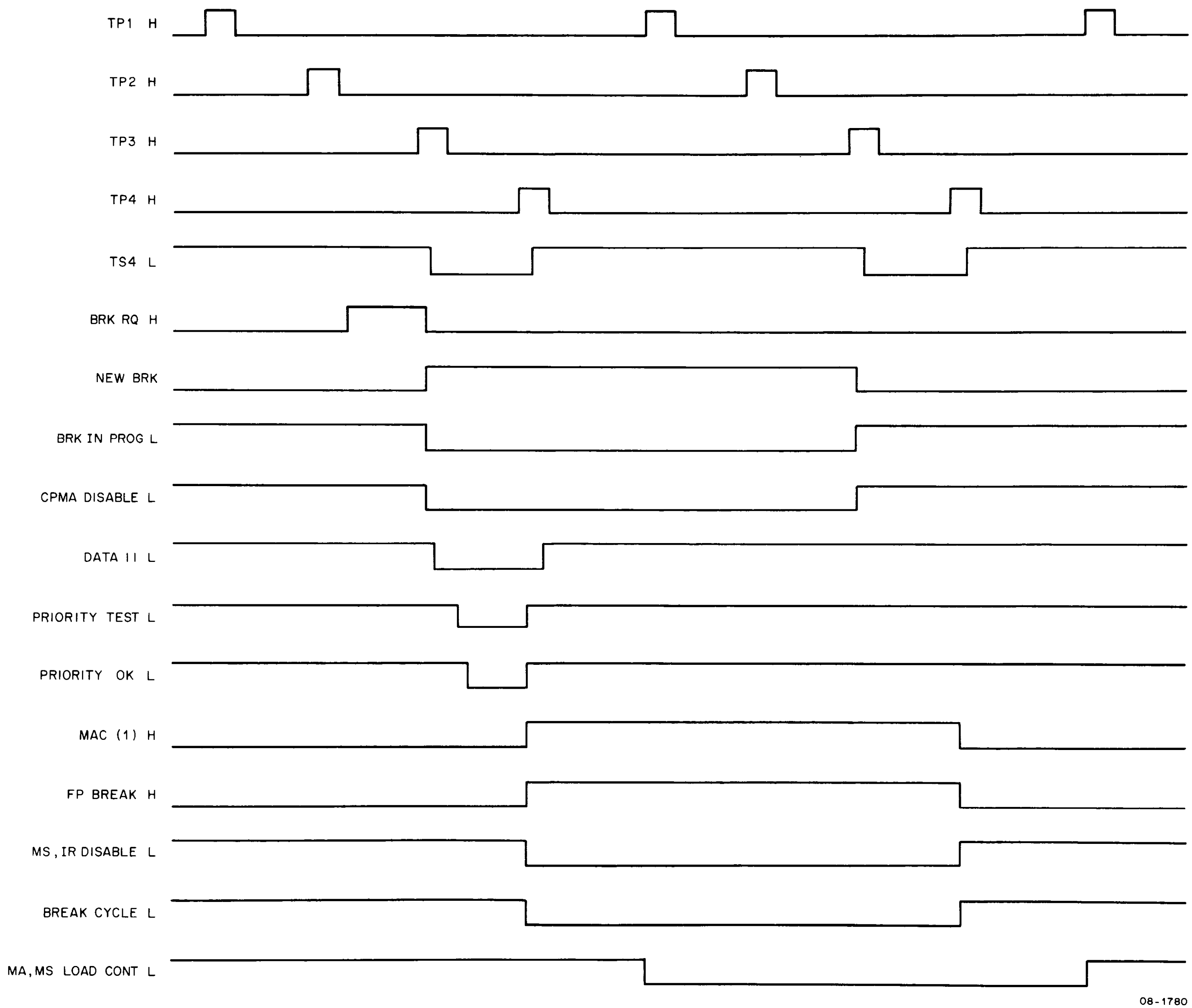
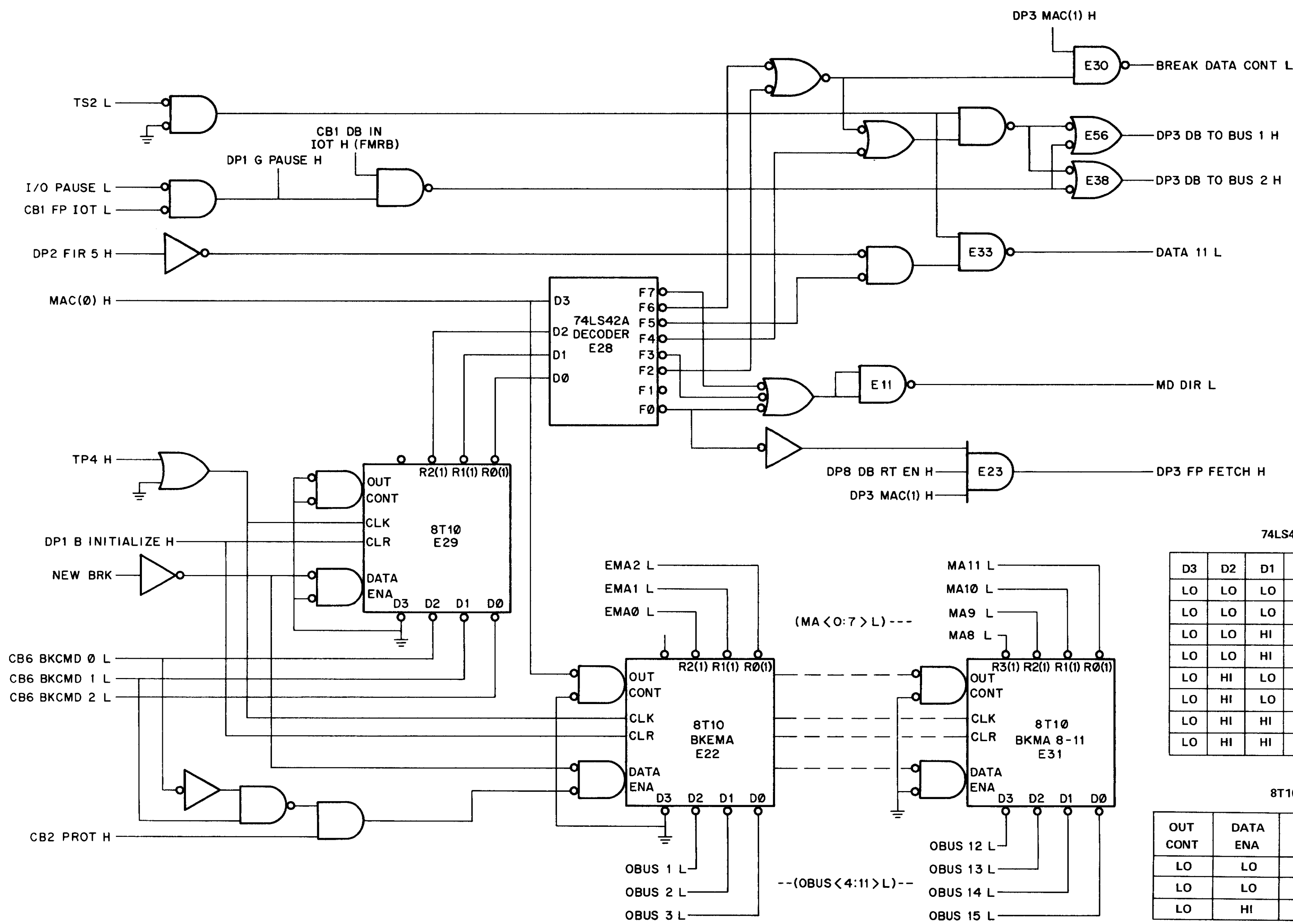


Figure 4-34 Timing, Data Break Control Signals



74LS42A

D3	D2	D1	D0	OUT LO
LO	LO	LO	LO	F0
LO	LO	LO	HI	F1
LO	LO	HI	LO	F2
LO	LO	HI	HI	F3
LO	HI	LO	LO	F4
LO	HI	LO	HI	F5
LO	HI	HI	LO	F6
LO	HI	HI	HI	F7

8T10

OUT CONT	DATA ENA	Dn	Rn (1)
LO	LO	LO	LO
LO	LO	HI	HI
LO	HI	X	NO CHANGE

08-1781

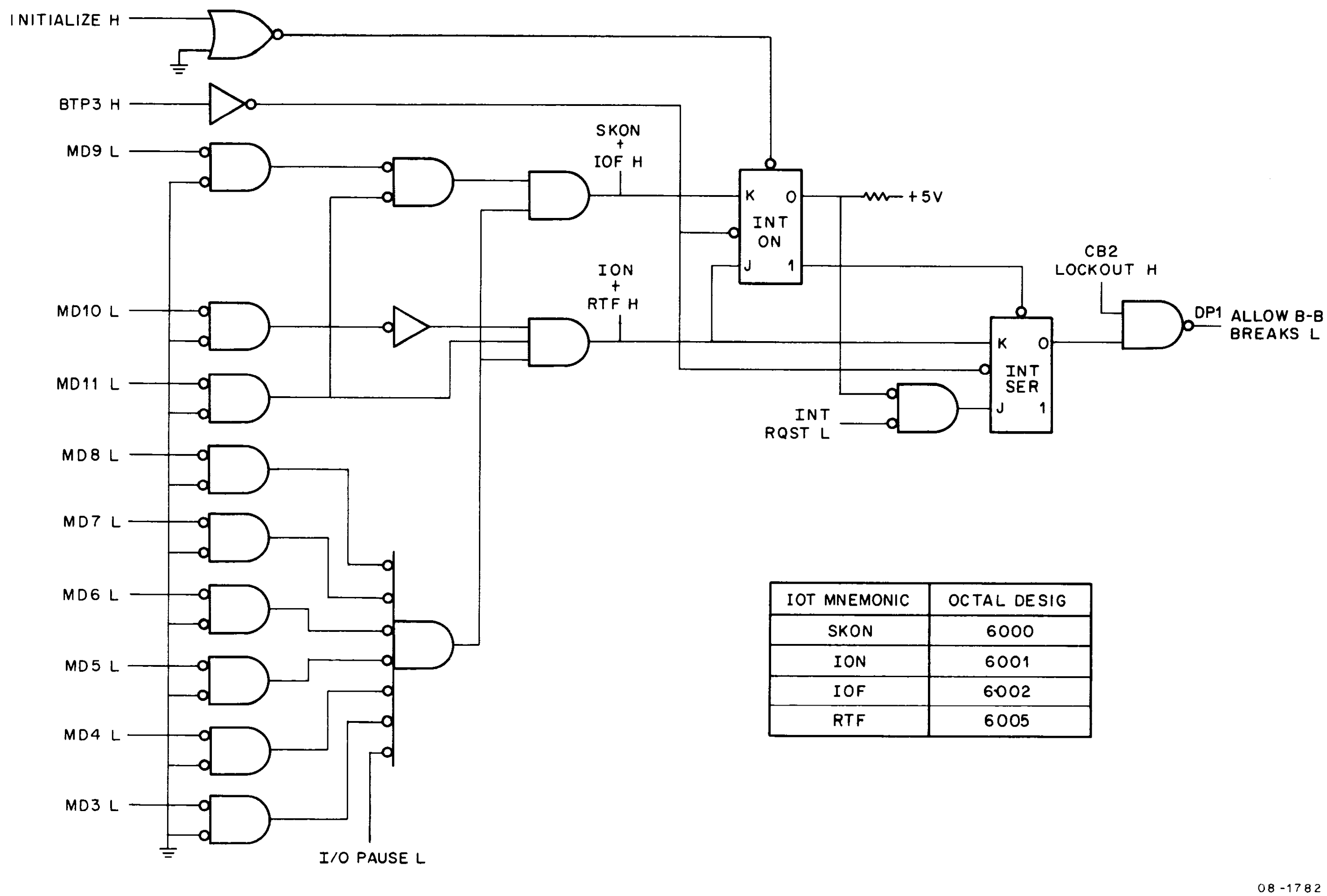
Figure 4-35 Data Break Logic, FPP Control

Table 4-23 Data Break Logic, FPP Control Signals

MAC (0) H	BKCMD 0 L	BKCMD 1 L	BKCMD 2 L	DECODER OUT LO	Signal Asserted	Result
LO	LO	LO	LO	f0	MD DIR L, BREAK DATA CONT L, FP FETCH H (if DB RT EN H and MAC (1) H are asserted)	Instruction Fetch – Read, DATA to FIR
LO	LO	LO	HI	f1	–	–
LO	LO	HI	LO	f2	DB to BUS 1, 2 H (During TS2)	Write (PROT BIT IGNORED); Used for APT Get and Put
LO	LO	HI	HI	f3	MD DIR L, BREAK DATA CONT L	Read (PROT BIT IGNORED); Used for APT Get and Put
LO	HI	LO	LO	f4	DB to BUS 1, 2 H (During TS2), BREAK DATA CONT L	ADM (if PROT = 1, BKEMA not Loaded)
LO	HI	LO	HI	f5	DATA 11 L (During TS2 if FIR5 is HI), BREAK DATA CONT L	Increment (If PROT = 1, BKEMA not Loaded)
LO	HI	HI	LO	f6	DB to BUS 1, 2 H (During TS2)	Write (If PROT = 1, BKEMA not Loaded)
LO	HI	HI	HI	f7	MD DIR L, BREAK DATA CONT L	Read (If PROT = 1, BKEMA not Loaded)

4.3.7 Lockout Logic

The logic in Figure 4-36 enables the FPP to use the Lockout mode. The logic monitors the MD lines to detect the IOT instructions that are involved with the CPU interrupt system. If the active interrupt system is turned off (SKON or IOF is issued), the INT ON flip-flop is cleared; this holds the INT SER flip-flop in the clear state and the FPP operates in the Lockout mode uninterrupted, providing the Lockout bit is set. However, if the interrupt system is turned on by ION or RTF, the INT ON flip-flop is set. If an INT RQST L is generated in the system, the INT SER flip-flop is set and the FPP goes to the Interleaved mode. At the conclusion of interrupt servicing, the ION or RTF instruction, which re-enables the interrupt system, also clears the INT SER flip-flop in the FPP. The INT ON flip-flop remains set, and the FPP automatically resumes operation in Lockout mode.



IOT MNEMONIC	OCTAL DESIG
SKON	6000
ION	6001
IOF	6002
RTF	6005

Figure 4-36 Lockout Logic

APPENDIX A
FPP8-A FIRMWARE SYNTAX

```
        THIS IS THE FORMAT OF THE FPP SOURCE CODE:

        ABSOLUTE         LABEL            DATA PATH        TIMING           CONTROL
        ADDRESS          (MNEMONIC)       STATEMENT        STATEMENT        STATEMEMT

1. ABSOLUTE ADDRESS
AN ASTERISK AS THE FIRST CHARACTER ANNOUNCES AN ABSOLUTE ADDRESS SETTING.
THE REMAINDER OF THE LINE MUST THEN BE AN OCTAL ADDRESS.

2. COMMENTS
A SLASH AS THE FIRST CHARACTER ON A LINE ANNOUNCES THAT THE LINE IS A COMMENT.

3. LABEL
MNEMONIC ADDRESS LABELS ARE SIX CHARACTERS OR LESS IN LENGTH, CONTAIN
ONLY A-Z AND 0-9, AND ARE TERMINATED WITH A COMMA-TAB PAIR.

4. DATA PATH STATEMENT

DATA PATH STATEMENTS ARE OF THE FOLLOWING FORM:

        NO OPERATION

             OR

        DESTINATION:=SOURCE[ARITHMETIC FUNCTION]SOURCE+1; DESTINATION:=SOURCE

SEVERAL DESTINATIONS MAY BE USED, WHERE LOGICALLY REASONABLE--EACH DESTINATION
IS SEPARATED FROM THE NEXT BY A COMMA-SPACE PAIR.  BY CONVENTION, THE FIRST
SOURCE IS THE FILE A SOURCE; THE SECOND ONE (AFTER THE ARITHMETIC FUNCTION)
IS THE FILE B OR CONSTANT.  THE +1 IS OPTIONAL--IT FORCES A CARRY INTO
THE ALU.  IF TWO ARITHMETIC STATEMENTS APPEAR ON THE SAME LINE,
THEY MUST:
        1. BE LOGICALLY NON-CONFLICTING
        2. BE SEPARATED BY A SEMICOLON-SPACE PAIR
GENERALLY, THE SECOND STATEMENT (IF USED) WILL BE EITHER OB:=MD
OR OB:=[0].

BKMA IS ALWAYS ONE OF THE DESTINATIONS EVERY TIME THERE
IS A T3 IN THE TIMING STATEMENT, AND A BKCMD:= MUST BE IN THE CONTROL
STATEMENT FOR EVERY T4 (EXCEPT FOR ADDRESSES 3, 6, 7, 13, 15 AND 17).

A LIST OF THE A SOURCES ARE:
        FPC
        X0
        BR
        OPADD
        APTP
        TEMA
        FIELD
        FACE
        FACM
        FACN
        FACP
        FACR
        FACS


        SC
        SCRATCHE
        SCRATCHM
        SCRATCHN
        SCRATCHP
        SCRATCHR
        SCRATCHS
        SCRATCHT
        MQM
        MQN
        MQP
        MQR
        MQS

IF FILE A IS USED BOTH AS A SOURCE AND A DESTINATION, THE SAME SOURCE AND
DESTINATION IS USED.
```

THE FILE B SOURCES ARE:

```
        0       (IMPLIED IF NOTHING IS SPECIFIED)
        K1      (PLUS 1)
        K2
        K3
        K14
        K2000
        K3777
        M1      (MINUS 1)
        M2
        M5
        M6
        M14
        M27
        M30
        M73
        TEMP<1:3>,0     (THE FIELD BITS FROM TEMP; 0 IN THE 12 LSB)
        FIR<9:11> OR FIR<5:11>  (THE FIR BITS ARE BITS FROM THE INSTRUCTION
                WORD WHICH ARE SAVED IN LATCHES IN THE DATA PATH.  THE CHOICE
                BETWEEN FIR<9:11> AND FIR<5:11> IS MADE BY AN EXTRA BIT
                ALSO LATCHED IN THE DATA PATH, WHICH RECORDS THE STATE OF BITS
                3 AND 4 OF THE INSTRUCTION WORD.  THESE TWO BITS MUST BE 0 AND 1
                RESPECTIVELY IN ORDER TO GET FIR<5:11>)
        DB      (FIELD BITS = 0)
        TEMP<1:3>,DB    (SAME AS ABOVE, EXCEPT THAT THE 3 MSB ARE FILLED
                WITH THE FIELD BITS FROM TEMP)
        FIR<6:8>        (RIGHT-JUSTIFIED--I.E, IN THE LSB)
        TEMP    (THESE EIGHT LOCATIONS ARE THE ONLY WRITABLE ONES IN FILE B)
        TEMP1
        TEMP2
        TEMP3
        TEMP4
        TEMP5
        TEMP6
        TEMP7
```

SINCE THERE ARE ONLY EIGHT WRITABLE LOCATIONS IN FILE B, THE WRITE LOCATIONS ARE:

```
        TEMP
        TEMP1
        TEMP2
        TEMP3
        TEMP4
        TEMP5
        TEMP6
        TEMP7
```

WRITING MUST BE TO A DIFFERENT FILE FROM THE ONE BEING READ.
TEMP1-TEMP6 ARE USED TO HOLD THE OPERAND FETCHED FROM MEMORY.
TEMP6 HOLDS THE EXPONENT; TEMP1 HOLDS THE MSB OF THE FRACTION; AND LESSER BITS OF FRACTION ARE STORED IN TEMP2-5 RESPECTIVELY. THUS MOST OF THE WORD MOVES, ETC TAKE PLACE VIA TEMP OR TEMP7.

THERE IS A SPECIAL OPERATION THAT INVOLVES BOTH A AND B, AND A SPECIAL ARITHMETIC OPERATION.

```
        1030!FPC<1:3>
```

THE FPC FIELD BITS ARE PLACED ON THE LSB OF THE B INPUT TO THE ALU. 1030 IS ORED WITH THE FPC FIELD BITS. THE ALU IS PLACED IN A "B ONLY" MODE (SO THAT THE FPC IS NOT ADDED TO THE WORD ON THE B INPUTS).

AS STATED ABOVE, THE ARITHMETIC FUNCTION IS ENCLOSED IN SQUARE BRACKETS, AND USED AS A DELIMITER BETWEEN THE A AND B INPUT FUNCTIONS.

15-BIT (ADDRESS CALCULATION) FUNCTIONS

FUNCTION	DESCRIPTION
[+]	15-BIT ADD
[2*]	15 BIT ADD FOLLOWED BY LEFT SHIFT. 0 IS SHIFTED INTO THE LSB; THE MSB IS LOST.
[3*]	THE CONTENTS OF THE B LEG OF THE ADDER ARE GATED ONTO THE A LEG OF THE ADDER (SHIFTED LEFT ONE PLACE). THE ADDER IS PLACED IN THE 15-BIT ADD MODE.
[6*]	A COMBINATION OF 2* AND 3*
[R3R]	THE A AND B LEGS OF THE ADDER ARE ADDED (15-BIT ARITHMETIC) AND THE RESULT ROTATED 3 PLACES RIGHT. THIS FUNCTION IS GENERALLY USED FOR MOVING FIELD BITS INTO POSITION.

12-BIT FUNCTIONS (USED FOR DATA MANIPULATION)

FUNCTION	DESCRIPTION
[12BIT]	THE 3 MSB ARE ALWAYS ZERO. THE 12 LSB ARE ADDED TOGETHER. THE MSB OF THE ALU ARE PLACED IN A SPECIAL MODE SO THAT THIS PART OF THE ALU MAY BE USED FOR OVERFLOW DETECTION.
[0]	(MAY BE USED IN EITHER 12 OR 15 BIT MODE.) ALL BITS ARE ZERO.

[SIGN] THE SIGN OF THE 12-BIT WORD ON THE A LEG OF THE ALU IS EXAMINED. IF THE SIGN BIT IS 0, THE OUTPUT OF THE ALU WILL BE ZERO. IF THE SIGN BIT IS 1, THE OUTPUT OF THE ALU WILL BE 7777.

[MINUS] A - B. TWO'S COMPLEMENT SUBTRACT, 12 BITS.

[EXPSIZE] THE SIGN OF A IS EXAMINED. IF THE SIGN OF A IS NEGATIVE, THE ALU OUTPUT IS 7777. IF THE SIGN OF A IS POSITIVE, THE CONTENTS OF B ARE ADDED TO A.

[OVFREC] OVERFLOW RECOVERY. THE ALU IS PLACED IN "12BIT" MODE. THE OUTPUT OF THE ALU IS SHIFTED RIGHT. THE COMPLEMENT OF THE SIGN IS SHIFTED INTO THE SIGN POSITION.

[SHL] 12 BIT LEFT SHIFT. 0 IS SHIFTED INTO THE VACATED BIT.

[SHR] 12 BIT RIGHT SHIFT. THE SIGN BIT IS SHIFTED INTO THE VACATED BIT POSITION.

[MDLST] LIKE MDS DESCRIBED BELOW, EXCEPT THAT THE RESULTING QUOTIENT BIT IS SHIFTED INTO THE DATA BUFFER. USED ONLY IN THE DIVIDE OPERATION.

[MDS] MULTIPLY-DIVIDE STEP. A BIT IS SAVED IN THE MAJOR REGISTERS TO DISTINGUSH BETWEEN MULTIPLY AND DIVIDE. THIS BIT IS SAVED AT THE SAME TIME THE FIR BITS ARE LATCHED.

IF THE OPERATION IS MULTIPLY: IF THE EXT BIT (DESCRIBED LATER) IS NOT ASSERTED, THE MSB OF THE DB (DB0) IS EXAMINED. IF THIS BIT IS ZERO, THE CONTENTS OF THE A LEG OF THE ALU ARE GATED TO THE SHIFT GATES, AND SHIFTED LEFT. IF THE MSB OF THE DB IS 1, AN ADD OF A AND B OCCURS BEFORE THE SHIFT. IN EITHER CASE, THE DB IS ROTATED LEFT ONE PLACE.

IF THE EXT BIT IS ASSERTED, THE SAME OPERATION AS DESCRIBED ABOVE TAKES PLACE, EXCEPT THAT THE CONDITIONAL ADD IS CONTROLLED BY DB11. NO ROTATING OF THE DB OCCURS.

IF THE OPERATION IS DIVIDE: THE SIGN OF THE WORD IN TEMP1-TEMP5 IS EXAMINED. DB11 IS ALSO EXAMINED. IF THESE TWO BITS ARE THE SAME, A IS ADDED TO B. IF THESE TWO BITS ARE DIFFERENT, B IS SUBTRACTED FROM A. THE RESULT IS SHIFTED LEFT ONE PLACE. IF THE STEP IS MDLST, RATHER THAN MDS, THE SIGN OF THE RESULT IS ALSO SHIFTED INTO THE DB.

A SPECIAL BIT, CALLED EXT IS USED TO CONTROL THE CARRY AND SHIFT LINK IN 12-BIT OPERATIONS. IF EXT IS NEGATED, THE OPERATIONS DESCRIBED ABOVE OCCUR. IF EXT IS ASSERTED, THE OPERATION PERFORMED IS SIMILAR TO THAT DESCRIBED ABOVE EXCEPT:

1. THE CARRY ASSERT IS IGNORED--THE CARRY LINK IS USED INSTEAD.
2. IF THE OPERATION IS A 12-BIT SHIFTING OPERATION OF SOME SORT, THE SHIFT LINK IS USED TO FILL THE VACATED BIT POSITION AND THE BIT WHICH WOULD NORMALLY BE LOST IS LOADED INTO THE SHIFT LINK. THE SHIFT LINK IS NOT CHANGED IF NO SHIFT IS PERFORMED.

THE EXT BIT ALSO HAS SPECIAL SIGNIFICANCE IN THE MDS OPERATION. SEE DESCRIPTION ABOVE.

5. TIMING STATEMENTS

THE FOLLOWING TIMING STATEMENTS WILL BE FOUND IN THE LISTING:

TS3	LEADING EDGE OF CLOCK PULSE COINCIDES WITH LEADING EDGE OF OMNIBUS SIGNAL TP2H; TRAILING EDGE COINCIDES WITH LEADING EDGE OF OMNIBUS SIGNAL TP3H.
BT1	FPP DATA BREAK AND OMNIBUS SIGNAL TP1
T2	OMNIBUS SIGNAL TP2H
T3	OMNIBUS SIGNAL TP3H
T4	OMNIBUS SIGNAL TP4H
FREE	FREE-RUNNING CLOCK IN THE FPP
FREE*	FREE-RUNNING CLOCK IN THE FPP

THE USE OF FREE AND FREE* IS A LITTLE BIT OF A KLUDGE. ORDINARILY, THE * FUNCTION WOULD BE IN A SEPARATE FIELD. HOWEVER, THIS CONVENTION WAS EMPLOYED TO KEEP THE WIDTH OF THE LISTING WITHIN BOUNDS. THE * BIT CONTROLS THE FILLING OF A SET OF FLAGS WHICH REFLECT THE STATE OF EITHER TEMP1-TEMP5 (IF THE * IS NOT PRESENT) OR SCRATCHM-SCRATCHS (IF THE * IS PRESENT). THE * IS NEVER PRESENT UNLESS THE FREE-RUNNING CLOCK IS ON. HENCE AT THE START OF ALL ARITHMETIC OPERATIONS, THE MOVABLE FLAGS REFLECT THE STATE OF TEMP1-5. THUS THE FPP CAN TEST FOR ZERO OPERANDS, ETC. AS SOON AS THESE INITIAL TESTS

ARE COMPLETE, AND BEFORE SCRATCH IS LOADED, THE FREE* TIMING STATEMENT APPEARS. FOR THE REMAINDER OF THE OPERATION, THE MOVEABLE FLAGS REFLECT THE STATE OF THE SCRATCH FILES BECAUSE OF THE CONTINUED PRESENCE OF THE *.

6. CONTROL STATEMENT

THE CONTROL STATEMENT GOVERNS INTERNAL HOUSEKEEPING OPERATIONS WITHIN THE FPP. THE PRIMARY USE OF CONTROL STATEMENTS IS TO GOVERN JUMPS, SUBROUTINE CALLS AND CONDITIONAL BRANCHES OF THE MICRO PC. SOME SECONDARY FUNCTIONS--SETTING OF VARIOUS BITS IN THE STATUS WORD, ETC, ARE ALSO DONE BY CONTROL STATEMENTS.

STATEMENT	EFFECT
NONE	THE MICRO PC IS INCREMENTED
BKCMD:=	SAME AS ABOVE, EXCEPT BITS 9-11 OF MICRO P IN ARE LOADED INTO THE BKCMD REGISTER OF THE MAJOR REGISTER BREAK CONTOL. THE LOADING OF THE BKCMD REGISTER HAPPENS AUTOMATICALLY BECAUSE A DATA BREAK WAS REQUESTED.
GO TO,	THIS IS AN UNCONDITIONAL JUMP OF THE MICRO PC. THE MICRO P IN BITS ARE LOADED INTO THE MICRO PC.
INSTR DISP 1	FIRST INSTRUCTION DISPATCH. THE MICRO P IN BITS ARE DETERMINED BY THE VARIOUS FAC FLAGS AND THE INSTRUCTION WORD ON THE MD LINES OF THE OMNIBUS. SEE SHEET 3 OF K-CS-M8410-0-9 FOR MORE COMPLETE DETAILS.
INSTR DISP 2	SECOND INSTRUCTION DISPATCH, USED BY DATA REFERENCE INSTRUCTIONS. THIS DISPATCH DIRECTS THE CONTROL TO THE FLDA, FSTA, GETARG OR GETN ROUTINE, DEPENDING ON THE CURRENT INSTRUCTION. SEE SHEET 5 OF K-CS-M8410-0-9.
INSTR DISP 3	THIS INSTRUCTION DISPATCH IS USED BY INSTRUCTIONS WHICH USED EITHER THE GETARG OR GETN ROUTINE AT INSTR DISP 2. INSTR DISP 3 DIRECTS THE CONTROL TO THE APPROPRIATE ARITHMETIC ROUTINE. SEE SHEET 12 OF K-CS-M8410-0-9 FOR DETAILS.
SUB,	THE CURRENT STATE OF THE MICRO PC IS SAVED IN THE SP REGISTER. THE MICRO P IN BITS ARE LOADED INTO THE MICRO PC. A FLIP-FLOP (COUNT SP) IS ALSO SET, SO THAT THE SP IS INCREMENTED AT THE NEXT MICRO PC CLOCK. THIS METHOD OF CALLING SUBROUTINES PLACES TWO IMPORTANT RESTRICTIONS ON THE CONTROL CODE--ALL SUBROUTINES MUST BE AT LEAST 2 INSTRUCTIONS LONG, AND ONLY ONE LEVEL OF SUBROUTINING IS PERMITTED.
CSUB,	LIKE "SUB," EXCEPT THAT "COUNT SP" IS SET ONLY IF THE BIT COUNTER CONTAINS ALL 1'S. ANOTHER CONTROL STATEMENT (PRESET BIT COUNT) LOADS -12 INTO THE BIT COUNTER. HENCE CSUB ALLOWS 12 SUCESSIVE CALLS TO A SUBROUTINE BEFORE PROCEEDING TO THE NEXT INSTRUCTION IN THE CALLING PROGRAM.
RETURN	RETURN FROM SUBROUTINE (EITHER TYPE). THE CONTENTS OF SP ARE LOADED INTO THE MICRO PC.

CONDITIONAL BRANCHES--IN ALL CASES, MICRO P IN IS LOADED INTO THE MICRO PC IF THE CONDITION IS MET. IF THE CONDITION IS NOT MET, THE MICRO PC ADVANCES TO THE NEXT INSTRUCTION IN SEQUENCE.

STATEMENT	BRANCH IF
IF DP,	CALCULATING MODE IS DP (24-BIT FIXED POINT)
IF EP,	CALCULATING MODE IS EP (FLOATING POINT, 60 BIT FRACTION)
IF NOT EP,	CALCULATING MODE IS EITHER 24 BIT FIXED POINT OR FLOATING POINT WITH 24-BIT FRACTION.
IF FS,	THE BIT IN THE COMMAND REGISTER INDICATING A 2-WORD ACTIVE PARAMETER TABLE IS SET.
IF ZIN,	THE BIT IN THE MAJOR REGISTERS INDICATING A ZERO WORD ON THE MD LINES IS SET.
IF OVFL,	THE ARITHMETIC OPERATION TWO STEPS BACK IN THE LISTING PRODUCED AN OVERFLOW
IF MOVE OK,	THE FIRST 13 BITS OF THE WORD IN SCRATCHM-SCRATCHN ARE ALL 1'S OR ALL 0'S. (I.E., BRANCH IF SCRATCH MAY BE NORMALIZED BY DOING WORD MOVES.)
IF NORMED,	THE WORD IN SCRATCHM, ETC, IS NORMALIZED. (BITS 0 AND 1 ARE NOT EQUAL, OR THE ENTIRE WORD IN SCRATCH IS ZERO.) NOTE: THIS TEST DOES NOT CHECK FOR SCRATCH = 6000 0000. SEE "FORBIDDEN" BELOW.
IF TO MEM,	CURRENT INSTRUCTION IS FADDM OR FMULM.
IF FACSGN,	BIT 0 OF FACM IS 1 (IE, FAC IS NEGATIVE)
IF FACZERO,	ENTIRE FAC FRACTION IS ZERO. NOTE: IF NOT IN EP MODE, HARDWARE IGNORES FACP, FACR AND FACS.
IF TEMPZERO,	THE TEMP FLAG INDICATING ZERO IS SET. NOTE: THE TEMP FLAGS ARE MOVEABLE, AND LOOK AT TEMP1-5 OR SCRATCH DEPENDING ON THE FREE/FREE* STATEMENT IN THE TIMING FIELD.

```
IF TEMPSGN,      THE TEMP FLAG INDICATING A NEGATIVE FRACTION IS SET.
                         THE NOTE ABOVE APPLIES.

IF FORBIDDEN,    SCRATCHM=SCRATCHN (OR SCRATCHM=SCRATCHS) =4000 0000.  CALCULATIONS
                         THAT RESULT IN 6000 0000 NORMALIZE ONE STEP TOO
                         MANY, TEST FOR THE FORBIDDEN NUMBER WITH THIS
                         BRANCH TEST, AND THEN SHIFT THE SCRATCH AREA
                         RIGHT ONE PLACE.

IF SGN,          THE SIGN OF THE ARITHMETIC OPERATION TWO STEPS BACK IN
                         THE LISTING WAS NEGATIVE.

IF EXPFL,        TEST THE STATE OF THE EXPONENT FLAG.  BRANCH IF IT IS
                         SET.  (THE EXPONENT FLAG REFLECTS THE STATE
                         OF THE SC REGISTER.  IT IS SET IF THE LAST OPERATION
                         LOADING THE SC PRODUCED EITHER A NEGATIVE RESULT
                         OR A POSITIVE RESULT AND AN OVERFLOW.  OTHERWISE
                         IT IS CLEARED.)

IF NZSET,        THE EXPONENT UNDERFLOW FLAG IS CLEARED, OR THE
                         ZTRAP BIT OF THE COMMAND REGISTER IS SET.
```

MISCELLANEOUS CONTROL OPERATIONS

```
STATEMENT        OPERATION
-------------------------------------------------------------------
SET TRAPI        SET THE TRAPI FLAG IN THE STATUS REGISTER

SET DIV0         SET THE DIV0 FLAG IN THE STATUS REGISTER

TEST OVFL        TEST OVERFLOW.  THIS OPERATION TESTS THE ARITHMETIC OPERATION
                         TWO STEPS BACK IN THE LISTING.
                                 DP: SET THE DPOVF FLAG
                                 NOT DP: IF SGN IS -, COMPLEMENT
                                         EXPOVF.  IF SGN IS +, COMPLEMENT
                                         UNDFLO

ENTER (DP OR FP OR EP) MODE
                 MICRO P IN 10 AND MICRO P IN 11 ARE LOADED INTO THE DP
                 AND EP FLOP-FLOPS, CAUSING A CHANGE IN CALCULATING
                 MODE.
                         BIT 10          BIT 11          NEW MODE
                         0               0               FP
                         0               1               DP
                         1               0               EP
                         1               1               ILLEGAL--NOT USED.

PRESET BIT COUNT
                 THE BIT COUNTER DESCRIBED IN THE CSUB OPERATION IS PRESET
                         TO -12 (1'S COMPLEMENT).


EXTEST           IF NONE OF THE CONDITIONS DESCRIBED BELOW IS MET, GO
                         TO FETCH (MICRO PC = 20)
                         IF ANY CONDITION IS MET, GO TO EXSTRT (MICRO PC
                         = 1000).
                                 CONDITIONS:
                                         FORCED EXIT FLAG SET
                                         DIV0 FLAG SET
                                         DPOVF FLAG SET
                                         EXPOVF FLAG SET
                                         UNDFLO FLAG SET, AND ZTRAP
                                                 COMMAND BIT ALSO SET.
```

APPENDIX B
FPP8-A – FPP12-A DIFFERENCES

THE FPP8-A IS A CODE-COMPATIBLE FPP WHICH PLUGS INTO THE OMNIBUS. IT WILL RUN FPP12-A FORTRAN IV WITHOUT MODIFICATION.

A. PHYSICAL DIFFERENCES

FPP12-A: 1 CABINET, CONTAINING POWER SUPPLY AND 6 MOUNTING PANELS OF LOGIC. THE FPP12-A REQUIRES KA-8E AND KD-8E IN ORDER TO RUN ON AN OMNIBUS-TYPE MACHINE. POWER CONSUMPTION IS 250 WATTS, PLUS THE POWER REQUIRED BY THE KA AND KD.

FPP8-A: 2 HEX MODULES WHICH PLUG INTO THE OMNIBUS. A SINGLE INTERCONNECTING CABLE BETWEEN THE FPP8-A MODULES USES STANDARD BERG 50-PIN HEADERS. OMNIBUS POWER REQUIRED IS +5 VOLTS AT 8.8 AMPERES (44 WATTS). NO KD-8E OR KA-8E IS REQUIRED; NO CONNECTIONS TO EXTERNAL PERIPHERALS ARE MADE FROM THE FPP8-A MODULES.

B. INSTRUCTION SET DIFFERENCES

1. FPP8-A IS AVAILABLE IN ONE FLAVOR ONLY--THE 60-BIT EXTENDED PRECISION MODE IS BUILT IN.

2. ALL UNDEFINED INSTRUCTIONS EXECUTE NO OPERATION IN THE FPP8-A. MOST UNDEFINED FPP12A INSTRUCTIONS EXECUTE NO OPERATION, BUT SOME ARE NOT TESTED.

3. MAINTENENCE IOTS ARE DIFFERENT.

4. THE "LOCKOUT" BIT, BIT 8 OF THE COMMAND REGISTER, ALLOWS THE FPP8-A COMPLETE ACCESS TO THE BREAK SYSTEM WHEN IT IS SET. (THE FPP12-A CAN TAKE BREAKS AT A MAXIMUM RATE OF ONE BREAK EVERY OTHER MEMORY CYCLE.) THE FPP8-A IS DESIGNED SO THAT IT CAN KEEP THE BREAK SYSTEM TIED UP FETCHING INSTRUCTIONS, OPERANDS, ETC. IF THE LOCKOUT BIT IS SET, THE FPP8-A WILL RELINQUISH THE BUS ONLY WHEN IT NEEDS TO DO SOME ARITHMETIC WORK BEYOND MERE ADDRESS CALCULATIONS. WHENEVER THE FPP8-A DISCERNS AN INTERRUPT SERVICE IS BEING PERFORMED BY THE PDP8-A, IT TEMPORARILY DISABLES THE LOCKOUT MODE AND RUNS AT HALF SPEED UNTIL THE NEXT ION INSTRUCTION IS GIVEN.

5. COMMAND REGISTER BITS 4, 5, 6 AND 7 WORK DIFFERENTLY IN THE FPP8-A. IN THE FPP12-A, THESE BITS CONTROL STORING OF OPERAND ADDRESS, X0, BR AND FAC RESPECTIVELY UPON EXIT. IN THE FPP8-A, THESE BITS ARE TESTED AS A UNIT. IF ALL FOUR BITS ARE 1, THE FPP8-A GOES TO A "FAST ENTRY AND EXIT" MODE, WHERE IT PICKS UP AND STORES ONLY THE FPC ON ENTRY AND EXIT. ANY OTHER COMBINATION OF BITS CAUSES THE FPP8-A TO PICK UP AND RESTORE THE ENTIRE APT.

CAUTION: THIS WILL WORK ON FORTRAN IV. THERE MAY BE SOME OTHER PEOPLE WHO HAVE WRITTEN THEIR OWN PROGRAMS WHERE THEY USE A SHORTER, BUT NOT THE SHORTEST, APT. IF SO, THEY MAY BE IN TROUBLE. NOTE, HOWEVER, THAT THE FAC IS AT THE END OF THE APT. HENCE THE ONLY PEOPLE WHO COULD POSSIBLY BE IN TROUBLE ARE THOSE WHO USE THE FOLLOWING CONFIGURATION OF BITS IN THE COMMAND REGISTER:

BIT 7=1; BIT 4=0; BITS 5 AND 6 ANYTHING. (DO NOT SAVE THE FAC, BUT SAVE OPERAND ADDRESS AND PERHAPS X0 AND/OR BR)

BITS 7 AND 4=1; BIT 6=0; BIT 5 ANYTHING. (DO NOT SAVE THE FAC OR OPERAND ADDRESS, BUT SAVE BR AND PERHAPS X0.)

BITS 7, 4 AND 6=1; BIT 5=0. (DO NOT SAVE FAC, OPERAND ADDRESS OR BASE REGISTER, BUT SAVE THE INDEX REGISTER POINTER.)

OR WHO STORE CONSTANTS IN OTHERWISE UNUSED LOCATIONS IN THE APT.

6. ALL EXECUTION TIMES ARE DIFFERENT. IN GENERAL, LOADS AND STORES ARE CONSIDERABLY FASTER IN THE FPP8-A; ADDS, SUBS AND SHORT MULTIPLIES ARE SLIGHTLY SLOWER; 60-BIT MULTIPLIES AND DIVIDES ARE 50% SLOWER IN THE FPP8-A. WITH THE LOCKOUT BIT SET, FORTRAN IV RUNS AT ABOUT THE SAME SPEED AS IT DID ON THE FPP12A.

7. THE FPP12-A HAS 4 GUARD BITS WHEN IT IS RUN IN FP OR DP MODE. THE FPP8-A HAS 12 GUARD BITS. BOTH FPPS USE NO GUARD BITS IN EXTENDED PRECISION (60-BIT) MODE.

8. THREE OF THE 5 TRAP INSTRUCTIONS ARE REPLACED BY NEW COMMANDS. TRAP3 AND TRAP4 REMAIN AS IN THE FPP12-A.

```
OP CODE:  101 000 CCC 000
MNEMONIC:  LTR
OPERATION:  LOAD TRUTH.  CCC (THE CONDITION) IS
        DEFINED IN THE SAME WAY AS FOR BRANCH INSTRUC-
        TIONS.  IE, CCC=0 MEANS DO IT IF FAC=0, ETC.
        IF CONDITION IS MET, LOAD FAC WITH A FLOATING
        1.0.  (IN DP MODE, FAC IS LOADED WITH 2000 0000.)
        IF CONDITION IS NOT MET, CLEAR FAC.

OP CODE:  110 00+ XR MSB
              ADDRESS LSB
MNEMONIC: LEA OR IMUL (DEPENDING ON MODE)
OPERATION:  IN FP OR EP MODE, LOAD EFFECTIVE ADDRESS.
        DO AN ADDRESS CALCULATION, AS THOUGH THIS WERE
        A 24-BIT DIRECT DATA REFERENCE INSTRUCION.  THEN
        DUMP THE RESULTING ADDRESS INTO BITS 9-23 OF THE
        FAC AND CHANGE TO DP MODE.
                IN DP MODE, FETCH OPERAND AND PERFORM
        A SIGNED INTEGER MULTIPLY ON THE CONTENTS OF THE
        FAC, LEAVING THE RESULT IN THE FAC.

OP CODE:  111 00+ XR OFFSET
MNEMONIC: LEAI OR IMULI (DEPENDING ON MODE)
OPERATION:  IN FP OR EP MODE, LOAD EFFECTIVE ADDRESS
        INDIRECT.  DO A SINGLE WORD INDIRECT ADDRESS
        CALCULATION, PLACE THE ADDRESS INTO FAC9-23, AND
        CHANGE TO DP MODE.
                IN DP MODE, FETCH OPERAND, CALCULATING
        THE ADDRESS USING THE INDIRECT ADDRESSING RULE.
        FETCH OPERAND, AND PERFORM A SIGNED INTEGER
        MULTIPLY BETWEEN THE OPERAND AND THE FAC.
        THE RESULTS OF THE MULTIPLY ARE LEFT IN THE FAC.
```

9. ABSOLUTELY NO ATTEMPT IS BEING MADE TO HAVE THE FPP8-A RUN ON A DW-8E. INDEED, THERE ARE FUNDAMENTAL REASONS WHY THE FPP8A CAN NEVER RUN ON THE EXTERNAL PDP-8 I/O BUS--THE INTERRELATIONSHIP BETWEEN FPP AND OMNIBUS TIMING SIGNALS, AND THE RELIANCE ON THE OMNIBUS ADD-TO-MEMORY DATA BREAK FEATURE.

10. OPADD BEHAVES SLIGHTLY DIFFERENTLY. IT IS UNAFFECTED BY JNX AND BRANCH INSTRUCTIONS.

11. THE FPP12-A WAS SELF-INCONSISTENT IN THAT FPICL DID NOT CHANGE THE STATE OF THE FPP FROM DP TO FP, BUT DID CHANGE IT FROM EP TO FP. THE FPP8-A WILL ALWAYS RETURN TO FP MODE ON AN FPICL, FPIST OR CAF IOT.

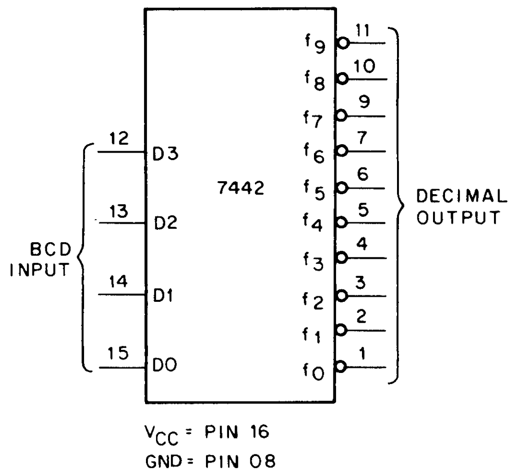
APPENDIX C
IC DESCRIPTIONS

This appendix describes the ICs listed below. The "S" in an IC designation indicates Schottky-clamped, TTL logic, while an "L" indicates a low-power device. Any such devices are functionally identical to TTL alone; for example, the 74LS151 and the 74151 are identical, and the two designations might be used in the same piece of literature.

74LS42
7475
8T10
8136
82S21
8234
8266
8613
74LS139
74LS151
74LS157
74LS158
74LS161
74LS181
74LS182
74LS194
74LS253
82S112
FPLA (14X48X8)

7442 4 LINE TO 1 LINE DECODER

These BCD-to-decimal decoders consist of eight inverters and ten 4-input NAND gates. The inverters are connected in pairs to make BCD input data available for decoding by the NAND gates.



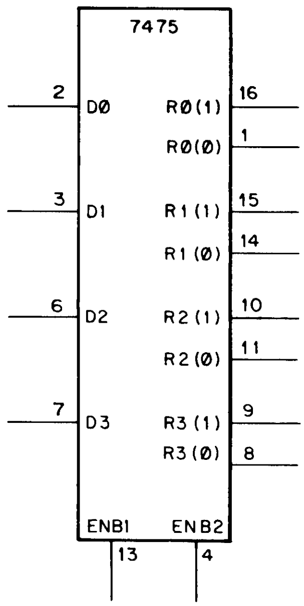
IC-7442

7442
TRUTH TABLE

BCD Input				Decimal Output									
D3	D2	D1	D0	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

7475 4-BIT BISTABLE LATCH

The 7475 latches are used for temporary storage of binary information. Information present at a data (D) input is transferred to the R output when the clock is high, and the R output will follow the data input as long as the clock remains high. When the clock goes low, the information present at the data input at the time of the transition is retained at the R output until the clock is permitted to go high. Input ENB1 is the clock input for data inputs D0 and D1. ENB2 is the clock input for data inputs D2 and D3.



VCC PIN 5
GND PIN 12

DN INPUTS		RN (1) OUTPUTS		CLOCK INPUTS
D0	0 1	0 1	R0	ENB1
D1	0 1	0 1	R1	
D2	0 1	0 1	R2	ENB2
D3	0 1	0 1	R3	

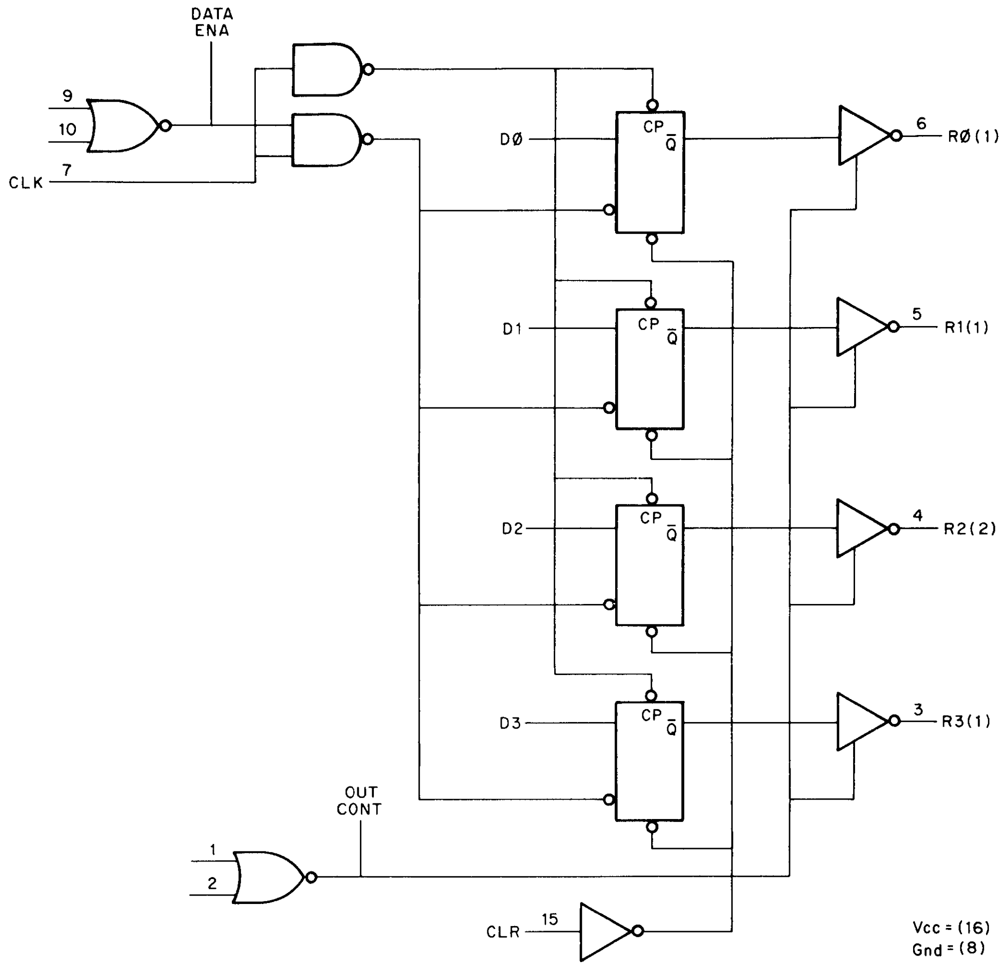
NOTE:

1. RN (O) Outputs = inverted RN (1) outputs.
2. ENB1 and ENB2 clock on negative going edge.

IC-7475

8T10 QUAD, D-TYPE, BUS FLIP-FLOP

The 8T10 outputs present a high impedance to the bus when disabled and active drive when enabled. When both the inputs and outputs are enabled, the output follows the data input when clocked; if the input is disabled, while the output is enabled, the output remains in the state it exhibited before the clock pulse.



LOGIC DIAGRAM

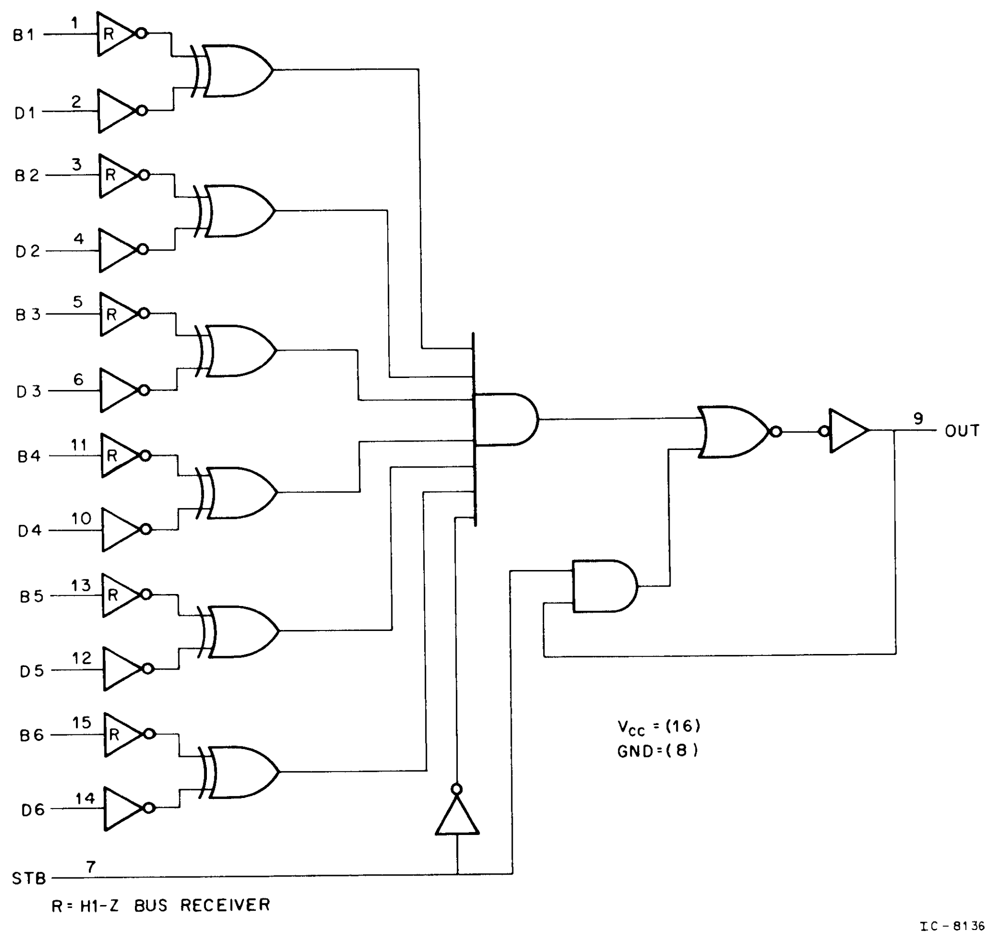
Dn	DATA ENA	OUT CONT	Rn (1)
LO	HI	HI	LO
HI	HI	HI	HI
X	LO	HI	PREVIOUS OUTPUT
X	X	LO	HI-Z

TRUTH TABLE

IC-8T10

8136 6-BIT, UNIFIED-BUS COMPARATOR

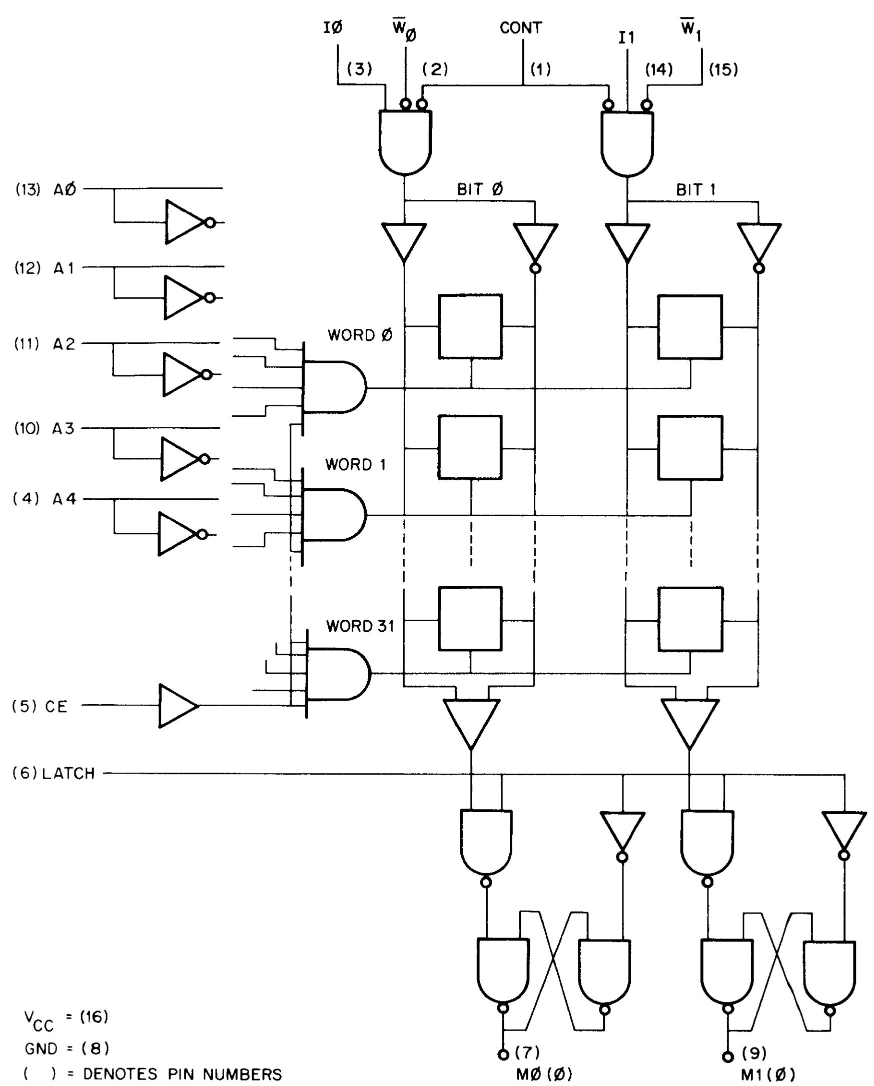
The 8136 compares two binary words (from 2 to 6 bits in length) and indicates matching bit-for-bit of the two words. Inputs for one word are TTL, while those of the second word are high impedance receivers driven by a terminated data bus. The transfer of information to the output occurs as long as the STB input is logic 0. Inputs may be changed while the STB input is at the logic 1 level without affecting the state of the output.

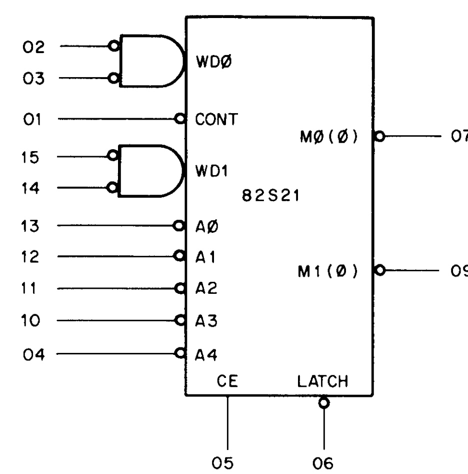


82S21 64-BIT, WRITE-WHILE-READ RAM

The 82S21 IC is organized in 32 words of 2 bits each. Words are selected through a 5-input decoder when the read-write enable input, CE, is a logic 1 (hi). $\overline{W0}$ and $\overline{W1}$ are the write inputs for bit 0 and bit 1 of the word selected. CONT is the write control input. When $\overline{WX}$ and CONT are both at logic 0, data on the I0 and I1 data lines are written into the addressed word. The read function is enabled when either $\overline{WX}$ or CONT is at logic 1.

An internal latch on the chip provides the write-while-read capability. When the latch control line, LATCH, is logic 1 and data is being read from the 82S21, the latch is effectively bypassed. The data at the output will be that of the addressed word. When LATCH goes from a logic 1 to a logic 0, the outputs are latched and remain latched regardless of the state of any other address or control line. When LATCH goes from 0 to 1, the outputs unlatch and the outputs become that of the present address word.





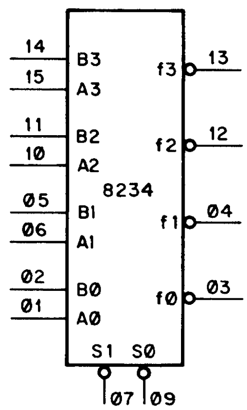
TRUTH TABLE

CE	CONT	$\overline{W}_0$	$\overline{W}_1$	LATCH	MODE	OUTPUTS
X	X	X	X	0	Output Hold	Data from last addressed word when CE = "1"
0	X	X	X	1	Read & Write Disabled	Disabled logic "1"
1	1	X	X	X	Read	Data stored in addressed word
1	0	1	1	X	Read	Data stored in addressed word
1	0	0	0	0	Write Data	Data from last word address when LATCH went from "1" to "0"
1	0	0	0	1	Write Data	Data being written into memory
1	0	0	1	X	Write Data into Bit 0 Only	If LATCH = 0: Data from last word address when LATCH went from "1" to "0"
1	0	1	0	X	Write Data into Bit 1 Only	If LATCH = 1: Data being written into the selected bit location and stored in other addressed location

IC-82S21

8234 2-INPUT 4-BIT MULTIPLEXER

The 8234 is a 2-input, 4-bit multiplexer designed for general-purpose data-selection applications. The 8234 features inverting data paths.



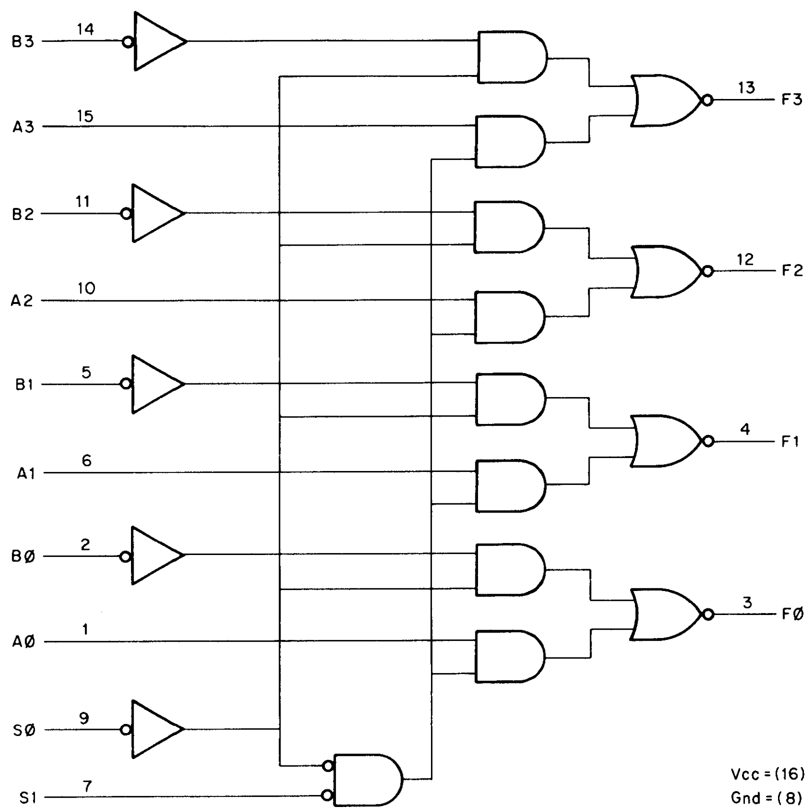
S1	S0	f
0	0	$\overline{B}$
1	0	$\overline{A}$
0	1	$\overline{B}$
1	1	1

GND = PIN 8
VCC = PIN 16

IC-8234

8266 2-INPUT, 4-BIT MULTIPLEXER

The 8266 is a 2-input, 4-bit multiplexer. Input selection is controlled by the S0 and S1 select lines.



LOGIC DIAGRAM

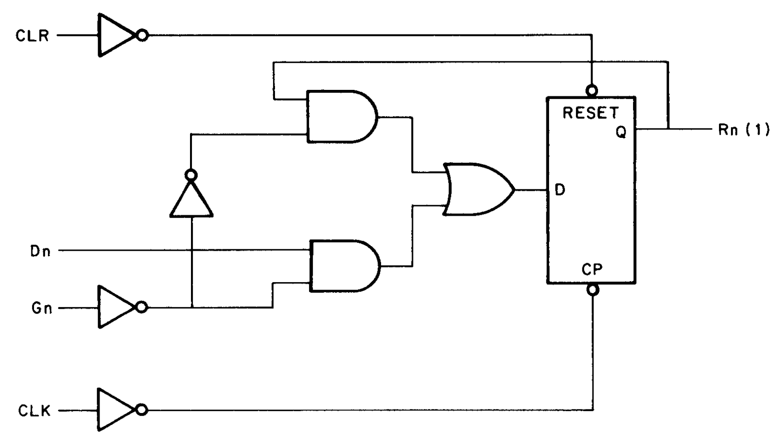
SO	S1	Fn
LO	LO	Bn
LO	HI	Bn
HI	LO	$\overline{An}$
HI	HI	HI

TRUTH TABLE

IC-8266

8613 QUAD, GATED D FLIP-FLOP

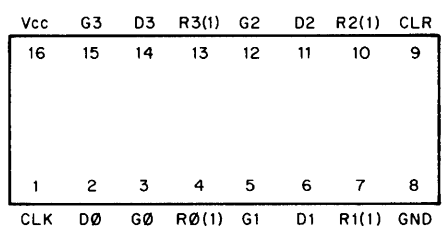
The 8613 is a positive-edge-triggered, quad, gated D flip-flop with direct clear and gated inputs. The gate, if set to a logical 1 level, will inhibit data entry from the data input.



LOGIC DIAGRAM (ONE FLIP FLOP)

Dn	Gn	CLR	Rn (1)
HI	LO	LO	HI
LO	LO	LO	LO
X	HI	LO	NO CHANGE
X	X	HI	LO

TRUTH TABLE

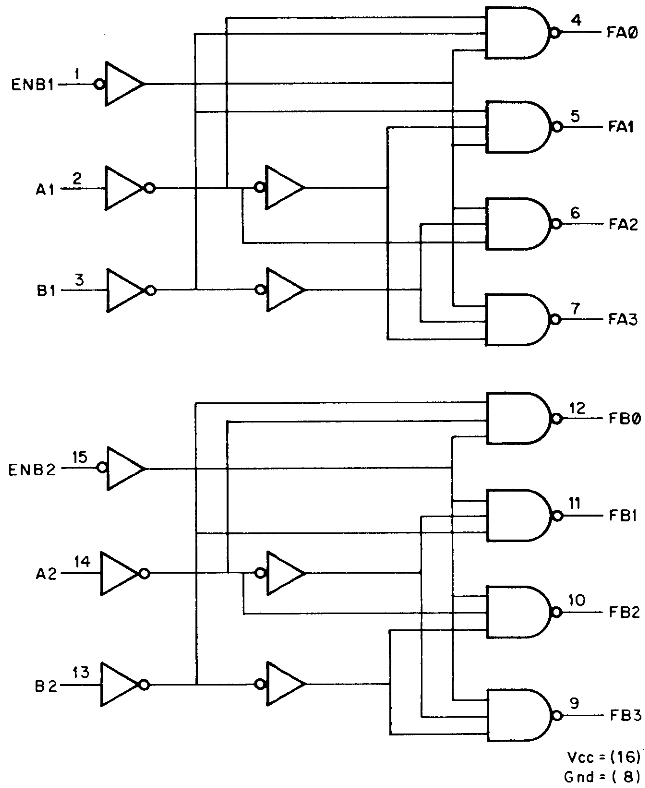


PIN LOCATOR

IC-8613

74LS139 DUAL, 2-LINE TO 4-LINE DECODER/DEMULTIPLEXER

The 74LS139 is a dual, 2-line to 4-line decoder/demultiplexer.



ENABLE	SELECT		OUTPUT			
ENB1	B1	A1	FA0	FA1	FA2	FA3
HI	X	X	HI	HI	HI	HI
LO	LO	LO	LO	HI	HI	HI
LO	LO	HI	HI	LO	HI	HI
LO	HI	LO	HI	HI	LO	HI
LO	HI	HI	HI	HI	HI	LO

FUNCTION TABLE
(SAME FOR OTHER HALF)

IC-74LS139

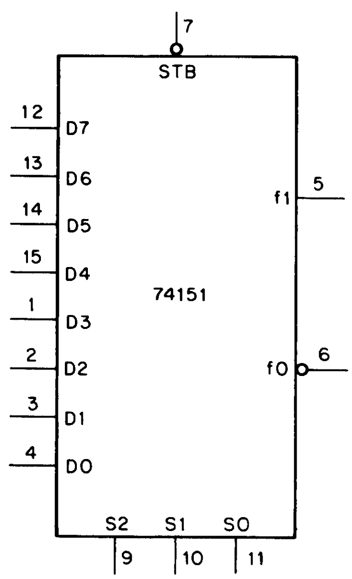
74LS151 8-INPUT, DATA SELECTOR/MULTIPLEXER

The 74LS151 is designed to be used in high-speed data routing applications. The element selects one of 8 data inputs as directed by the binary address inputs and provides both true and complementary data when the strobe input goes low.

74151 TRUTH TABLE

Inputs												Outputs	
S2	S1	S0	STB	D0	D1	D2	D3	D4	D5	D6	D7	f1	f0
X	X	X	1	X	X	X	X	X	X	X	X	0	1
0	0	0	0	0	X	X	X	X	X	X	X	0	1
0	0	0	0	1	X	X	X	X	X	X	X	1	0
0	0	1	0	X	0	X	X	X	X	X	X	0	1
0	0	1	0	X	1	X	X	X	X	X	X	1	0
0	1	0	0	X	X	0	X	X	X	X	X	0	1
0	1	0	0	X	X	1	X	X	X	X	X	1	0
0	1	1	0	X	X	X	0	X	X	X	X	0	1
0	1	1	0	X	X	X	1	X	X	X	X	1	0
1	0	0	0	X	X	X	X	0	X	X	X	0	1
1	0	0	0	X	X	X	X	1	X	X	X	1	0
1	0	1	0	X	X	X	X	X	0	X	X	0	1
1	0	1	0	X	X	X	X	X	1	X	X	1	0
1	1	0	0	X	X	X	X	X	X	0	X	0	1
1	1	0	0	X	X	X	X	X	X	1	X	1	0
1	1	1	0	X	X	X	X	X	X	X	0	0	1
1	1	1	0	X	X	X	X	X	X	X	1	1	0

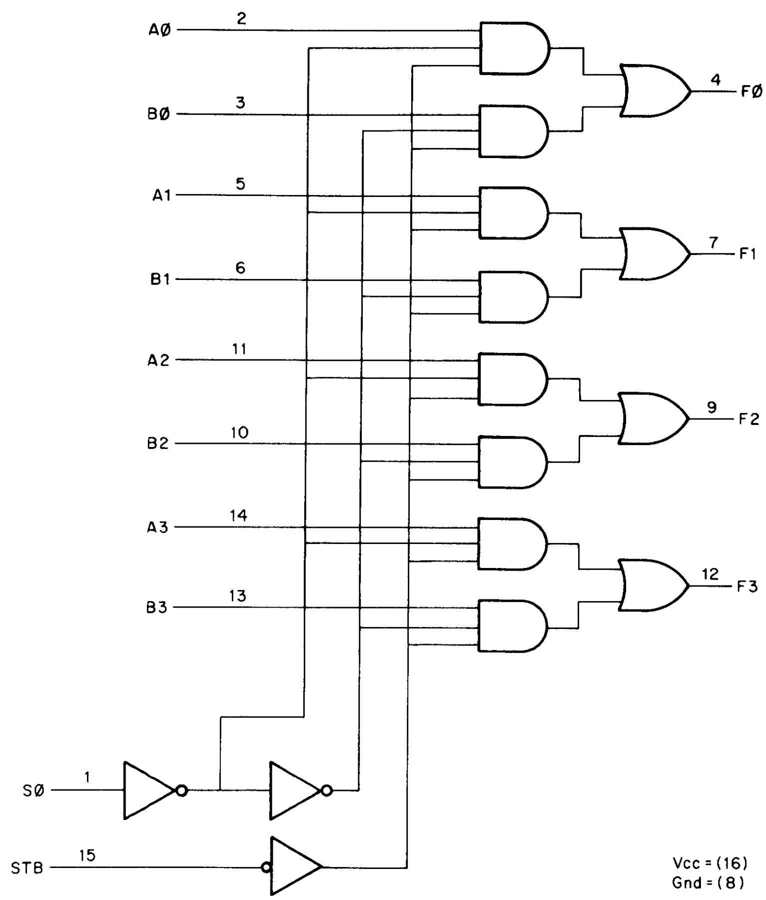
When used to indicate an input, X = irrelevant.



IC-74151

74LS157/74LS158 QUAD 2-INPUT DATA SELECTORS/MULTIPLEXERS

The 74LS157 and 74LS158 are quadruple 2-input data selectors/multiplexers. The 74LS158 features inverting data paths.



LOGIC DIAGRAM (SHOWN FOR 74LS157)

STB	S0	An	Bn	Fn 74LS157	Fn 74LS158
HI	X	X	X	LO	HI
LO	LO	LO	X	LO	HI
LO	LO	HI	X	HI	LO
LO	HI	X	LO	LO	HI
LO	HI	X	HI	HI	LO

TRUTH TABLE

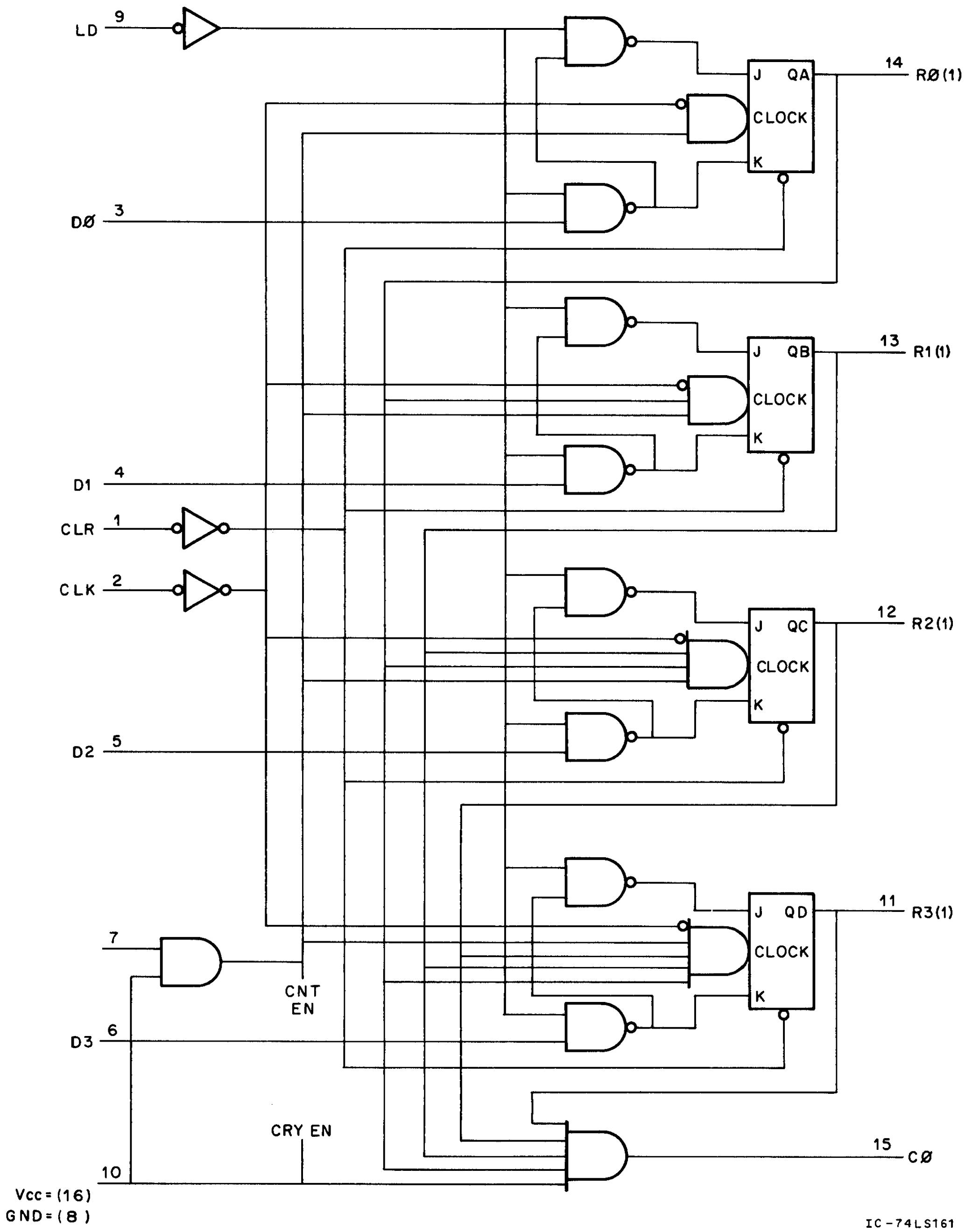
IC - 74LS157

74LS161 4-BIT BINARY COUNTER

The 74LS161 is a synchronous, presettable, 4-bit binary counter. It has an internal carry look-ahead that enables totally synchronous high-speed counting. All counting flip-flops are triggered simultaneously from a common clock buffer, counting on the positive-going edge of the clock input.

All counters are synchronously presettable to either state. When the LD line is low, the next rising edge of the clock transfers into the counting register data present on the Dn lines.

The clear function is asynchronous and a low on the CLR line sets all outputs low regardless of the state of the clock or of any other input.



74181 4-BIT ARITHMETIC LOGIC UNIT, ACTIVE HIGH DATA

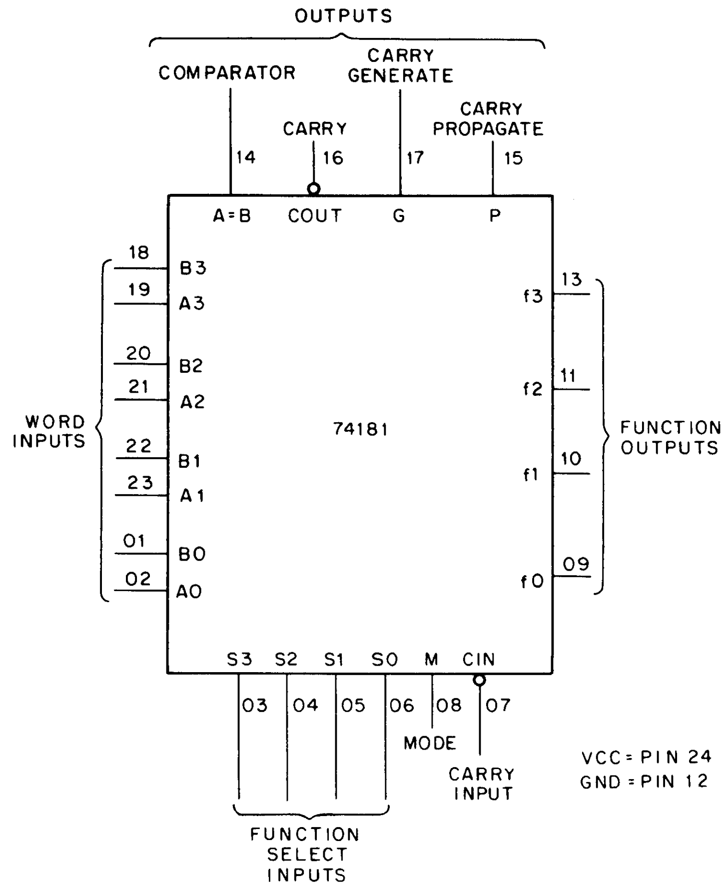
The 74181 performs up to 16 arithmetic and 16 logic functions. Arithmetic operations are selected by four function-select lines (S0, S1, S2, and S3) with a low-level voltage at the mode control input (M), and a low-level carry input. Logical operations are selected by the same four function-select lines except that the mode control input (M) must be high to disable the carry input.

Subtraction is accomplished by 1's complement addition where the 1's complement of the subtrahend is generated internally. The resultant output is A-B-1, which requires an end-around or forced carry to provide A-B.

74181
TABLE OF LOGIC FUNCTIONS

Function Select				Output Function	
S3	S2	S1	S0	Negative Logic	Positive Logic
L	L	L	L	$f = \overline{A}$	$f = \overline{A}$
L	L	L	H	$f = \overline{AB}$	$f = \overline{A + B}$
L	L	H	L	$f = \overline{A} + B$	$f = \overline{A}B$
L	L	H	H	f = Logical 1	f = Logical 0
L	H	L	L	$f = \overline{A + B}$	$f = \overline{AB}$
L	H	L	H	$f = \overline{B}$	$f = \overline{B}$
L	H	H	L	$f = \overline{A \oplus B}$	$f = A \oplus B$
L	H	H	H	$f = A + \overline{B}$	$f = A\overline{B}$
H	L	L	L	$f = \overline{A}B$	$f = \overline{A} + B$
H	L	L	H	$f = A \oplus B$	$f = \overline{A \oplus B}$
H	L	H	L	f = B	f = B
H	L	H	H	f = A + B	f = AB
H	H	L	L	f = Logical 0	f = Logical 1
H	H	L	H	$f = A\overline{B}$	$f = A + \overline{B}$
H	H	H	L	f = AB	f = A + B
H	H	H	H	f = A	f = A

With mode control (M) high: C_{in} irrelevant
For positive logic: logical 1 = high voltage
logical 0 = low voltage
For negative logic: logical 1 = low voltage
logical 0 = high voltage



IC-74181

74181
TABLE OF ARITHMETIC OPERATIONS

Function Select				Output Function	
S3	S2	S1	S0	Low Levels Active	High Levels Active
L	L	L	L	f = A minus 1	f = A
L	L	L	H	f = AB minus 1	f = A + B
L	L	H	L	$f = A\overline{B}$ minus 1	$f = A + \overline{B}$
L	L	H	H	f = minus 1 (2's complement)	f = minus 1 (2's complement)
L	H	L	L	f = A plus $[A + \overline{B}]$	f = A plus $A\overline{B}$
L	H	L	H	f = AB plus $[A + \overline{B}]$	f = [A + B] plus $A\overline{B}$
L	H	H	L	f = A minus B minus 1	f = A minus B minus 1
L	H	H	H	$f = A + \overline{B}$	$f = A\overline{B}$ minus 1
H	L	L	L	f = A plus [A + B]	f = A plus AB
H	L	L	H	f = A plus B	f = A plus B
H	L	H	L	$f = A\overline{B}$ plus [A + B]	f = $[A + \overline{B}]$ plus AB
H	L	H	H	f = A + B	f = AB minus 1
H	H	L	L	f = A plus A†	f = A plus A†
H	H	L	H	f = AB plus A	f = [A + B] plus A
H	H	H	L	$f = A\overline{B}$ plus A	f = $[A + \overline{B}]$ plus A
H	H	H	H	f = A	f = A minus 1

With mode control (M) and C_{in} low
† Each bit is shifted to the next more significant position.

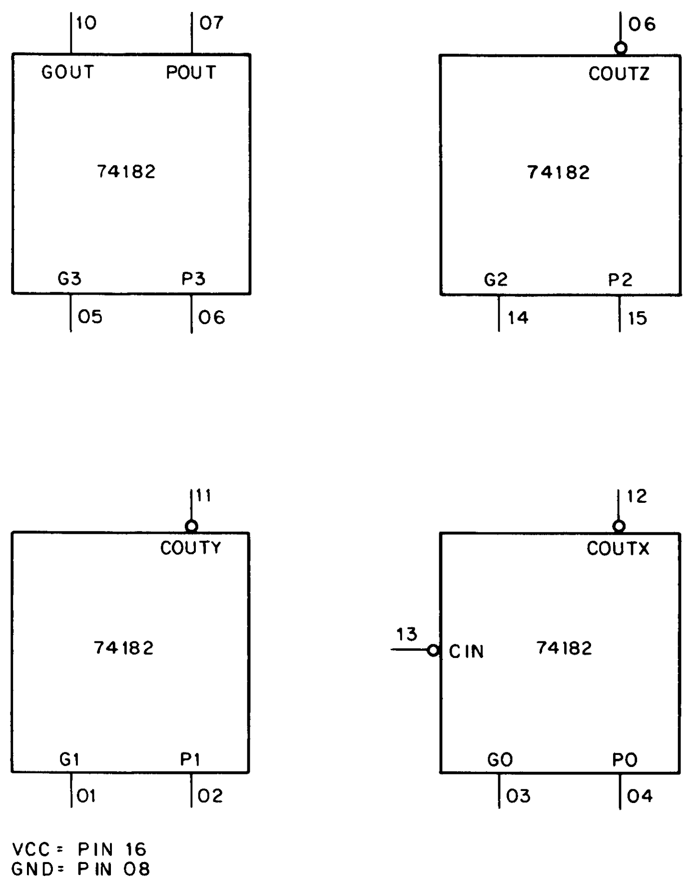
74182 LOOK-AHEAD CARRY GENERATOR

The 74182 Look-Ahead Carry Generator, when used with the 74181 ALU, provides carry look-ahead capability for up to n-bit words. Each 74182 generates the look-ahead (anticipated carry) across a group of four ALUs and, in addition, other carry look-ahead circuits may be employed to anticipate carry across sections of four look-ahead packages up to n-bits.

Carry inputs and outputs of the 74181 ALU are in their true form, and the carry propagate (POUT) and carry generate (GOUT) are in negated form.

PIN DESIGNATIONS

Designation	Pin No.	Function
G0, G1, G2, G3	3, 1, 14, 5	ACTIVE-LOW CARRY GENERATE INPUTS
P0, P1, P2, P3	4, 2, 15, 6	ACTIVE-LOW CARRY PROPAGATE INPUTS
CIN	13	CARRY INPUT
COUTX, COUTY, COUTZ	12, 11, 9	CARRY OUTPUTS
GOUT	10	ACTIVE-LOW CARRY GENERATE OUTPUT
POUT	7	ACTIVE-LOW CARRY PROPAGATE OUTPUT
V_{CC}	16	SUPPLY VOLTAGE
GND	8	GROUND

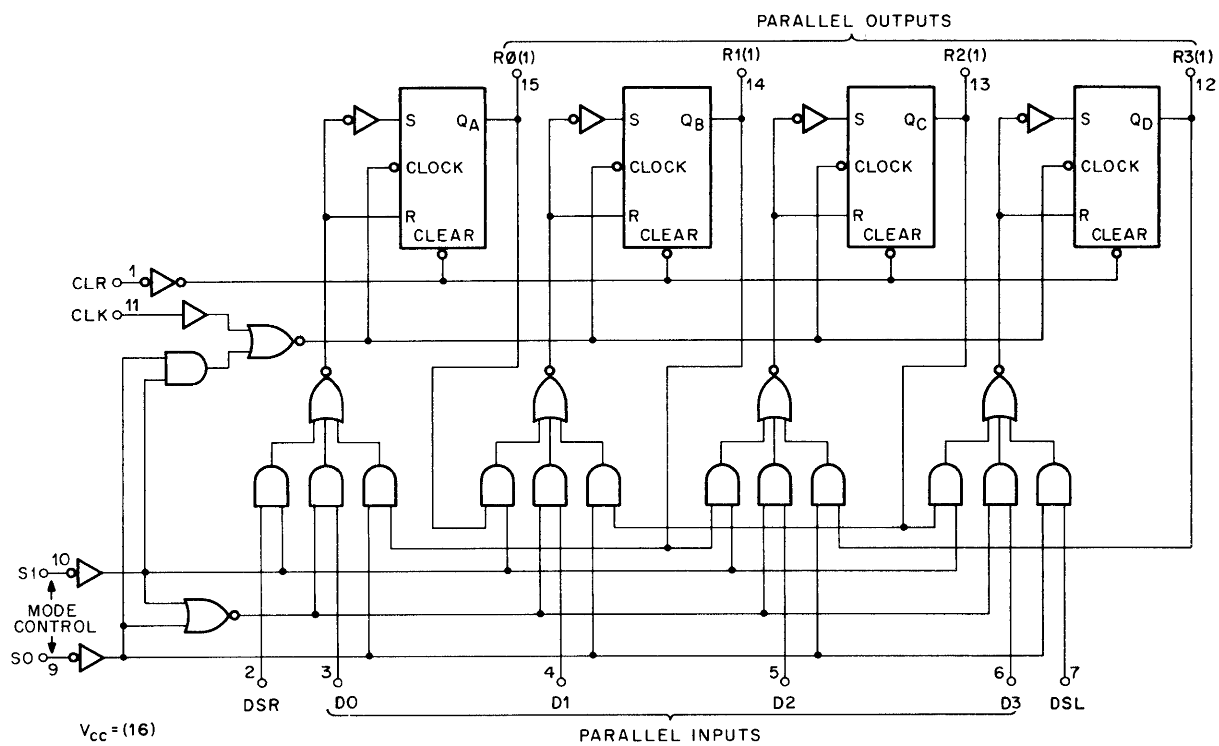


IC-74182

74LS194 4-BIT BIDIRECTIONAL SHIFT REGISTER

The 74LS194 is a 4-bit bidirectional shift register.

In the parallel-load mode, data is loaded into the associated flip-flop and appears at the outputs after the positive transition of the clock input. During loading, serial data flow is inhibited. Shift right is accomplished synchronously with the rising edge of the clock pulse when S0 is high and S1 is low. Serial data for this mode is entered at the shift-right data input (DSR). When S0 is low and S1 is high, data shifts left synchronously and new data is entered at the shift-left serial input (DSL). Clocking of the flip-flops is inhibited when both mode-control inputs are low.



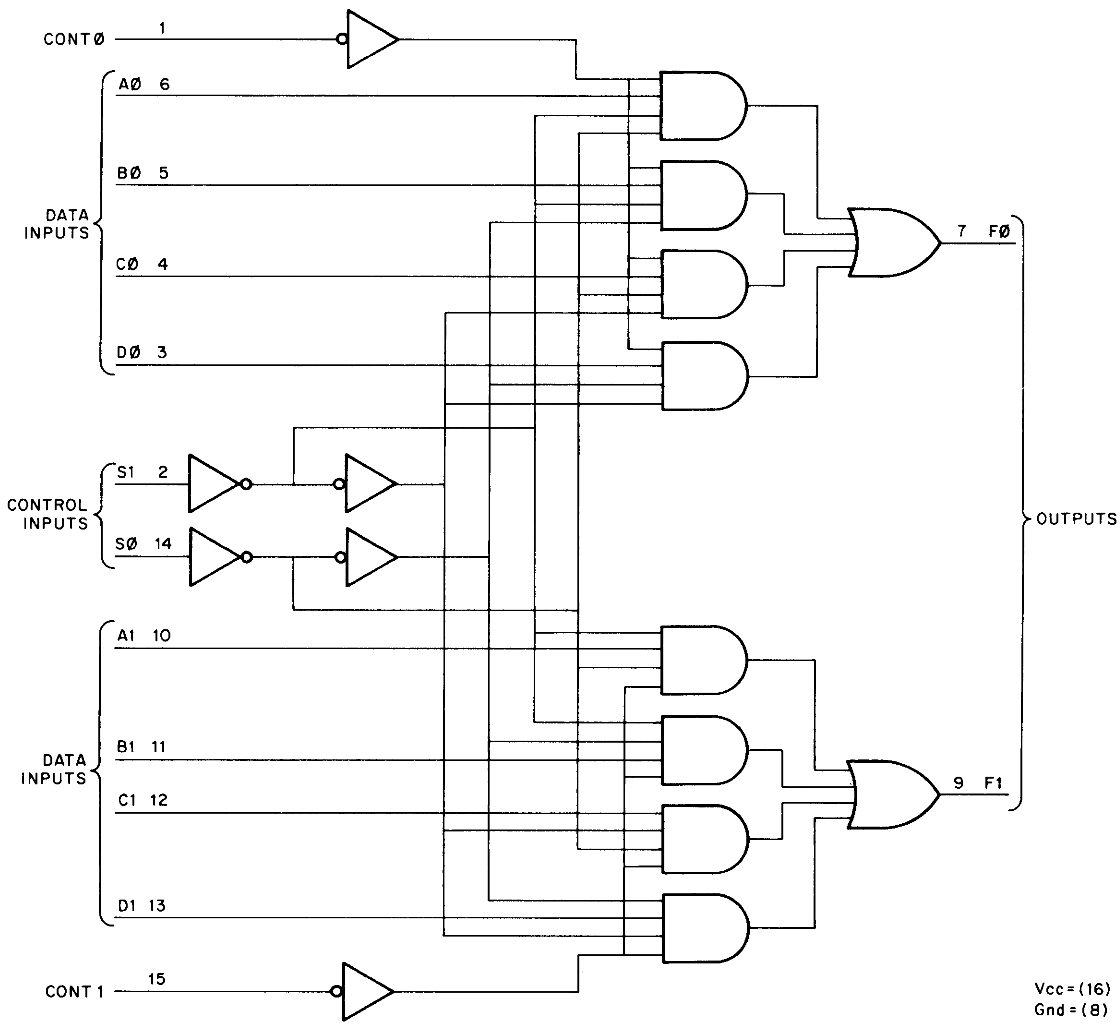
LOGIC DIAGRAM

	MODE CONTROL	
	S1	SO
PARALLEL LOAD	H	H
SHIFT RIGHT (IN THE DIRECTION Q_A TOWARD Q_D)	L	H
SHIFT LEFT (IN THE DIRECTION Q_D TOWARD Q_A)	H	L
INHIBIT CLOCK (DO NOTHING)	L	L

IC-74LS194

74LS253 DUAL, 4-LINE TO 1-LINE DATA SELECTOR

The 74LS253 is a dual, 4-line to 1-line data selector/multiplexer.



LOGIC DIAGRAM

S1	S0	An	Bn	Cn	Dn	CONTn	Fn
X	X	X	X	X	X	HI	Z
LO	LO	LO	X	X	X	LO	LO
LO	LO	HI	X	X	X	LO	HI
LO	HI	X	LO	X	X	LO	LO
LO	HI	X	HI	X	X	LO	HI
HI	LO	X	X	LO	X	LO	LO
HI	LO	X	X	HI	X	LO	HI
HI	HI	X	X	X	LO	LO	LO
HI	HI	X	X	X	HI	LO	HI

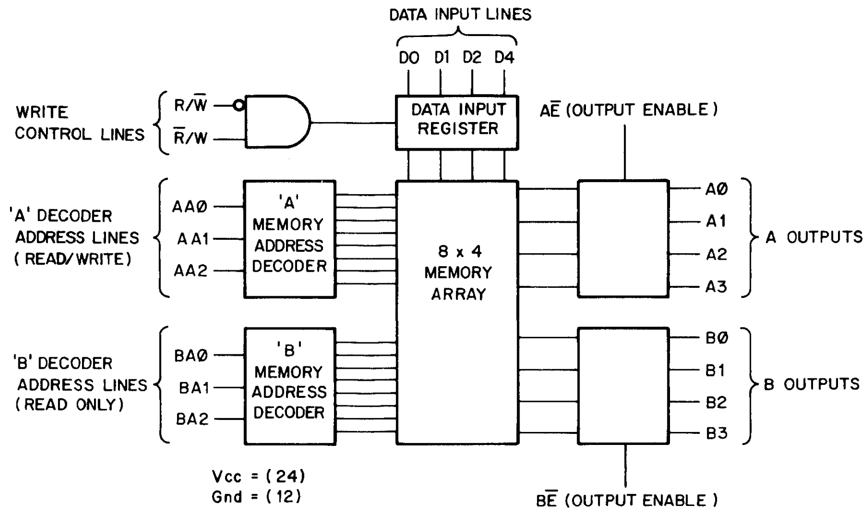
X = DON'T CARE
Z = HI IMPEDANCE

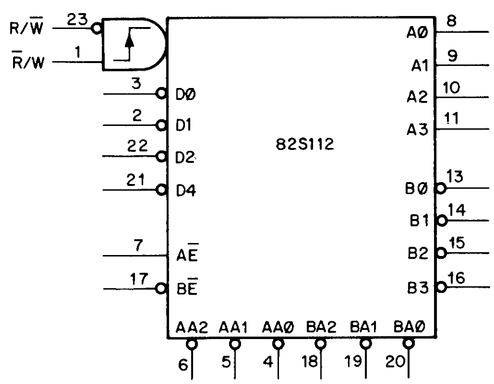
TRUTH TABLE

IC -74LS253

82S112 32-BIT, MULTIPORT MEMORY

The 82S112 IC is a TTL, 32-bit, multiport memory organized in 8 words of 4 bits each. Stored data is addressed through 2 independent sets of 3-input decoders, and read out when the corresponding output enable line is low. Two separate word locations can, therefore, be read at the same time by enabling both the A and B output drivers. In addition, data can be read and written at the same time by utilizing the "A" address to specify the location of the word to be written, and the "B" address to specify the word to be read.





LOGIC REPRESENTATION

DATA IS WRITTEN INTO WORD ADDRESSED BY 'A' DECODER WHEN R/W̄ IS LO AND R̄/W IS HI. HIGH LEVEL IN APPEARS AS HIGH LEVEL OUT.

FUNCTION TABLE

					OUTPUTS	
R̄/W	R/W̄	AĒ	BĒ	MODE	A	B
LO	X	HI	HI	OUTPUTS DISABLED	HI	HI
LO	X	HI	LO	READ	HI	DATA
LO	X	LO	HI	READ	DATA	HI
LO	X	LO	LO	READ	DATA	DATA
HI	HI	HI	HI	READ	HI	HI
HI	HI	HI	LO	READ	HI	DATA
HI	HI	LO	HI	READ	DATA	HI
HI	HI	LO	LO	READ	DATA	DATA
HI	LO	HI	HI	WRITE	HI	HI
HI	LO	HI	LO	WRITE	HI	DATA B ADDRESS
HI	LO	LO	HI	WRITE	DATA BEING WRITTEN	HI
HI	LO	LO	LO	WRITE	DATA BEING WRITTEN	DATA B ADDRESS

IC-82S112

FPLA

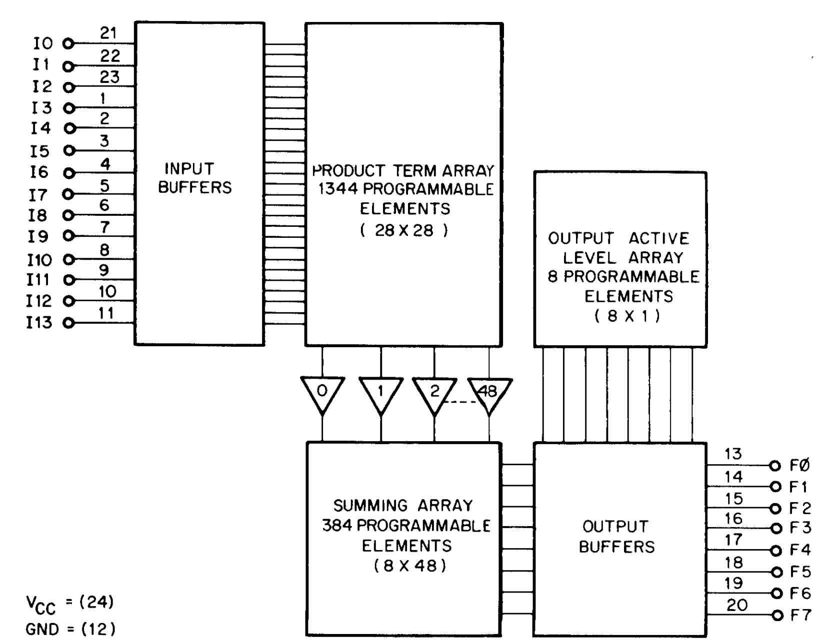
The FPLA (Field Programmable Logic Array) is a logic element designed to produce a sum of product terms at its outputs. The device has 14 inputs and 8 outputs. It can have as many as 48 product terms, each term having as many as 14 variables; each output provides a sum of selected product terms. The FPLA is functionally equivalent to a collection of AND gates which may be ORed at any of its outputs. Since some functions are more easily represented in their inverted form, the output level is also programmable to either a high or low active level.

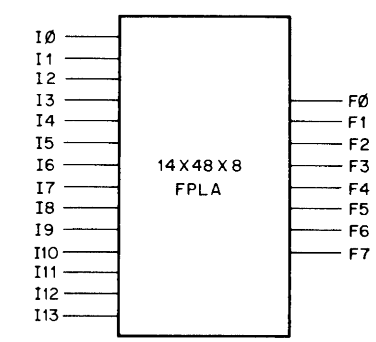
Figure ICFPLA shows a logic diagram, the logic representation, and a truth table of the FPP8-A FPLA.

TRUTH TABLE

	INPUTS														OUTPUTS							
	I13	I12	I11	I10	I09	I08	I07	I06	I05	I04	I03	I02	I01	I00	F07	F06	F05	F04	F03	F02	F01	F00
00	H	L	H	H	H	–	–	–	–	–	–	–	–	H	–	–	–	–	L	–	L	L
01	H	H	L	H	H	L	H	H	H	–	–	–	–	H	–	–	–	–	L	L	L	–
02	H	H	L	H	H	L	H	H	L	–	–	–	–	H	–	–	–	–	L	L	L	L
03	H	H	H	H	H	L	H	H	H	–	–	–	–	H	–	–	–	L	–	–	–	–
04	H	H	H	H	H	L	H	H	L	–	–	–	–	H	–	–	–	L	–	–	L	–
05	H	H	L	H	H	L	H	L	H	–	–	–	–	H	–	–	–	L	–	L	–	–
06	H	H	L	H	H	L	H	L	L	–	–	–	–	H	–	–	–	L	L	–	–	–
07	H	H	H	H	H	H	H	L	H	–	–	–	–	H	–	L	–	L	–	–	–	–
08	H	H	H	H	H	H	H	L	L	–	–	–	–	H	–	–	–	L	L	L	–	L
09	H	L	L	H	H	–	–	–	–	–	–	–	–	H	–	–	–	L	L	L	L	–
10	L	H	H	H	H	–	–	–	–	–	–	–	–	H	–	–	–	L	L	L	L	–
11	–	–	–	H	L	–	–	–	–	–	–	–	L	H	–	–	L	–	–	–	–	–
12	–	–	–	H	L	–	–	–	–	–	–	–	H	H	–	–	L	–	–	–	–	L
13	–	–	–	L	H	–	–	–	–	–	–	–	–	H	–	–	L	–	–	L	–	–
14	L	L	H	H	H	–	–	–	–	–	–	–	–	H	–	–	L	–	–	L	–	–
15	–	–	–	L	–	L	H	H	H	–	–	–	–	H	–	–	L	–	–	L	–	L
16	L	L	–	H	H	L	H	H	H	–	–	–	–	H	–	–	L	–	–	L	–	L
17	–	–	–	L	–	H	H	H	H	–	–	–	–	H	–	–	L	–	–	L	L	–
18	L	L	–	H	H	H	H	H	H	–	–	–	–	H	–	–	L	–	–	L	L	–
19	–	–	–	L	L	–	–	–	–	–	–	–	–	H	–	–	L	–	L	L	–	–
20	L	L	L	H	H	–	–	–	–	–	–	–	–	H	–	–	L	–	L	L	–	–
21	L	H	L	H	H	H	–	–	–	–	–	–	–	H	–	L	–	–	–	–	–	L
22	–	H	L	H	H	H	H	–	–	L	–	–	–	H	–	–	–	–	–	L	L	–
23	–	H	L	H	H	H	H	H	L	–	L	–	–	H	–	–	–	–	–	L	L	–
24	–	H	L	H	H	H	H	L	H	–	H	–	–	H	–	–	–	–	–	L	L	–
25	–	H	L	H	H	H	H	L	L	–	–	–	–	H	–	–	–	–	–	L	L	–
26	–	H	L	H	H	H	L	H	H	H	–	–	–	H	–	–	–	–	–	L	L	–
27	–	H	L	H	H	H	L	H	L	–	H	–	–	H	–	–	–	–	–	L	L	–
28	–	H	L	H	H	H	L	L	H	H	L	–	–	H	–	–	–	–	–	L	L	–
29	–	H	L	H	H	H	L	L	L	–	–	L	–	H	–	–	–	–	–	L	L	–
30	–	H	L	H	H	H	H	H	H	H	–	–	–	H	L	–	–	–	L	–	L	–
31	–	H	L	H	H	H	H	H	L	–	H	–	–	H	L	–	–	–	L	–	L	–
32	–	H	L	H	H	H	H	L	H	H	L	–	–	H	L	–	–	–	L	–	L	–
33	–	H	L	H	H	H	L	–	H	L	–	–	–	H	L	–	–	–	L	–	L	–
34	–	H	L	H	H	H	L	H	L	–	L	–	–	H	L	–	–	–	L	–	L	–
35	–	H	L	H	H	H	L	L	H	–	H	–	–	H	L	–	–	–	L	–	L	–
36	–	H	L	H	H	H	L	L	L	–	–	H	–	H	L	–	–	–	L	–	L	–
37	H	H	L	H	H	L	L	–	–	–	–	–	–	H	L	–	–	–	–	–	–	–
38	H	H	H	H	H	L	L	–	–	–	–	–	–	H	L	–	–	–	–	–	–	–
39	H	H	H	H	H	L	H	L	–	–	–	–	–	H	L	–	–	–	–	–	–	–
40	H	H	H	H	H	H	L	L	–	–	–	–	–	H	L	–	–	–	–	–	–	–
41	L	H	L	H	H	L	–	–	–	–	–	–	–	H	L	–	–	–	–	–	–	–
42	H	H	H	H	H	H	L	H	H	–	–	–	–	H	L	–	–	–	–	–	–	–
43	H	H	H	H	H	H	L	H	L	–	–	–	–	H	–	L	–	–	L	–	–	–
44	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
45	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
46	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
47	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

H = HIGH, L = LOW, '–' = DON'T CARE, IF INPUT, OR NOT CONNECTED, IF OUTPUT





FPP8-A Maintenance Manual
EK-FPP8A-MM-001

Reader's Comments

CUT OUT ON DOTTED LINE

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

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

What features are most useful? ______________________

What faults do you find with the manual? ______________________

Does this manual satisfy the need you think it was intended to satisfy? ______________________

Does it satisfy *your* needs? ____________ Why? ______________________

Would you please indicate any factual errors you have found. ______________________

Please describe your position. ______________________

Name ______________________ Organization ______________________

Street ______________________ Department ______________________

City ____________ State ____________ Zip or Country ____________

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

Digital Equipment Corporation
Technical Documentation Department
146 Main Street
Maynard, Massachusetts 01754

printed in U.S.A.