



GE MEDICAL SYSTEMS INSTITUTE

AP 120/B
MAINTENANCE MANUALS

GE MEDICAL SYSTEMS INSTITUTE

AP120B MAINTENANCE MANUAL

TABLE OF CONTENTS

Section 1	Book 1 for Audio Tapes
Section 2	Book 2 for Audio Tapes
Section 3	AP Maintenance Manual
Section 4	AP Processor Manual
Section 5	I.R.P. Manual
Section 6	AP Handouts
Section 7	Backplane Signal Glossary
Section 8	Course Outline
Section 9	Special Handouts
Section 10	Tests

GE MEDICAL SYSTEMS INSTITUTE

AP120B COURSE OBJECTIVES

The student will become familiar with the Array Processor and how it fits into the CT/T System. This will be done through the use of block diagrams.

Also, with the use of block diagrams, the student will be able to maintain and troubleshoot the Array Processor. With most emphasis on the use of diagnostics and troubleshooting with diagnostics.

The primary result will be board level replacements and, if possible, component level replacement.

Lab Record Book

Week 1

Date: 19

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	
Instructions	Handout Manual How does AP fit into Sys. Basic Structure	AP Art AP Art Review to date	AP Art Finish diagnostic	Review test Review Course		
Basic Structure	AP Path How to run diag. "Accept"/"Run"	AP Art AP Art AP Art	1/5 Test AP Com AP Com	See Boards AP Diagnostic Test	Diplomas Dismiss Class	
Lunch		Lunch	Lunch	Lunch	Lunch	
Obs		Obs	Obs	Review System Review System		
• Student run diagnostics	• diagnosis failure	• diagnosis failure	• diagnosis failure	Review System Diplomas	Final Test	
AP Art		• repair boards *	• repair boards *	Review System Diplomas		
EVENTS	Events	Events	Events	Events	Events	* NOTE: if possible on-hand
MUR 1	MUR 3	MUR 2	MUR 2	MUR 2	MUR 2	

BOOK 1

FAST RECONSTRUCT PROCESSOR

AP120B

-PRELIMINARY -

CT/T Fast Reconstruct Processor

MEDICAL SYSTEMS INSTITUTE

MEDICAL SYSTEMS DIVISION

GENERAL ELECTRIC COMPANY

MILWAUKEE, WISCONSIN

July, 1978

Dave Anderson, General Electric
with
Sam Martin, Floating Point
Instructors

GE MEDICAL SYSTEMS INSTITUTE

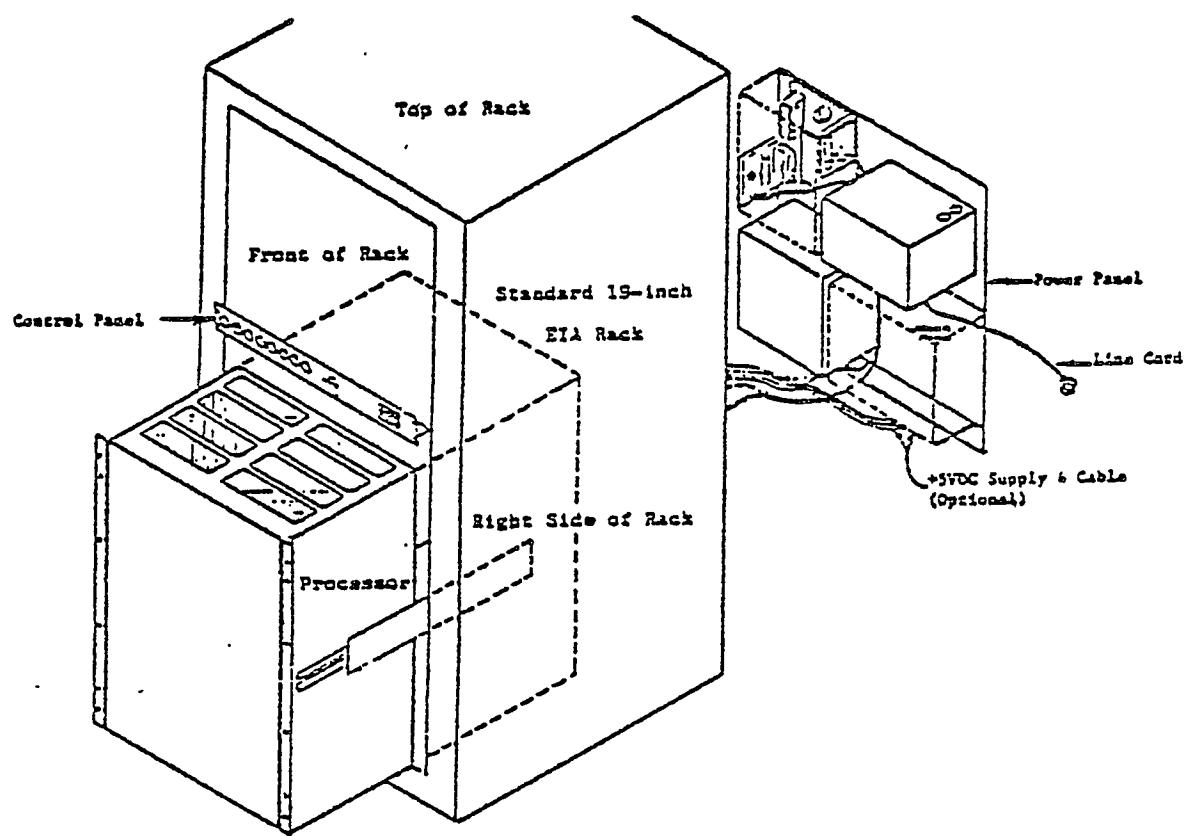


FIGURE INTRO1

The AP-120B Hardware Rack Configuration

GE MEDICAL SYSTEMS INSTITUTE

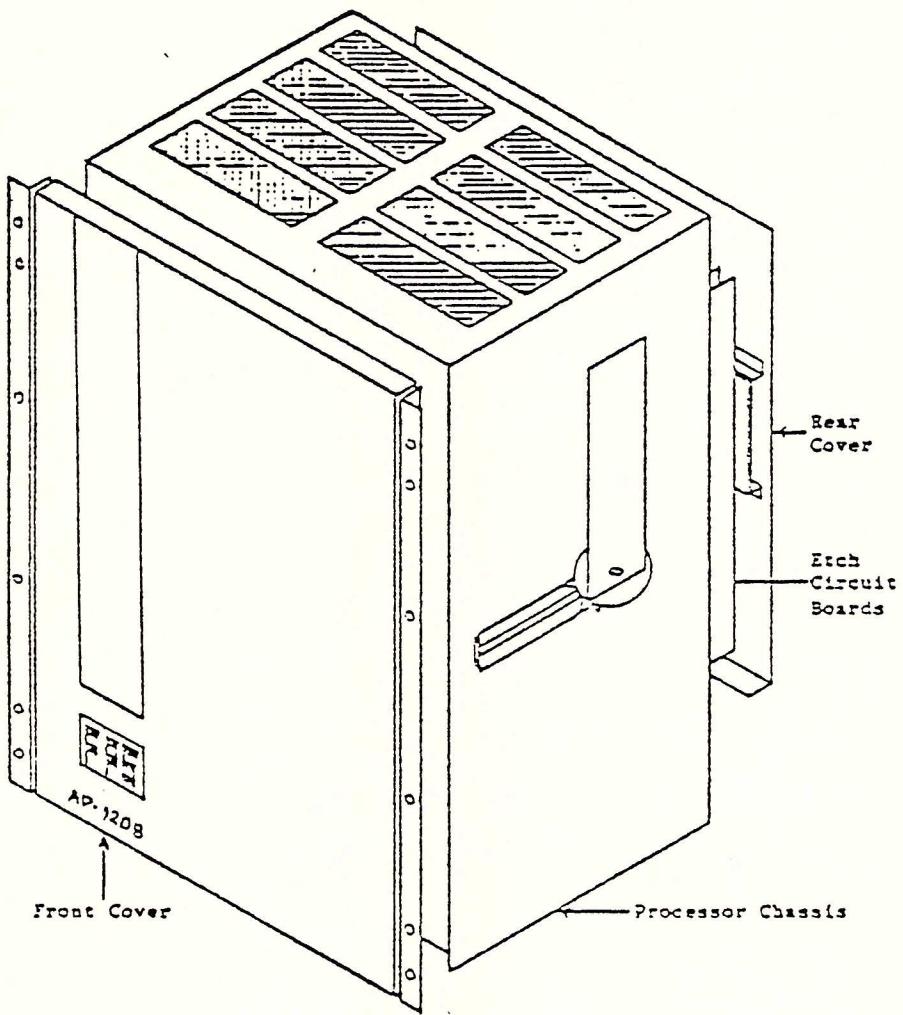


FIGURE INTRO2

The Processor

GE MEDICAL SYSTEMS INSTITUTE

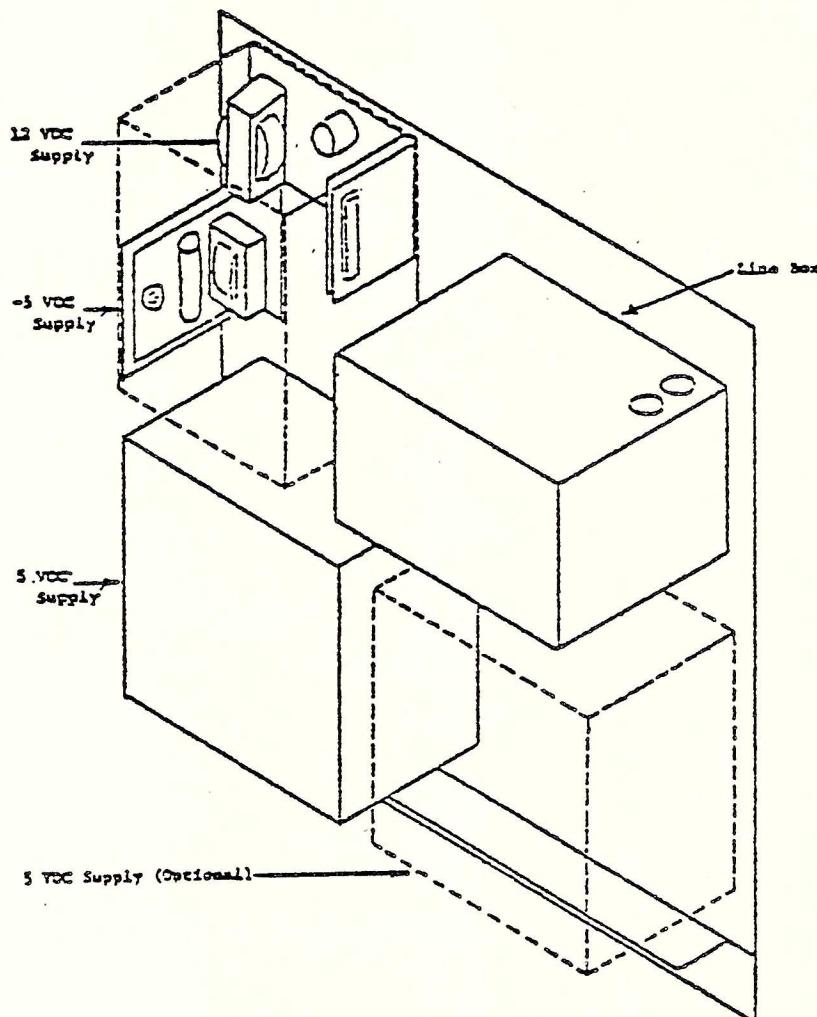
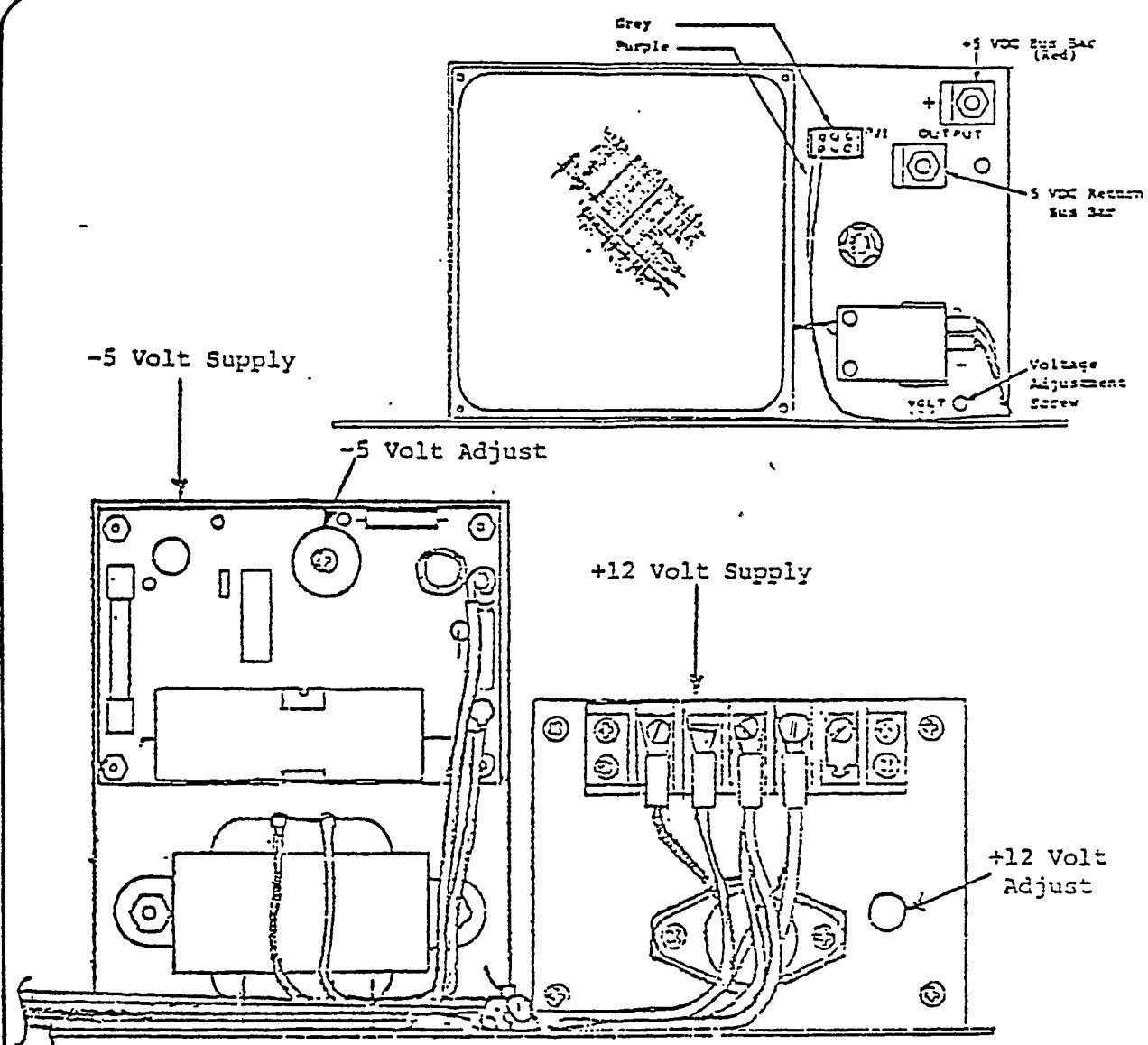


FIGURE INTRO3

The Power Panel

GE MEDICAL SYSTEMS INSTITUTE



SUPPLY	VOLTAGE ADJUSTMENT	TOLERANCE	MEASURING POINT
+5 VDC	+5.15	± 0.05	Backplane
+12 VDC	+12.00	± 0.05	Backplane
-5VDC	-5.00	± 0.05	Backplane

FIGURE INTRO4

AP-120B Power Supplies

GE MEDICAL SYSTEMS INSTITUTE

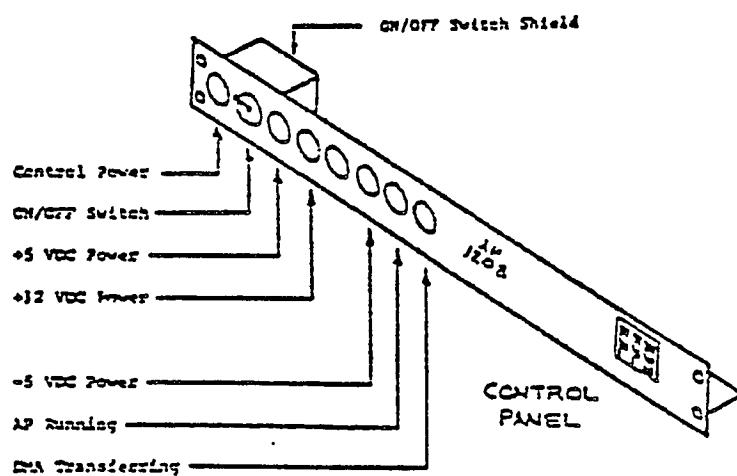


FIGURE INTRO5

The Control Panel

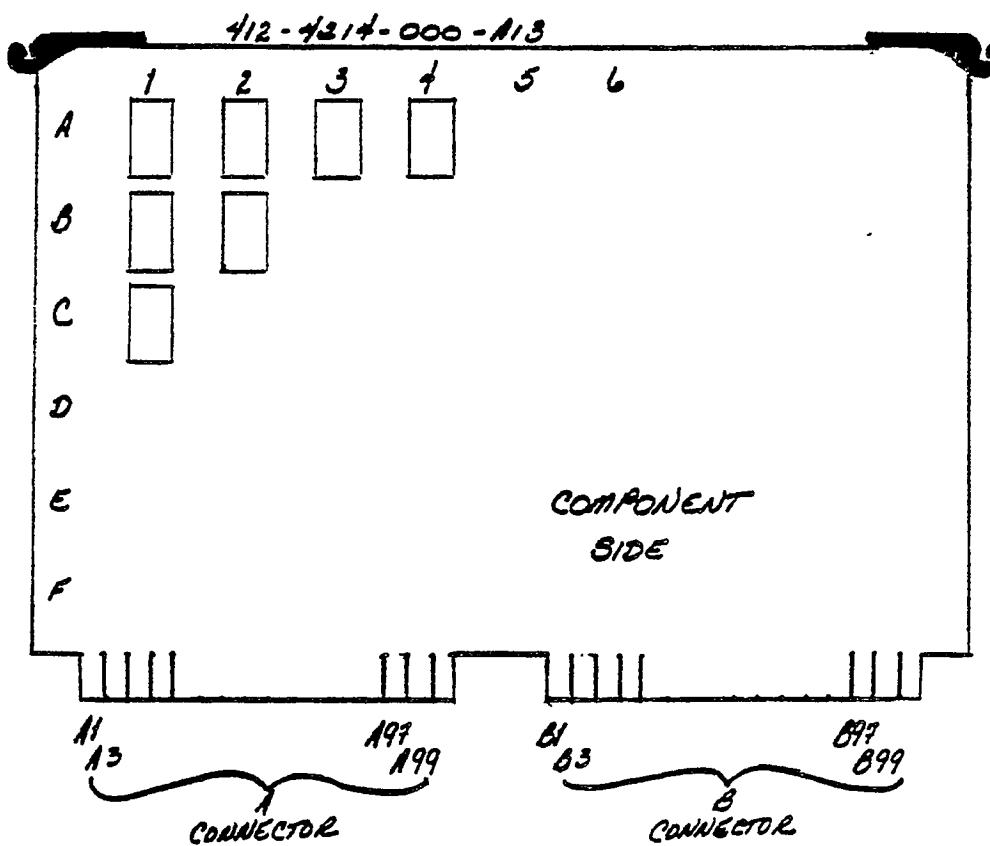


FIGURE INTRO6

AP-120B Etch Circuit Board

GE MEDICAL SYSTEMS INSTITUTE

AP-120B Organization Overview

The AP-120 is a high-speed arithmetic processor.

Cycle: 167 nanoseconds

Word Size: 38-bit word size

Registers: 64 accumulators (Data Pad) 38-bits wide.
16 Address Registers (SPAD) 16-bits wide.

Memory: 64-bit wide Program Source (PS) memory in
256 word increments to 4096 words.

38-bit wide Main Data Memory (MD) in 8192 word
increments to 64K words.

38-bit Table Memory (TMRAW/TMPOM) in 512 word
increments to 64K words.

Arithmetic: 16-bit Integer Address Processor (SPAD).

13 operation Floating Adder (FA) in two stage
pipeline.

3-stage pipelined Floating Multiplier (FM).

Both FA and FM produce normalized, rounded
results with overflow/underflow detection and
correction.

Processor Control: Conditional branches based on 25 conditions (15
locations forward or 16 locations backwards).
Global jumps to anywhere in 4K of PS either
absolute or relative.

Hardware subroutine return stack allowing nested
routines up to 16 levels deep.

Instructions to read and write PS memory thus
allowing the processor to bootstrap and to alter
programs dynamically.

Input/Output: Internal programmed I/O Bus (38-bit) and DMA
to/from Main Data Memory.

Host Interface: Programmed I/O and DMA interface to host CPU
including bidirectional format conversions on
four different formats:

16-bit integer

32-bit integer

32-bit single precision floating-point

32-bit single precision IBM floating-point

GE MEDICAL SYSTEMS INSTITUTE

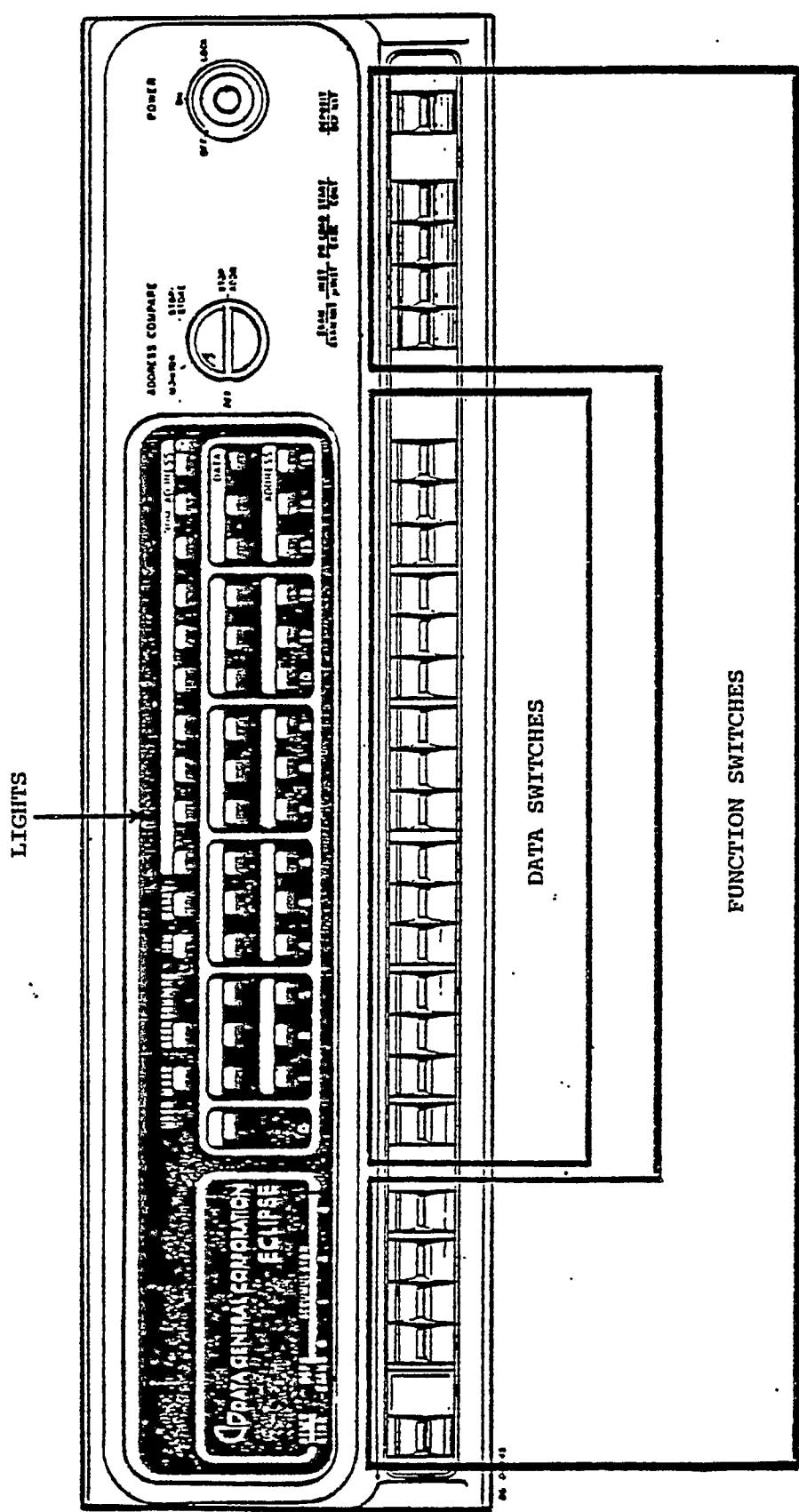


FIGURE VFPI
Data General's Eclipse Operator's Console

GE MEDICAL SYSTEMS INSTITUTE

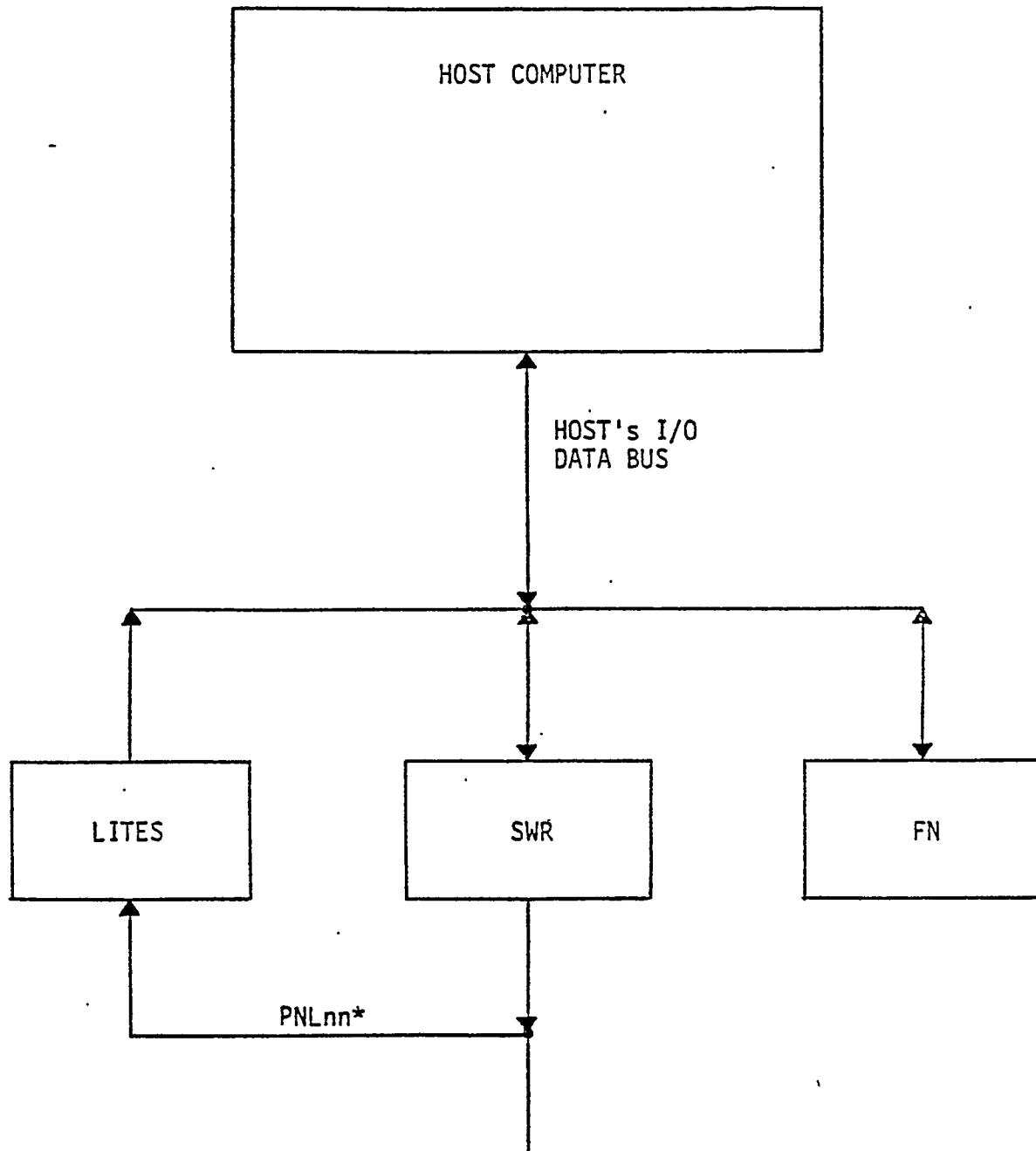


Figure VFP2
AP-120B Virtual Front Panel

GE MEDICAL SYSTEMS INSTITUTE

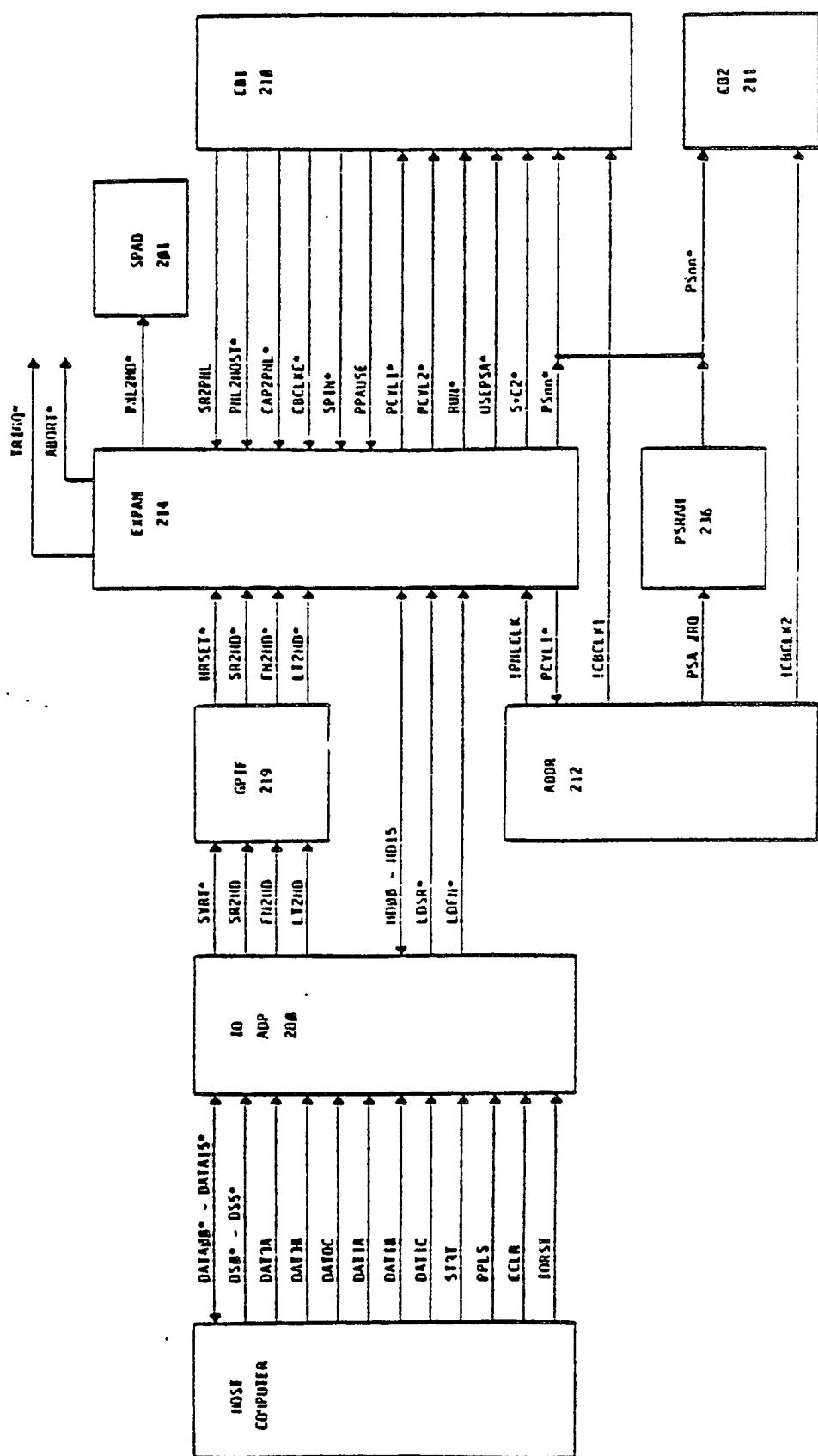


Figure VFP3 .
Virtual Front Panel Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

VFP GLOSSARY (CONTINUED)

PCYL1*	Panel Cycle One
PCYL2*	Panel Cycle Two
PNL2HOST*	Enable Lites Register to be Loaded from Panel Bus
PNL2MD*	Panel to Main Data Register (Causes a Main Data Reference)
PPLS	Pulse I/O Pulse from Host
PPAUSE	Processor Pause
PSnn*	Program Source Memory Output Bits 00 through 63
PSAZRO	Program Source Zero, Disables Output of Program Source Memory
RUN*	Processor is Currently Executing Instructions
S+C2*	Start or Continue Processor Execution
SPIN*	Suspend Processor Execution
SR2PNL	Switch Register to Panel Enable
SR2HD	Switch Register to Host Data Enable
SR2HD*	Switch Register to Host Data Enable (Low True)
STRT	Start I/O Pulse from Host
SYRT*	System Reset
TRIGQ	Array Processor's Hardware Trigger Point
USEPSA*	Use the Contents of PSA as the Next Program Source Address

GE MEDICAL SYSTEMS INSTITUTE

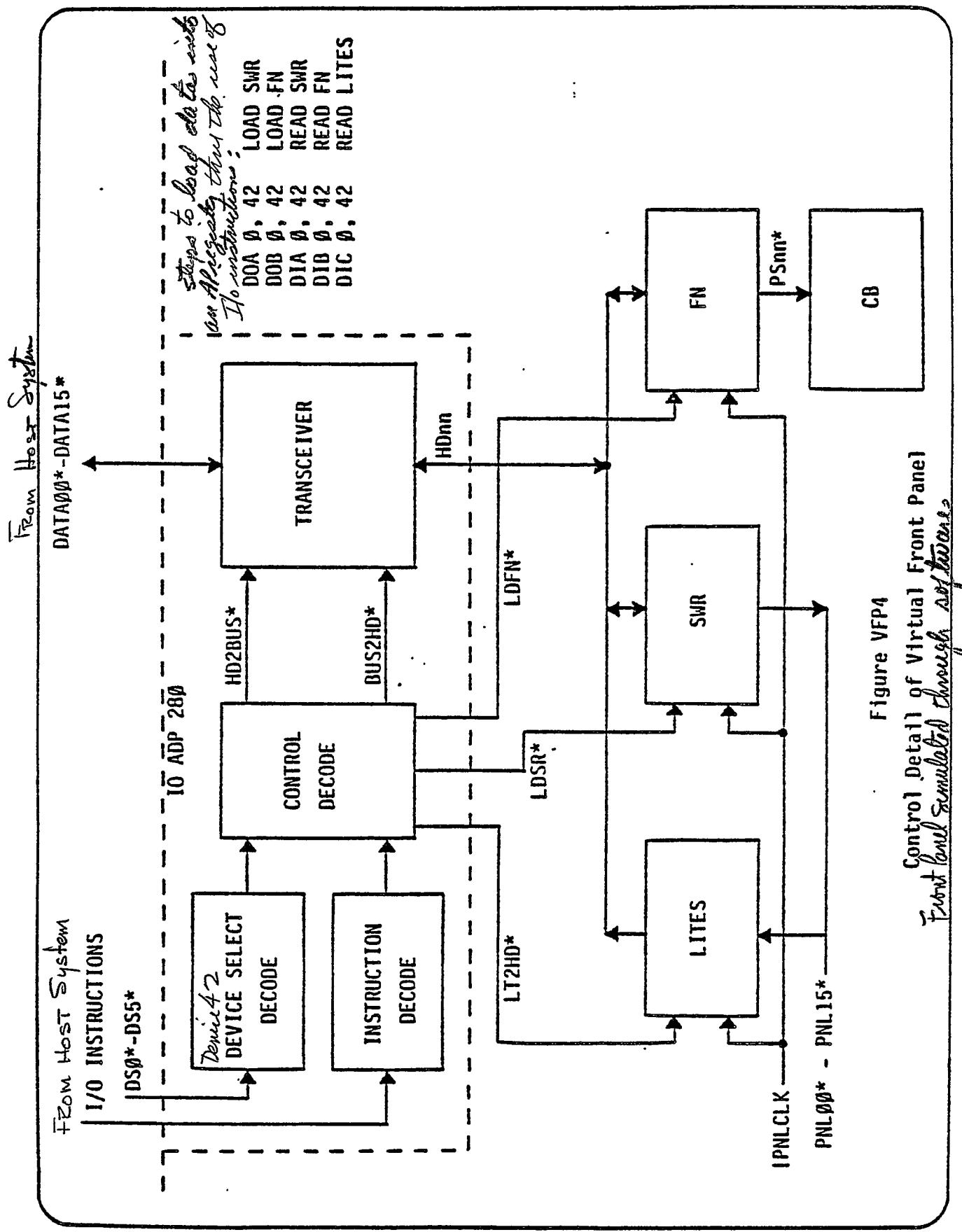


Figure VFP4

Control Detail of Virtual Front Panel
Front panel simulated through software,

GE MEDICAL SYSTEMS INSTITUTE

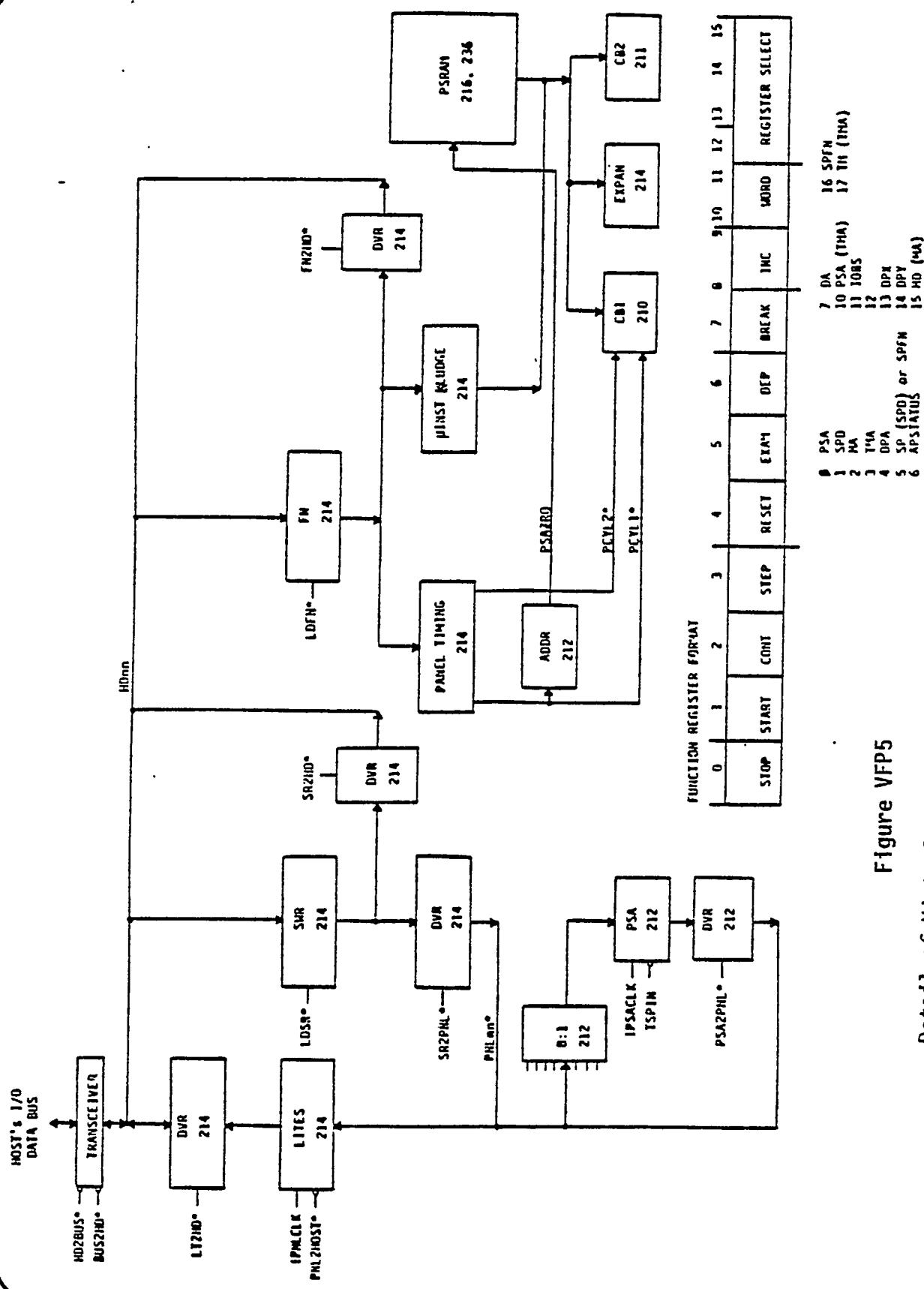


Figure VFP5
Detail of Virtual Front Panel Data Paths

GE MEDICAL SYSTEMS INSTITUTE

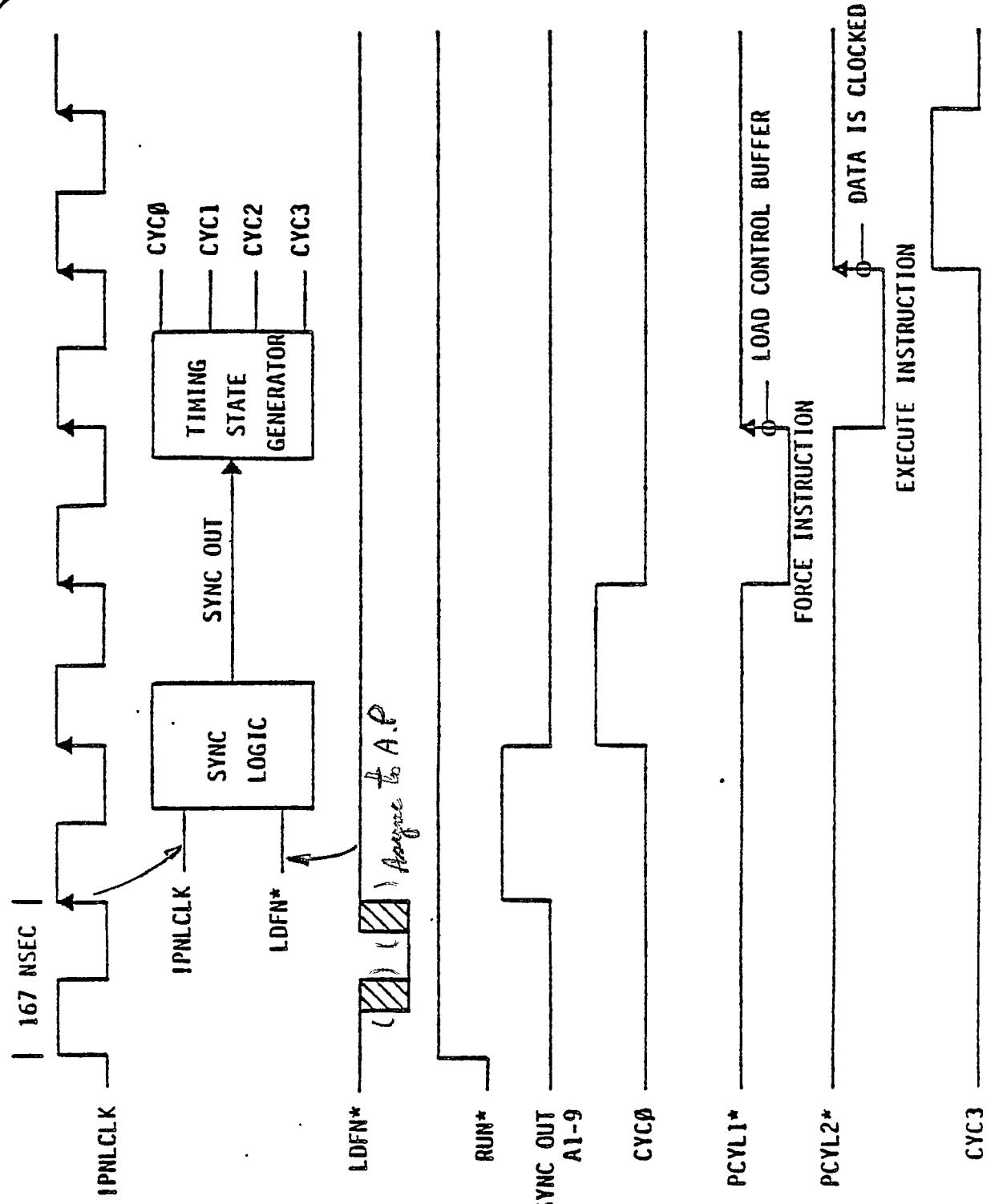


Figure VFP6
Virtual Front Panel Timing

GE MEDICAL SYSTEMS INSTITUTE

"MICRO-INSTRUCTIONS FORCED BY PANEL LOGIC

"INSTR

- 010103	PNLIT; RPSA	Examine PSA (PSA) PNL LITES
110000		
000000		
000000		
010103	PNLIT; RSPD	Examine SPD (SPD) PNL LITES
111000		
000000		
000000		
010103	PNLIT; RMA	Examine MA (MA) PNL LITES
112000		
000000		
000000		
010103	PNLIT; RTMA	Examine TMA (TMA) PNL LITES
113000		
000000		
000000		
010103	PNLIT; RDPA	Examine DPA (DPA) PNL LITES
114000		
000000		
000000		
010103	PNLIT; RSPFN	Examine SPFN SPFN PNL LITES
115000		
000000		
000000		
010103	PNLIT; RAPS	Examine APSTATUS (APSTATUS) PNL LITES
116000		
000000		
000000		
010103	PNLIT; RDA	Examine DA (DA) PNL LITES
117000		
000000		
000000		
011120	RPSOT	Examine PS, WORDS PSØ PNL LITES
000000		
000000		
000000		

GE MEDICAL SYSTEMS INSTITUTE

011220	RPSIT	Examine PS, WORD1 PS1 PNL LITES
000000		
000000		
000000		
011124	RPS2T	Examine PS, WORD2 PS2 PNL LITES
000000		
000000		
000000		
011224	RPS3T	Examine PS, WORD3 PS3 PNL LITES
000000		
000000		
000000		
010107	DBELIT; IN; DB=INBS	Examine IOBS EXP IOBS DPBS PNL LITES
144000		
001000		
000000		
010113	DBHLIT; IN; DB=INBS	Examine IOBS HMAN IOBS DPBS PNL LITES
144000		
001000		
000000		
010117	DBLLIT; IN; DB=INBS	Examine IOBS LMAN IOBS DPBS PNL LITES
144000		
001000		
000000		
010114	DBLLIT; DB=ZERO	CB, WORD3
000000		
000000		
000000		
010104	DBELIT; DB= DPX (-4)	Examine DPX, EXP DPX (-4) DPBS PNL LITES
000000		
003000		
000000		
010110	DBHLIT; DB=DPX (-4)	Examine DPX HMANN DPX (-4) DPBS PNL LITES
000000		
003000		
000000		
010114	DBLLIT; DP=DPX (-4)	Examine DPX LMANN DPX (-4) DPBS PNL LITES
000000		
003000		
000000		

GE MEDICAL SYSTEMS INSTITUTE

010104 000000 004000 000000	DBELIT; DS=DPY (-4)	Examine DPY, EXP DPY (-4) DPBS PNL LITES
010110 000000 004000 000000	DBHLIT; DP=DPY (-4)	Examine DPY DPY (-4) DPBS PNL LITES
010114 000000 004000 000000	DBLLIT; DB=DPY (-4)	Examine DPY, LMAN DPY (-4) DPBS PNL LITES
010104 000000 005000 000000	DBELIT; DB=MD; SPMDAV	Examine MD, EXP MD DPBS PNL LITES
010110 000000 005000 000000	DBHLIT; DB=MD; SPMDAV	Examine MD, HMAN MD DPBS PNL LITES
010114 000000 005000 000000	DBLLIT; DB=MD; SPMDAV	Examine MD, LMAN MD DPBS PNL LITES
010114 000000 006000 000000	DBLLIT; DB=SPFN	Examine SPFN, WORD3 SPFN FPBS PNL LITES
010104 000000 007000 000000	DBELIT; DB=TM	Examine TM, EXP TM DPBS PNL LITES
010110 000000 007000 000000	DBHLIT; DB=TM	Examine TM, HMAN TM DPBS PNL LITES
010114 000000 007000 000000	DBLLIT; DB=TM	Examine TM, LMAN TM DPBS PNL LITES

GE MEDICAL SYSTEMS INSTITUTE

011030	JMPP	Deposit PSA (SWR)	PNL	PSA
000000				
000000				
000000				
010143	SWDB; LDSPD	Deposit SPD (SWR)	PNL	SPD
101000				
000000				
000000				
010143	SWDB; LDMA	Deposit MA (SWR)	PNL	MA
102000				
000000				
000000				
010143	SWDB; LDTMA	Deposit TMA (SWR)	PNL	TMA
103000				
000000				
000000				
010143	SWDB; LDDPA	Deposit DPA (SWR)	PNL	DPA
104000				
000000				
000000				
010143	SWDB; LDSP	Deposit SP (SPD) (SWR)	PNL	SP (SPD)
105000				
000000				
000000				
010143	SWDB; LDAPS	Deposit APSTATUS (SWR)	PNL	APSTAT
106000				
000000				
000000				
010143	SWDB; LDDA	Deposit DA (SWR)	PNL	DA
107000				
000000				
000000				
011160	WPSQT	Deposit PS, WORD0 (SWR)	PNL	PS
000000				
000000				
000000				
011260	WPSIT	Deposit PS WORD1 (SWR)	PNL	PS
000000				
000000				
000000				

GE MEDICAL SYSTEMS INSTITUTE

011164 000000 000000 000000	WPS2T	Deposit PS WORD2 (SWR) PNL PS
011264 000000 000000 000000	WPS3T	Deposit PS, WORD3 (SWR) PNL PS
010144 000000 040000 000000	SWDBE; DPX (-4) < DB	Deposit DPX, EXP (SWR) PNL DPBS DPX (-4)
010150 000000 040000 000000	SWDBH; DPX (-4) < DB	Deposit DPX, HMAN (SWR) PNL DPBS DPX (-4)
010154 000000 040000 000000	SWDBL; DPX (-4) < DB	Deposit DPX, LMAN (SWR) PNL DPBS DPX (-4)
010144 000000 010000 000000	SWDBE; DPY (-4) < DB	Deposit DPY, EXP (SWR) PNL DPBS DPY (-4)
010150 000000 010000 000000	SWDBH; DPY (-4) < DB	Deposit DPY, HMAN (SWR) PNL DPBS DPY (-4)
010154 000000 010000 000000	SWDBL; DPY (-4) < DB	Deposit DPY, LMAN (SWR) PNL DPBS DPY (-4)
010144 000000 000000 000300	SWDBE; MI<DB	Deposit MD, EXP (SWR) PNL DPBS MD
010150 000000 000000 000300	SWDBH; MI<DB	Deposit MD, HMAN (SWR) PNL DPBS MD

GE MEDICAL SYSTEMS INSTITUTE

010154	SWDBL; MI<DB	Deposit MD, LMAN (SWR) PNL DPBS MD
000000		
000000		
000300		
000000	INCMA	INC MA
000000		
000000		
000020		
000000	INCTMA	INC TMA
000000		
000000		
000001		
000000	INCDPA	INC DPA
000000		
000000		
000004		
001403	LDSPNL 0; LDAPS	RESET
106000		
000000		
000000		
011030	JMPP	START
000000		
000000		
000000		

GE MEDICAL SYSTEMS INSTITUTE

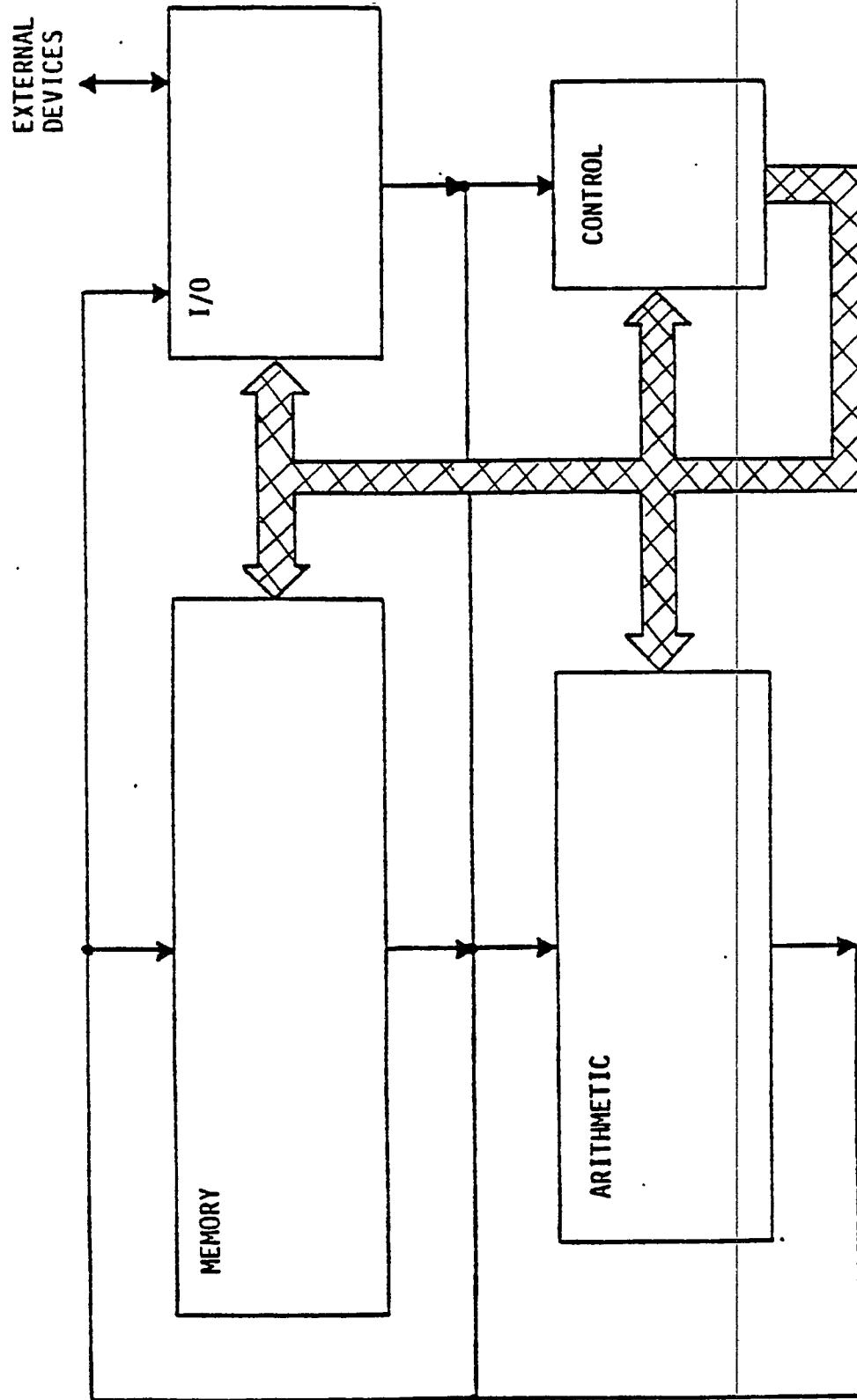


FIGURE SYS 1
Functional Elements of a General Purpose Computer

GE MEDICAL SYSTEMS INSTITUTE

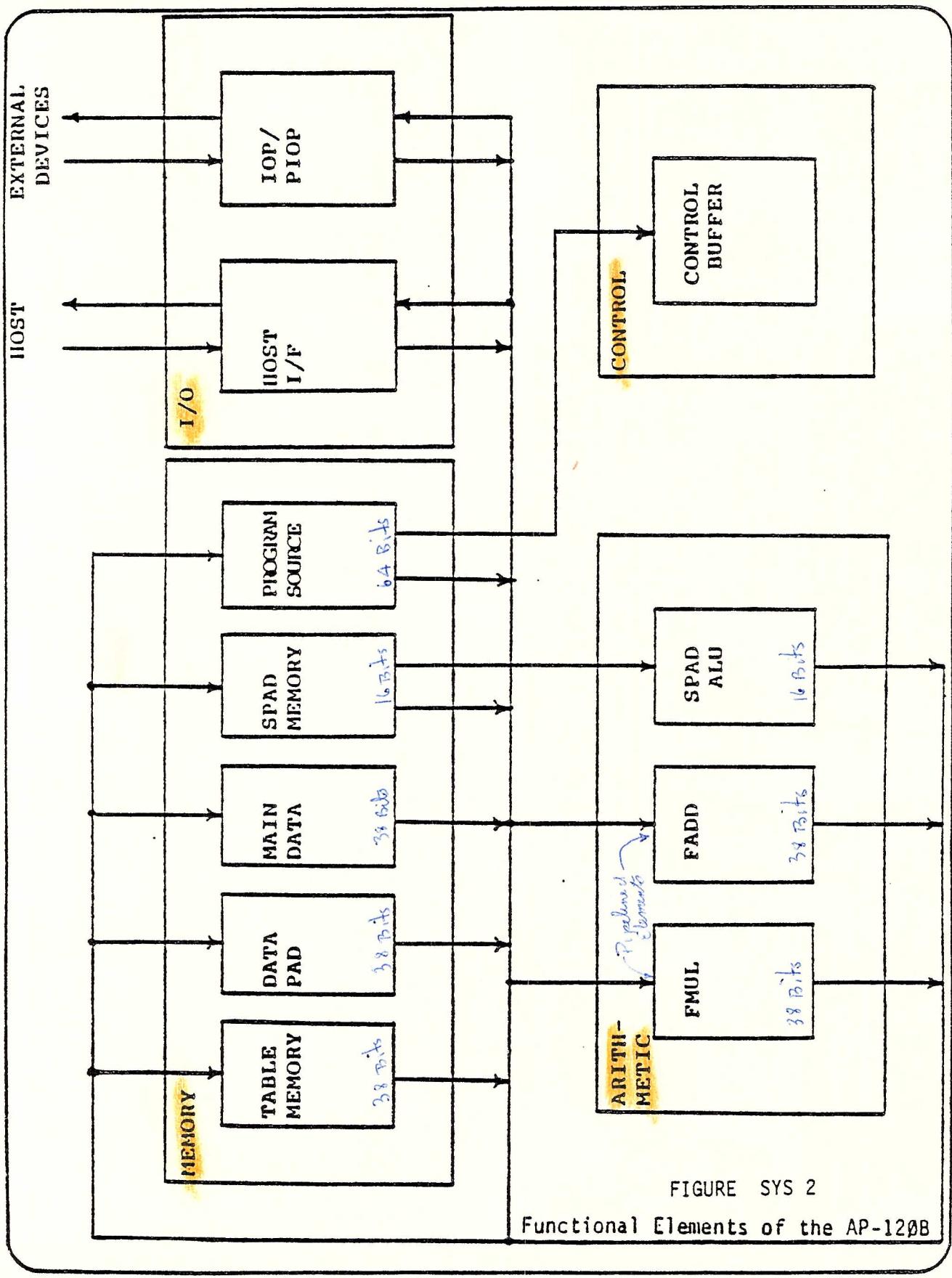


FIGURE SYS 2
Functional Elements of the AP-120B

GE MEDICAL SYSTEMS INSTITUTE

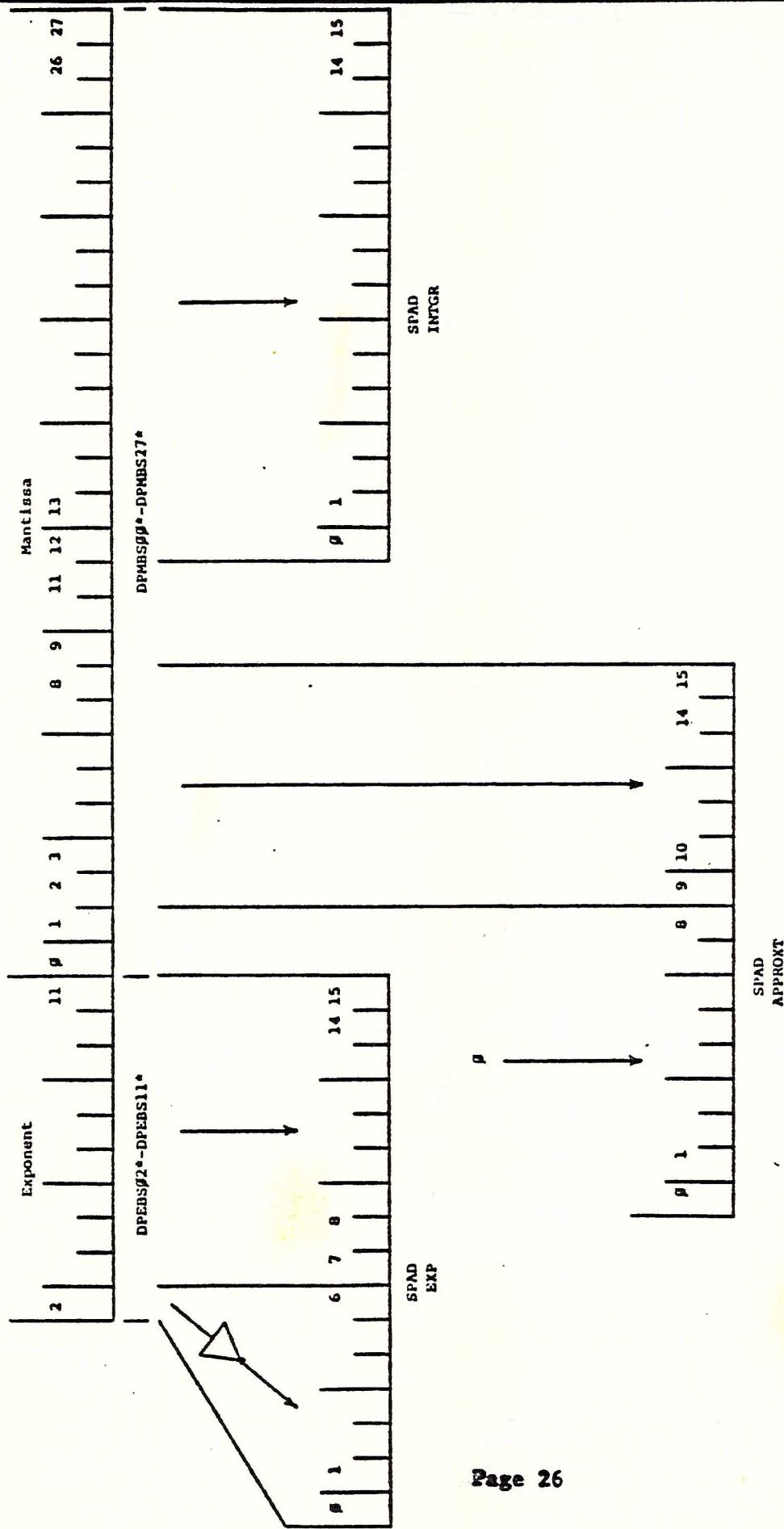


FIGURE SYS 5
Data Pad Bus to SPAD bit layout

GE MEDICAL SYSTEMS INSTITUTE

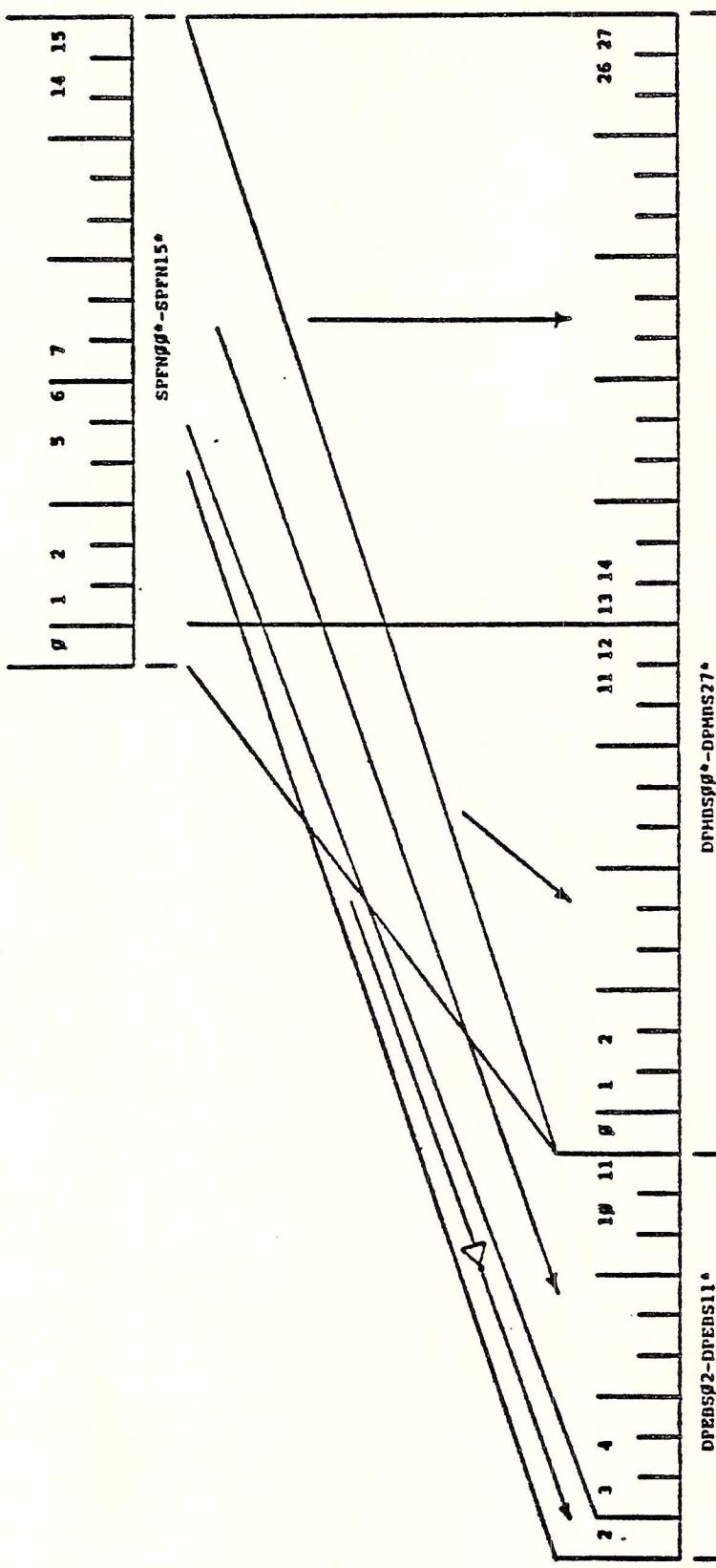


FIGURE SYS 6
SPPN to Data Pad Bus bit format

GE MEDICAL SYSTEMS INSTITUTE

TESTS

Intro 1 The Array Processor is implemented as three hardware subassemblies. These subassemblies are:

- A. The Processor, the Power Supplies, the Control Panel.
- B. The Card Cage, the Power Panel, the Control Element.
- C. The Processor, the Power Panel, the Control Panel.
- D. The Control Element, the Power Unit, the Status Element.

Intro 2 The Array Processor memory elements and their word sizes are:

- A. MD-38, PS-64, TM-38
- B. MD-38, DPX-16, TM-64
- C. DPY-38, TM-38, PS-16
- D. TM-64, PS-38, SPAD-38

Intro 3 Which of the following statements are true?

- A. The Array Processor has three arithmetic elements.
- B. The Array Processor has two integer arithmetic elements.
- C. The Array Processor has three floating point arithmetic elements.
- D. Both A and C are true.

Intro 4 The Branch range of the Array Processor's conditional Branches are:

- A. 4K
- B. -2048 to + 2047
- C. -15 to +16
- D. -16 to +15

GE MEDICAL SYSTEMS INSTITUTE

VFP 1 Make the appropriate choice to fill in the blanks. The sequence of events that must take place to deposit data into an internal Array Processor register using the Virtual Front Panel is:

Deposit the data into the _____ register, and then deposit an encoded value into the _____ register.

- A. Function, Switch
- B. Program Source Address, Function
- C. Switch, Function
- D. Switch, Lites

VFP 2 Fill in the blanks. In order to deposit data into the Program Source Address register from the Virtual Front Panel, the contents of the _____ register were enabled onto the _____, the 8:1 multiplexer was selected to 4 and the data was clocked into the Program Source Address register.

- A. Function, Host Data Bus
- B. Switch, Panel Bus
- C. Lites, Panel Bus
- D. Function, Panel Bus

VFP 3 The correct Function register value to cause an examine of the Program Source Address register is:

- A. 001000
- B. 042000
- C. 011000
- D. 002000

VFP 4 The LDFN* causes the data to be clocked into the Function register and:

- A. The micro-instruction to be kludged onto the Program Source.
- B. The Switch register to be enabled onto the Panel Bus.
- C. The Panel cycle timing to be initiated.
- D. The Program Source Address register to be loaded.

GE MEDICAL SYSTEMS INSTITUTE

VFP 5 During PCYLT* (P cycle 1)

- A. The Program Source Address register is enabled to the Panel Bus.
- B. The Switch register is enabled to the Panel Bus.
- C. The micro-instruction is Kludged onto the Program Source lines.
- D. The Lites register is enabled to the Host Data.

VFP 6 During PCYL2* (P Cycle 2)

- A. The contents of the Program Source Address register is enabled to the Panel Bus.
- B. The Lites register is loaded from the Panel Bus.
- C. The instruction in the Control Buffer is executed.
- D. All of the above.

GE MEDICAL SYSTEMS INSTITUTE

Sys 1 Fill in the blanks. The data paths M1BSnn*, M2BSnn*, A1BSnn* and A2BSnn* are _____ source, and _____ destination paths.

- A. Multiple, Single
- B. Single, Single
- C. Single, Multiple
- D. Multiple, Multiple

Sys 2 Fill in the blanks. The data paths FAnn* and FMnn* are _____ source, and _____ destination paths.

- A. Multiple, Single
- B. Single, Single
- C. Single, Multiple
- D. Multiple, Multiple

Sys 3 Fill in the blanks. DPBSnn* and IOBSnn* are _____ source and _____ destination buses.

- A. Multiple, Single
- B. Single, Single
- C. Single, Multiple
- D. Multiple, Multiple

Sys 4 Fill in the blanks. The output of MD and _____ may be loaded into _____ and _____.

- A. FM, A2, M2
- B. TM, M1, A1
- C. FA, A1, M1
- D. FA, M2, A2

Sys 5 Fill in the blanks. The output of TM and _____ may be loaded into _____ and _____.

- A. FM, A2, M2
- B. FM, A1, M1
- C. FA, A1, M1
- D. FA, A2, M2

GE MEDICAL SYSTEMS INSTITUTE

AP120B WORKBOOK ANSWER SHEET

Intro 1	A	B	C	D
Intro 2	A	B	C	D
Intro 3	A	B	C	D
Intro 4	A	B	C	D
UFP 1	A	B	C	D
UFP 2	A	B	C	D
UFP 3	A	B	C	D
UFP 4	A	B	C	D
UFP 5	A	B	C	D
UFP 6	A	B	C	D
Sys 1	A	B	C	D
Sys 2	A	B	C	D
Sys 3	A	B	C	D
Sys 4	A	B	C	D
Sys 5	A	B	C	D

BOOK 2

FAST RECONSTRUCT PROCESSOR

AP120B

GE MEDICAL SYSTEMS INSTITUTE

2.2 CONTROL UNIT

The control unit, as illustrated by Figure 2-1, consists of:

- program source memory (PS)
- program source address (PSA) register
- control buffer (CB) with decoding logic
- subroutine return stack (SRS)

The operation of the AP is controlled by the execution of 64-bit instruction words which reside in program source (PS) memory. The program word for the next instruction to be performed is selected by the address in the program source address (PSA) register. At the initiation of the next machine cycle, this program word is transferred to the control buffer (CB) where it is decoded and executed. The PSA is incremented by one unless a branch in the current instruction causes the PSA to move to another location in program source memory. Access to program source memory and instruction decoding is overlapped so that the AP can operate at a 6-MHz rate (167ns).

Branching is accomplished in two ways. A short-range branch is provided by adding the 5-bit branch displacement field to the current PSA. This gives a branch range of from -20₈ to +17₈. A long-range jump to any location in PS is accomplished by loading the desired target address into PSA.

Subroutine jumps are made by a JSR instruction which saves the current PSA in the subroutine return stack and sets PSA to the subroutine address. Return is via a return, which loads the PSA with the last entered return address on the SRS.

Subroutine return address (SRA) is the subroutine return stack pointer, which is automatically incremented or decremented as subroutines are called and returns are made from the subroutine.

GE MEDICAL SYSTEMS INSTITUTE

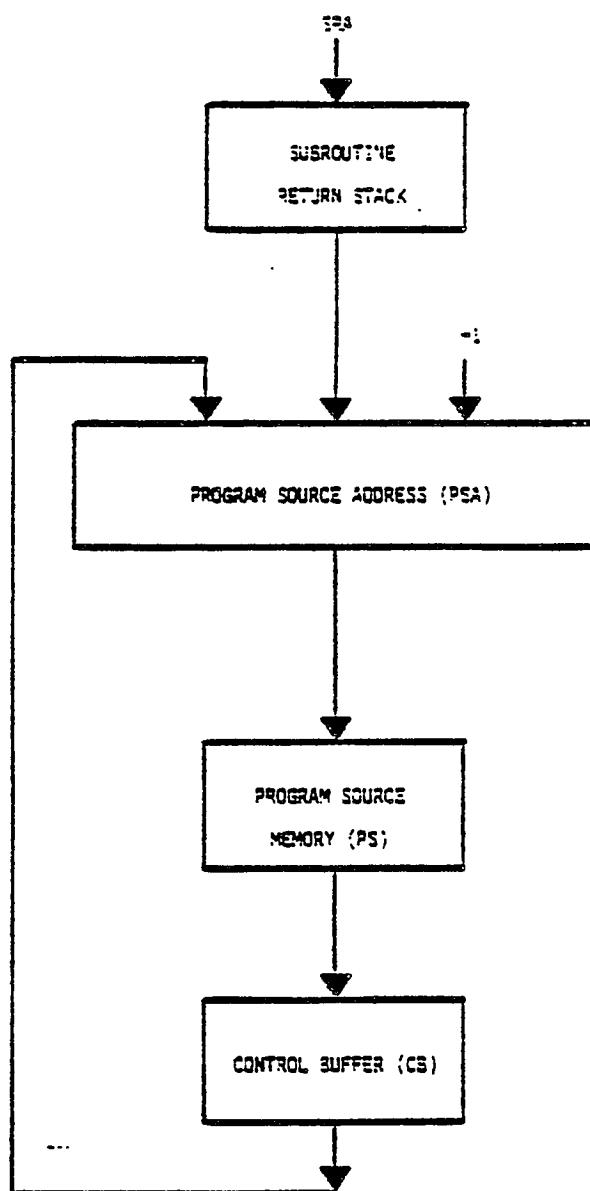


Figure 2-1 Control Unit

GE MEDICAL SYSTEMS INSTITUTE

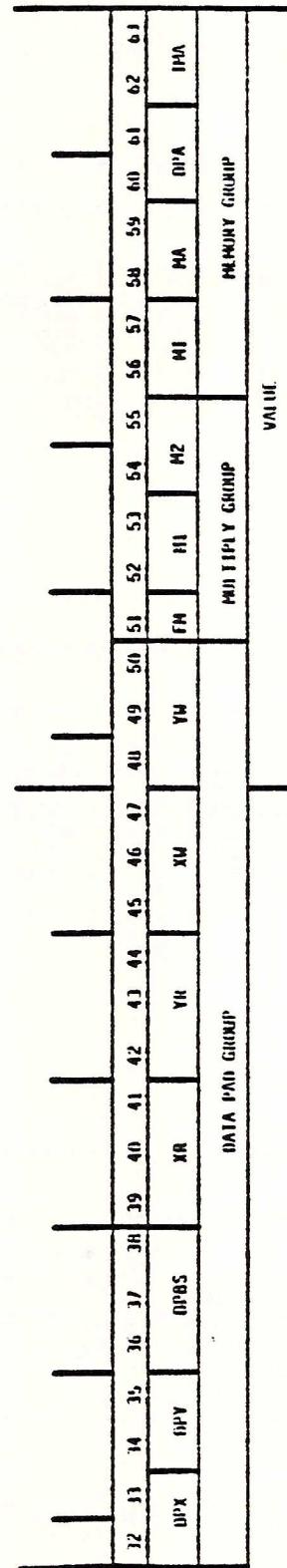
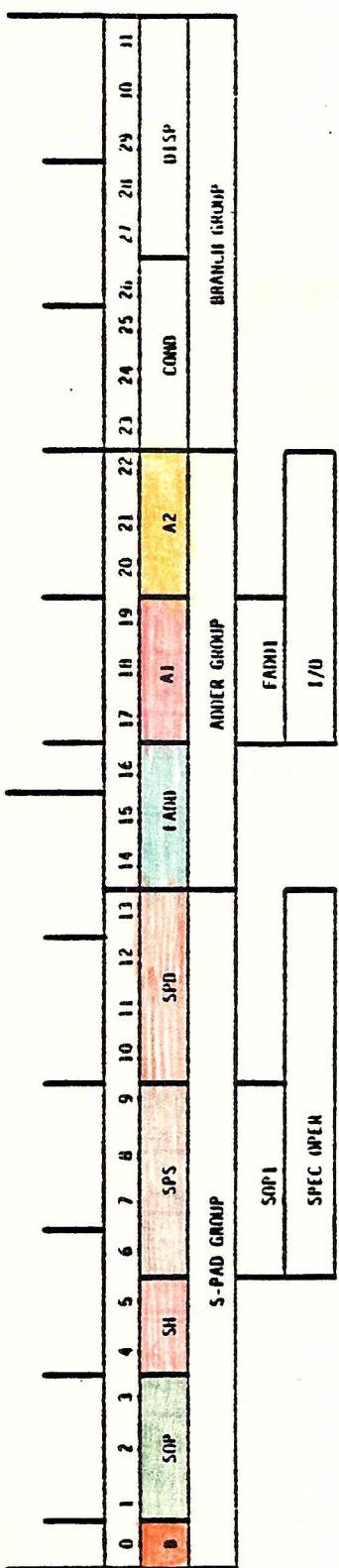


Table B-1 AP Instruction Field Layout

→ 64 bit format ↴

GE MEDICAL SYSTEMS INSTITUTE

Table A-3 AP Instruction Summary

UNCONDITIONAL FIELDS Each of the following fields may be used in any given instruction word.

OCTAL CODE	FIELD NAME										OCTAL CODE
	S	SOP	SOP1	S4	S5	SPO	FADD	FADD1	A1	A2	
0	NOP	SOP1	NOP	NOP	(S-PAD)	(S-PAD)	FADD	FADD1	NOP	NC	0
1	SPEC	WATEXP	L	Source	Dest.		FSUBR	FIX	FM	FA	1
2	ADD	WATHMN	RR	Reg.)	Reg.)		FSUB	FIWT	DPX	DPY	2
3	SUB	WRTL4N	R				FADD	FCSLT	DPY	DPY	3
4	MUL						FEQV	FSM2C	TN	MD	4
5	MOV	NOP					FAND	F2ESM	ZERO	ZERO	5
6	AND	NOP					FOR	FSCALE	ZERO	MOPX	6
7	OR	NOP					IO	FABS	ZERO	EDPY	7
10	EQV	NOP									10
11	CLR										11
12	INC										12
13	DEC										13
14	COM										14
15	LDSPNL										15
16	LDSPPE										16
17	LDSP1										17
	LDSP2										

OCTAL CODE	FIELD NAME										OCTAL CODE
	COND	DISP	DPY	DPY	DPBS	XR	YQ	XW	YW	SM	
0	NOP	(Branch	NOP	NOP	ZERO	(DPX	(DPY	(DPX	(DPY	NOP	0
1	#	Displace-	DS	DB	INBS	Read	Read	Write	Write	FMUL	1
2	BR	ment)	FA	FA	VALUE*	Index)	Index)	Index)	Index)		2
3	BINTRO	(0-37)	FM	FM	DPX						3
4	BICN				DPY	(0-7)	(0-7)	(0-7)	(0-7)		4
5	BIOZ				MD						5
6	BFPE				SPFN						6
7	RETURN				TM						7
10	BFED										10
11	BFHE										11
12	BFGE										12
13	BFGT										13
14	SEQ										14
15	BNE										15
16	BGE										16
17	BGT										17

OCTAL CODE	FIELD NAME						OCTAL CODE
	M1	M2	M1	MA	DPA	TMA	
0	FM	FA	NOP	NOP	NOP	NOP	0
1	DPX	DPX	FA	INCHMA	INCOPA	INCTHA	1
2	DPY	DPY	FM	DECMA	DECOPA	DECTHA	2
3	TM	MD	DB	SETMA	SETOPA	SETTHA	3

* This instruction uses a 16-bit immediate VALUE as a constant or address (in bits 48-63 of this instruction). The YW, FM, M1, M2, MA and DPA fields are then disabled for this instruction word.

GE MEDICAL SYSTEMS INSTITUTE

Table A-4 SPEC Fields

SPEC FIELDS One of the SPEC Fields may be used per instruction word. The 3-oct Fields ('0, S0P, S0P1, SH, SPS, and SPO) are then disabled for this instruction.

OCTAL CODE	FIELD NAME								OCTAL CODE
	SPEC	STEST	H0STPNL	S0TPSA	PSEVEN	P0S0D0	PS	S0TEXIT	
0	STEST	BFLT	PNLLIT	JMPA*	RPS8A*	RPS1A*	RPSLA*	NOP	0
1	H0STPNL	BLT	DBELIT	JSRA*	RPS2A*	RPS3A*	RPSFA*	SETEXA*	1
2	SPMDA	BNC	DBHLIT	JMP*	RPS0*	RPS1*	RPSL*	NOP	2
3	NOP	BZC	DBLLIT	JSR*	RPS2*	RPS3*	RPSF*	SETEX*	3
4	NOP	BDSN	NOP	JMPT	RPS0T	RPS1T	RPSLT	NOP	4
5	NOP	BDBZ	NOP	JSRT	RPS2T	RPS3T	RPSFT	SETEXT	5
6	NOP	BIFN	NOP	JMP	NOP	NOP	RPSLP	NOP	6
7	NOP	BIF2	NOP	JSRP	NOP	NOP	RPSFP	SETEXP	7
10	S0TPSA	NOP	SW0B	NOP	WPS8A*	WPS1A*	WPSLA*	NOP	10
11	PSEVEN	NOP	SW0BE	NOP	WPS2A*	WPS3A*	WPSRA*	NOP	11
12	P0S0D0	NOP	SW0BH	NOP	WPS0*	WPS1*	WPSL*	NOP	12
13	PS	NOP	SW0BL	NOP	WPS2*	WPS3*	WPSR*	NOP	13
14	SETEXIT	BFL0	NOP	NOP	WPS0T	WPS1T	WPSLT	NOP	14
15	NOP	BFL1	NOP	NOP	WPS2T	WPS3T	WPSRT	NOP	15
16	NOP	BFL2	NOP	NOP	NOP	NOP	WPSLP	NOP	16
17	NOP	BFL3	NOP	NOP	NOP	NOP	WPSRP	NOP	17

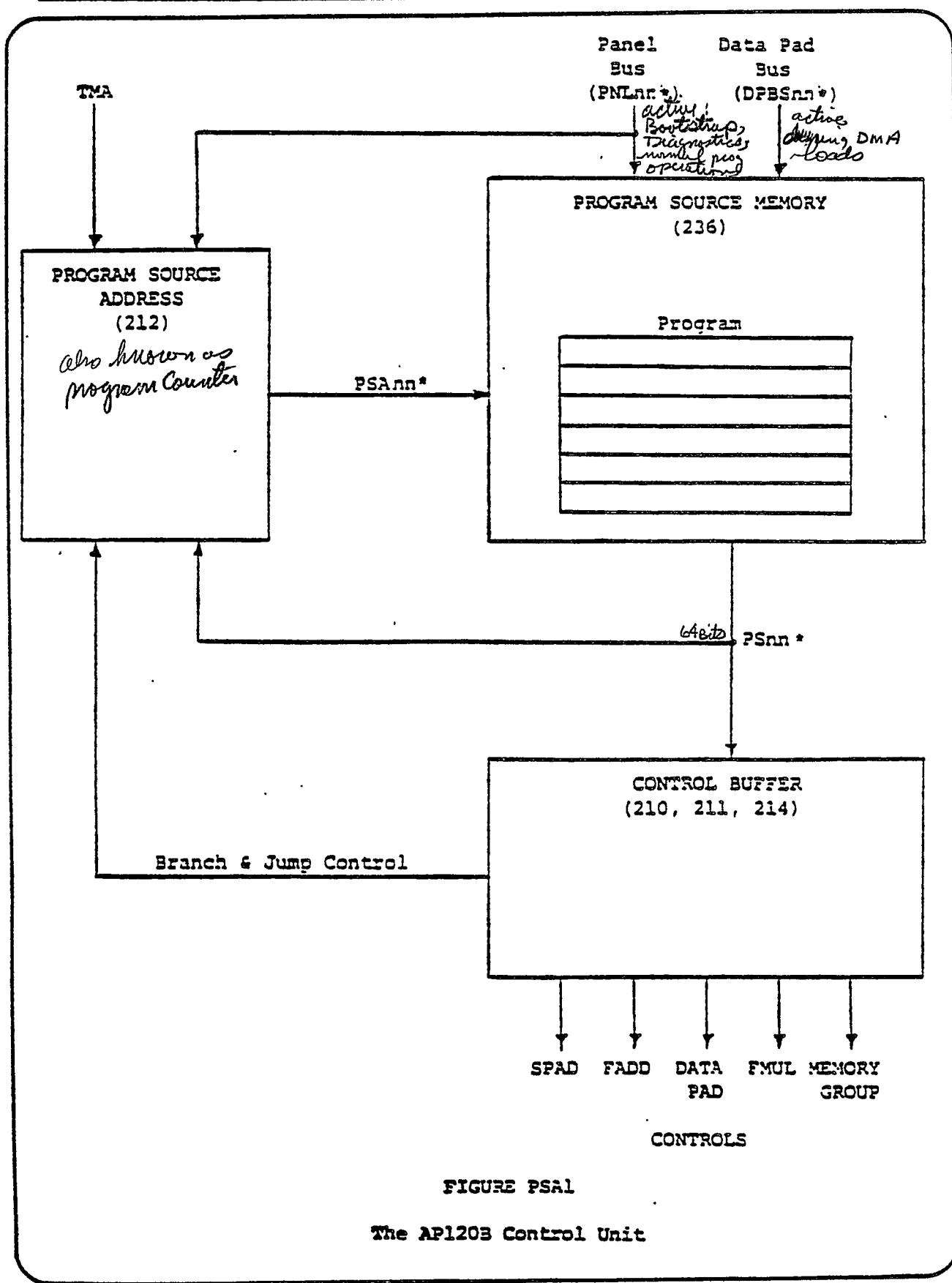
* This instruction uses a 16-bit integer VALUE (in bits 48-63 of the instruction word). The YW, FM, M1, M3, M4, TMA, and PDA Fields are then disabled for this instruction word.

Table A-5 I/O Fields

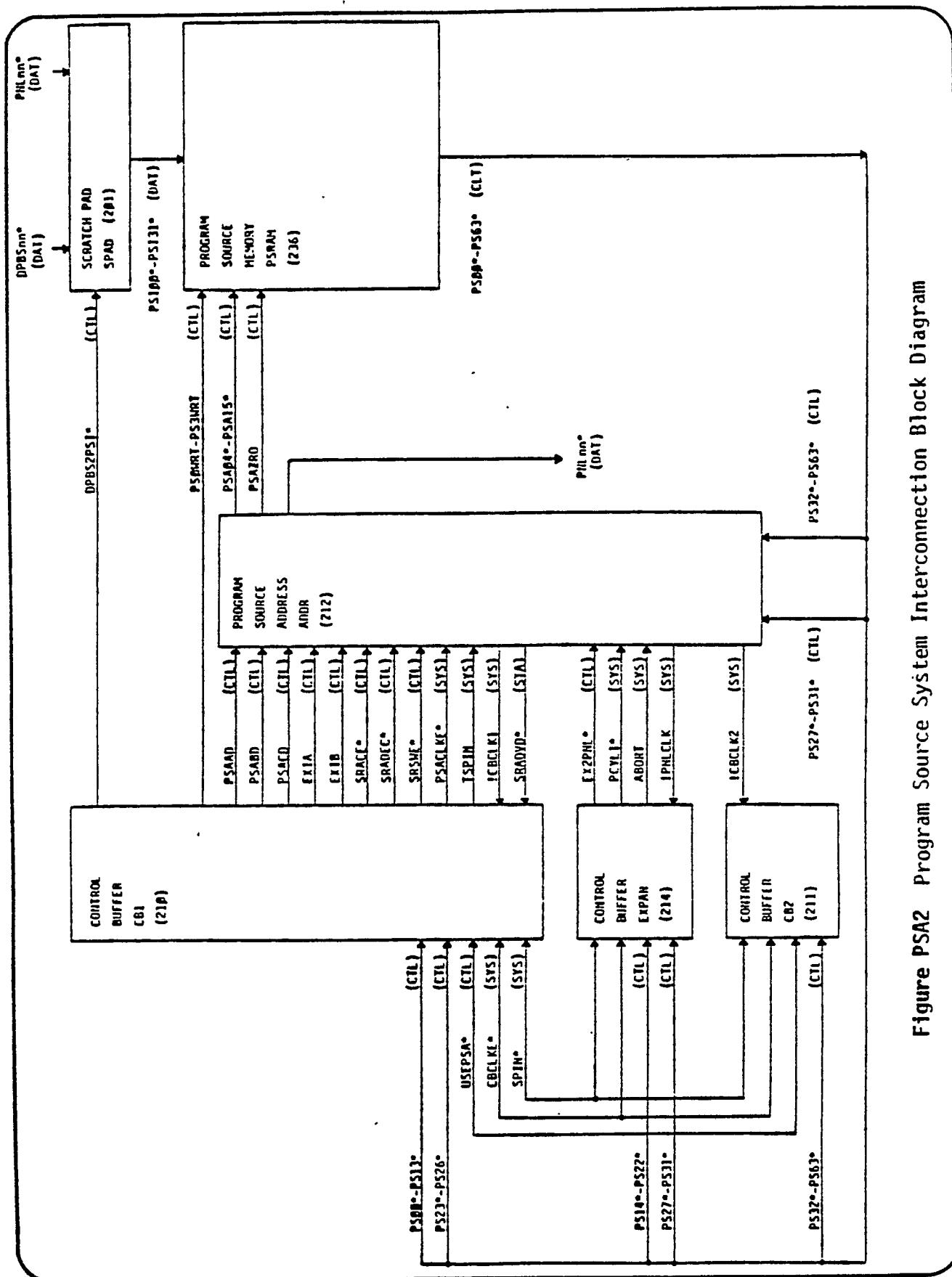
I/O FIELDS One of the I/O fields may be used per instruction word. The floating adder fields (FADD, FADD1, A1, and A2) are then disabled for this instruction word.

OCTAL CODE	FIELD NAME							OCTAL CODE
	IO	L0REG	R0REG	INOUT	SENSE	FLAG	CONTROL	
0	L0REG	NOP	RPSA	OUT	SNSA	SFL0	HALT	0
1	R0REG	LDSPD	RSPD	SPROUT	SPININ	SFL1	I0RST	1
2	SPMDAV	LDMA	RTMA	OUTDA	SNSADA	SFL2	INTEN	2
3	NOP	LDTMA	RTMA	SPOTDA	SPNADA	SFL3	INTA	3
4	INOUT	LDOPA	R0PA	IN	SNSB	CFL0	REFR	4
5	SENSE	LDSP	RSPFN	SPININ	SPINB	CFL1	ARTEX	5
6	FLAG	LDAPS	RAPS	OUTDA	SNSBDA	CFL2	ARTMAN	6
7	CONTROL	LD0A	R0A	SPINDA	SPN0BA	CFL3	NOP	7

GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

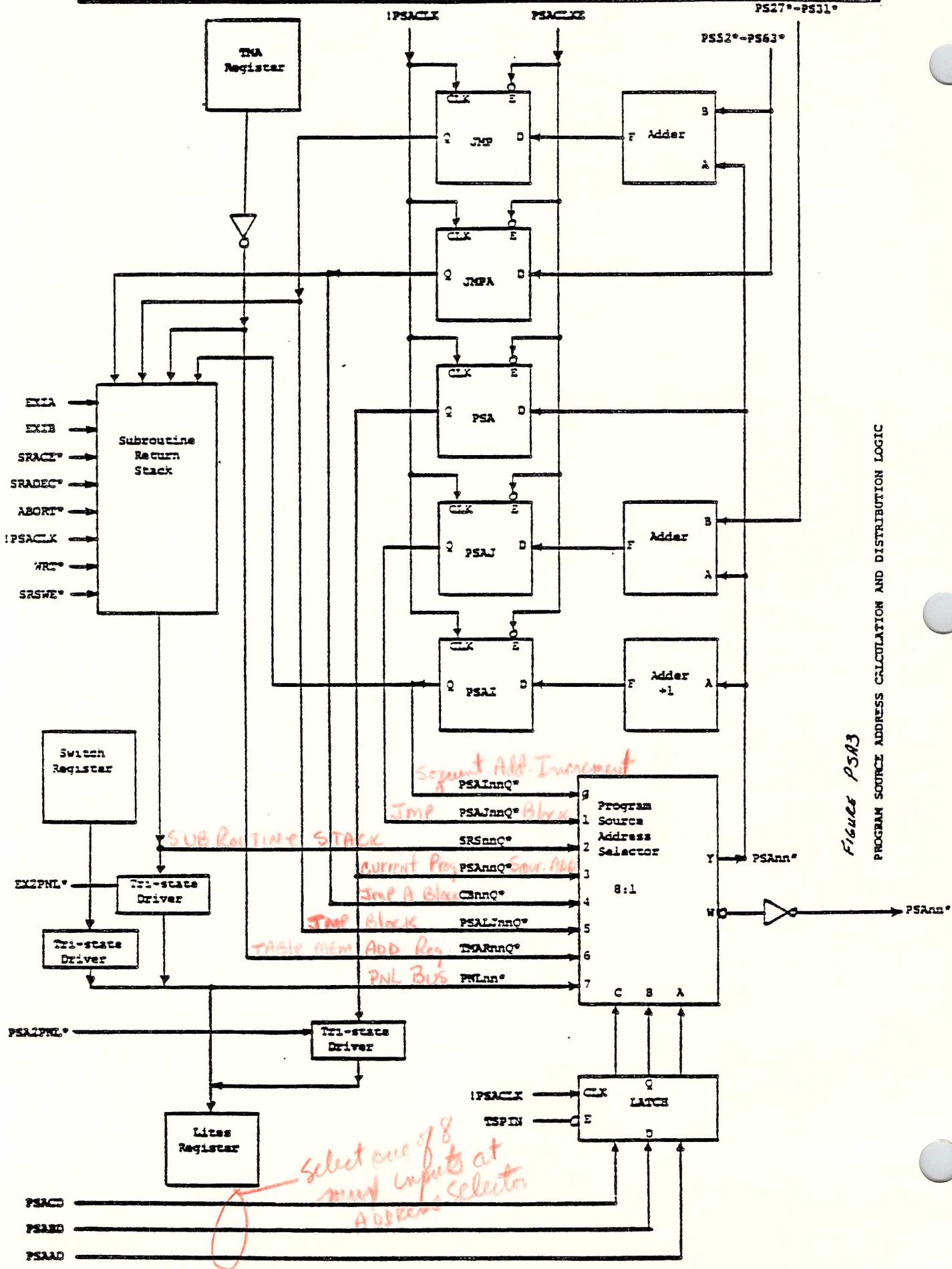


Figure P5.3 PROGRAM SOURCE ADDRESS CALCULATION AND DISTRIBUTION LOGIC

GE MEDICAL SYSTEMS INSTITUTE

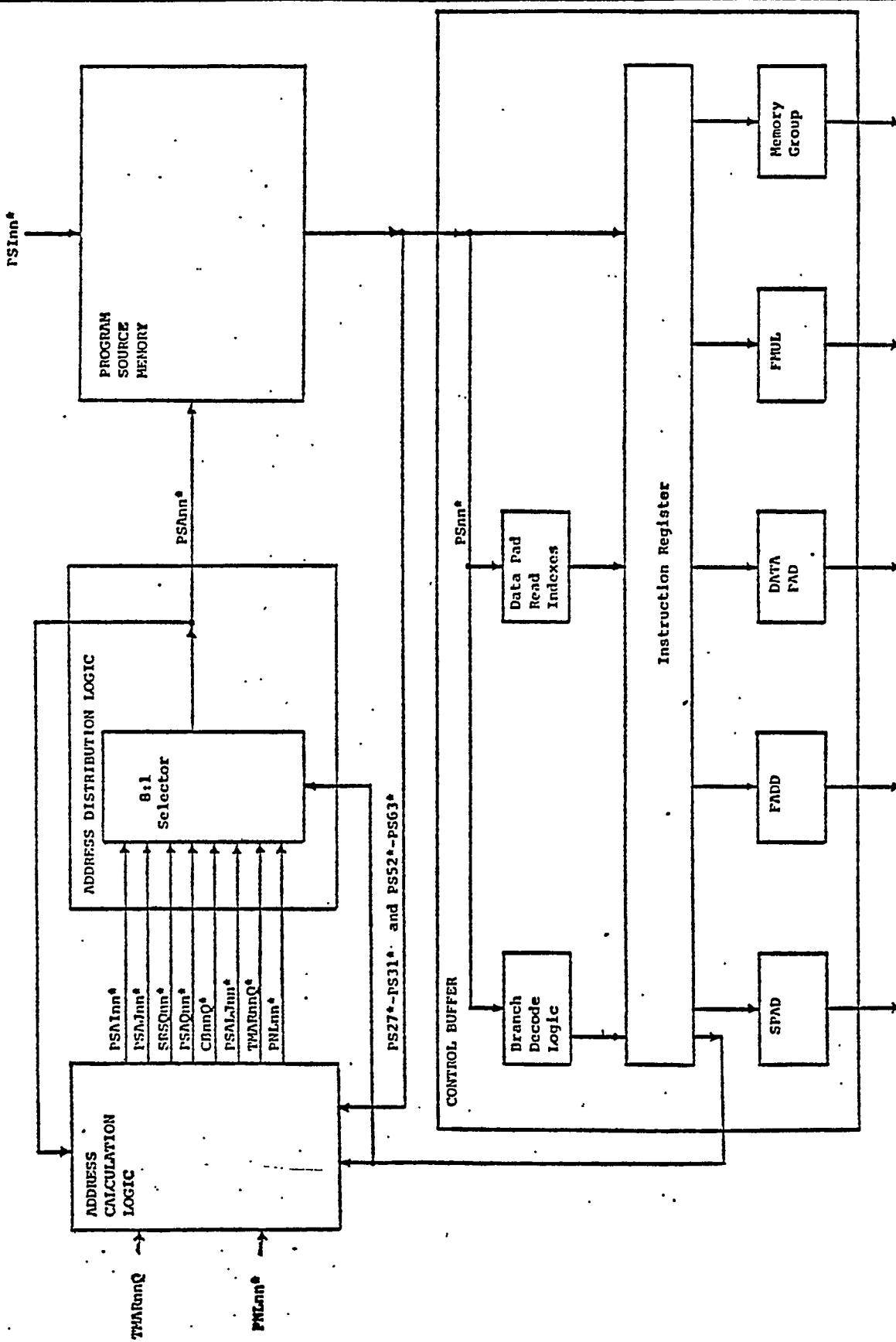


Figure PSA4 Program Source Address Logic

GE MEDICAL SYSTEMS INSTITUTE

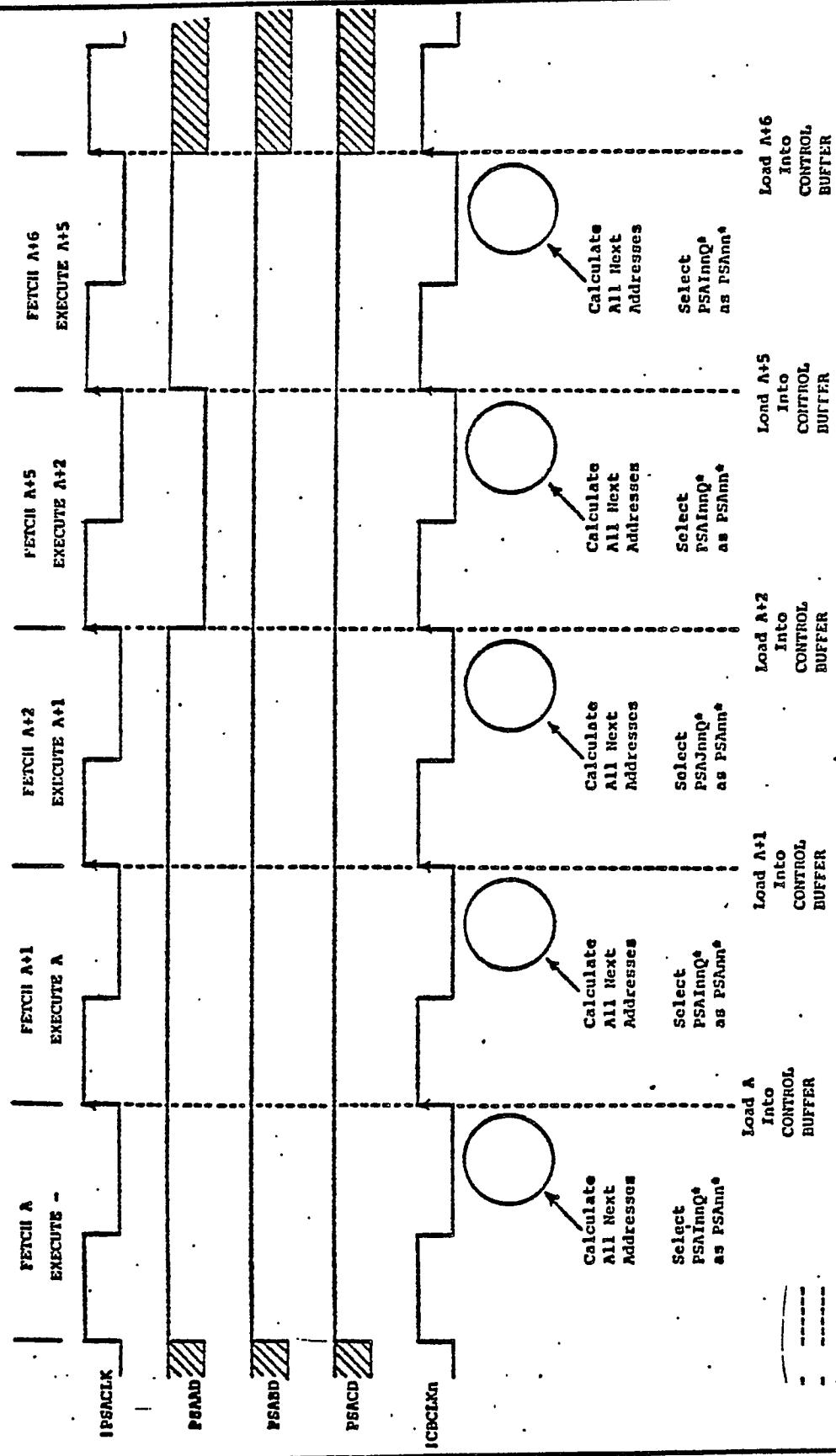


FIGURE FSA
PROGRAM SOURCE ADDRESS TIMING

GE MEDICAL SYSTEMS INSTITUTE

Program Source Signal Glossary

ABORT*	Reset the Array Processor
!CBCLK1	Control Buffer (CBL) one clock
!CBCLK2	Control Buffer (CB2) two clock
CBCLKE*	Control Buffer clock (load) enable
DPBSnn*	38 bits of the Data Pad Bus
DPBS2PSI*	
EX2PNL*	
EXIA	
EXIB	
PCYLI*	Panel Cycle One
PNLnn*	
!PNLCLK	Virtual front PaNeL CLoCK
PSnn*	Program Source Memory Output Bits 00 through 63
PS0WRT-PS3WRT	
PSA04*-PSA15*	
PSAAD	
PSABD	
PSACD	
PSACLKE*	
PSAZRO	Program Source Zero, Disables Output of Program Source Memory
PSInn	
SPIN*	Suspend instruction execution
SRACE*	
SRADEC*	
SRAOUD*	
SRSWE*	
TSPIN	
USEPSA*	Use the Contents of PSA as the Next Program Source Address

GE MEDICAL SYSTEMS INSTITUTE

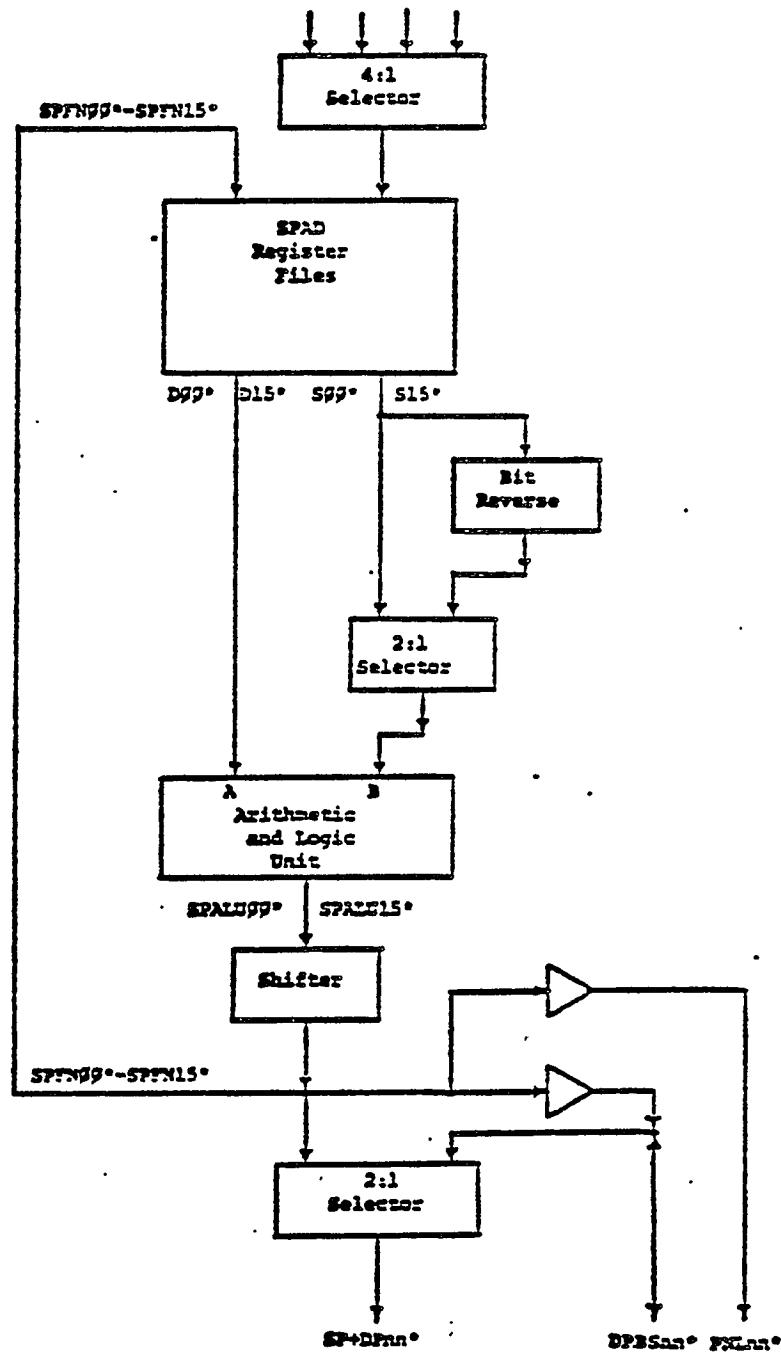
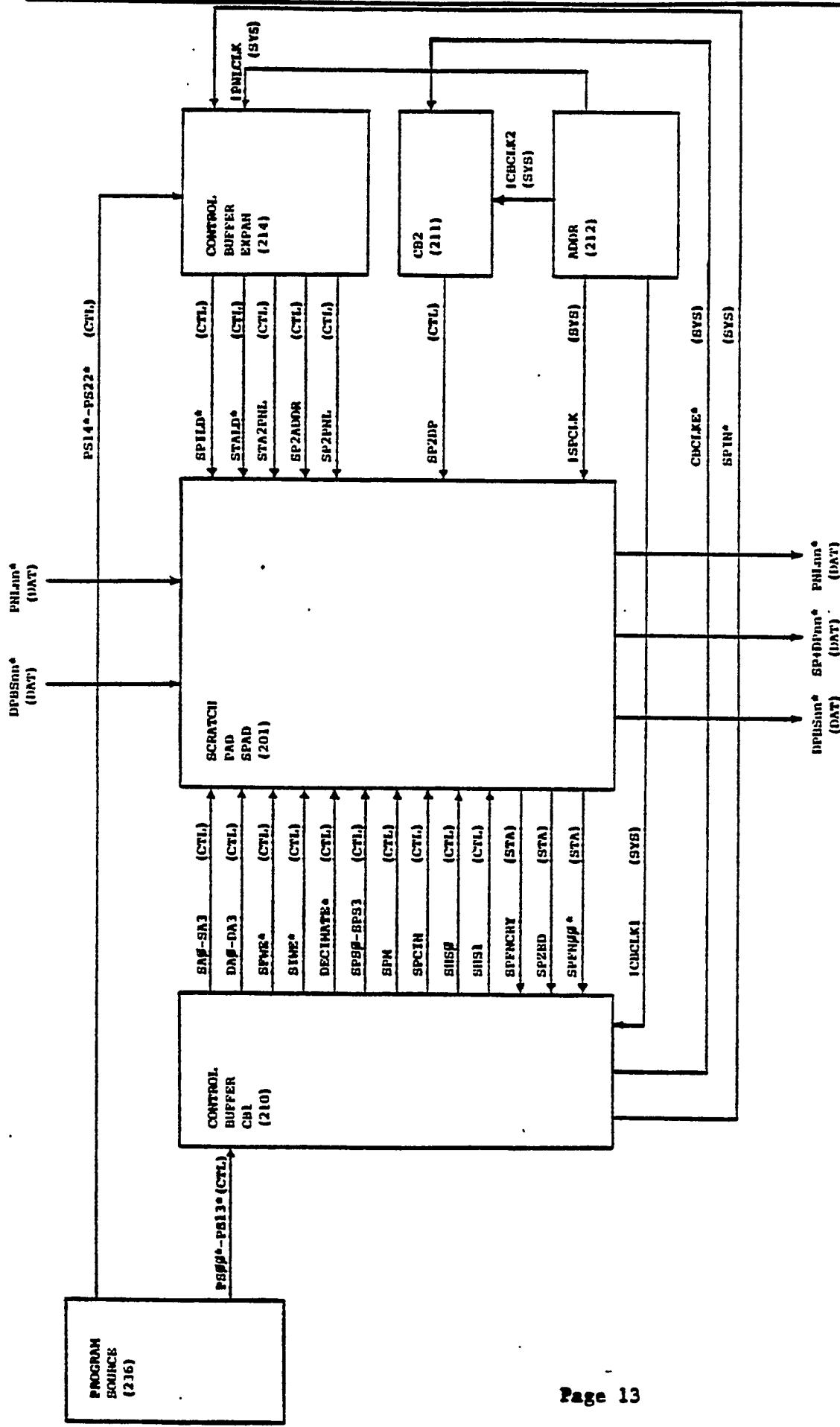


FIGURE SPAD1
SPAD Block Diagram

GE MEDICAL SYSTEMS INSTITUTE



SPAD2

SPAN Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

SPAD Signal Glossary

!CBCLK1 -	Control Buffer (CB1) one clock
!CBCLK2 -	Control Buffer (CB2) two clock
CBCLKE* -	Control Buffer clock (load) enable
DA0-DA3 -	SPAD Destination Address bits 0 through 3
DECIMATE* -	Use the contents of the bit reversed source as the B input to the SPAD arithmetic and logic unit
DPBSnn* -	38 bits of the Data Pad Bus
!PNLCLK -	Virtual front PaNeL CLOCK
PS03*-PS13* -	Program Source bits 03 through 13 which control the operation of SPAD
PS14*-PS22* -	Program Source bits 14 through 22 which control I/O operations
SA0-SA3 -	SPAD Source Address bits 0 through 3
SFWE* -	Causes the SPAD register addressed by the SPAD destination register (DA0-DA3) to be loaded from SPFN
SHS0 -	SPAD shifter control
SHS1 -	SPAD shifter control
SIWE* -	Causes the SPAD register addressed by the SPAD destination register (DA0-DA3) to be loaded from the output of the 4:1 mux
SP2ADDR -	2:1 selector (mux) control signal which causes SPFN to be enabled to SP+DP when true and the least significant 16 bits of the DPSS to be enabled to SP+DP when false
SP2DP -	Enables SPFN onto the Data Pad bus
SP2PNL -	Enables SPFN onto the PaNeL bus
SPCIN -	Forces a carry into the SPAD arithmetic and logic unit
!SPCLK -	SPAD clock which is generated from master system clock
SPFN03* -	The most significant bit of SPFN (the sign bit)
SPFNCRY -	The carry bit out of the shifter
SPILD* -	Loads the least significant 16 bits of the Data Pad bus into the SPAD register addressed by the destination register
SPIN* -	Suspend instruction execution
SPM -	SPAD arithmetic and logic Mode control
SPSG-SPS3 -	SPAD arithmetic and logic summer (function) controls
SPZED -	SPAD zero detect; true if SPFN is zero
STA2PNL -	Enable the status register contents to the Panel bus
STALD* -	Enables the status register to be loaded

GE MEDICAL SYSTEMS INSTITUTE

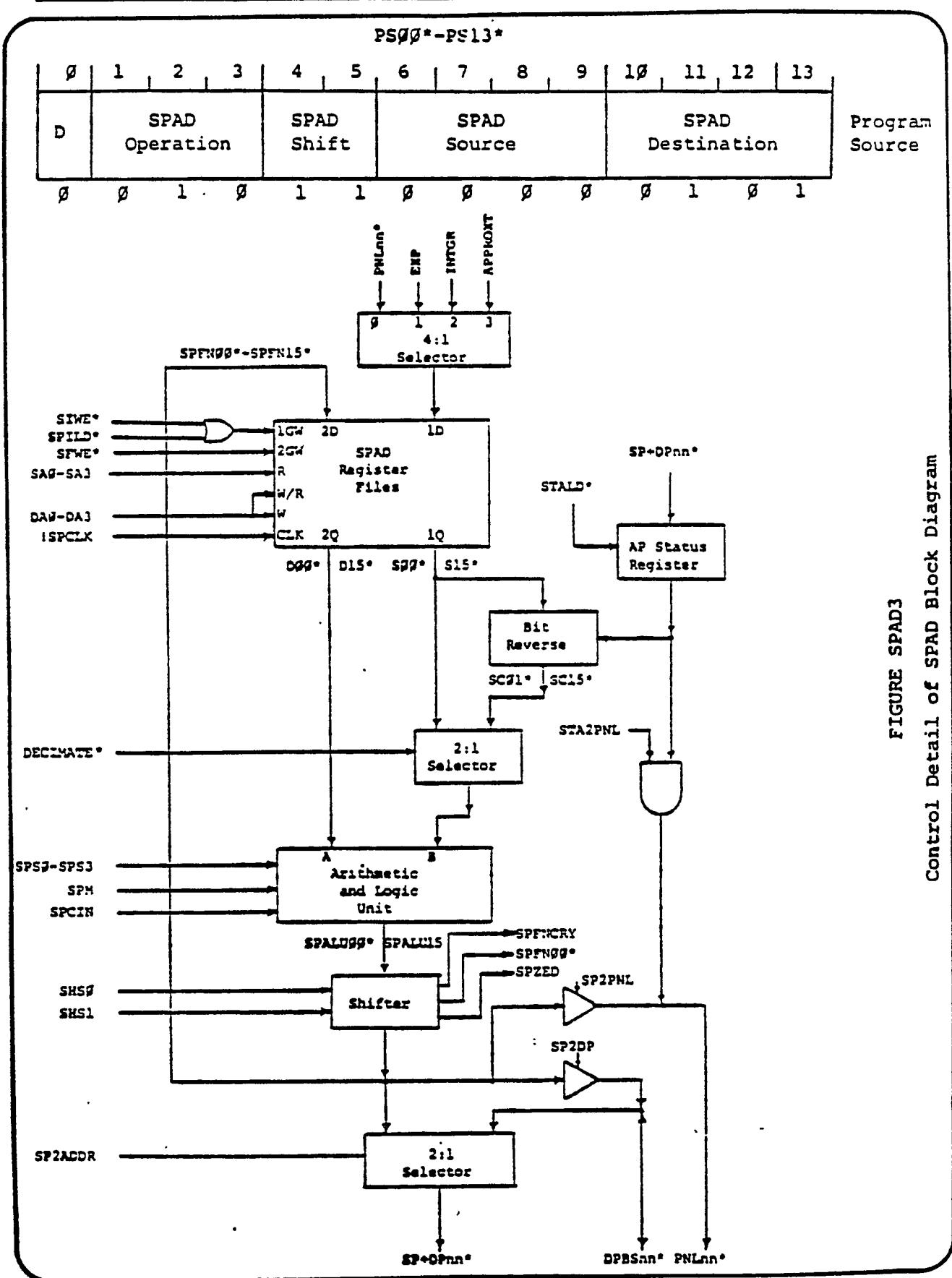


FIGURE SPAD3
Control Detail of SPAD Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

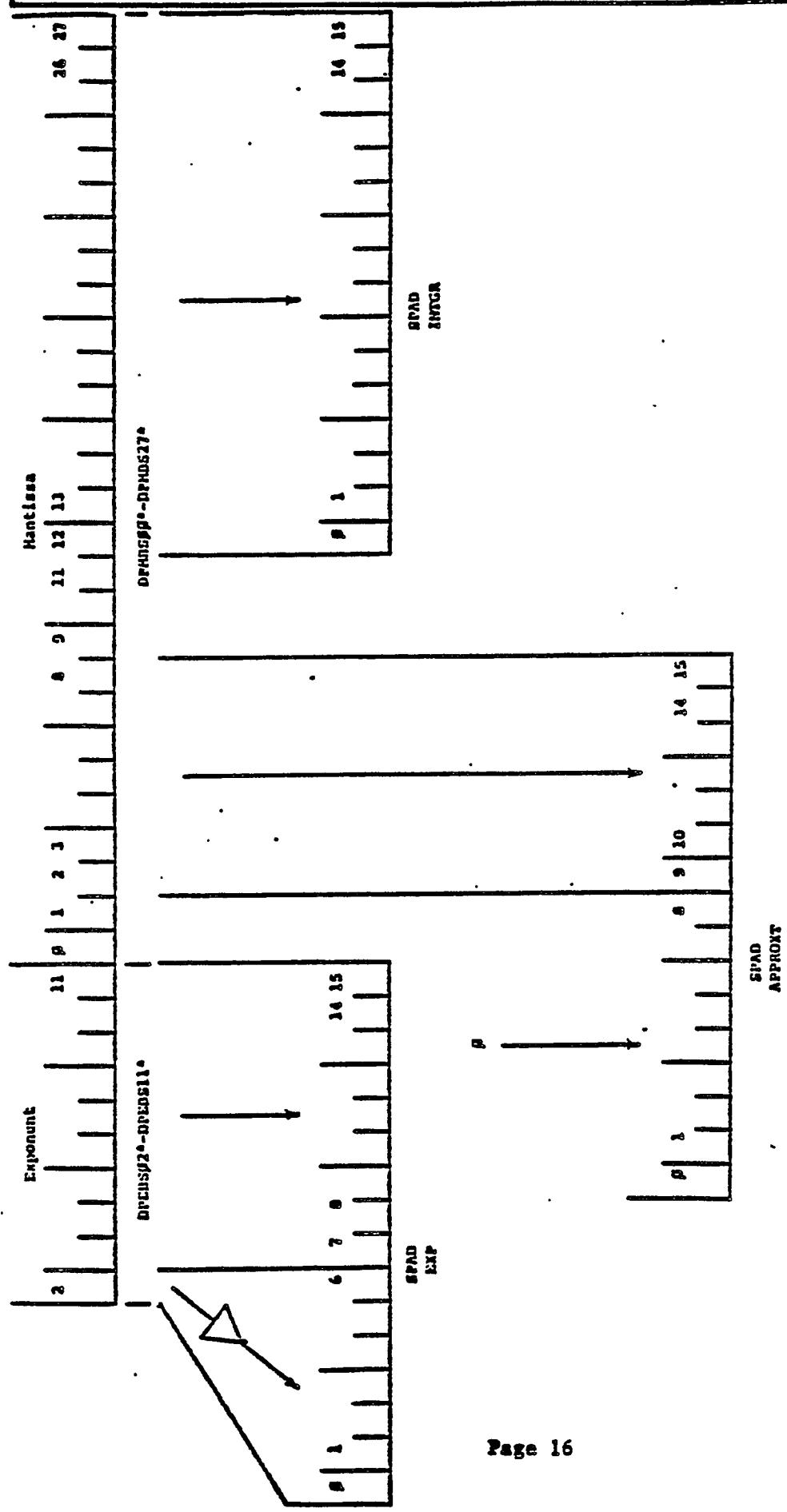


FIGURE SPAD4

Data Pad Bus to SPAD bit layout

GE MEDICAL SYSTEMS INSTITUTE

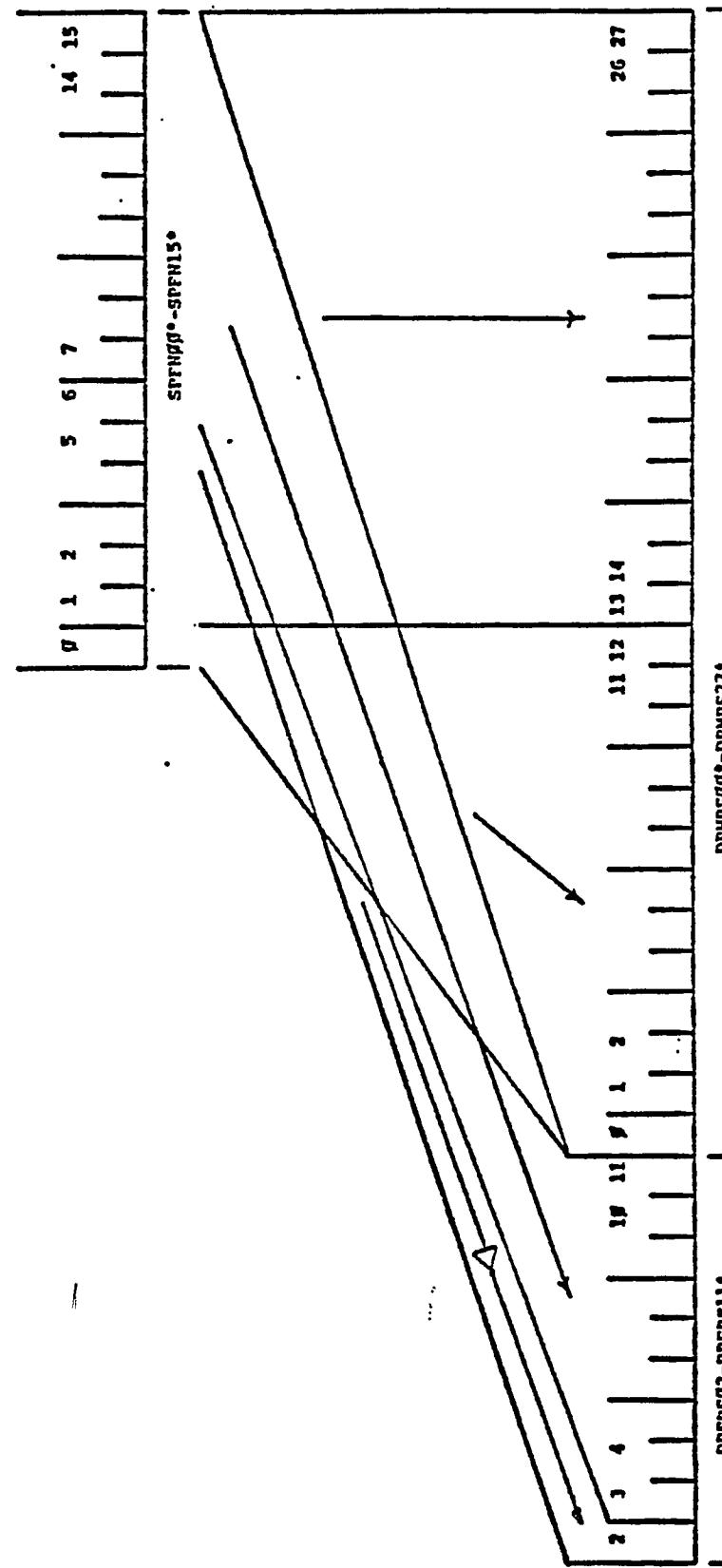


FIGURE SPADS5

SPIR to Date and Bus format

GE MEDICAL SYSTEMS INSTITUTE

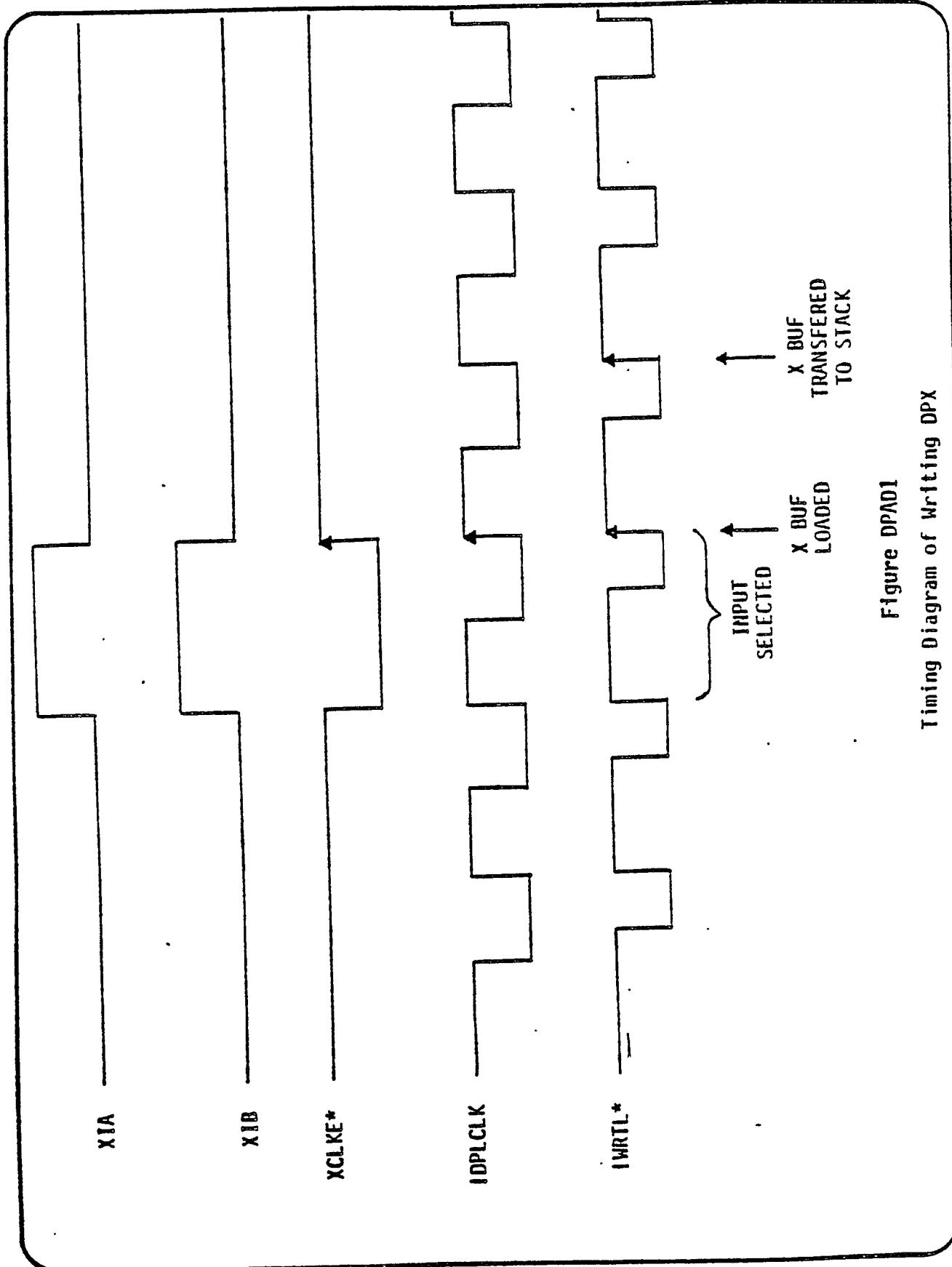


Figure DPX1
Timing Diagram of Writing DPX

GE MEDICAL SYSTEMS INSTITUTE

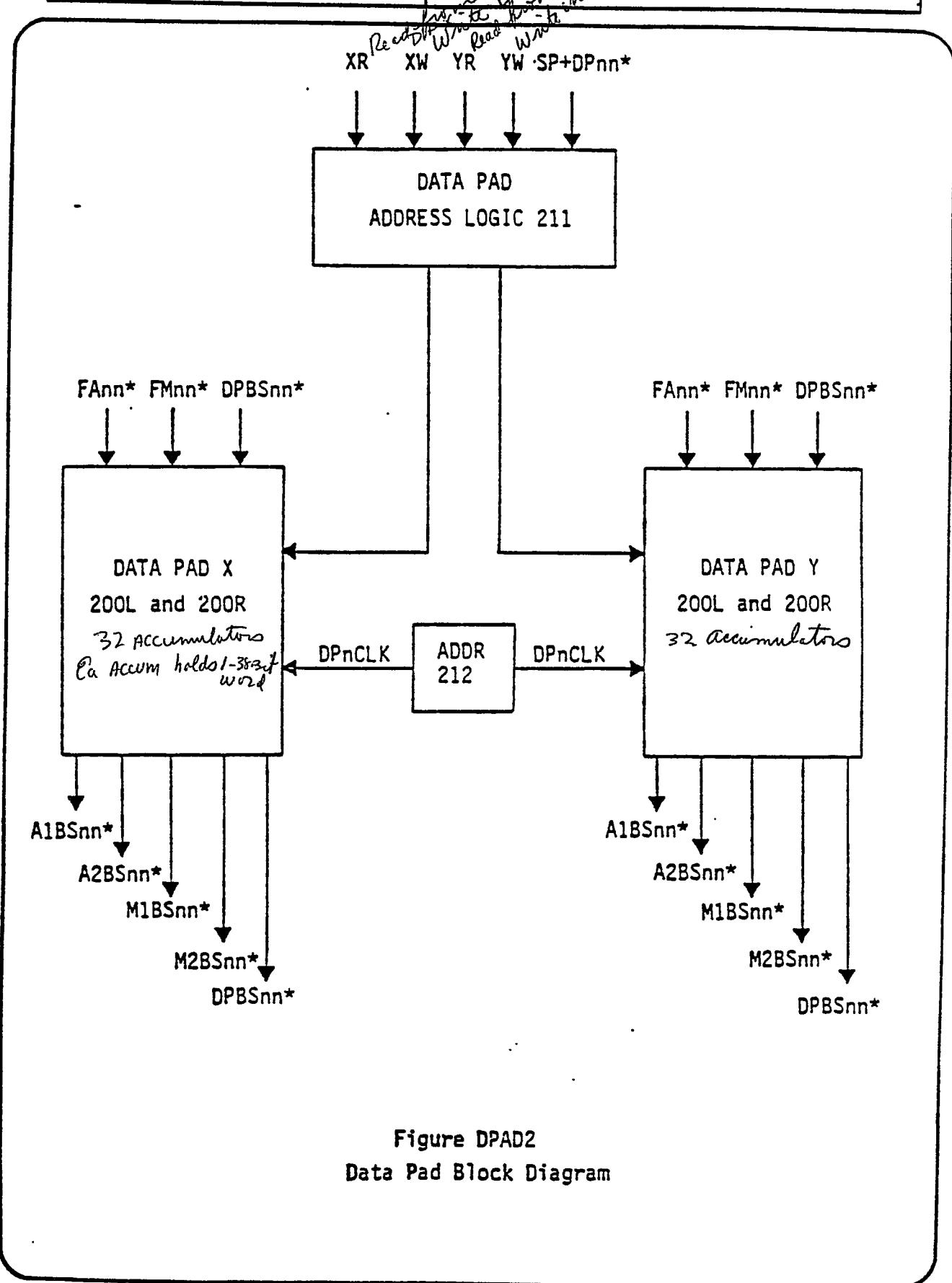


Figure DPAD2
Data Pad Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

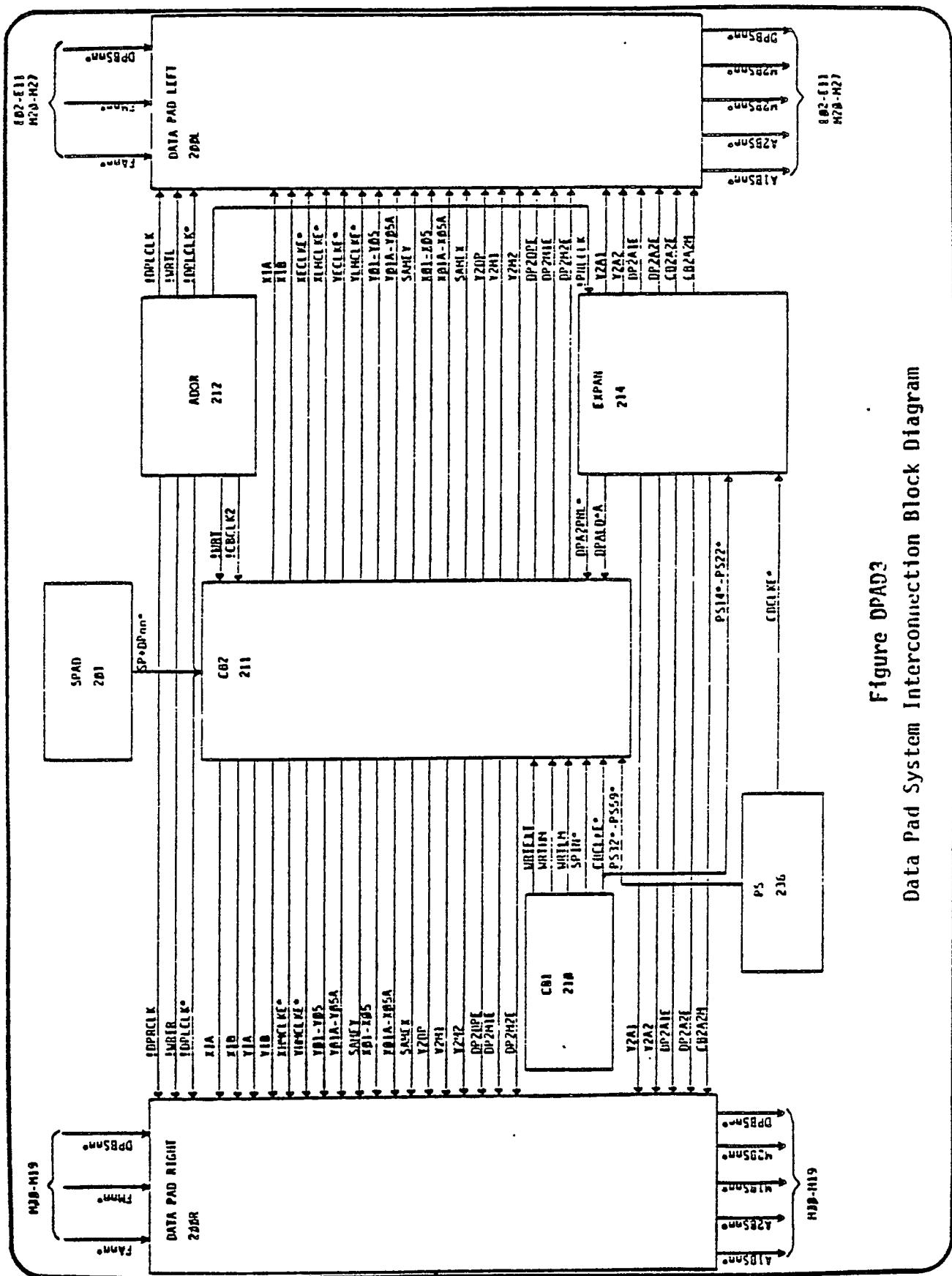


Figure DPA3
Data Pad System Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

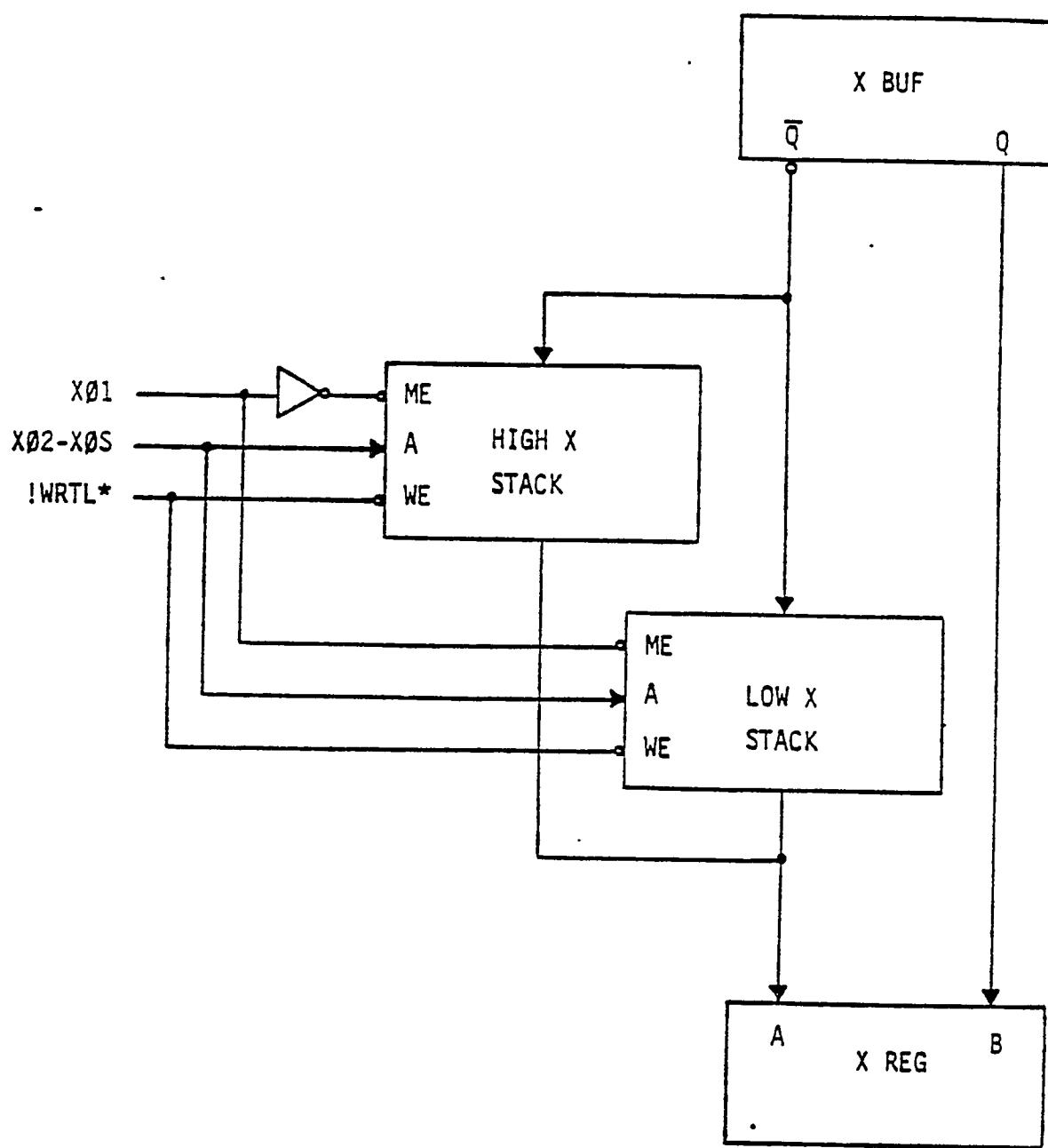


Figure DPAD4
Data Pad Memory Element Detail Showing Stack Address

GE MEDICAL SYSTEMS INSTITUTE

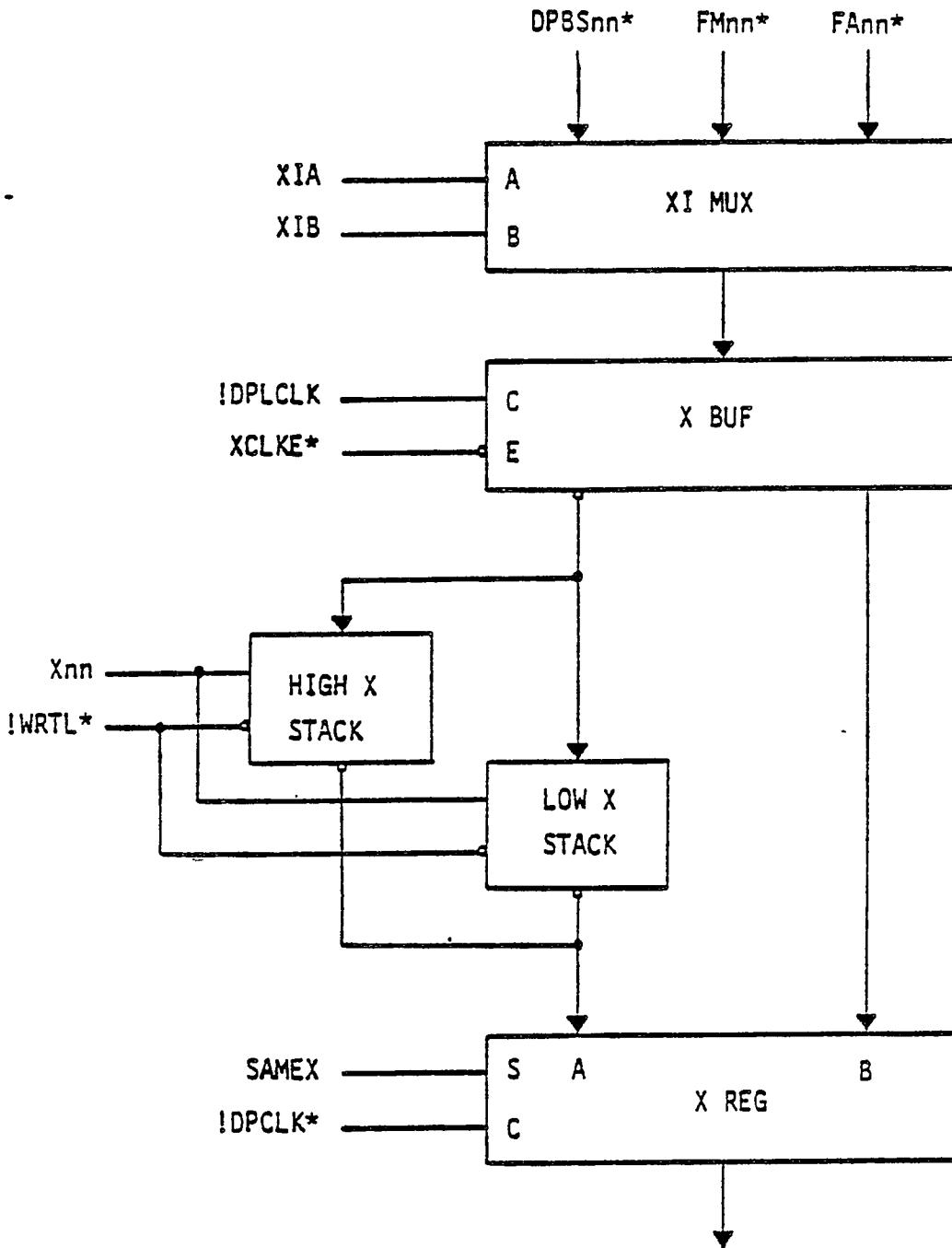


Figure DPAD5
Block Diagram of DPX

GE MEDICAL SYSTEMS INSTITUTE

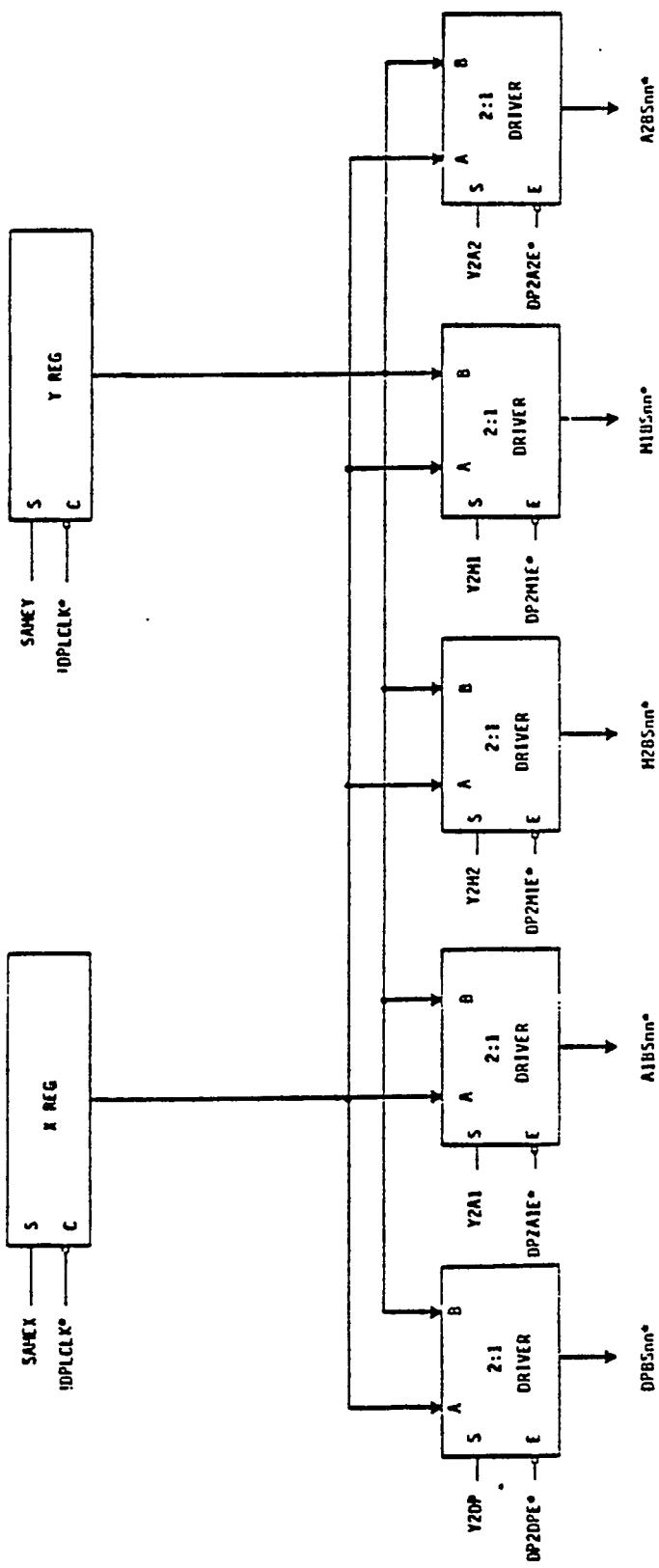


Figure DPADE
Data Pad Output Logic

GE MEDICAL SYSTEMS INSTITUTE

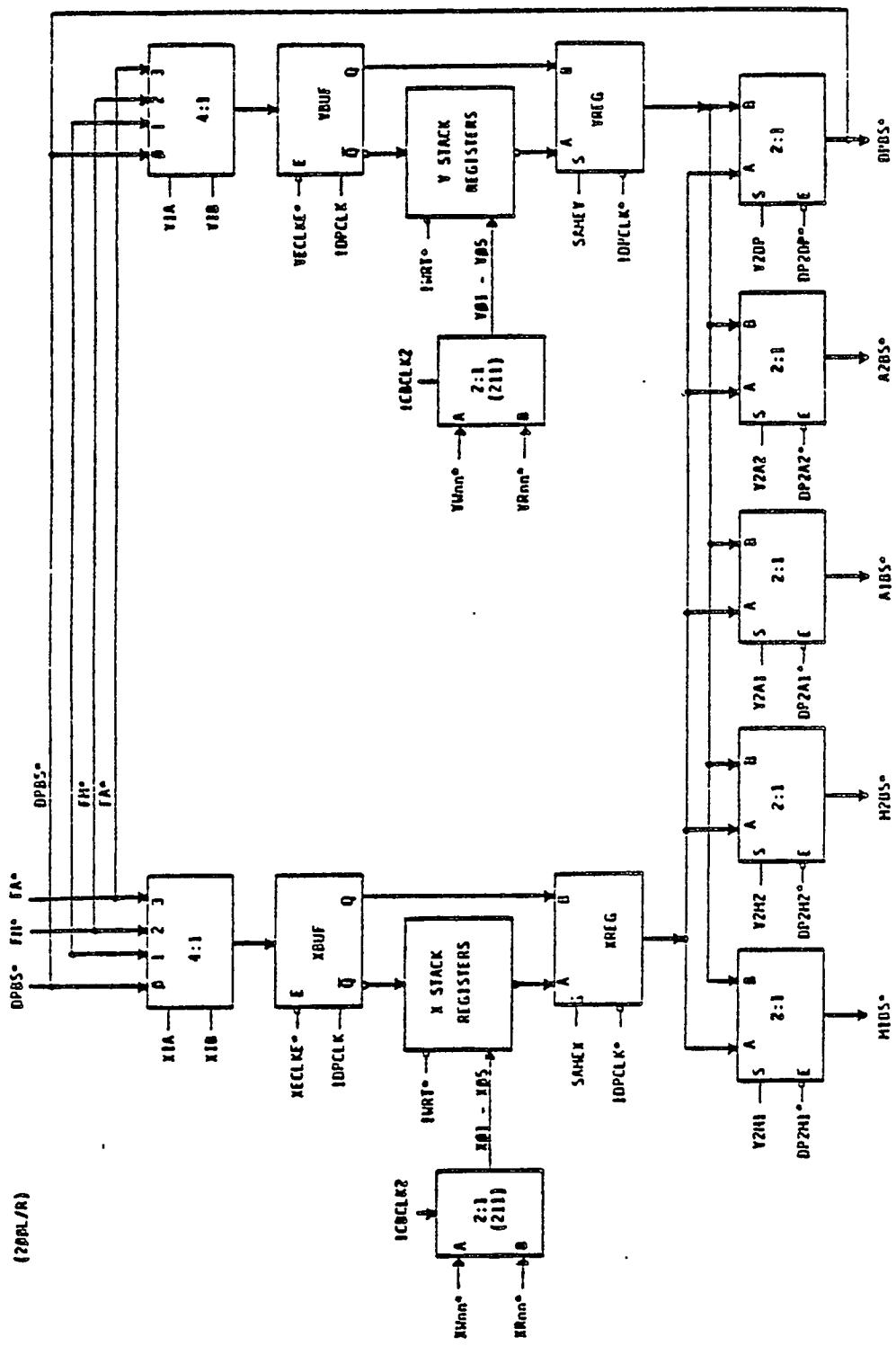


Figure DPAD7
Data Pad

GE MEDICAL SYSTEMS INSTITUTE

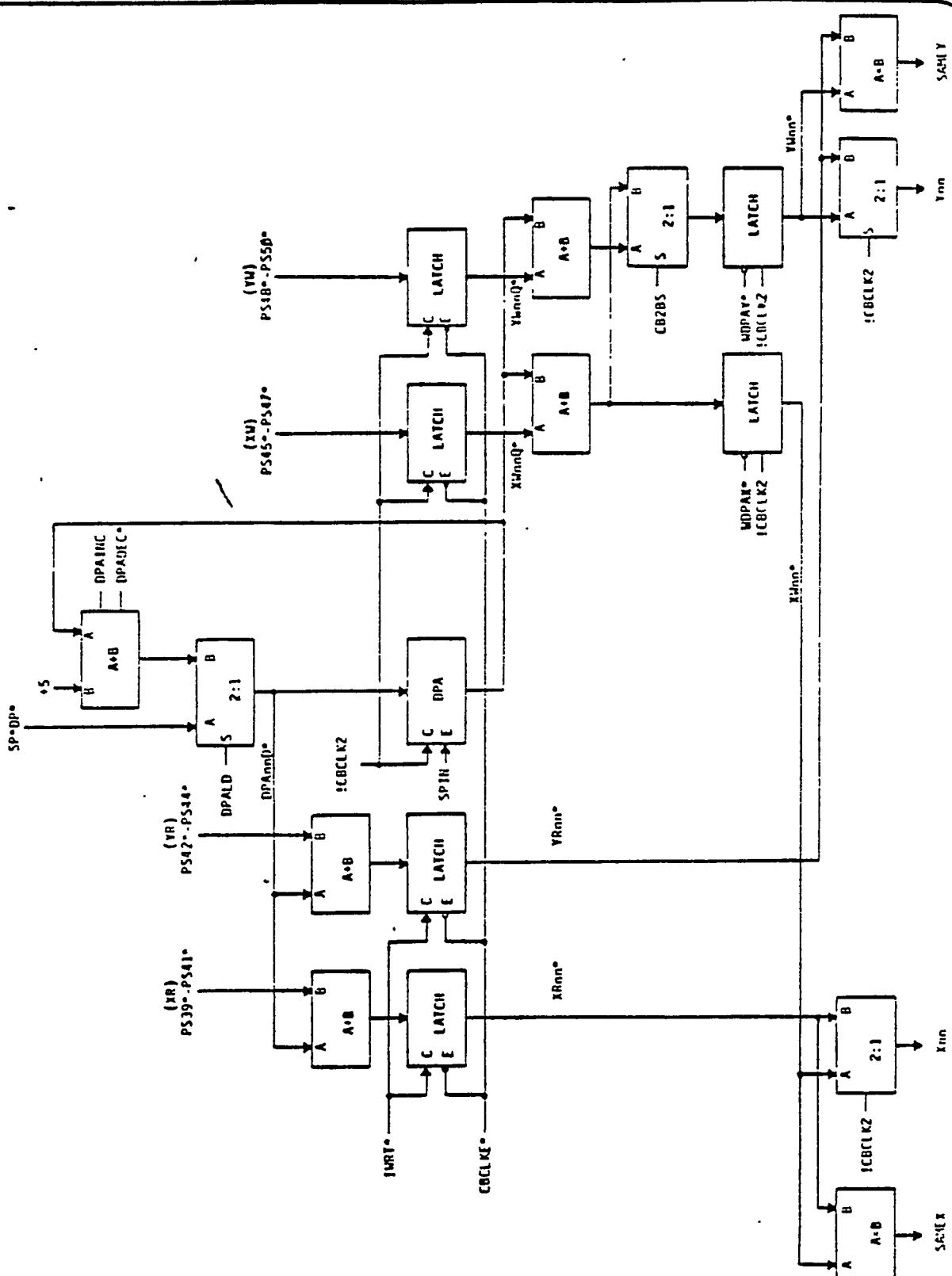


Figure DPAD8 Data Pad Address Logic

GE MEDICAL SYSTEMS INSTITUTE

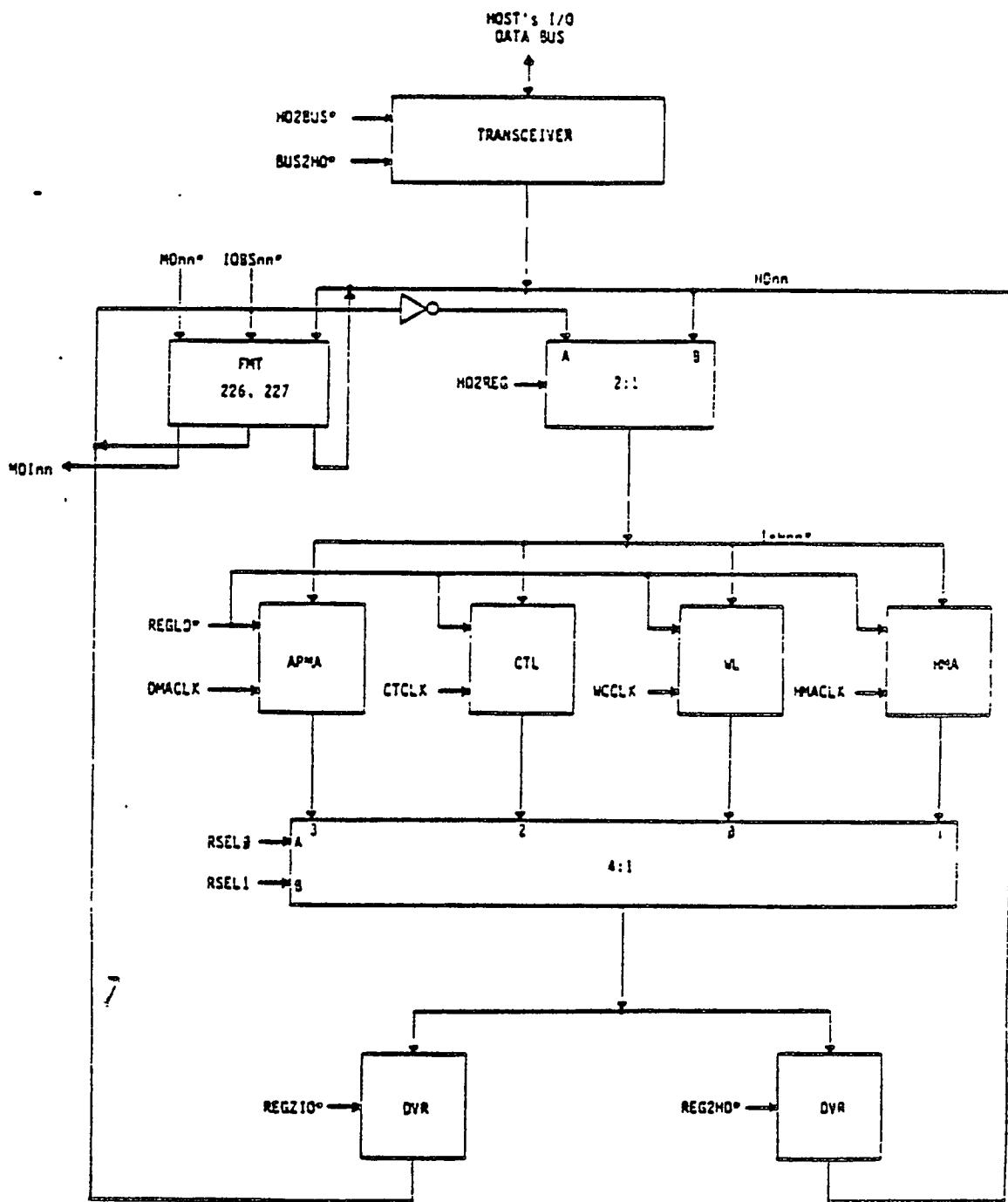


FIGURE IF2
Array Processor to Host Interface Data Paths

GE MEDICAL SYSTEMS INSTITUTE

SYSTEM CLOCK TIMING

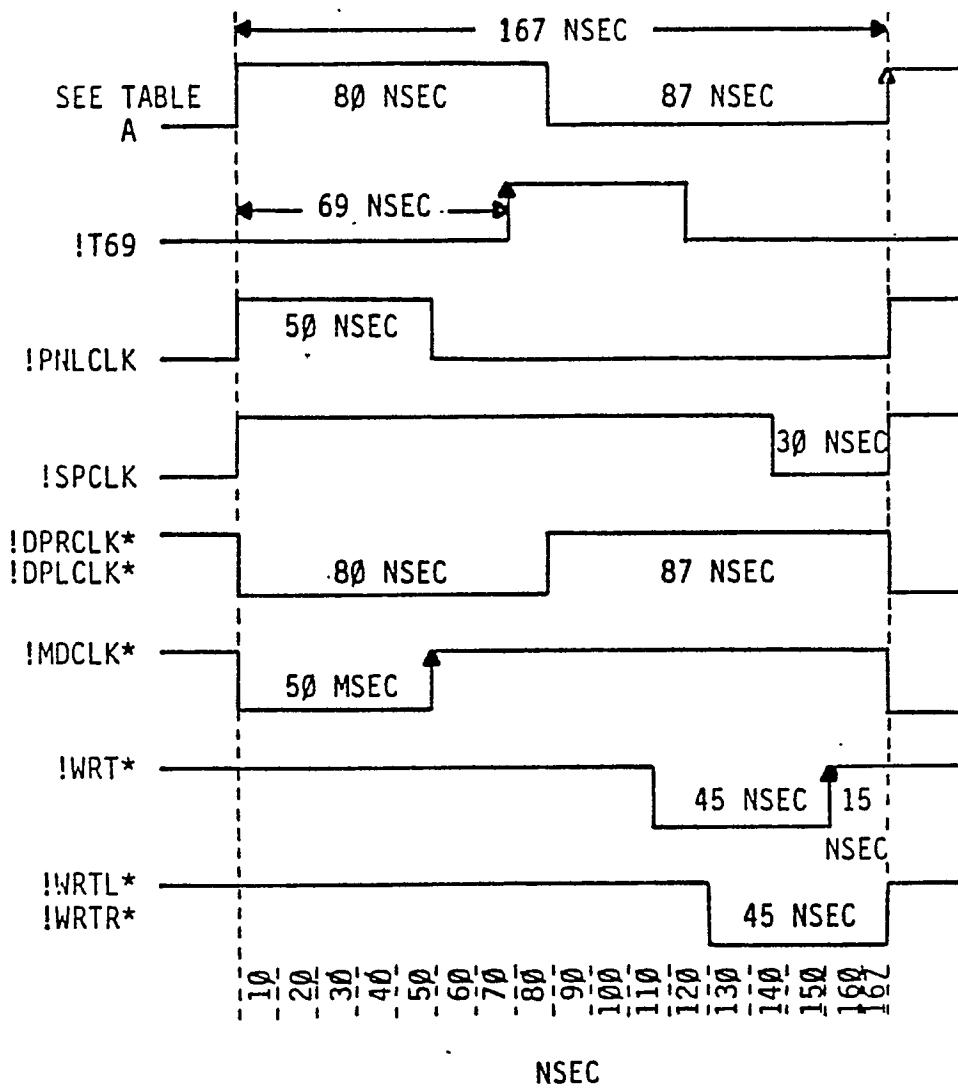


TABLE A

!CBCLK1
 !CBCLK2
 !CBCLK
 !DPLCLK
 !DPRCLK
 !FACLK2
 !FACLK3
 !FMCLKA
 !FMCLKB
 !FMCLKC
 !MCLK
 !MICLK
 !PSACLK
 !TMCLK

THESE CLOCKS
 ARE ALL 167
 NSEC AS SHOWN

AP-120B System Clock Timing

GE MEDICAL SYSTEMS INSTITUTE

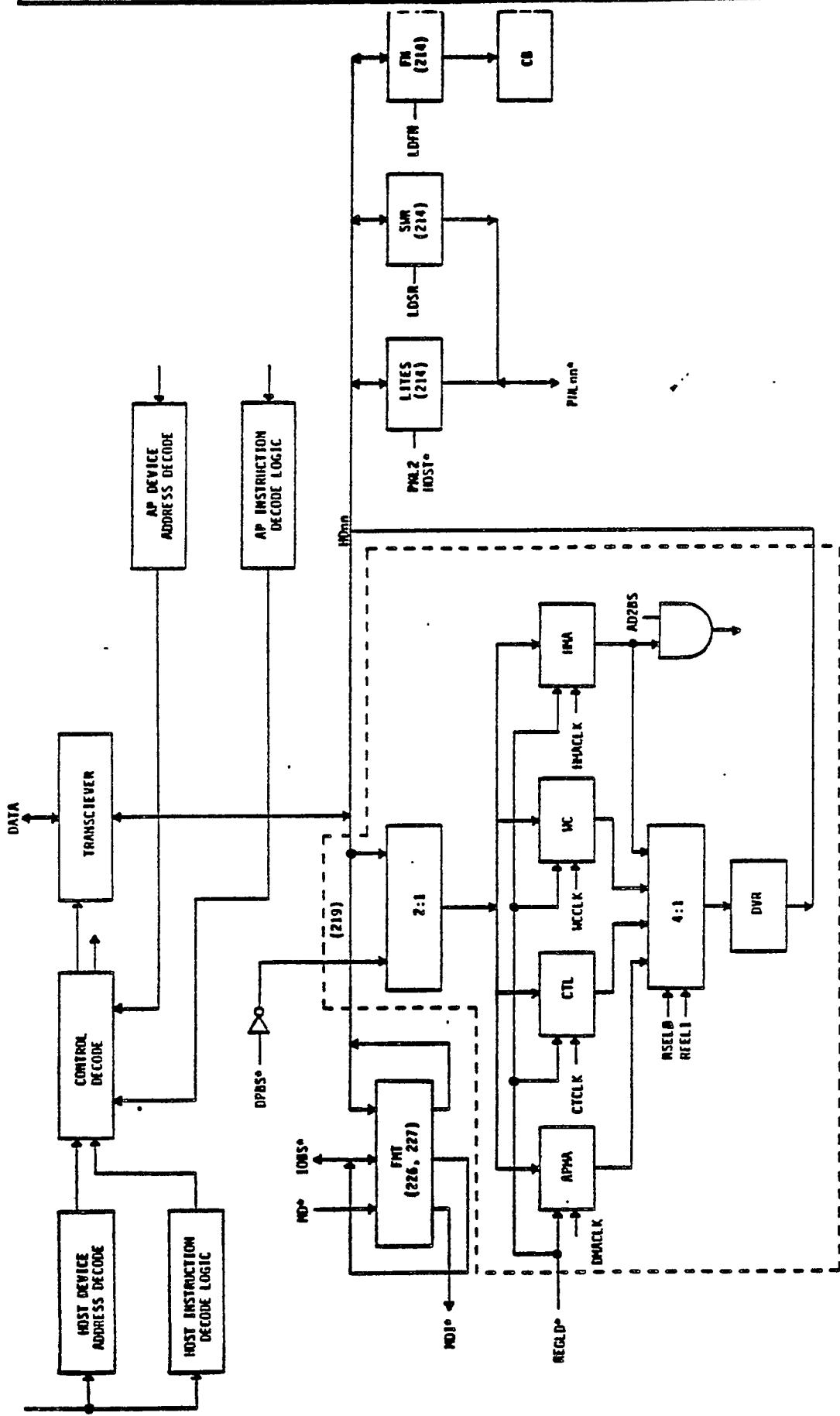


Figure IF1
Interface Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

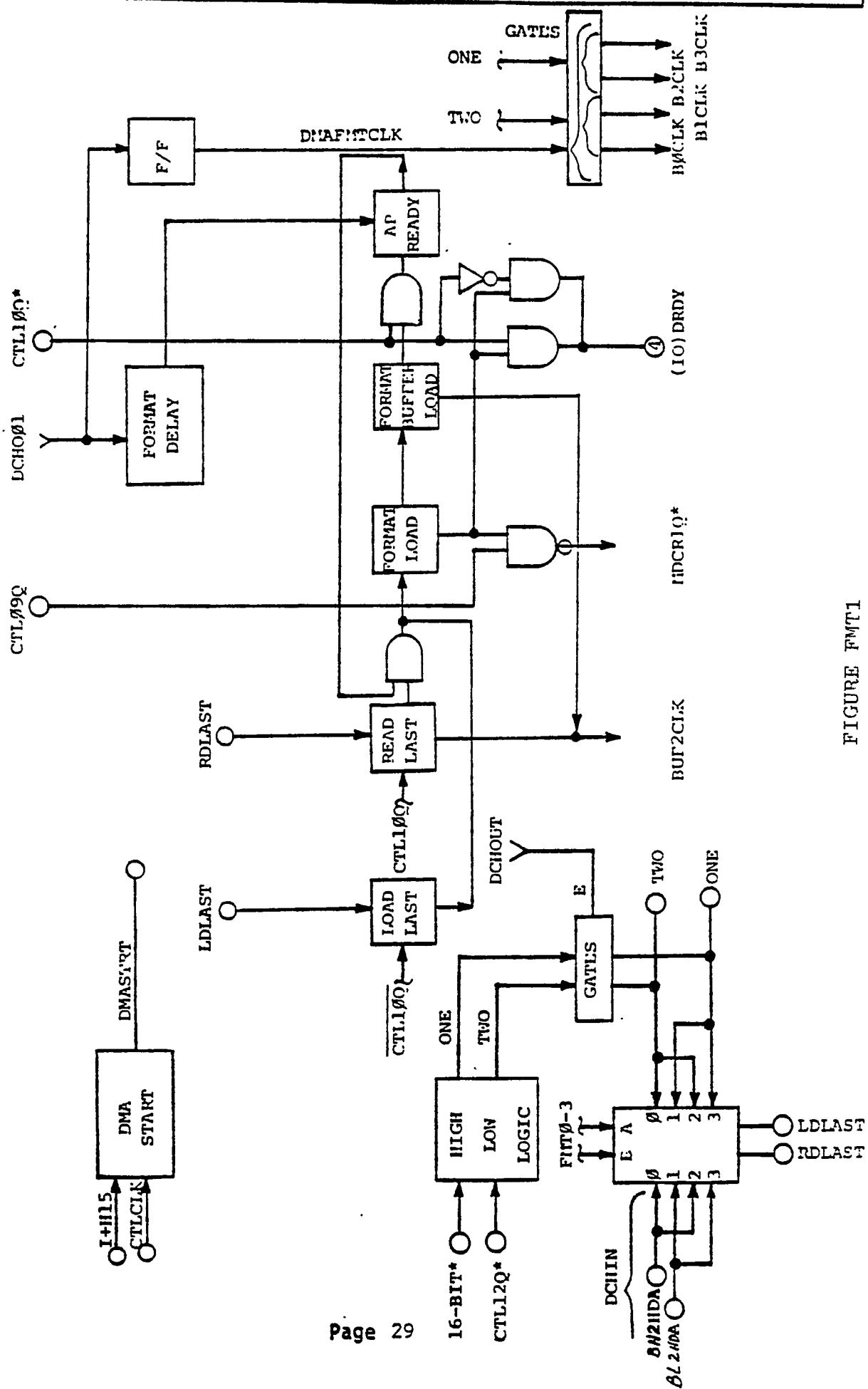


FIGURE FMT1

GE MEDICAL SYSTEMS INSTITUTE

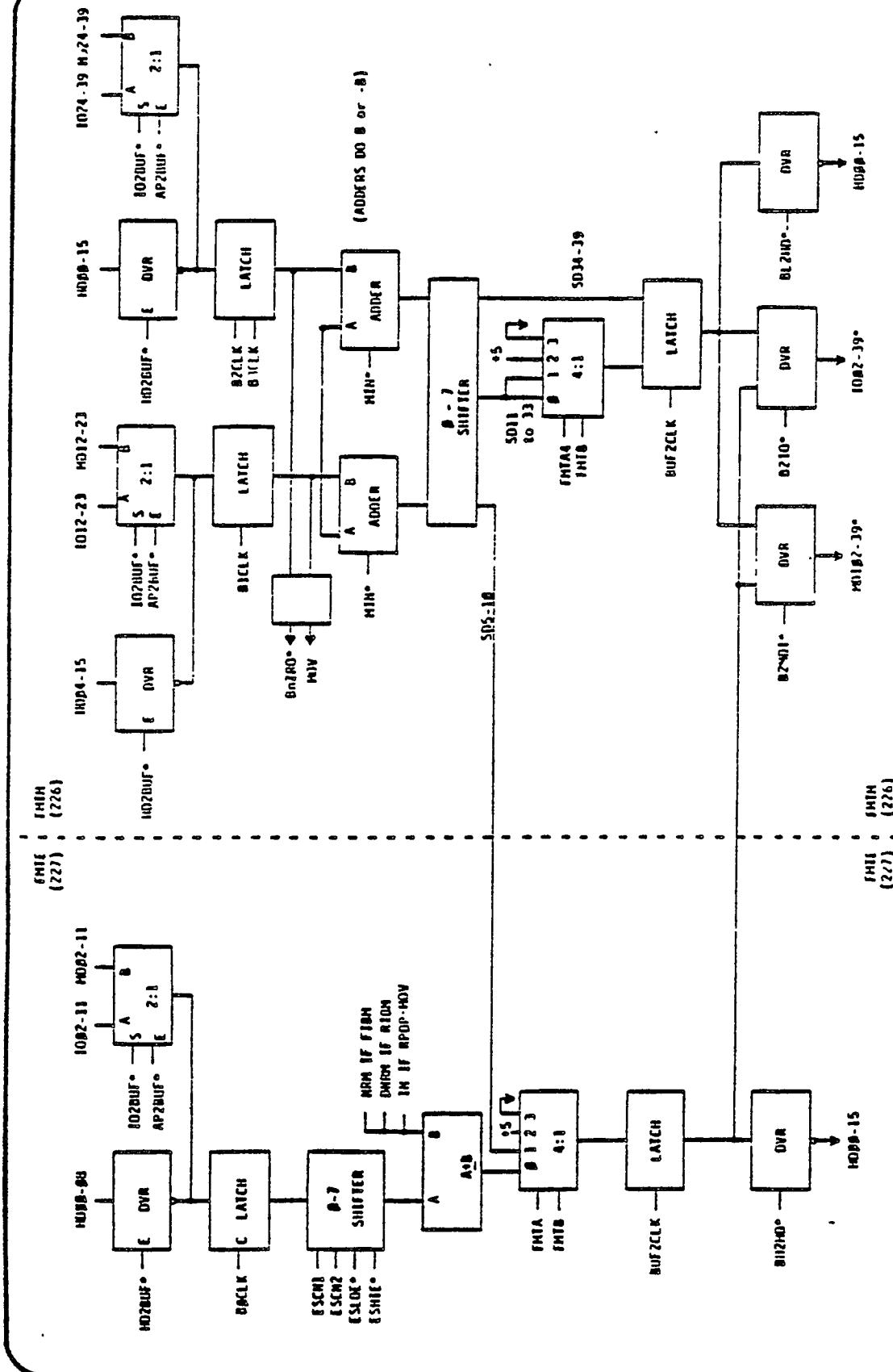
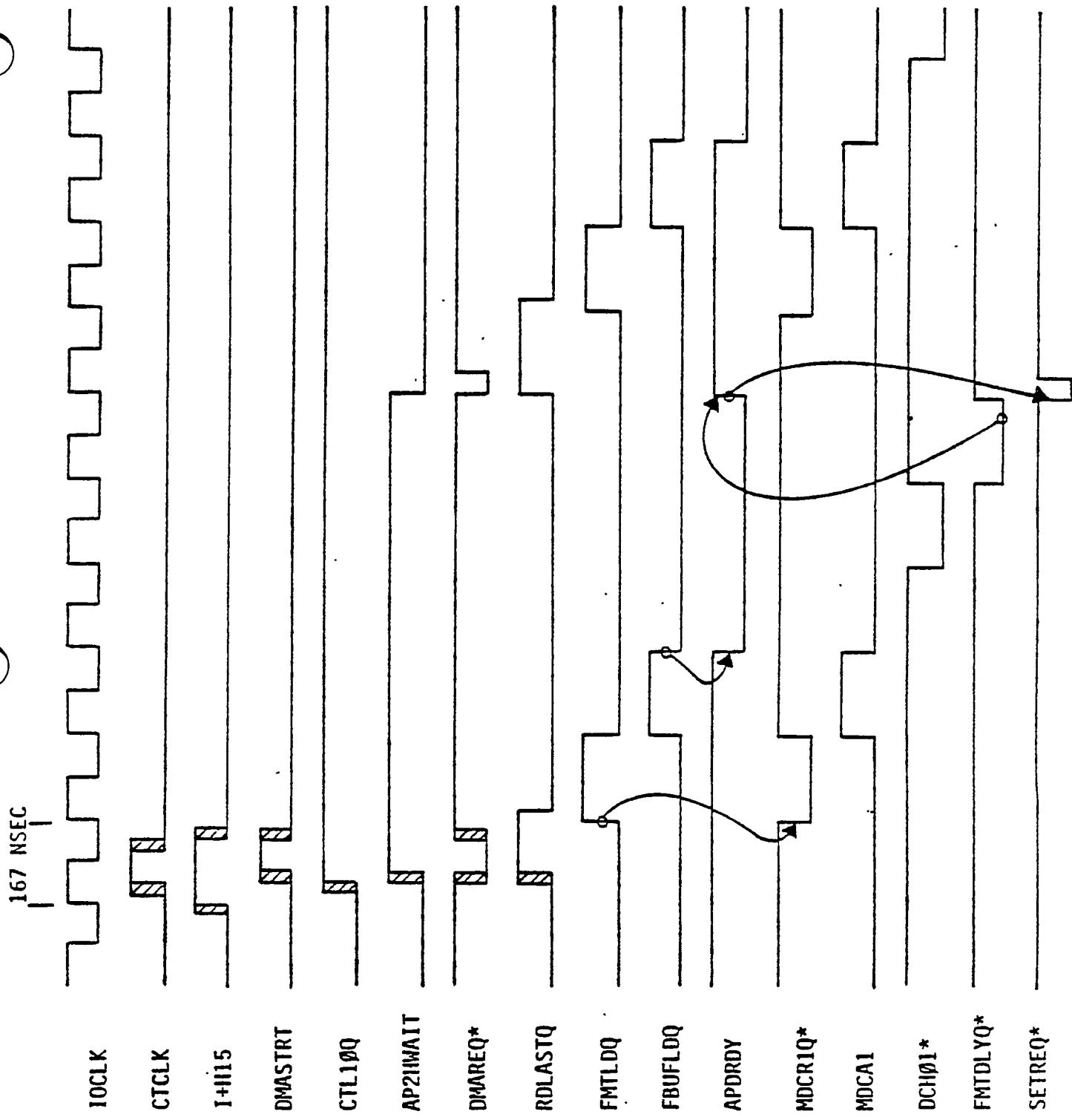


FIGURE FMT2
Formatter Block Diagram

GE MEDICAL SYSTEMS INSTITUTE



DMA to DMA Startup

GE MEDICAL SYSTEMS INSTITUTE

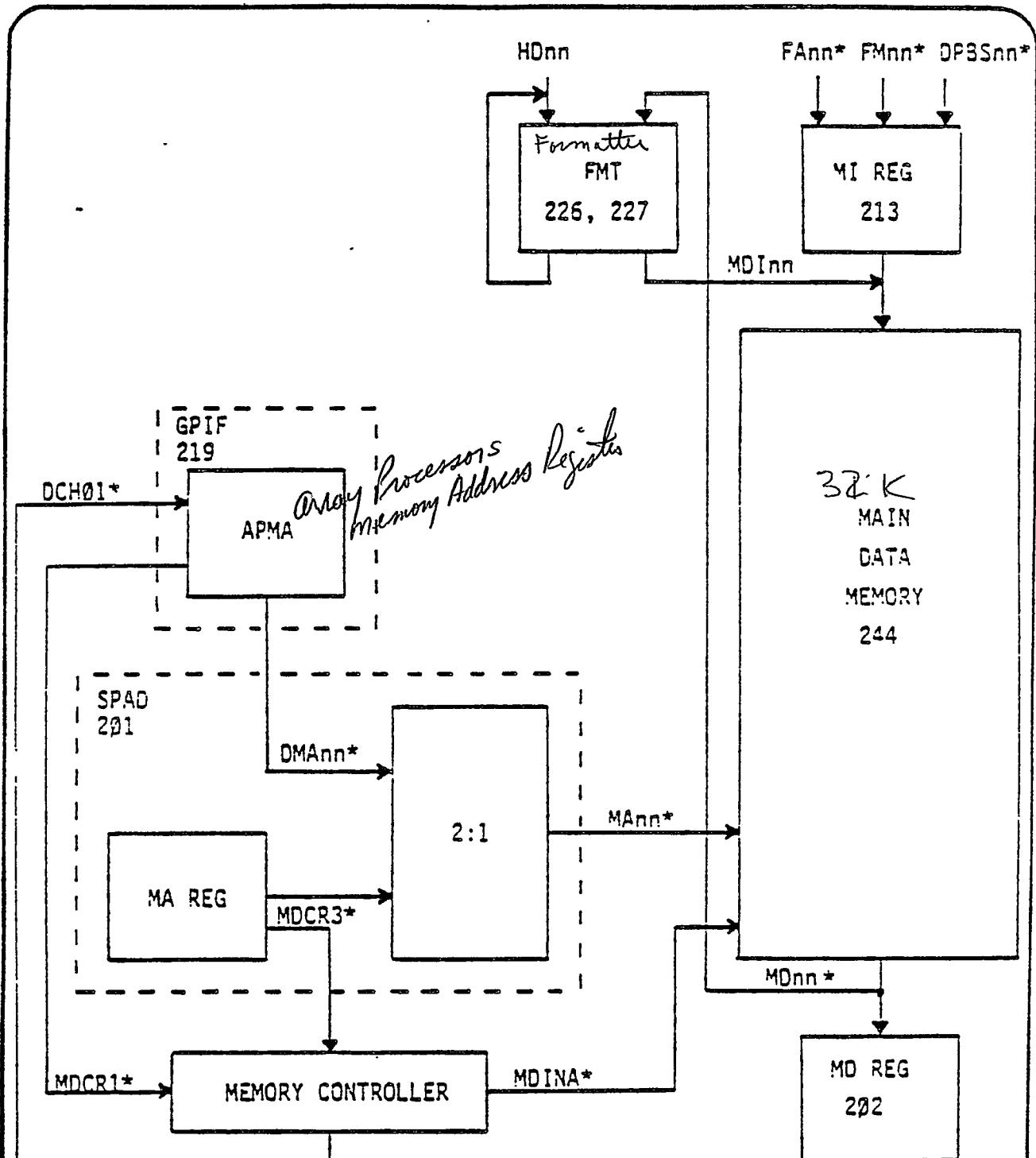


Figure MD1
Main Data System Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

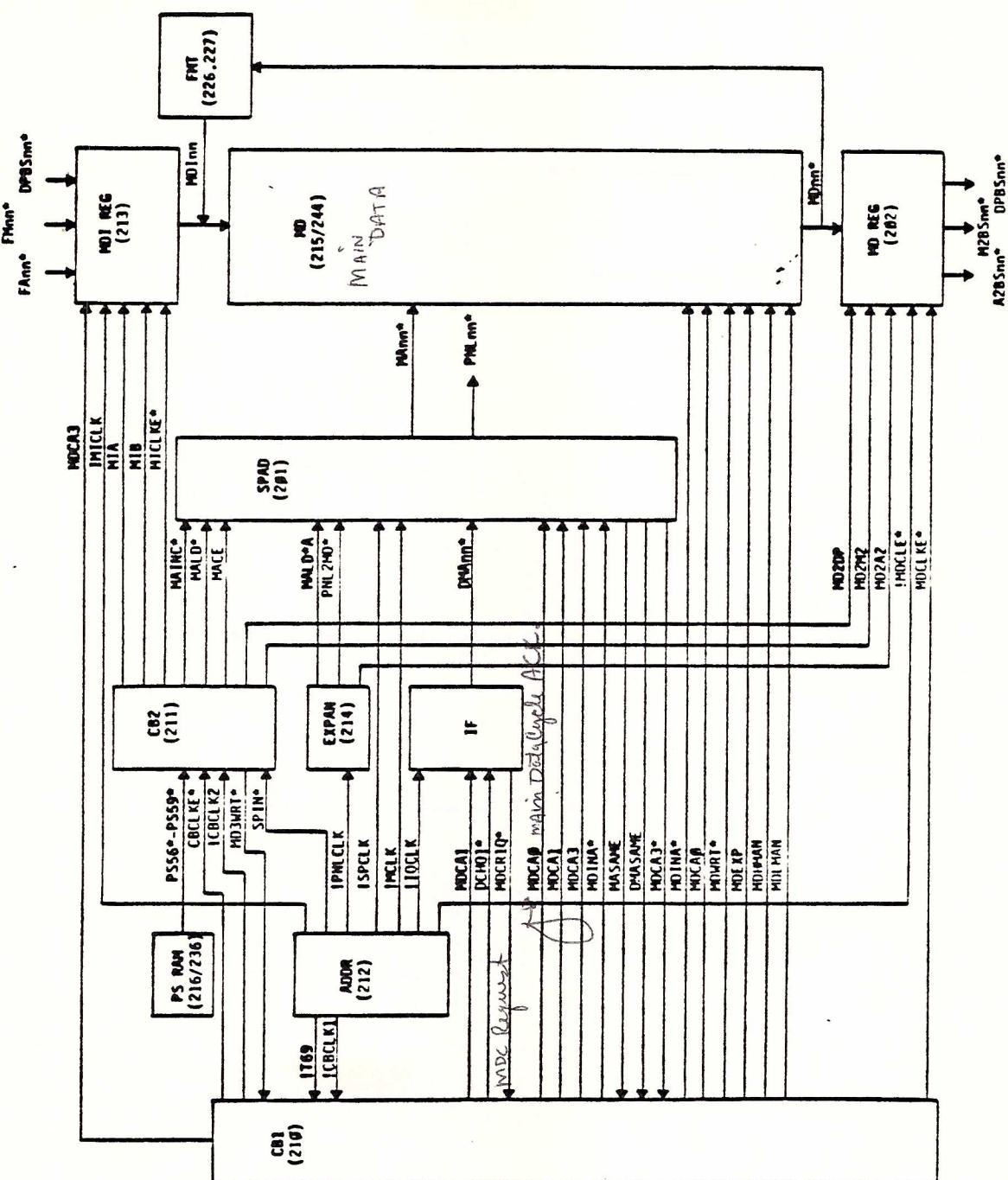


Figure MB2

GE MEDICAL SYSTEMS INSTITUTE

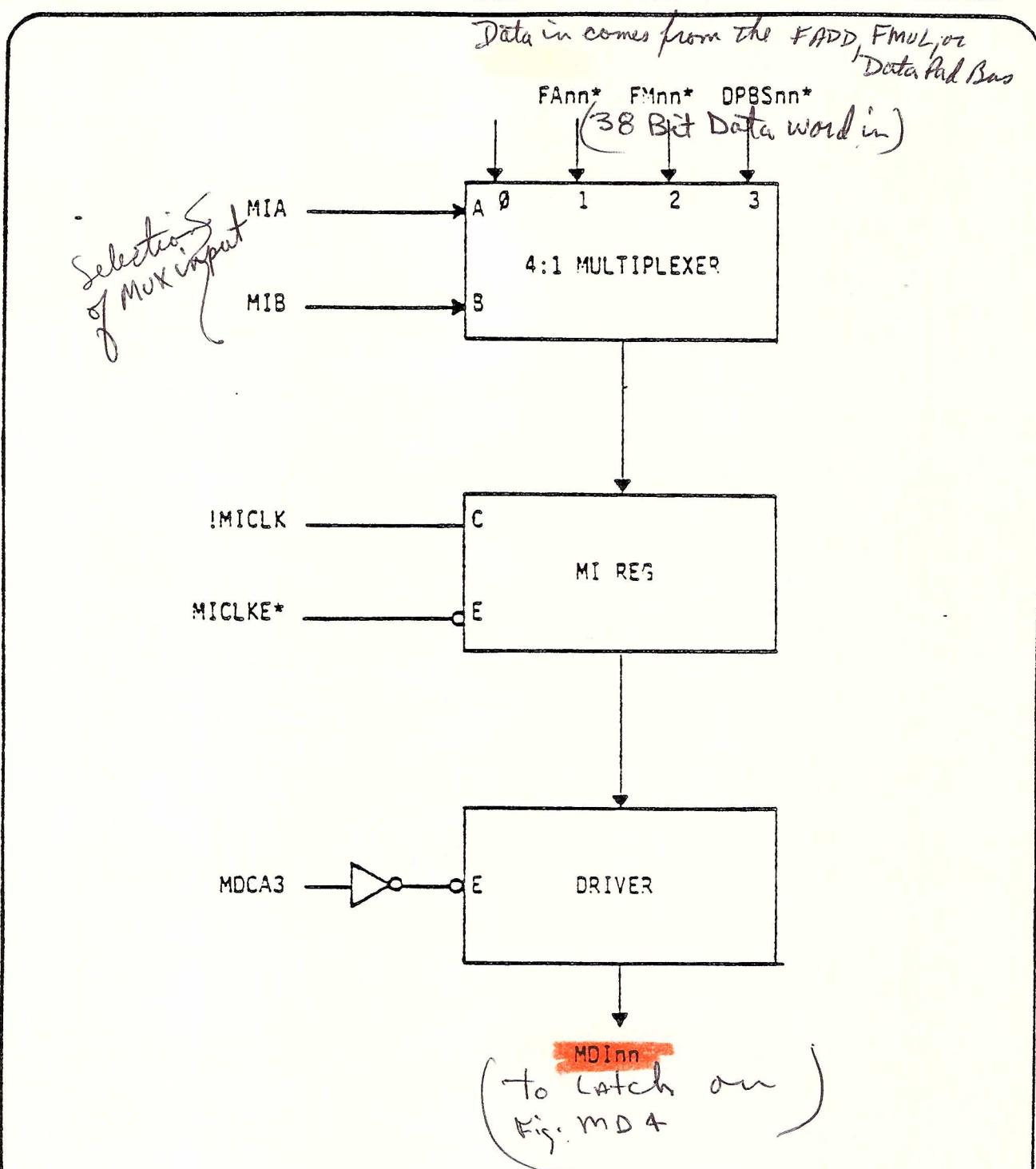
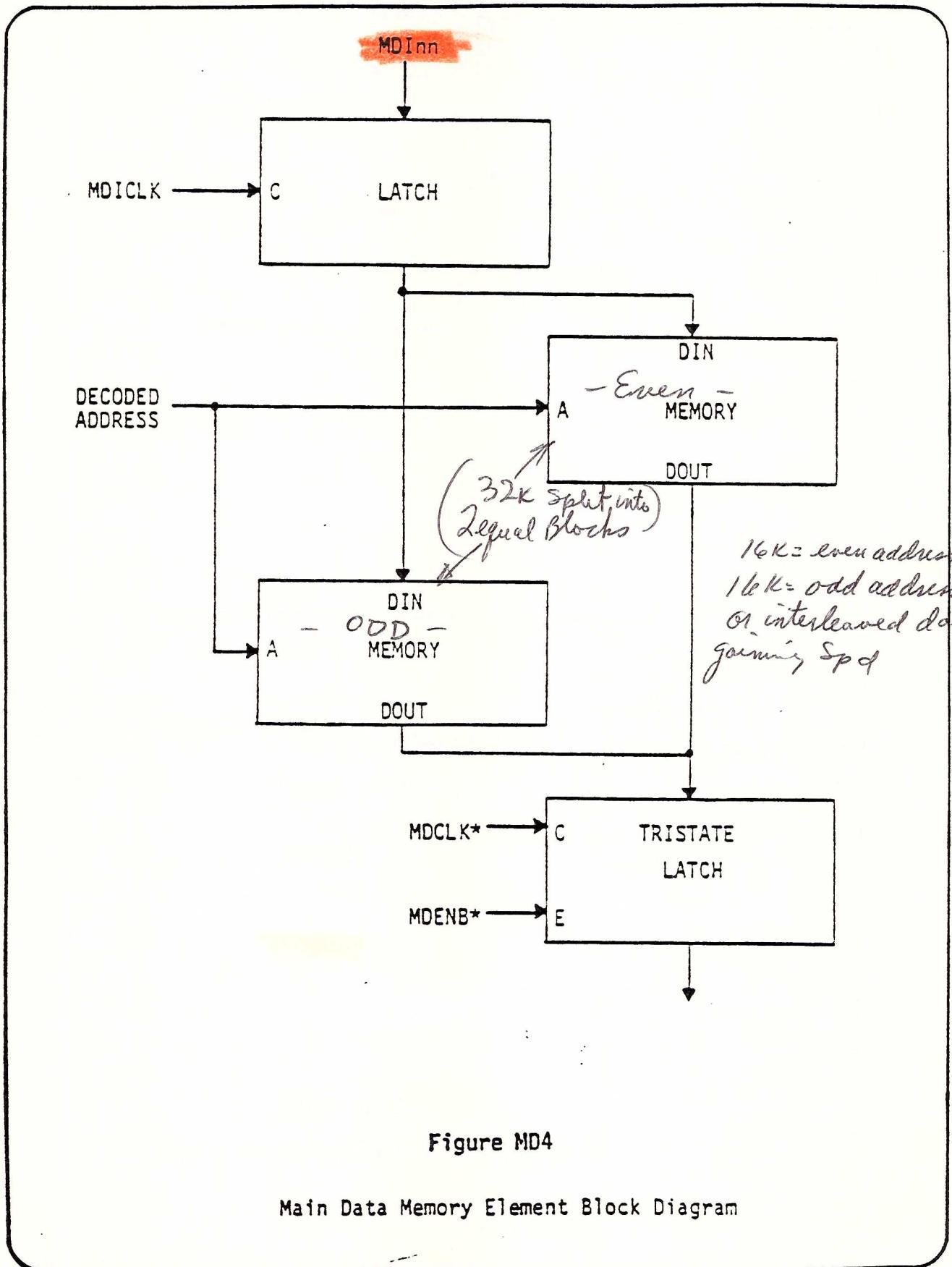


Figure MD3

MI Reg Block Diagram

GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

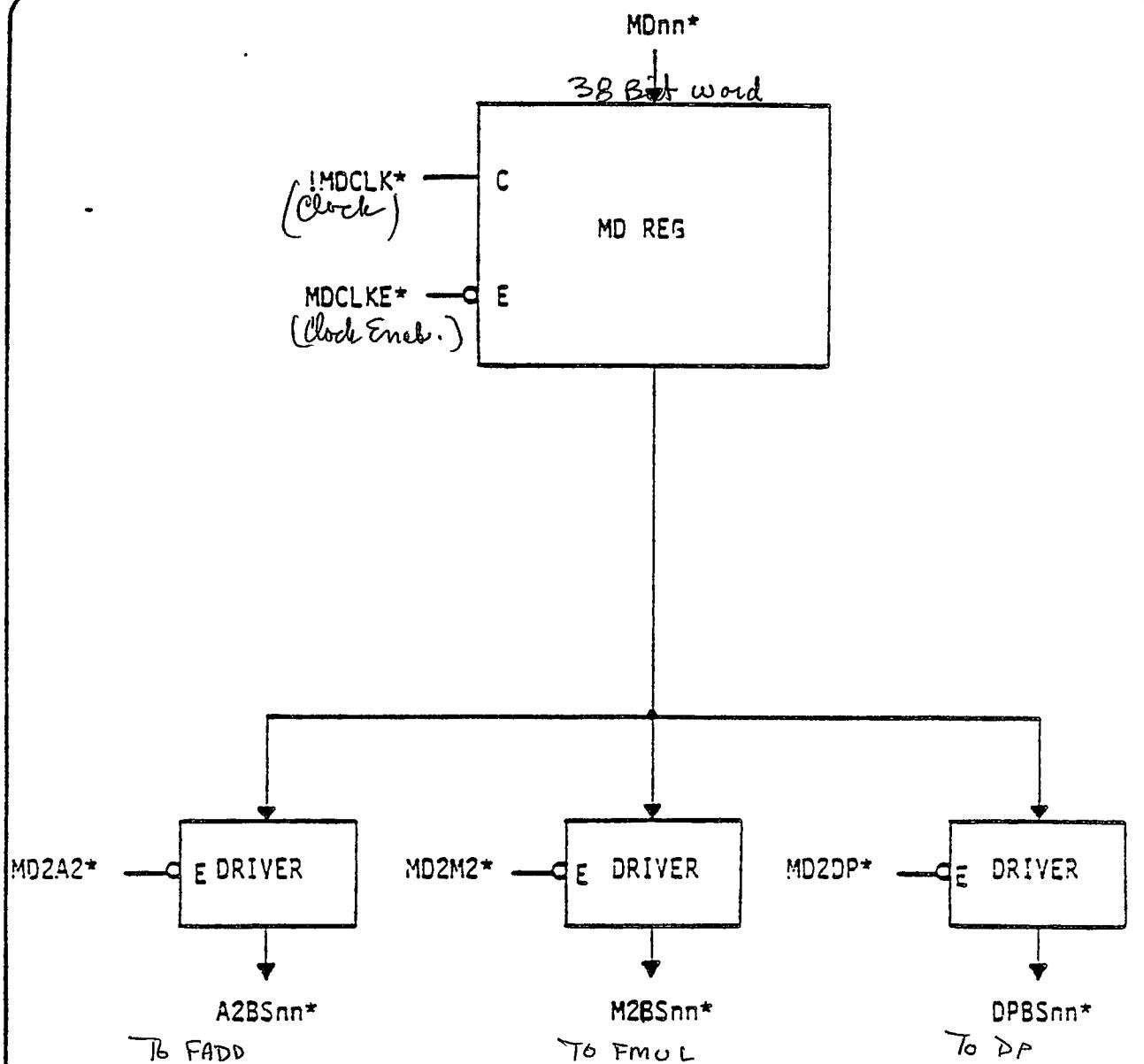


Figure MD5

Main Data Register Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

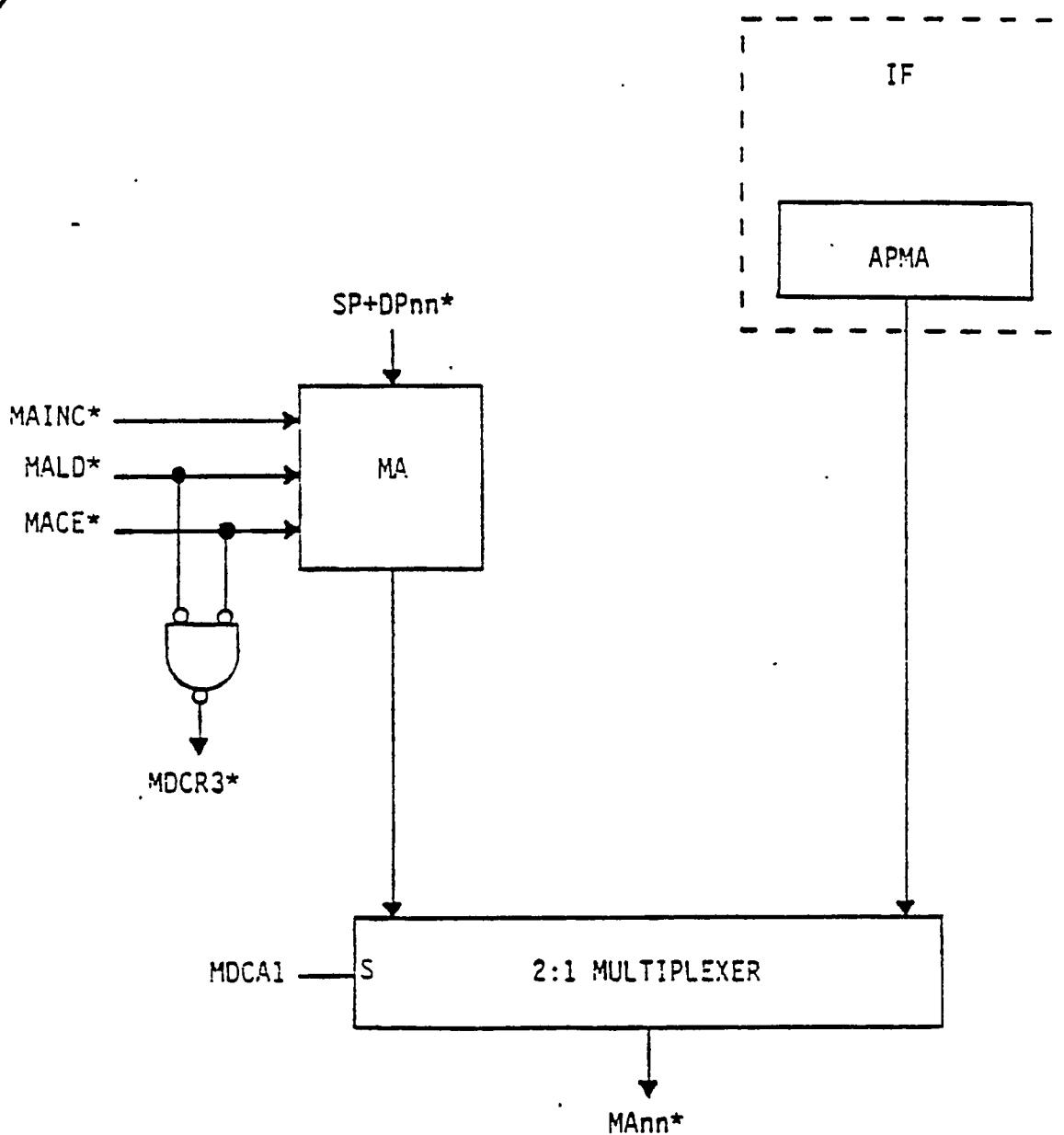


Figure MD6
Memory Address Simplified Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

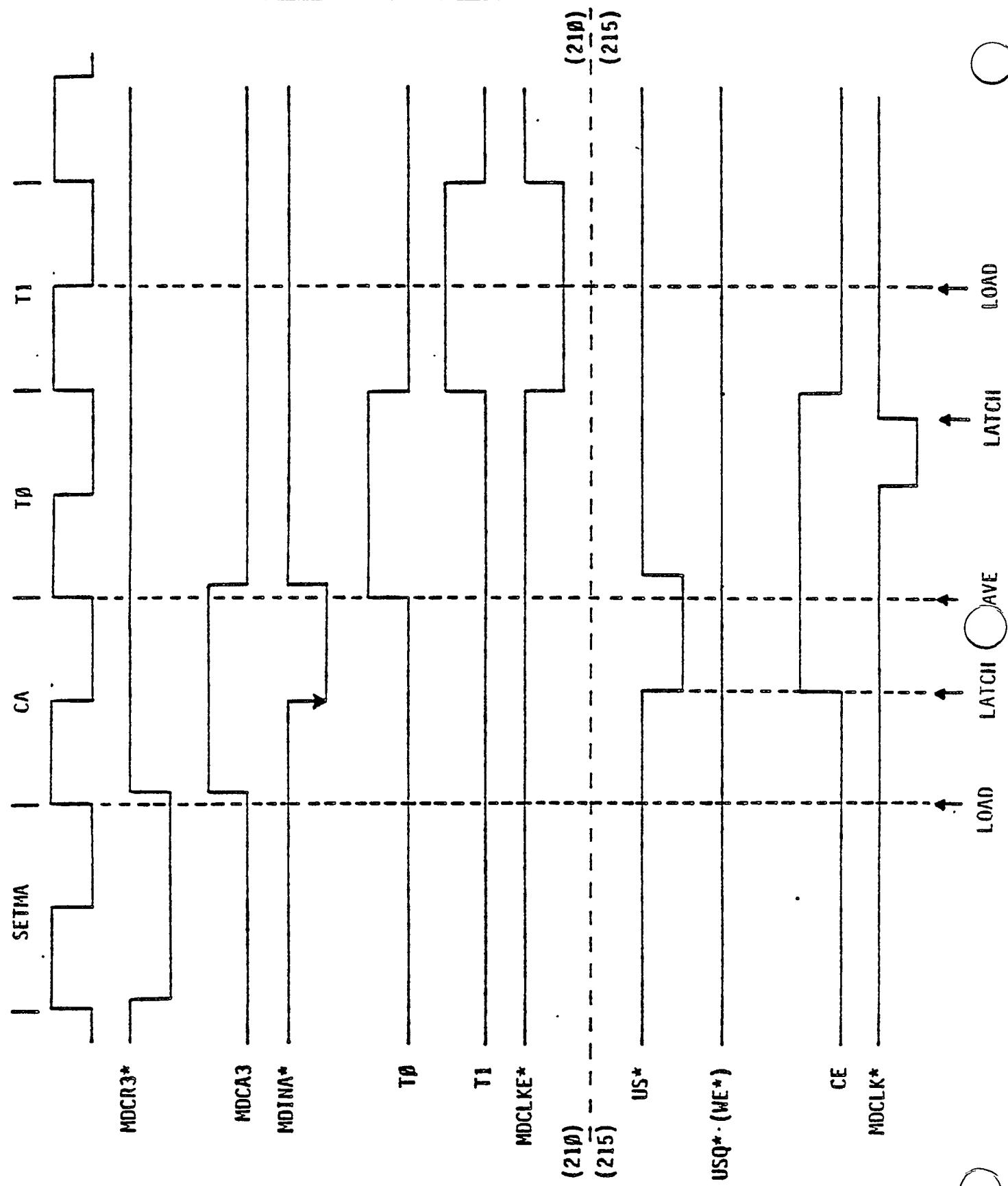
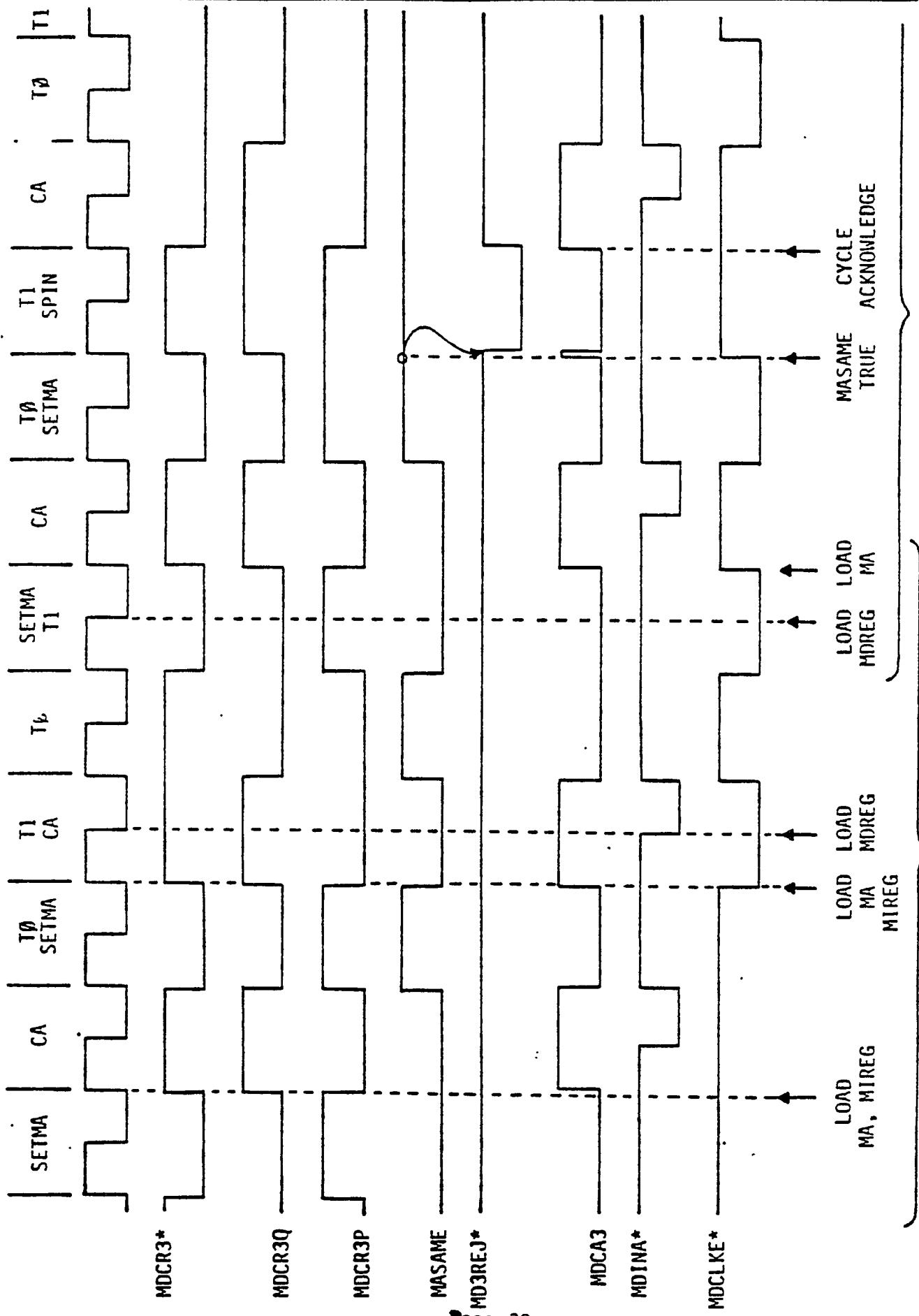


Figure MD7
MD TIMING (ISOLATED CYCLE)

GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

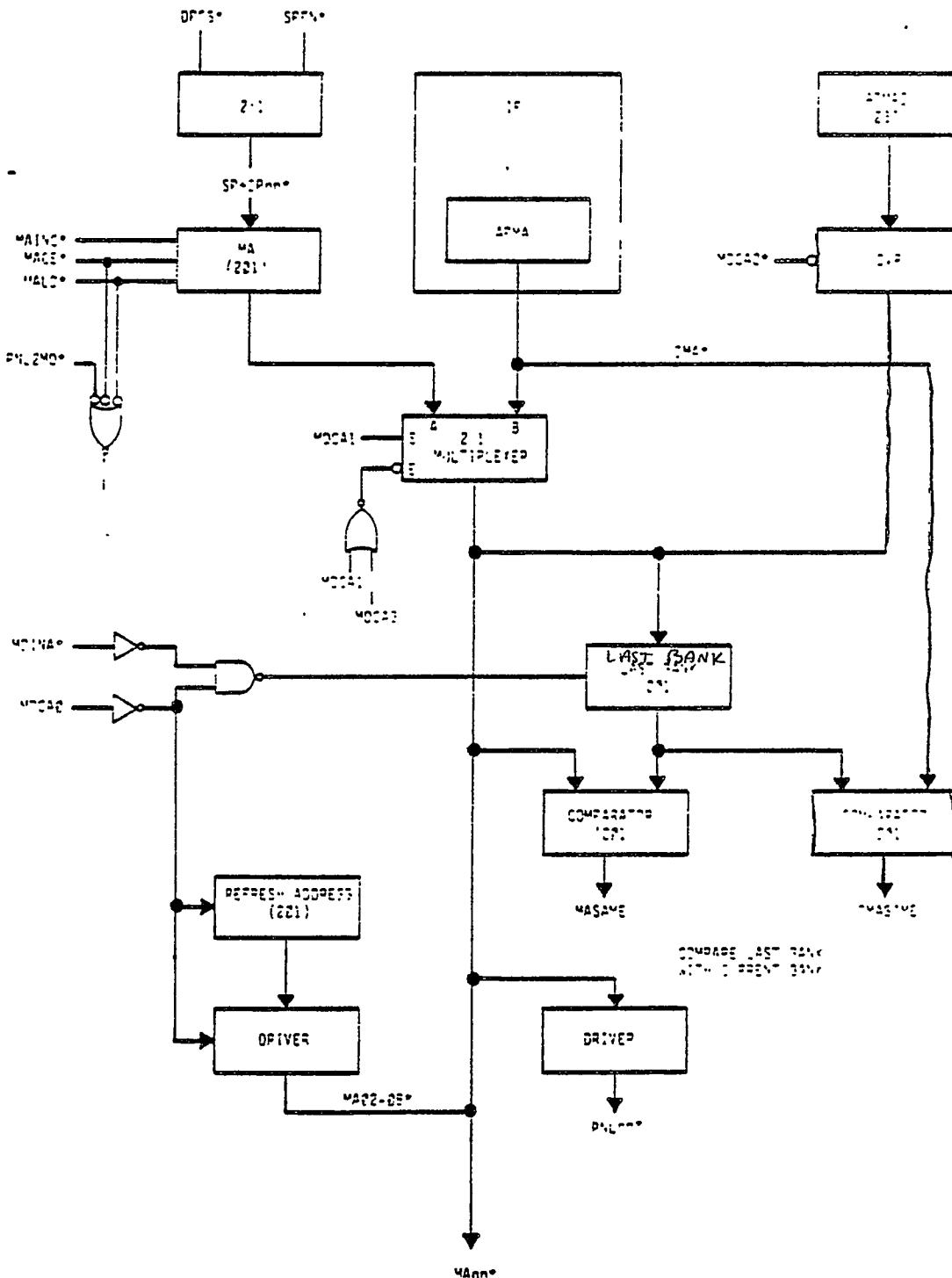
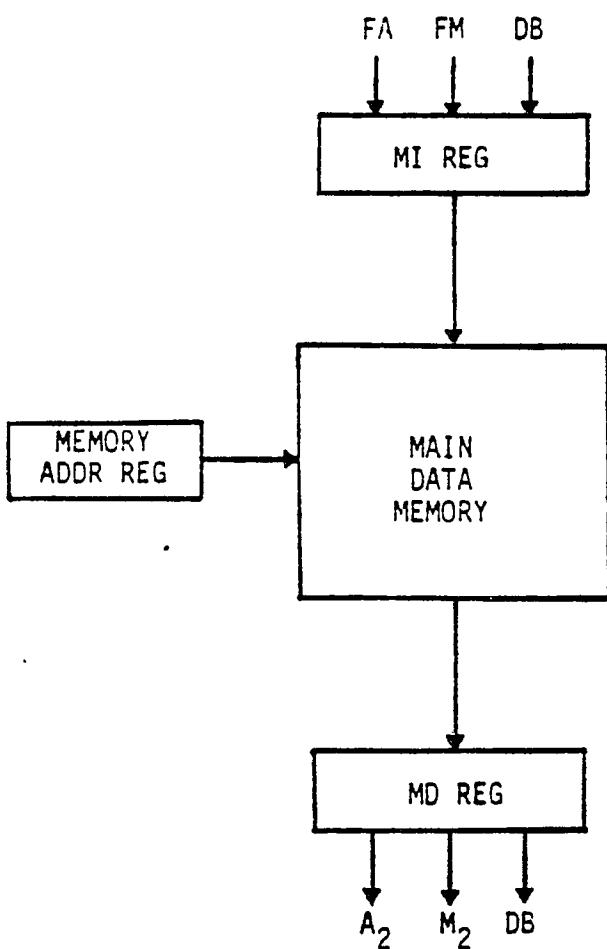


FIGURE MD8
Memory Address Block Diagram

GE MEDICAL SYSTEMS INSTITUTE



MAIN DATA SPECIFICATIONS

38-bits wide
64K words directly addressable
Expandable to 512K words with
Page Select
2-way Interleave

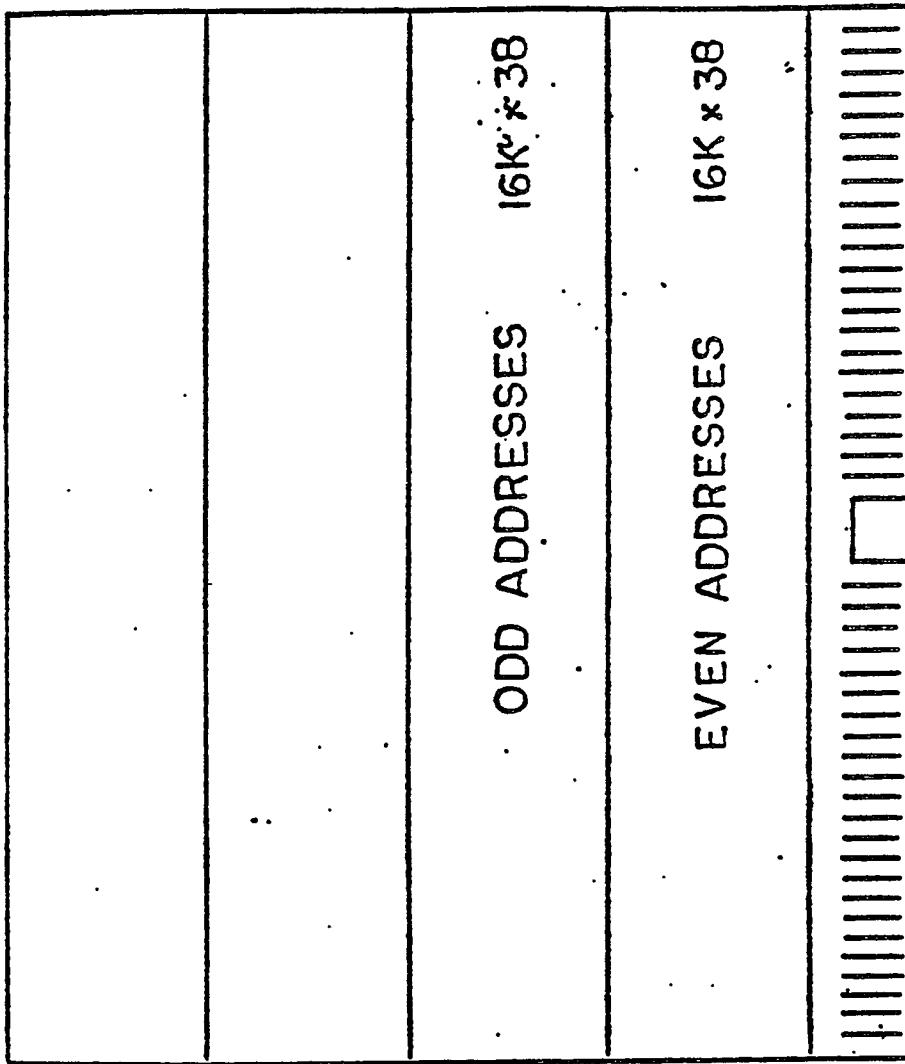
Standard Memory
333 Nsec cycle time

Fast Memory
167 Nsec cycle time

FIGURE MD9

GE MEDICAL SYSTEMS INSTITUTE

COMPONENT SIDE UP



GE MEDICAL SYSTEMS INSTITUTE

BACKPLANE SIGNAL GLOSSARY - Continued

!WR1L*	Write pulse for DPL
!WRTR*	Write pulse for DPR
"GND	Extra Grounds not included in the standard set provided by the motherboard.
A1CLKD*	A1 Clock Data (low true) causes A1CLK
A1EBS02* to A1EBS11*	A1 Exponent Bus
A1MBS00* to A1MBS27*	A1 Mantissa Bus
A2CLKD*	A2 Clock Data causes A2CLK
A2EBS02* to A2EBS11*	A2 Exponent Bus
A2MBS00* to A2MBS27*	A2 Mantissa Bus
ABORT*	Internal System Reset Line
B0CLK	Byte 0 Clock to FMT Bd.
B1CLK	Byte 1 Clock to FMT Bd.
B2CLK	Byte 2 Clock to FMT Bd.
B3CLK	Byte 3 Clock to FMT Bd.
B2IO	FMT Buffer to I/O Bus Enable
BH2ED	FMT Buffer High to Host Data Enable
BL2ED	FMT Buffer Low to Host Data Enable
BS2A1	A1BS to A1 input select line
BS2A2	A2BS to A2 input select line
BS2M1	M1BS to M1 select line
BS2M2	M2BS to M2 select line
BUF2CLK	FMT BUFFER #2 Clock

GE MEDICAL SYSTEMS INSTITUTE

BACKPLANE SIGNAL GLOSSARY

!ALCLK	Clock for A1 Register of FA
!A2CLK	Clock for A2 Register of FA
!CBCLK1	Clock for Bd. 210, CB-1
!CBCLK2	Clock for Bd. 211, BC-2
!DPLCLK	Clock for Bd. 200L DPAD
!DPLCLK*	Inverted clock for Bd. 200L DPAD
!DPRCLK	Clock for Bd. 200R DPAD
!DPRCLK*	Inverted clock for Bd. 200R DPAD
!FACLK2	Clock for Bd. 204, FADD2
!FACLK3	Clock for Bd. 205, FADD3
!FMCLKA	Clock for Bd. 206, FMULA
!FMCLKB	Clock for Bd. 207, FMULB
!FMCLKC	Clock for Bd. 208, FMULC
!IOCLK	Clock for Interface Bd
!M1CLK	Clock for M1 Register of FM
!M2CLK	Clock for M2 Register of FM
!MCLK	Clock for Main Data Memory Cards
!MDCLK*	Inverted Clock for MDREG Bd. <u>202</u>
!MICLK	Clock for MIREG Bd. <u>213</u>
!PNLCLK	Clock for Panel Logic on EXPAN, Bd. 214
!SPCLK	Clock for SPAD, Bd. 201
!T69	Clock delayed by 69ns for MD Timing Bd. 210
!TMCLK	Clock for TMREG Bd. <u>209</u>
!WRT*	Low true write pulse used to write PS, and the Subrouting Return Stack

GE MEDICAL SYSTEMS INSTITUTE

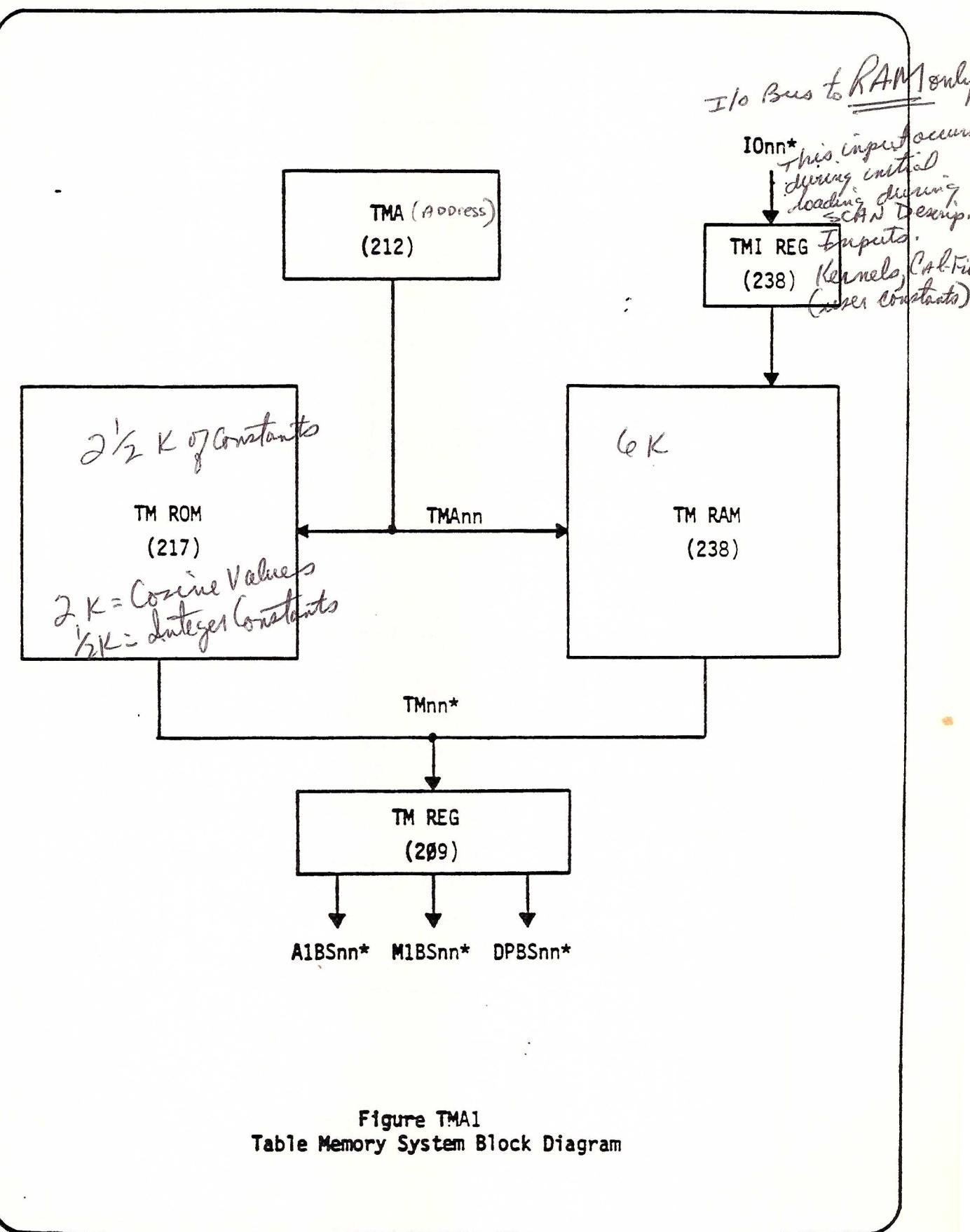
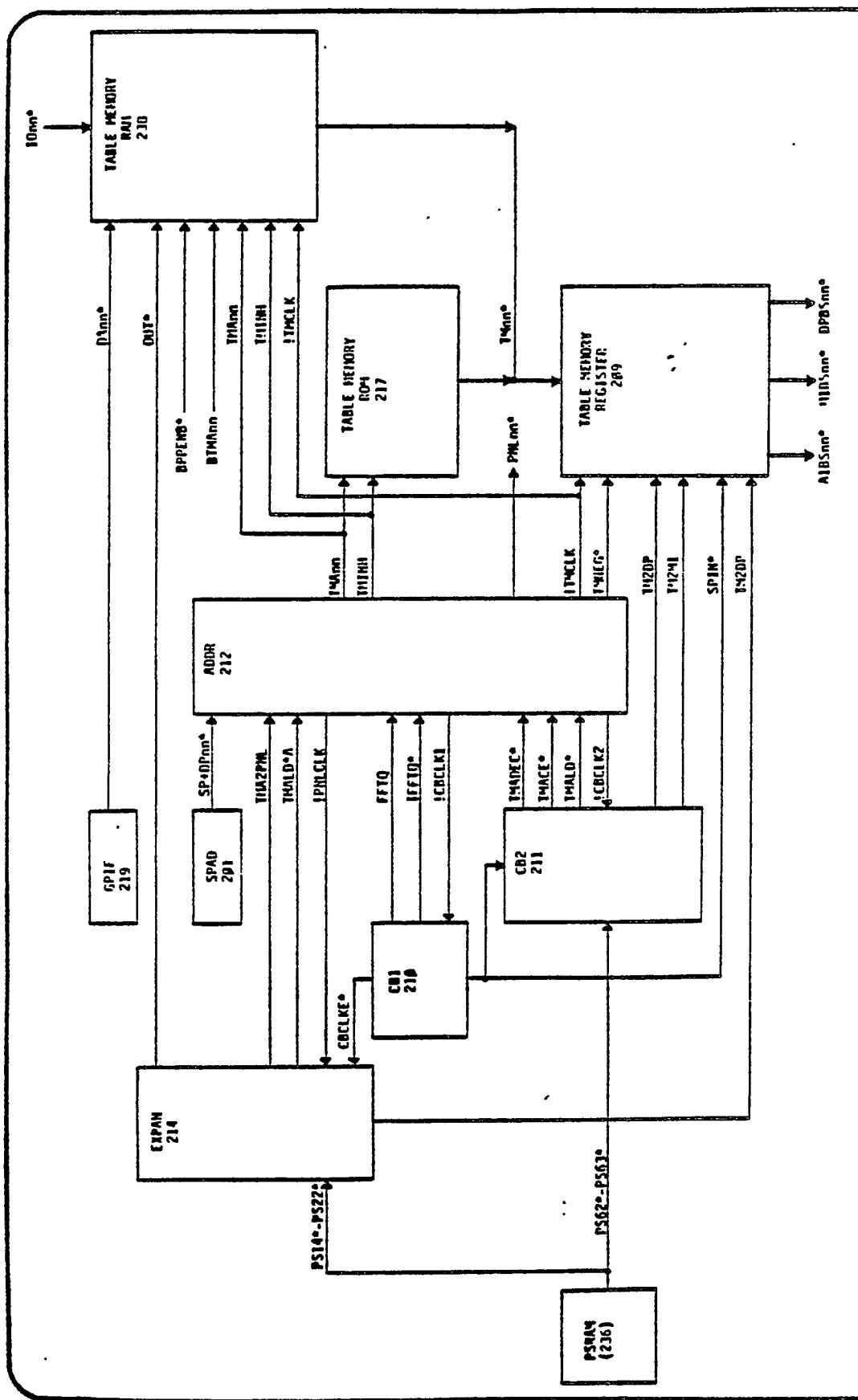


Figure TMA1
Table Memory System Block Diagram

GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

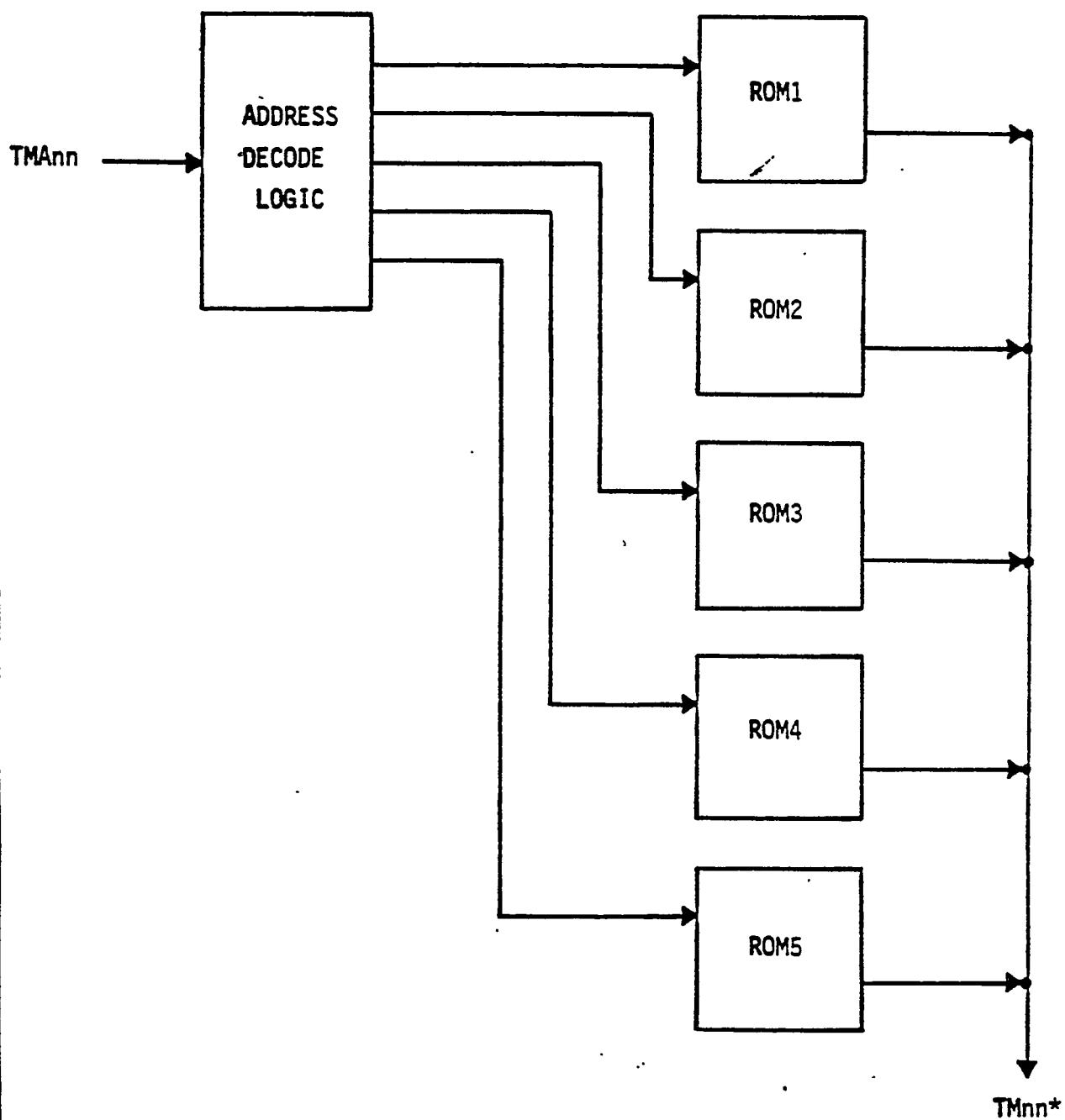


Figure TMA3
TM ROM Memory Elements

GE MEDICAL SYSTEMS INSTITUTE

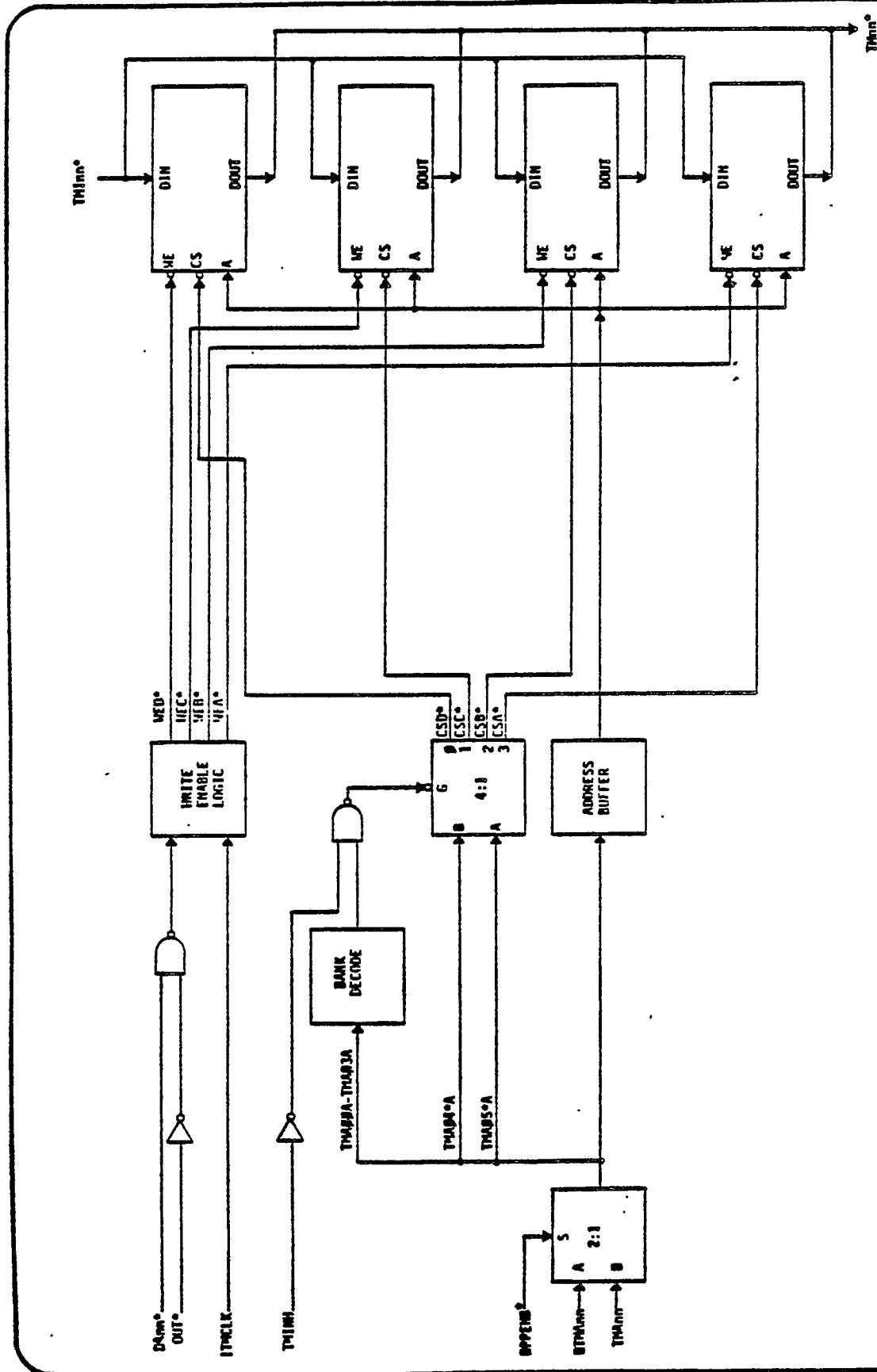


Figure TMA4
TMRAM Memory Elements

GE MEDICAL SYSTEMS INSTITUTE

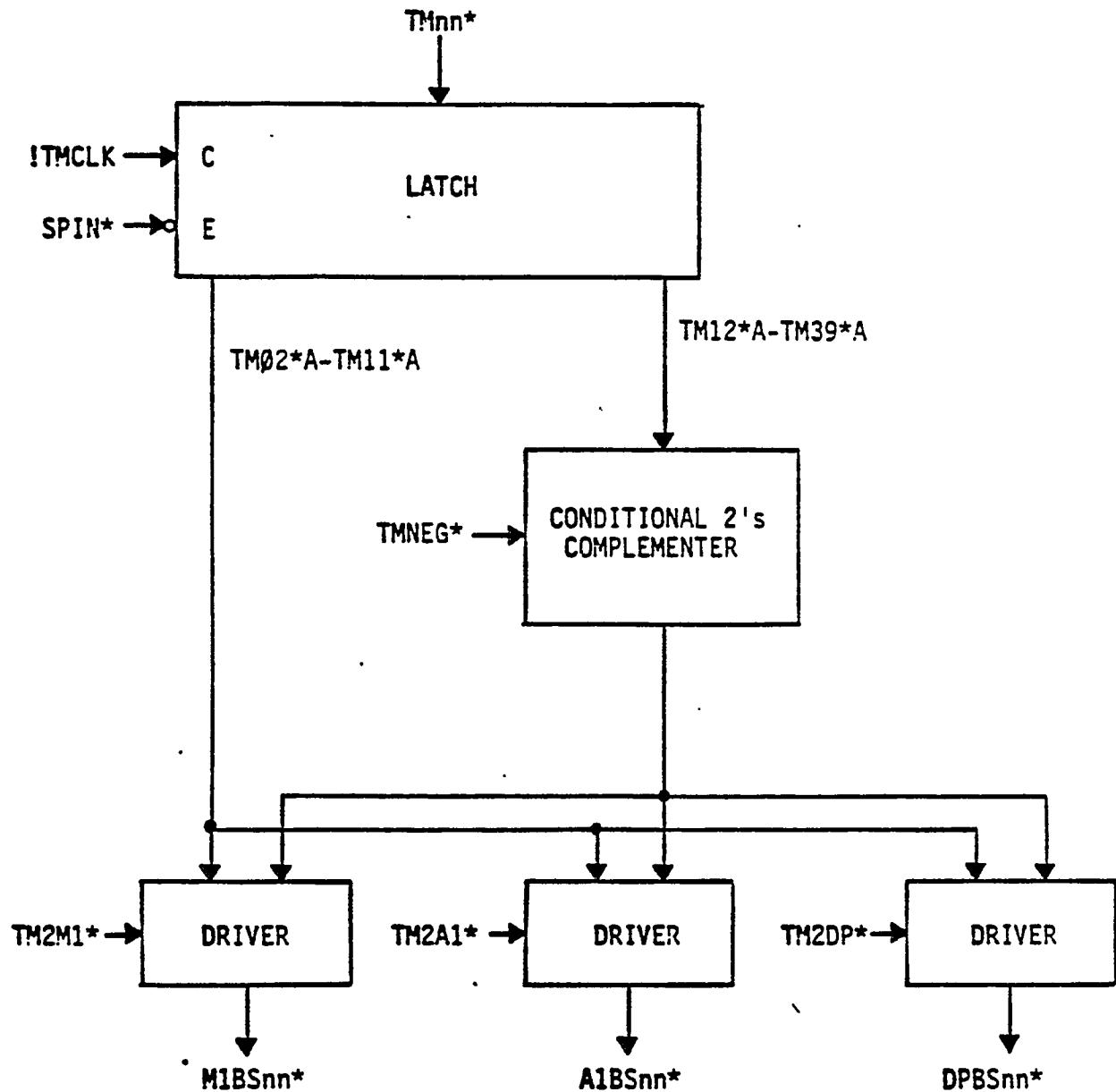


Figure TMA5
Table Memory Register Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

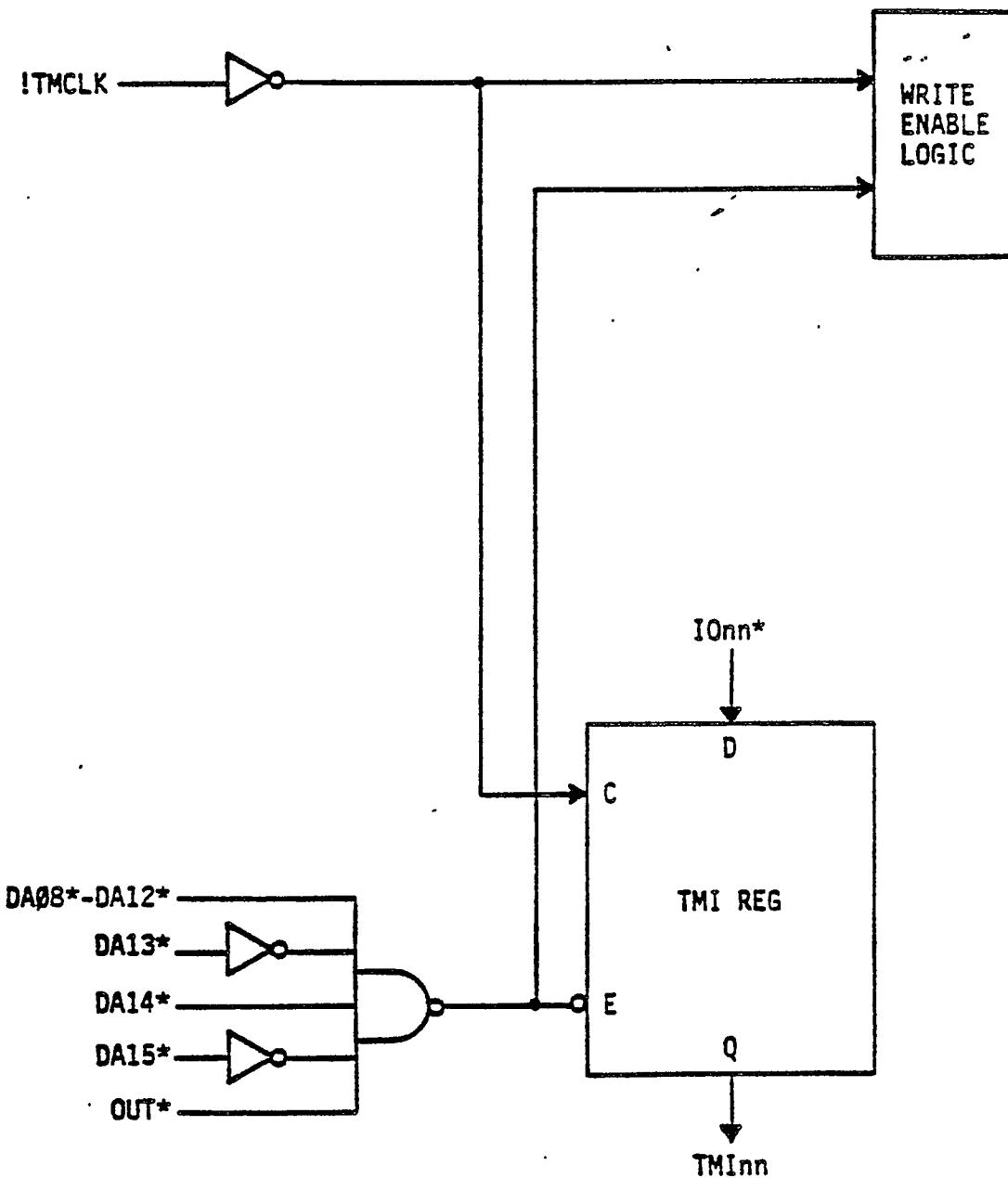


Figure TMA6
Table Memory Input Register Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

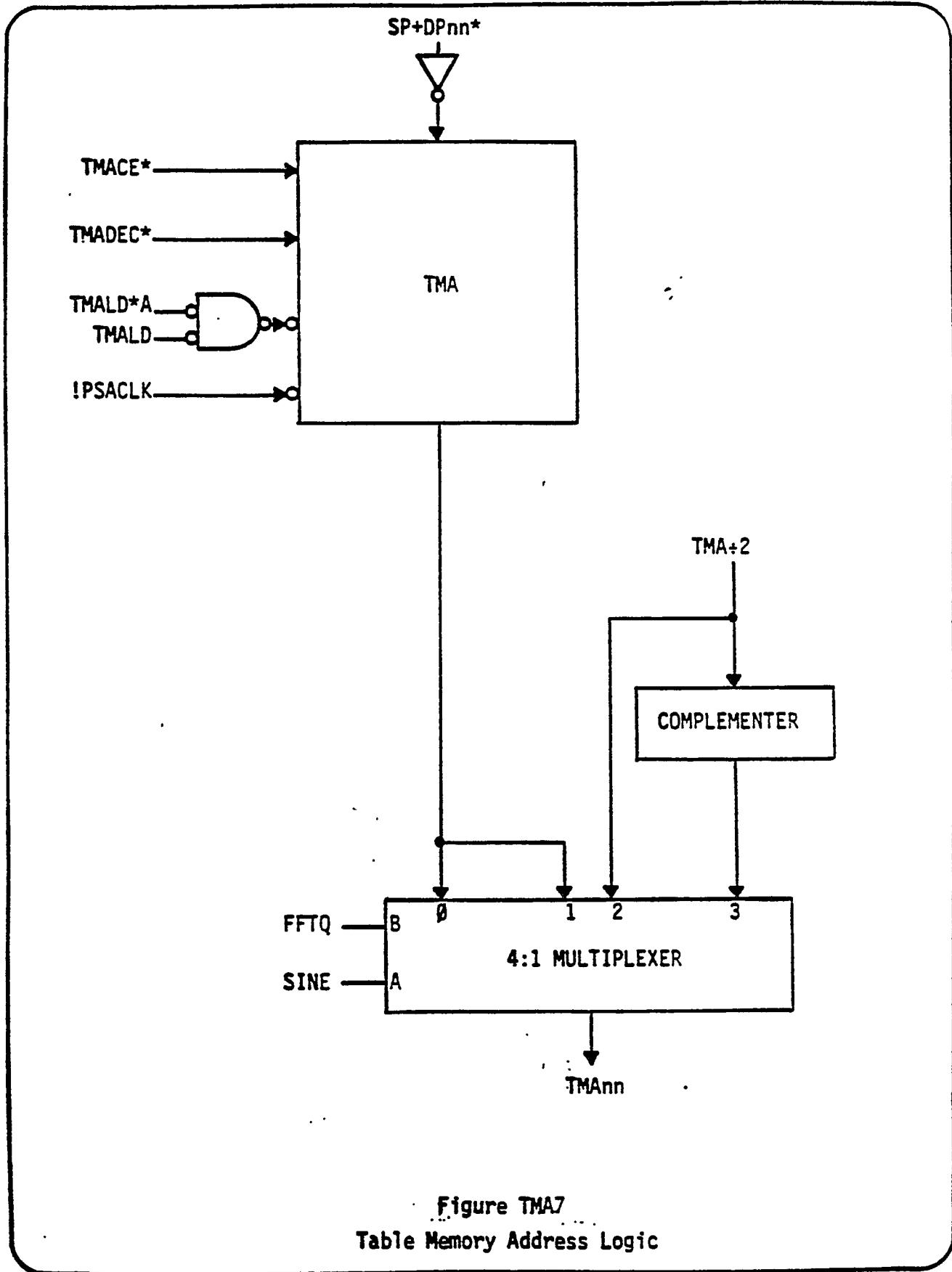


Figure TMA7
Table Memory Address Logic

GE MEDICAL SYSTEMS INSTITUTE

238 BOARD FULL BD

COMPONENT SIDE UP

EVEN BITS TM₀₂ - TM₃₈

ADDRESS
(6000 - 7777)

ODD BITS TM₀₃ - TM₃₉

ADDRESS
(4000 - 5777)

EVEN BITS TM₀₂ - TM₃₈

ADDRESS
(2000 - 3777)

ODD BITS TM₀₃ - TM₃₉

ADDRESS
(0 - 1777₈)



EXPONENT	HIGH MANTISSA	LOW MANTISSA
02	11	12
	23	24
		39

TABLE MEMORY WORD FORMAT

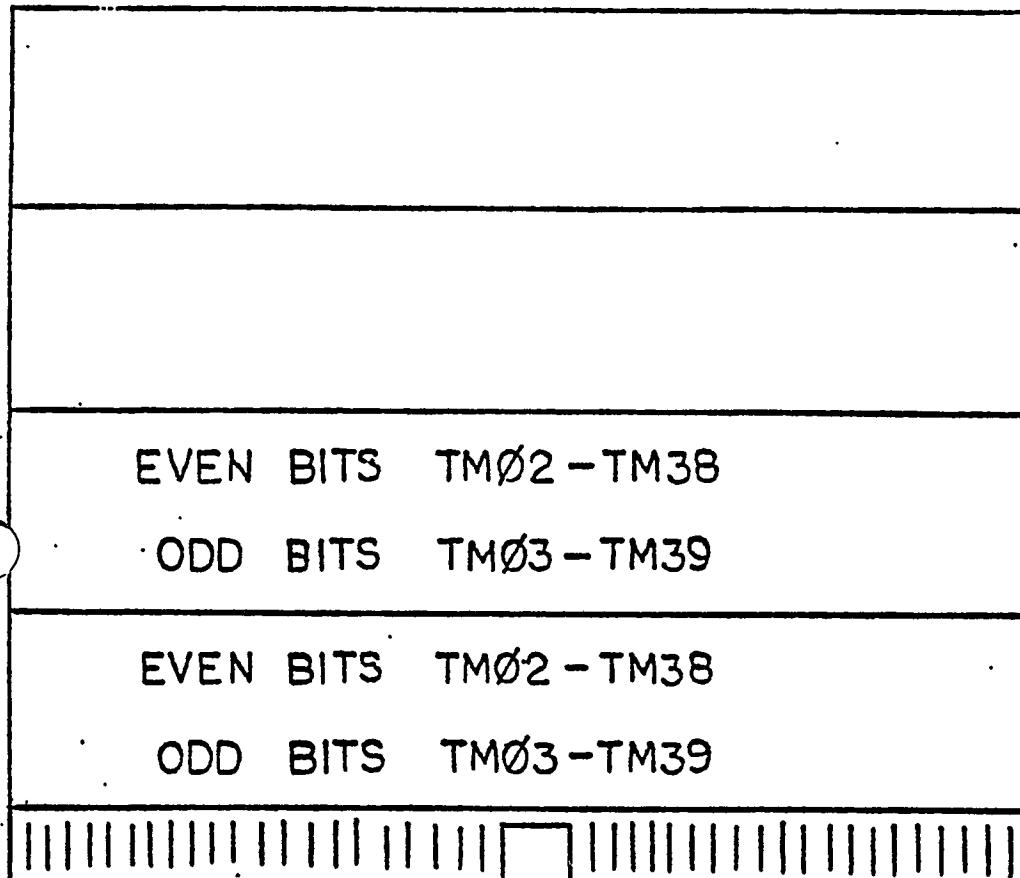
0 0 1 7 7 7 | 0 0 7 7 7 7 | 1 7 7 7 7 7

ERROR MESSAGE DATA FORMAT

GE MEDICAL SYSTEMS INSTITUTE

238 BOARD HALF BD

COMPONENT SIDE UP



EXONENT	HIGH MANTISSA	LOW MANTISSA	
02	11	12	23 24
001777	007777	177777	39

TABLE MEMORY WORD FORMAT

ERROR MESSAGE DATA FORMAT

GE MEDICAL SYSTEMS INSTITUTE

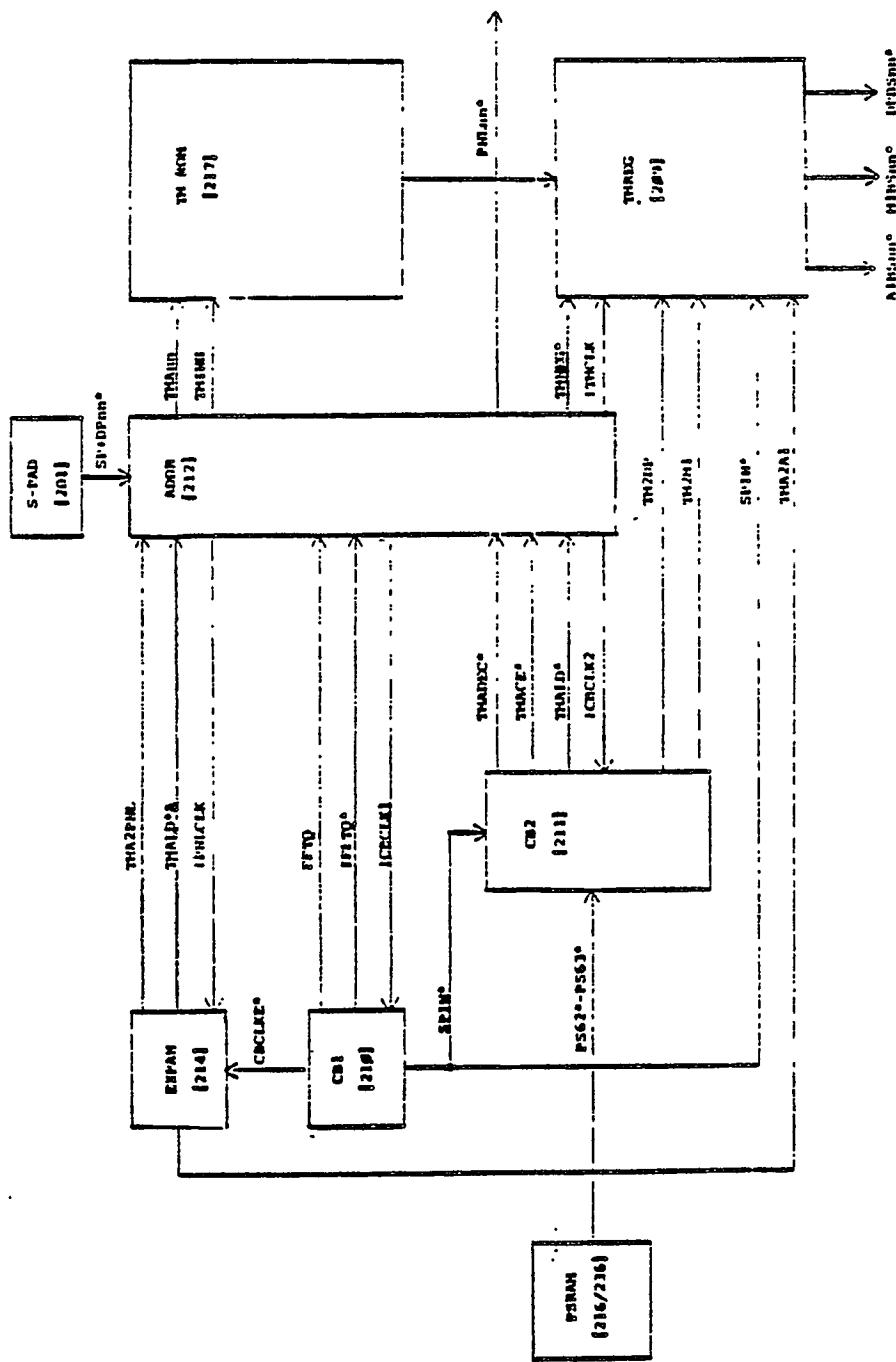


Figure 5-13 Table Memory ROM Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

217 BOARD
TM ROM

COMPONENT SIDE UP

0-511 (0-777₈)

512-1023 (1000-1777₈)

1024-1535 (2000-2777₈)

1536-2047 (3000-3777₈)

2048-2559 (4000-4777₈)



GE MEDICAL SYSTEMS INSTITUTE

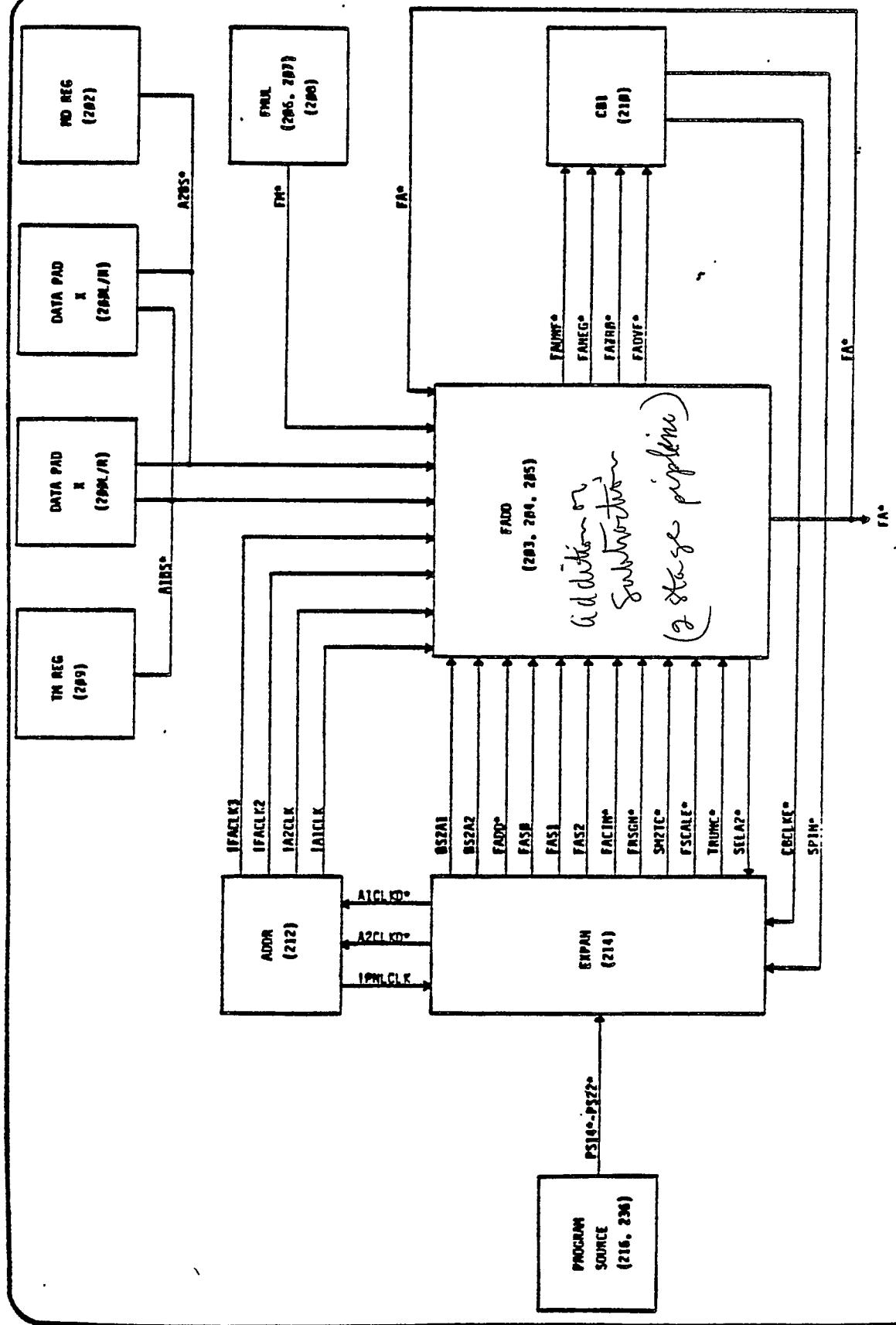


Figure FADDI
Floating Adder System

GE MEDICAL SYSTEMS INSTITUTE

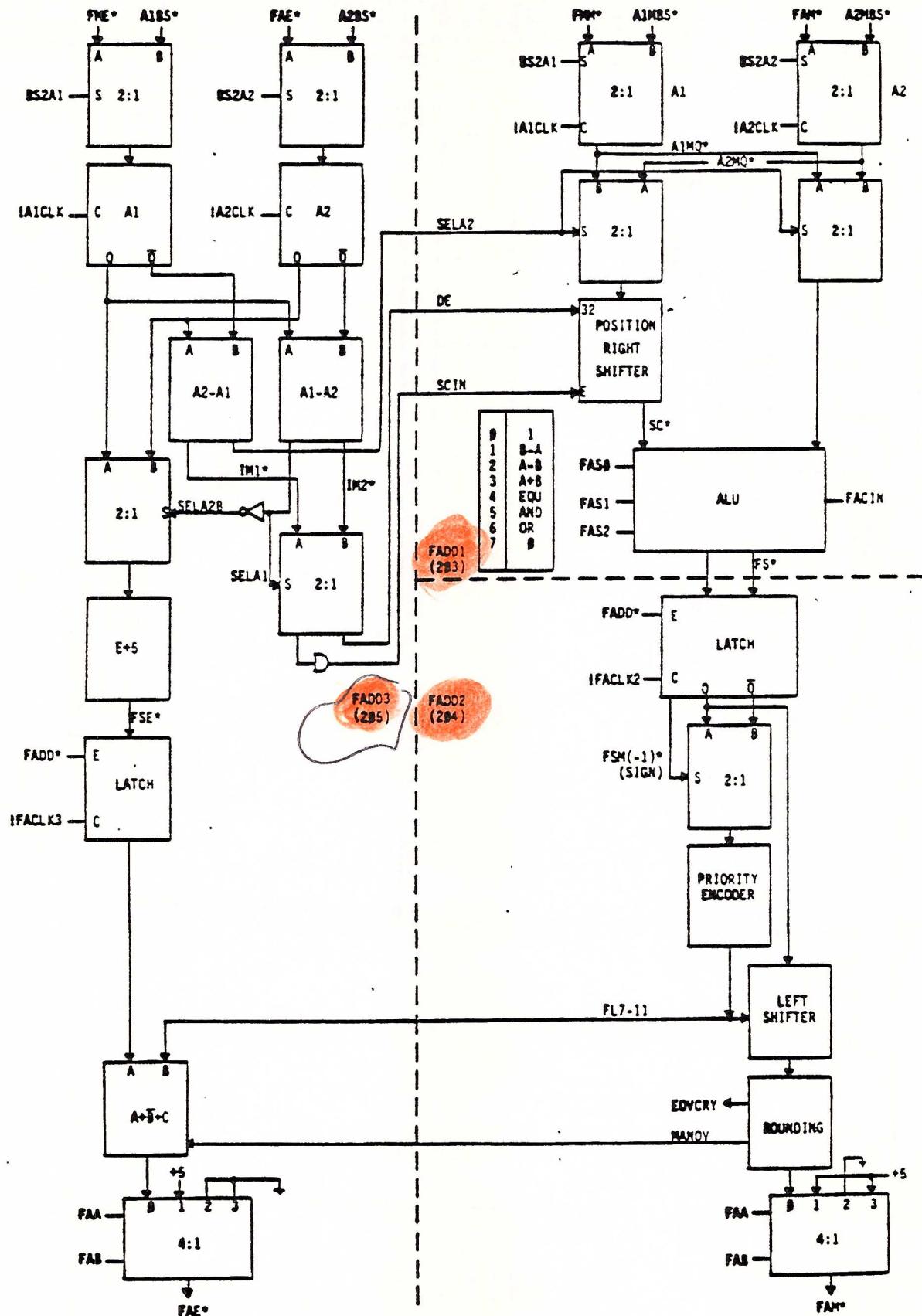


Figure FADD2
Floating Adder Block Diagram
Page 57

GE MEDICAL SYSTEMS INSTITUTE

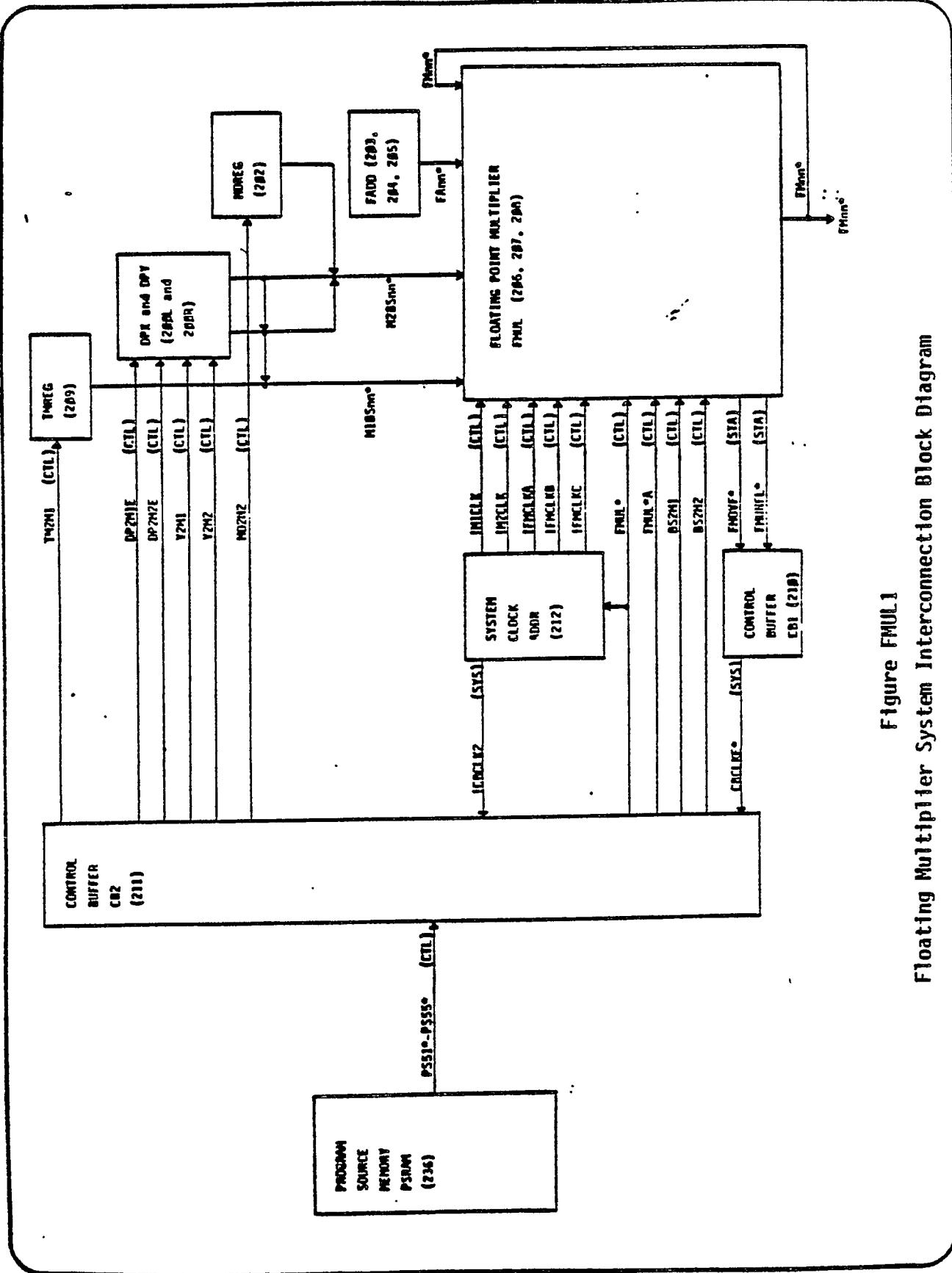
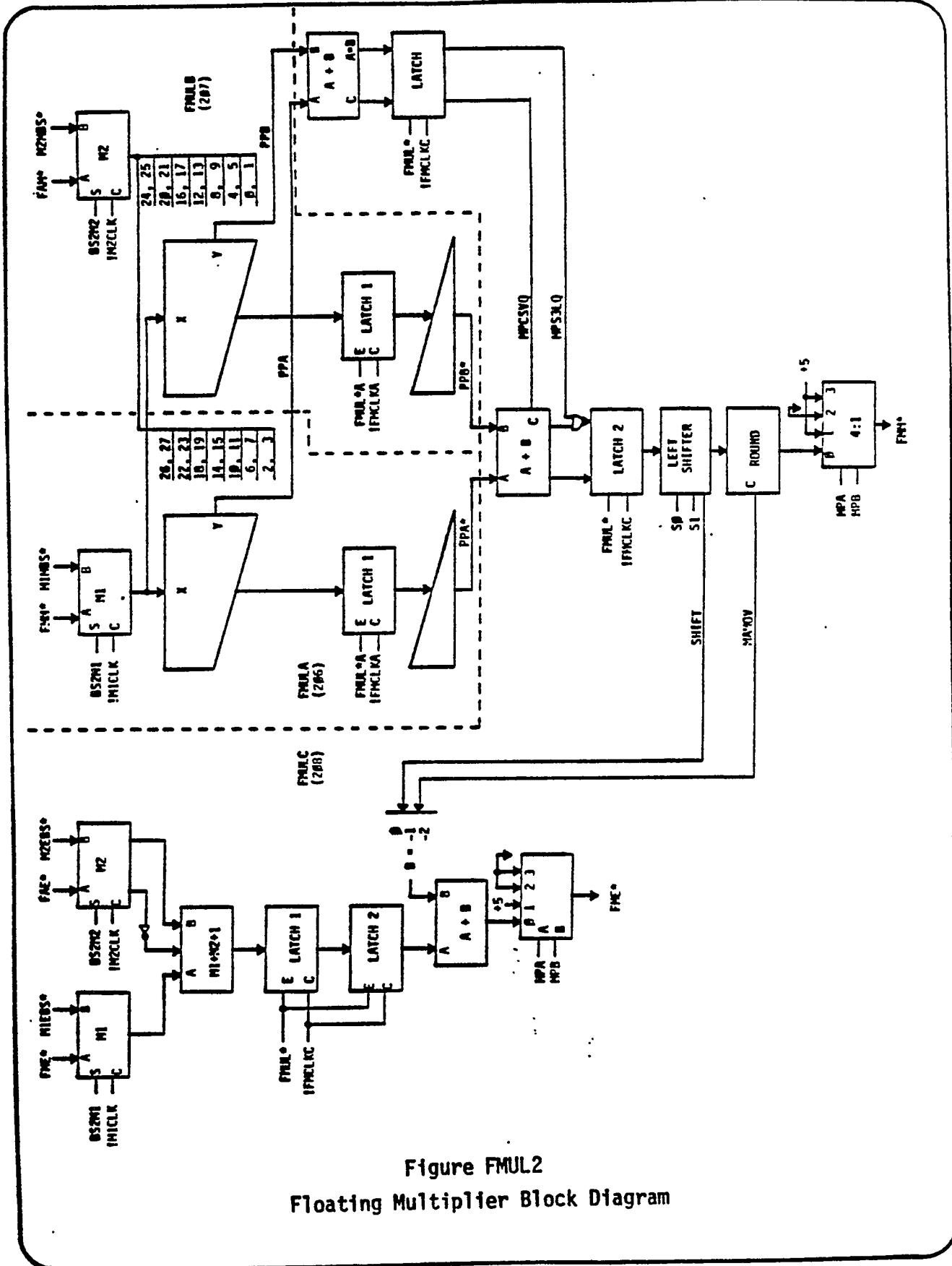
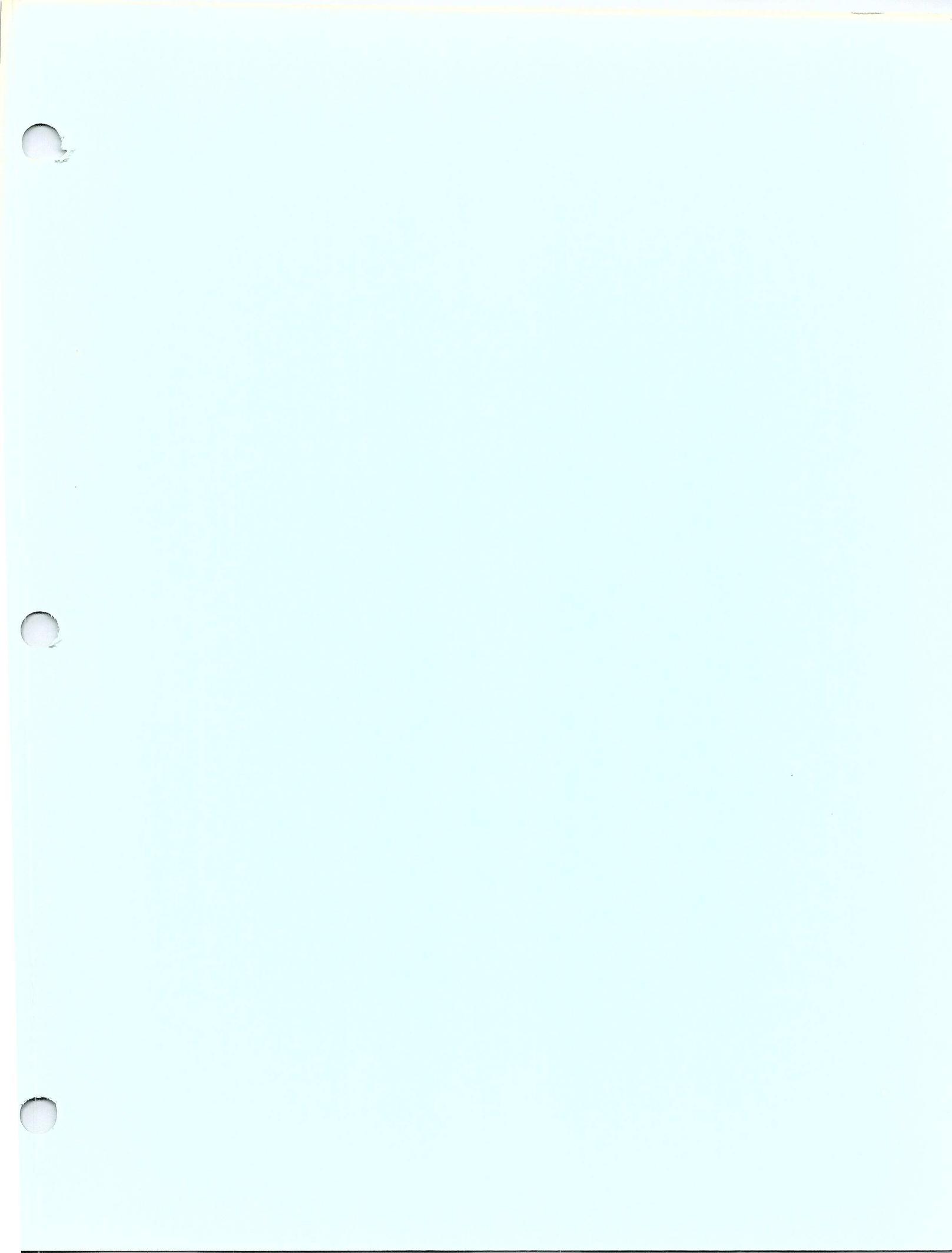


Figure FMUL1
Floating Multiplier System Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE





**Array Processor
Maintenance
Manual**

FAST RECONSTRUCT PROCESSOR

AP120B

by FPS Technical Publications Staff

**Maintenance
Manual
860-7270-000**

PROPRIETARY INFORMATION

1st Edition, January 1978
Publication No. FPS-7270

NOTICE

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

Floating Point Systems, Inc. assumes no responsibility for any errors which may appear in this publication.

PROPRIETARY INFORMATION

This document contains proprietary information and is supplied for identification, maintenance, engineering evaluation or inspection purposes only and shall not be duplicated or disclosed without written permission of

FLOATING POINT SYSTEMS, INC.

By accepting this document the recipient agrees to make every effort to prevent unauthorized use of this information.

Copyright © 1978 by Floating Point Systems, Inc.
Beaverton, Oregon 97005

All Rights reserved. No part of this publication may be reproduced in any form or by any means without permission in writing from the publisher.

Printed in USA

CONTENTS

		Page
CHAPTER 1	INTRODUCTION	
1.1	PURPOSE	1-1
1.2	SCOPE	1-1
1.3	GENERAL DESCRIPTION	1-2
1.4	PHYSICAL DESCRIPTION	1-5
1.4.1	Processor	1-5
CHAPTER 2	SITE PLANNING AND PREPARATION	
2.1	INTRODUCTION	2-1
2.2	PHYSICAL REQUIREMENTS	2-1
2.2.1	Processor	2-1
2.2.2	Power Panel	2-2
2.2.3	Control Panel	2-4
2.3	ELECTRICAL REQUIREMENTS	2-5
2.4	ENVIRONMENTAL REQUIREMENTS	2-6
CHAPTER 3	INSTALLATION AND CHECKOUT	
3.1	INTRODUCTION	3-1
3.2	ACCEPTANCE OF DELIVERY	3-1
3.3	UNPACKING	3-1
3.3.1	Outer Carton	3-1
3.3.2	Inner Carton	3-2
3.3.3	Verification	3-2
3.4	RACK INSTALLATION	3-3
3.4.1	Rack Slides	3-5
3.4.2	Power Panel	3-8
3.4.3	Mounting the Processor	3-9
3.4.4	Control Panel	3-9
3.5	SYSTEM INTERCONNECTION	3-10
3.5.1	Power Panel to Control Panel	3-10
3.5.2	Power Panel to Processor	3-11
3.6	START-UP	3-14
3.6.1	Line Power	3-14
3.6.2	DC Power Supplies	3-15
3.6.3	Final Installation	3-17

	Page
CHAPTER 4	CHECKOUT AND TEST
4.1	4-1
4.2	4-1
4.3	4-1
4.3.1	4-2
4.3.2	4-2
4.3.3	4-2
4.3.4	4-2
	4-2
CHAPTER 5	THEORY OF OPERATION
5.1	5-1
5.2	5-1
5.2.1	5-3
5.2.2	5-10
5.2.3	5-20
5.3	5-22
5.3.1	5-22
5.3.2	5-31
5.3.3	5-39
5.4	5-47
5.4.1	5-47
5.4.2	5-52
5.4.3	5-65
5.4.4	5-68
5.4.5	5-76
5.4.6	5-92
5.4.7	5-107
5.5	5-118
5.5.1	5-118
5.5.2	5-123
CHAPTER 6	INTERFACES
6.1	6-1
6.2	6-1
6.2.1	6-3
6.2.2	6-7
6.3	6-15
6.3.1	6-15
6.3.2	6-21
6.3.3	6-21
6.3.4	6-23
6.3.5	6-23

ILLUSTRATIONS

Figure No.	Title	Page
1-1	The Processor	1-7
1-2	Typical AP-120B Configuration	1-8
1-3	Power Panel	1-9
1-4	Control Panel	1-10
2-1	AP-120B Configuration for Rack-Mounting	2-3
3-1	Rack Mounting Detail (Untapped)	3-6
3-2	Rack Mounting Detail (Tapped)	3-7
3-3	AP-120B System Interconnections	3-11
3-4	+5 VDC Power Supply	3-13
3-5	Bus Bar Bolting Detail	3-13
3-6	-5 VDC and +12 VDC Power Supplies	3-16
5-1	AP-120B - Simplified Block Diagram	5-2
5-2	Data Pad System Block Diagram	5-26
5-3	Block Diagram of DPX	5-27
5-4	Timing Diagram of Writing DPX	5-28
5-5	Detail Showing Stack Address	5-29
5-6	Data Pad Output Logic	5-30
5-7	Main Data System Interconnection Block Diagram	5-34
5-8	MI Reg Block Diagram	5-35
5-9	Main Data Memory Element Block Diagram	5-36
5-10	Main Data Register Block Diagram	5-37
5-11	Memory Address Simplified Block Diagram	5-38
5-12	Table Memory System Block Diagram	5-42
5-13	Table Memory ROM Interconnection Block Diagram	5-43
5-14	Table Memory Address Logic	5-44
5-15	Table Memory ROM Block Diagram	5-45
5-16	Table Memory Register Block Diagram	5-46
5-17	Internal Floating-Point Format	5-65
5-18	Adder (FADD) Pipeline Operation	5-93
5-19	Pushing Values Through the Adder (FADD)	5-94
5-20	Interconnection Block Diagram	5-101
5-21	FADD Hardware Block Diagram	5-102
5-22	Flow Chart AP-120B Floating Adder Logic	5-103
5-23	Floating Adder Input Latches	5-104
5-24	Input Latches and Exponent Comparison Logic	5-105
5-25	Exponent Alignment Logic	5-106
5-26	Floating Multiplier Interconnection Block Diagram	5-108
5-27	Multiplier (FMUL) Pipeline Operation	5-111
5-28	Pushing Values Through the Multiplier (FMUL)	5-112
5-29	Floating Multiplier Exponent Logic	5-116
5-30	Floating Multiplier Mantissa Logic	5-117
5-31	AP-120B Block Diagram	5-121
5-32	Program Source Address Logic	5-122
5-33	S-PAD Interconnection Block Diagram	5-125
5-34	S-PAD Block Diagram	5-126

Figure No.	Title	Page
6-1	Timing Diagram	6-9
6-2	Signal Load Variations	6-14
6-3	AP-120B I/O Bus Timing (Fast Device)	6-25
6-4	I/O Bus Timing with Spin	6-26
6-5	AP-120B DMA Bus Timing	6-28
6-6	Circuit Board Specifications	6-30
6-7	DMA Interface	6-31
6-8	Bread Board 221	6-33
6-9	Programmed I/O Interface	6-36

TABLES

Table No.	Title	Page
1-1	Related Publications	1-1
2-1	Electrical Specification Summary	2-5
3-1	Hardware Shipment Checklist	3-3
3-2	AP-120B Mounting Hardware	3-4
5-1	Table Memory Addressing	5-40
5-2	4-Bit, 2's Complement, Binary Number System	5-61
5-3	Range of Exponent Values	5-64
5-4	Adding a 512 Bias	5-64
5-5	Examples of Decimal Floating-Point Numbers	5-66
5-6	Examples of AP-120B Internal Floating-Point Numbers	5-66
5-7	Table of Rules for Booth's Algorithm	5-86
5-8	Table of Rules of 3-Bit Booth's Algorithm	5-91
5-9	Octal Decode of the FADD and FADD1 Fields	5-96
5-10	Octal Decode of the A1 and A2 Fields	5-96
6-1	Controller to AP Signals	6-10
6-2	AP to Controller Signals	6-11
6-3	Host Data Bus	6-12
6-4	Host Memory Address Lines	6-13
6-5	I/O Signal Names and Functions	6-24
6-6	DMA Signal Names and Functions	6-27
6-7	Recommended Drivers	6-29
6-8	Loading Rules	6-32
6-9	Recommended Interface Pinouts	6-34
6-10	Board 219 (UNIV) Pinouts	6-37
6-11	Board 226 (TMM) Pinouts	6-38

CHAPTER 1

INTRODUCTION

1.1 PURPOSE

The purpose of this manual is to provide the information necessary to understand, install, use and maintain the AP-120B. The array processor is a pipelined, parallel processor which can be interfaced to any one of a variety of host computers to provide a powerful, cost-effective processor for high-density, high-speed computation. Throughout the remainder of this manual, the array processor is referred to as either the "AP-120B" or the "AP".

1.2 SCOPE

This manual provides hardware information related to the AP-120B. The manual first covers site planning and installation so that the AP can be properly installed and started up. The next section covers the check-out and test procedures that are used to verify proper AP operation. The bulk of the manual contains detailed theory of operation for the AP-120B. This theory is written to the functional and schematic diagram level as an aid in understanding the AP hardware.

This manual is limited to discussions of the AP-120B hardware. Additional information on the AP-120B, such as software descriptions or interface descriptions, is available in the related manuals listed in Table 1-1 below. Any of these manuals can be ordered from Floating Point Systems, P.O. Box 23489, Portland, Oregon 97223.

Table 1-1 Related Publications

Manual	Number
Processor Handbook	7259-02
Programmers Manual	7319
APDEBUG Manual	7364
AP-120B Diagnostic Software Manual*	7284-01
AP-120B Math Library (Part 1)	7288-02
AP-120B Math Library (Part 2)	7288-03

1.3 GENERAL DESCRIPTION

The AP-120B is a pipelined, parallel processor. This processor contains two floating-point pipelined arithmetic elements, multiple memories and multiple buses. This structure greatly reduces the time required for processing information because it allows various system elements to handle tasks in parallel rather than sequentially.

The array processor may be functionally divided into five main units: memory, arithmetic, address, interface and control. Each of these five units is briefly described below:

Memory The AP-120B contains four main memory elements that, because of the multiple bus structure, operate independently. Thus, simultaneous operation of the memories is possible. These four memories are:

- | | |
|------------------|---|
| * program source | - stores the 64-bit microencoded instruction word |
| * table memory | - stores frequently-used constants such as sines and cosines |
| * data pad | - fast access memory composed of two elements (X and Y), each of which may be considered 32 individual accumulators |
| * main data | - prime storage area for data |

Arithmetic The AP-120B contains three arithmetic elements, two of which are specifically designed for floating-point operations. These elements are:

- | | |
|------------------|--|
| * floating-point | - a 2-stage, pipelined adder that performs arithmetic and logic operations. Because a new input may be entered into the pipeline stream every cycle, a new add can be started every 167ns although it takes 333ns to perform an add. |
|------------------|--|

The results from the adder are normalized, rounded and error-checked.

- * floating-point multiplier (FMUL)
 - a 3-stage, pipelined multiplier. Although it requires three cycles to complete the multiplication (500ns), the pipeline method permits a new multiply operation to be started every 167ns.

Multiplier results are normalized, rounded and error-checked.
- * scratchpad ALU (SPAD)
 - an ALU and sixteen 16-bit registers used to perform integer arithmetic and overhead functions in parallel with FADD and FMUL operation. Such overhead functions include: loop counting address indexing and control functions required by the program.

Address The AP-120B contains five registers used for addressing the system's memories, peripherals and scratchpad (SPAD) registers. These five registers are:

- * table memory
 - serves as a pointer to the location containing the desired constant in table memory.
- * data pad address register
 - contains an address that is combined with an index value in the instruction word to access a specific location in the data pad memory.
- * memory address register
 - serves as a pointer to the desired location in main data memory.
- * device address register
 - contains the address of the external device to be used with the AP-120B.
- * scratchpad destination
 - contains the address of the scratchpad register that is to receive the output of the scratchpad ALU.

Interface The AP-120B contains a number of interface elements. Three of these elements are registers that make up a "simulated" or "virtual" front panel. The five remaining elements are used when performing DMA transfers between AP-120B and the host computer. The interface elements are:

- * switch register - used by the host to load data and/or addresses into the AP-120B. This register is similar to a normal computer switch register except that information is loaded or read under computer control, rather than entered by means of front panel switches.
- * lights register - simulates front panel lights in order to display the contents of internal AP-120B registers.
- * function register - provides the remaining front panel controls required by a computer such as stop, start, deposit, examine, step, continue, etc.
- * formatter - converts the input word to the 38-bit floating-point format required by the AP-120B and vice versa. Typically, the formatter supports four different formats, as determined by the host computer.
- * host memory address register - serves as a pointer to host computer memory locations involved in DMA transfers. This register always operates in either auto-increment or auto-decrement mode.

- * array processor memory address register - serves as a pointer to the AP-120B main data memory locations involved in DMA transfers. This register always operates in either auto-increment or auto-decrement mode.
- * word count register - keeps track of the number of words transferred during a DMA operation.
- * control register - used for control and status information when performing DMA transfers.

Control The AP-120B contains a number of control elements that ensure the processor operates properly. Such control elements include clocking circuits, status registers, control gating, etc. It is important to remember that the AP-120B is a synchronous processor and therefore, control elements are extremely important.

Although programming information is not included in this manual, it should be noted that the AP-120B uses a 64-bit instruction word and a 38-bit floating-point data word. The data word consists of a 10-bit exponent and a 28-bit mantissa. For more information on instruction and data word formats, refer to the AP-120B Processor Handbook.

1.4 PHYSICAL DESCRIPTION

This section provides a detailed description of the physical characteristics of the three equipment groups that comprise the AP-120B. The three equipment groups are physically separate pieces of hardware (each with a separate serial number tag) and will subsequently be called the processor, the power panel, and the control panel.

1.4.1 Processor

The processor consists of a 31-slot card cage assembly, a minimum of 22 etch circuit boards (ECB), a front cover and a rear cover. (See Figure 1-1.) Assembled (ECB, front and rear covers in place), the processor is physically 62.23 cm. (24-1/2 inches) high, 31.12 cm. (12-1/4 inches) deep, and 44.78 cm. (17-5/8 inches) wide. The AP-120B is designed to mount in a standard 48.78 cm. (19-inch) EIA. Thus, the width measurement incorporates the distance the rack slides extend from the card cage chassis. Figure 1-2 illustrates a typical AP-120B configuration.

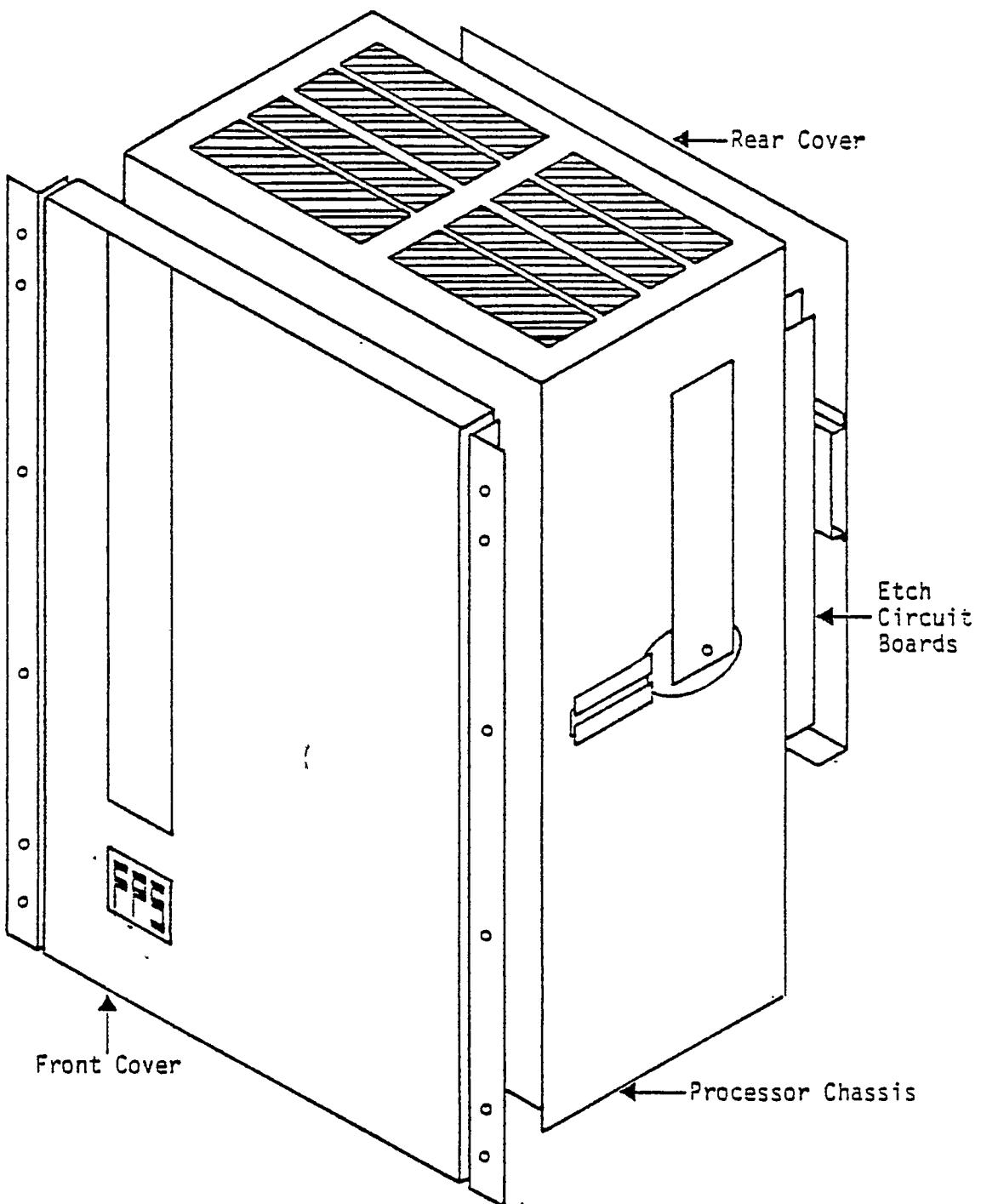
1.4.1.1 Power Panel

Mounted on the power panel is one or two 5V/120A or 150A DC power supply, a +12V/3A DC power supply, a -5V/3A DC power supply, and a line box. Figure 1-3 is a visual presentation of the power panel. The line box houses a terminal board, a power relay, and the line filters. When assembled, the power panel is 52.70 cm. (20-3/4 inches) high, 48.18 cm. (18-31/32 inches) wide, and 15.56 cm. (6-1/8 inches) deep. The power panel specifies the line voltage to be used.

The power panel, like the processor, is designed to be mounted in a 48.26 cm. (19-inch) EIA rack. The standard installation has the power panel occupying the back of the rack immediately behind the processor. The power panel mounts directly on the rack. This panel is also hinged to allow access to the power supplies for maintenance without removal of the power panel from the rack.

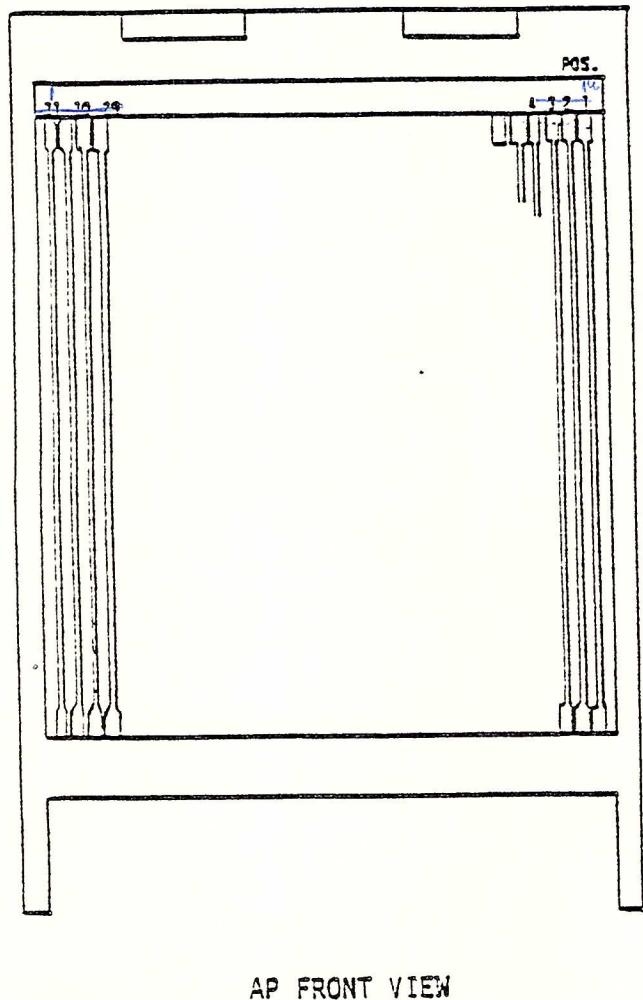
1.4.1.2 Control Panel

The control panel consists of the power ON/OFF switch, the power status indicators, and the processor status indicators. See Figure 1-4. Physically, the control panel is 4.29 cm. (1-11/16 inches) high (4.45 cm. [1-3/4 inch] rack increment), 48.18 cm. (18-31/32 inches) wide, and 4.45 cm. (1-3/4 inches) deep. This subassembly is also designed to be rack-mounted and may be mounted at the front of the rack (with the processor) or at the rear of the rack (with the power panel). It should be noted that the control panel voltage must match the power panel voltage as noted on the serial number tag.



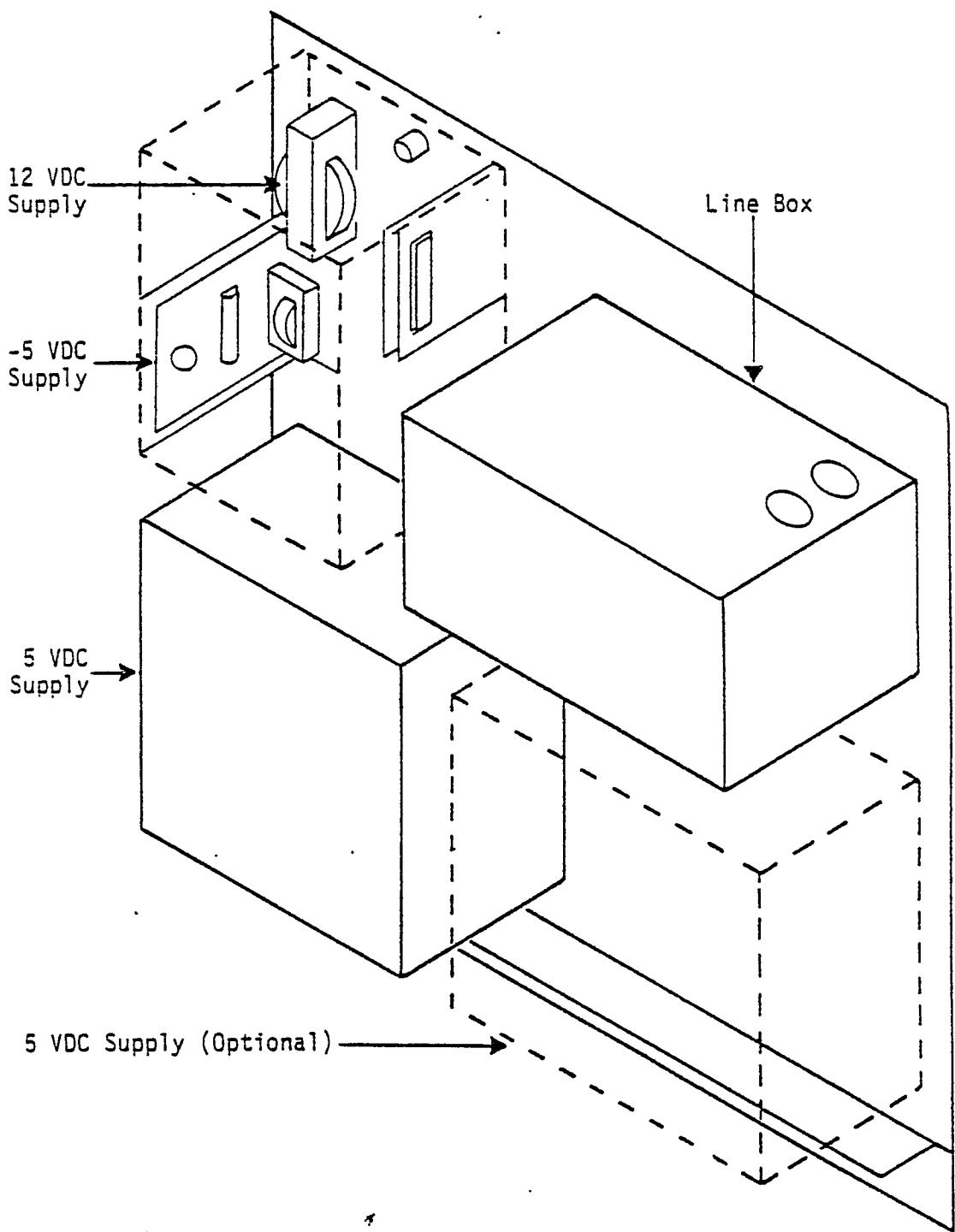
0181

Figure 1-1 The Processor



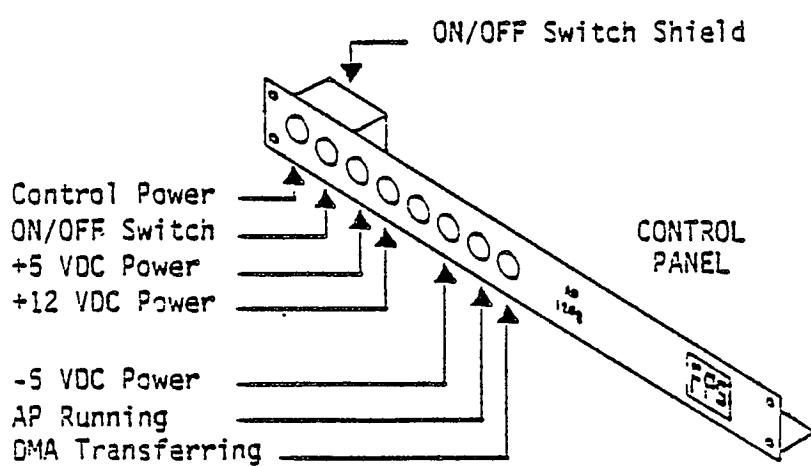
<u>POS.</u>	<u>BOARD</u>	<u>TYPE</u>
1	219	GENERAL INTERFACE
2	227	FORMATTER
3	226	---
4	238	---
5	238	TABLE MEMORY
6	215 275	
7	215 238	
8	215 276	
9	215 214	DATA MEMORY
10	215 244	
11	215 244	
12	215 244	
13	216 236	PROGRAM SOURCE
14	216 236	
15	201 201	SCRATCH PAD
16	212 7	TM ADDRESS REGISTERS
17	214	INST. & FRONT PANEL REG
18	210	DEST. ADDRESS REGISTERS
19	211	DATA PAD ADDR. REGISTER
20	202	DATA MEMORY OUT. REGISTER
21	217	TM ROM
22	209	TM OUTPUT REGISTER
23	213	DATA MEMORY IN. REGISTER
24	200 R	DATA PAD RIGHT
25	200 L	DATA PAD LEFT
26	203	
27	204	FLOATING-POINT ADDER
28	205	
29	206	
30	207	FLOATING-POINT MULTIPLIER
31	208	

Figure 1-2 Typical AP-120B Configuration



0182

Figure 1-3 Power Panel



0227

Figure 1-4 Control Panel

CHAPTER 2

SITE PLANNING AND PREPARATION

2.1 INTRODUCTION

This section provides a summary of the equipment characteristics of the AP-120B essential for pre-installation site planning and preparation.

2.2 PHYSICAL REQUIREMENTS

The physical requirements that must be considered for installation are the space to be occupied, ventilation, and the system connections that are required. The AP-120B is designed to be rack-mounted in a standard 48.26 cm. (19-inch) EIA rack with a rack depth of between 50.80 and 63.50 cm. (20 and 25 inches). Floating Point Systems recommends that the user allow 76.20 cm. (30 inches) of free vertical rack space for the installation of the AP-120B.

2.2.1 Processor

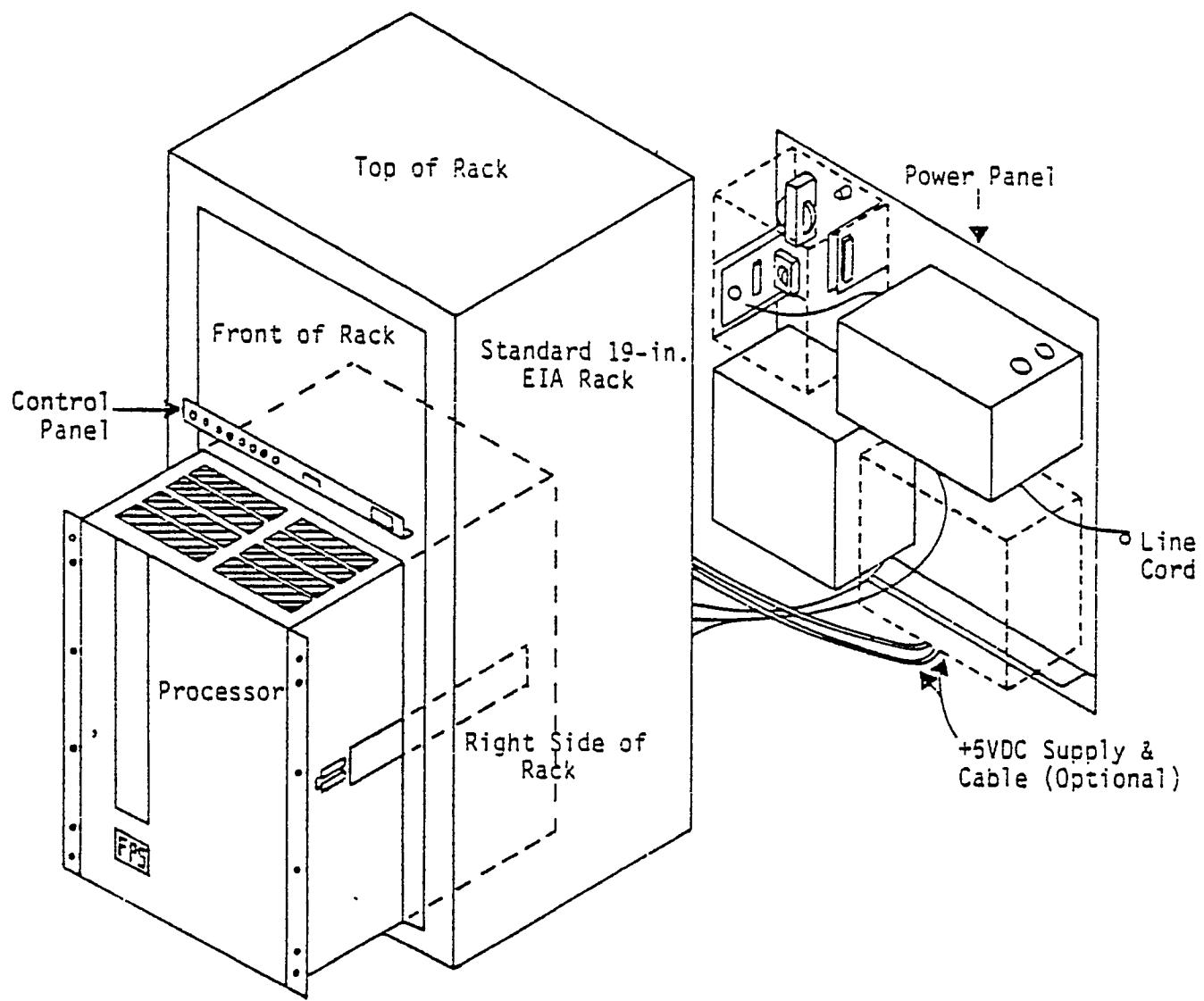
The AP-120B physically requires 62.23 cm. (24-1/2 inches) of vertical rack space and will be mounted in the forward portion of the rack on rack slides. A minimum of an additional 4.45 cm. (1-3/4 inches) above the processor (in the vertical plane) are necessary for proper air circulation. This can be occupied by the control panel, blank panels, or other equipment which does not fully occupy the space adjacent to the AP-120B.

Five position, tilt-type rack slides are used on the processor to allow access to the backplane, etch circuit boards, fans, and connectors during installation and maintenance. The rack slides and hardware (nuts, bolts, etc.) are included for mounting in racks from 50.80 to 63.50 cm. (20 to 25 inches) deep.

The processor must interface to the host processor via an interface cable. This interface cabling is, in most cases, supplied by Floating Point Systems. It is necessary that the processor be physically located in a place that will allow the interface cabling to connect from the host processor to the AP-120B. Because the processor is mounted on rack slides, it is recommended that an extra 63.50 cm. (25 inches) of slack be left in this cabling to allow the processor to operate in its fully-extended position. As examples of typical interface cabling lengths, the Nova/Eclipse interface cabling provided by FPS is 3 m., 4.80 cm. (10 feet) and the Unibus cable for the PDP-11 series is 1 m., 52.40 cm. (five feet) in length.

2.2.2 Power Panel

The power panel mounts directly on the rack immediately behind the processor assembly. Figure 2-1 shows the orientation of the processor, the power panel and the control panel with respect to the rack. As specified in Section 1.2.3.3, the power panel requires 52.70 cm. (20-3/4 inches) of free vertical rack space in the rear of the rack. If the rack is less than 50.80 cm. (20 inches) deep, it is possible that there will not be enough clearance between the processor and the power panel to allow the power panel to be mounted immediately behind the processor.



0180

Figure 2-1 AP Configuration for Rack Mounting

Consideration must also be given for the interconnection of the power panel to the processor, to the control panel, and to the electrical service.

Between the power panel and the processor, there are three interconnecting cables: the four separate +5 VDC cables 5 VDC, the control, the +12 VDC and the -5 VDC, and the AC fan voltage. The +5 VDC cabling provided is 1 m., 42.24 cm. (56 inches) in length and connects from the +5 VDC/150A power supply on the power panel to the front of the processor. The +12 VDC and the -5 VDC cabling is 1 m., 6.68 cm. (42 inches) long and connects to the rear of the processor. It should be noted that the processor is also capable of sliding forward 50.80 cm. (20 inches) on its rack slides. Thus, the physical layout of the power panel with respect to the processor must leave enough slack in the cabling to allow for this 50.80 cm. (20 inches) of travel. See Figure 3-3 for an interconnection block diagram.

There are two cables from the control panel going to the power panel. One cable is 1 m., 6.68 cm. (42 inches) long, lights the control power indicator and connects the ON/OFF switch. The other is 1 m., 6.68 cm. (42 inches) long for the DC supply and function indicators.

The cord provided to connect the power panel to the electrical service is 1 m., 82.88 cm. (six feet) in length. The cable originates at the line box on the power panel and utilizes a standard three-prong plug to connect to the service.

2.2.3 Control Panel

The two primary physical considerations in the placement of the control panel are its physical size and its cabling requirement. The control panel is to be mounted directly on the rack and requires a 4.45 cm. (1-3/4 inch) rack increment. Mounting hardware is included. The control panel may be either mounted in the front or the rear of the rack. Further, it may be mounted above or below the processor or power panel. When front mounted, the control panel assures free-air for one end of the processor and the status of the AP-120B is visible. Rear mounting minimizes total rack space and reduces the chances of accidentally switching the AP-120B power off.

There are two cables at the control panel. One contains the power and processor status. The other contains the ON/OFF control voltage. The physical requirements for the ON/OFF control cabling are defined in Section 2.2.2. The power and processor status cabling originates at the processor and routes by the power panel, terminating at the control panel, and is 1 m., 6.68 cm. (42 inches) in length. The ON/OFF control cabling originates at the control panel and terminates at the line box on the power panel.

2.3 ELECTRICAL REQUIREMENTS

The AP-120B can be supplied in three different power configurations as specified in Table 2-1. The total power required for the AP-120B system is approximately 1200 watts (the actual power required is configuration-dependent), and a low impedance service is advised. FPS strongly recommends that the AP-120B be given its own electrical service and the service should be rated for 20 amperes (or 10 amperes for 220V or 240V applications) in order to provide a low impedance source. This low impedance source is critical for proper operation of the peak rectifying supplies used.

Table 2-1 Electrical Specification Summary

Nominal Voltage Voltage (VRMS)	Voltage Range (VRMS)	Minimum Line Voltage (P-P)	Frequency (Hz)	Minimum Line Capacity (A)	Fuses (A) Line Control	
115	100-125	280	50-60	20	15	250ma
230	200-240	560	50-60	10	10	1

If the customer specifies the 115V power option, the AP-120B will come with a 1 m., 82.88 cm. (six-foot) power cord with a U.S. standard three-wire (with ground) male cord cap. For 220V applications inside the U.S., a 15A U.S. standard 220V male cord cap is supplied on the 1 m., 82.88 cm. (six-foot) power cord. For European applications, a 15A male IEC cord cap is supplied on the 1 m., 82.88 cm. (six-foot) power cord.

2.4 ENVIRONMENTAL REQUIREMENTS

Temperature, humidity, vibration, and dust are the four environmental factors that should be considered prior to the installation of the AP-120B.

FPS specifies the AP-120B to operate in environments of 10 degrees C to 40 degrees C. This temperature specification should be derated one degree C per 2500 feet (762 m) above sea level for 60 Hz operation and 5 degrees C for 50 Hz operation. This temperature specification (10 degree C to 40 degree C) is at a relative humidity of 0 percent to 90 percent. If the user's environment has the possibility of varying outside of this temperature/humidity specifications, some type of environmental conditioning is recommended.

With respect to vibration, typical data processing environments suitable for commercial computers are adequate for the AP-120B. The AP-120B, as shipped, is equipped with an air filter that filters the dirt particles out of the incoming cooling air. This air filter allows the AP-120B to be installed in most environments. If an extremely dirty environment is encountered, it may be necessary to clean the filter daily (or more often) or provide an alternate means of air filtering such as an electrostatic air cleaner in proximity to the AP-120B.

CHAPTER 3

INSTALLATION AND CHECKOUT

3.1 INTRODUCTION

This section provides the user with the information necessary to accept delivery, unpack, install, connect, and start up the AP-120B.

3.2 ACCEPTANCE OF DELIVERY

The acceptance of delivery of the AP-120B, unless otherwise negotiated, occurs at the factory. For insurance purposes, however, the shipping carton that contains the AP-120B should be carefully examined for apparent damage prior to signing the carrier's receipt of delivery. If there is damage to the packing carton, this should be noted on the shipping receipt prior to signing.

3.3 UNPACKING

All of the AP-120B hardware and one complete set (more if ordered) of the hardware documentation are shipped in a single carton. Note, however, that documentation will not be shipped until FPS receives a completed non-disclosure form from the customer stating that documentation will be used only for maintenance purposes. The shipping carton weighs approximately 67.95 kilograms (150 pounds) and is 83.82 cm. (33 inches) long, 68.58 cm. (27 inches) wide, and 68.58 cm. (27 inches) high. The AP-120B hardware inside this carton is separately boxed to ensure against damage during shipment. The packaging cartons and packing materials should be saved in case it is necessary to reship the equipment.

3.3.1 Outer Carton

Prior to opening the container, the packing slip affixed to the exterior of the carton should be removed and placed in a secure location. This packing slip is on the top of the carton and will be used to verify that no shortages exist in the shipment.

After removing the packing slip, the top of the packing carton should be opened, the documents removed and placed with the packing slip, and the upper shock pad (rectangular white plastic foam liner) removed. It is also possible that the I/O adapter (not used on all units) or the host-to-AP interconnect cable, if provided by FPS, or both, should be removed at this point. If so, its presence will be obvious; if not, it is packed elsewhere. The documentation will be checked later to ensure a complete shipment.

3.3.2 Inner Cartons

Open the inner carton and remove the two equipment packages and the bubble-packed control panel. Remove the smaller package first. The larger of the two equipment packages contains the processor, and the smaller, the power panel. Open the processor package, slide the processor out, and remove the plastic bag from around the processor. If the host-to-AP interconnect cable or the I/O adapter was to be provided by FPS and was not found above (see Section 3.3.1), it will be found in the processor package. Remove the processor, host-to-AP interconnect, and the I/O adapter to a secure area. Place the plastic bag and the packing material into the package. Open the power panel package and slide out the power panel and rack slides. Remove the rack slide box from the packing strips around the power panel. The power panel should then be placed with the processor in a secure area. Unwrap the bubble-pack from around the control panel and unbox the rack slides. Place the packing strips and bubble-pack inside the empty power panel package. Replace both the processor package and the power panel package in the inner carton. Replace the upper shock pad and close the outer carton.

3.3.3 Verification

After unpacking, both the hardware and the documentation should be checked to ensure that a complete shipment was made. The packing slip should be checked to see that all the specified items arrived in the packing carton. The packing slip reflects the shipment of the equipment as specified in the customers purchase order. It should be noted that if the purchase order specifies such things as the interface, the size of main data, or the size of program source, that these are etch circuit boards that are internal to the AP-120B chassis and have been installed in the processor at the factory prior to shipment.

With the documentation package is a sheet of paper titled "AP-120B Check List for Documentation". The actual documentation received should be checked piece-by-piece against this sheet to ascertain the presence of all specified documentation. If shortages do exist, these should be noted by name as specified on the checklist and FPS should be notified immediately.

At the shipping-dock level, verification of hardware shipment should be limited to the major functional units as specified in Table 3-1. The processor, the power panel, and the control panel each have a serial number tag affixed to them. Located at the end of the serial number is a letter ("A" for the processor, "B" for the power panel, and "C" for the control panel) added to help identify these subassemblies. The presence of the three subassemblies should be verified.

Table 3-1 Hardware Shipment Checklist

	Present	Absent
Functional unit		
Processor		
Power Panel		
Control Panel		
Rack Slides		
Host-to-AP Cable(s)		
I/O Adapter*		
Extender Card**		

* Not necessary on some Host CPU's

** This will be supplied with first unit only.

The presence of the rack slides, the host-to-AP interconnect cable (if supplied by FPS), and the extender card (this is sent with the first AP-120B supplied) should be verified. Again, if shortages exist, these should be recorded and FPS should be notified immediately.

The power panel and the control panel should be checked to verify that they operate on the same AC input power, and that this is the power option ordered. The three power options available are specified in Table 2-1. The serial number tag on the power panel and the control panel have a section labeled "volts". These should be checked to ensure that they both specify the same voltages, and that this voltage option is the one that was ordered.

3.4 RACK INSTALLATION

The rack installation of the AP-120B follows a logical sequence. After checking the area, set aside for the AP-120B installation to make sure it is clear and free of obstruction. The first step is to locate and mount the rack slides. Next, the power panel is located and mounted on the back of the rack. Then, the processor is inserted into the rack on its rack slides. Finally, the control panel is located and mounted. Table 3-2 gives a detailed list of the mounting hardware supplied by FPS.

Table 3-2 A2 Mounting Hardware

- (1) Set Racksides.
- (2) 12 OHS, 12 CW, 12 Captive Nuts (10-32 x 3/4) for Processor.
- (3) 20 BHS, 16 FHS, 20 LW, 50 FW, 4 nuts, 4 Nut Plates (10-32) for Racksides.
- (4) 4 BHS, 4 FW, 4 LW, 4 Captive Nuts (10-32 x 1/2) for Power Panel, bottom.
- (5) 4 OHS, 4 CW, 4 Captive Nuts (10-32 x 3/4) for Power Panel, top.
- (6) 4 OHS, 4 CW, 4 Captive Nuts (10-32 x 3/4) for Control Panel.
- (7) 2 BHS, 4 FW, 4 LW, 2 nuts w/LW (10-32 x 1/2) for Power Cables.

	10-32 BHS x ₈ #10	10-32 BHS x ₈ #10	10-32 FHS x ₂	10-32 LW x ₄	10-32 FW x ₂	Lock Washers #10	Cup Washers #10	Flat Washers #10	10-32 Nut w/Lock Washer #10	Nut Plate 10-32	Captive Nut 10-32
Processor					12		12				12
Racksides		20	16			20		50	4	4	
Power Panel, Bottom	4					4		4			4
Power Panel, Top					4		4				4
Control Panel					4		4				4
DC Power Cables	2					4		4	2		
TOTAL	6	20	16	20	28	20	58	6	4	24	
%/Loss Allowance	7	22	17	22	30	22	60	7	5	26	

These are to be packaged in plastic bags and packages with the racksides to be shipped with unit:

BHS = Binding Head Screw
 OHS = Oval Head Screw
 FHS = Flat Head Screw
 LW = Lock Washer
 FW = Flat Washer
 CW = Cup Washer

0229

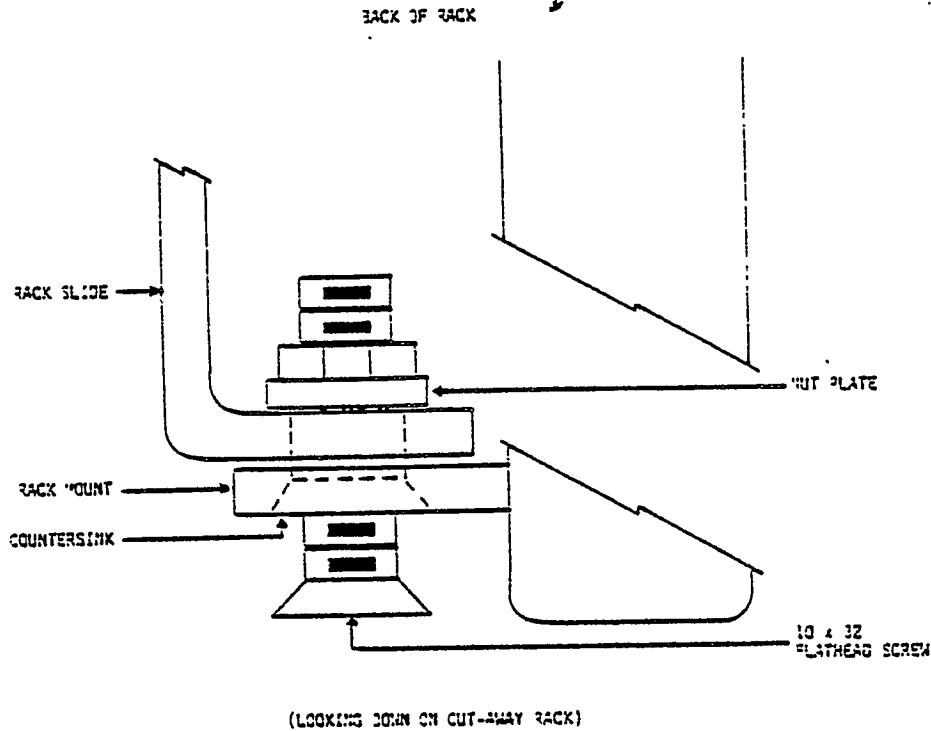
3.4.1 Rack Slides

Prior to locating the rack slides in the rack, the 76.20 cm. (30 inches) of free vertical rack space into which the AP-120B will be installed should be visually checked to ensure that there is nothing else occupying this space. The distance from the front to the rear of the rack mounts should be checked to ensure that this distance is greater than 50.80 cm. (20 inches) and less than 63.50 cm. (25 inches).

The first step in mounting the rack slides is to determine the uppermost limit of the 76.20 vertical centimeters (30 inches) of rack space set aside for the AP-120B installation. (This point should be between two 1.27 cm. [1/2 inch] spaced holes.) This will be the reference point from which further measurements will be made. From this reference point measure down 4.45 cm. (1-3/4 inches) (one rack increment) and mark this spot (note that this 4.45 cm. [1-3/4 inches] space may be occupied by a unit above, if the bottom of the unit permits free air passage to the rear of the rack, and if the control panel is to be mounted elsewhere.) This now becomes the theoretical top edge of the processor. (Again this should be between two holes that are 1.27 cm. [1/2 inch] apart.) Measure down 31.12 cm. (12-1/4 inches) (7 rack intervals) and mark this spot. This now is the center line for the rack slides and should be between two holes that are spaced 1.27 cm. (1/2 inch) apart.

There are generally two types of mounts in the standard 48.26 cm. (19-inch) EIA rack. In one, the mounting holes are tapped and in the other they are not. If the rack you have is one that has tapped mounting holes, skip the next paragraph. If your rack mounting holes are not tapped, it will be necessary for you to countersink the second hole above and the second hole below the center line of the rack slides. This is necessary to allow the front panel of the processor to mount flush in the rack. FPS provides all the necessary bolts, nuts, nut plates and washers for installation of the AP-120B.

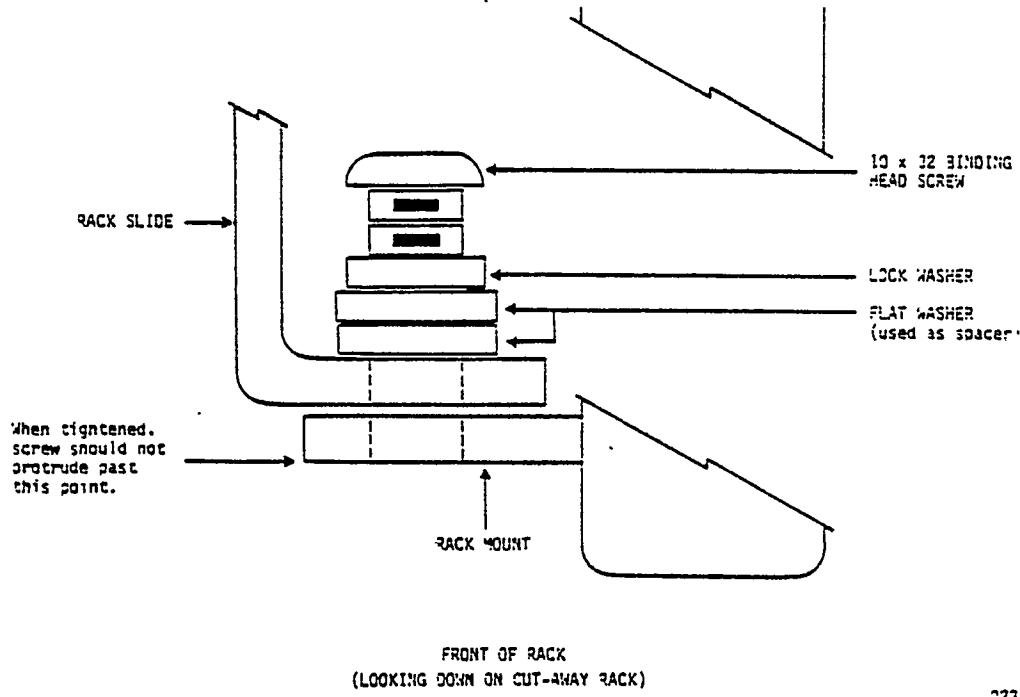
After countersinking these two holes in the front of the rack on both the left and right sides, mount the rack slide on the rack using flat-head 10-32 screws and nut plates. See Figure 3-1 for a diagram showing the positioning of the various parts. After mounting the rack slides to the front of the rack, position the rack slide extensions in place and determine the two holes on each side to be used for mounting the rack slide in the rear of the rack. Now that these have been determined, countersink these four holes (two on each side) as was done in the front of the rack. Now, bolt the rack slide extensions on the rack using the flat-head 10-32 screws and nut plates in the same manner as was used in the front of the rack. Then, using the 10-32 binding head screws, lockwashers, and nuts (four sets required) bolt the rack slides to the rack slide extensions. Then SECURELY tighten all rack slide and rack slide extension bolts.



0220

Figure 3-1 Rack Mounting Detail (Untapped)

If mounting the AP in a rack that has tapped mounting holes, the measurements determining the center line of the rack slides should have been made (as specified above) prior to this time. The rack slide will mount on the rack. Use the two holes above and two holes below this center line. The 10-32x3/8 binding head screws and flat washers are provided to mount the rack slide. In order for the front panel of the processor to mount flush in the rack, it is necessary that the 10-32 screws not protrude past the edge of the mounting flange. Install the rack slides as shown in Figure 3-2 using the required number of flat washers to ensure that the screws do not protrude.



3231

Figure 3-2 Rack Mounting Detail (Tapped)

After installing both the left and right rack slides to the front of the rack, the holes used to mount the rack slide extensions should be determined by positioning the rack slide extension at the end of the rack slide and marking the appropriate four holes. The rack slide extension should then be mounted to the rear of the rack in a similar fashion as the rack slides ensuring that the 10-32 binding head screws do not protrude. The latter is necessary to allow the power panel to mount properly. Now, using the 10-32 binding head screws, lock washers, and nuts (four sets required) bolt the rack slides to the rack slide extensions. Then SECURELY tighten all rack slide and rack slide extension bolts.

3.4.2 Power Panel

After the rack slides have been mounted, the power panel should be located and mounted in the rear of the rack. (Note that the correct orientation of the power panel is with the piano hinge at the bottom of the rack.) From the center line of the rack slide (between two holes spaced 1.27 cm. [1/2 inch] apart) measure up 22.86 cm. (nine inches) and mark this hole. This should be done on both sides prior to installation of the power panel. This is the hole in which the upper screw of the power panel will be placed.

After locating the two upper holes in the rack, assemble two 10-32 oval head screws (provided by FPS) and cup washers to be used to mount the power panel. The power panel will be installed by lifting the power panel into position on the rack and placing the two screws into the holes which were located above. These two screws will then support the weight of the panel while the rest of the screws are placed through the panel into the rack.

After assembling the two 10-32 screws with cup washers and determining the holes in the rack to be used, the upper hole on the left and right of the power panel should be aligned with the holes in the rack. The screw assembly is inserted through the upper hole in the power panel, and screwed into the rack mount. This should be done for both the left and right sides. Insert the other two 10-32 oval headed screws with cup washers into the rack through the middle holes in the power panel. (These may require the clip-type speed nuts if the rack is untapped.) These should not be mounted securely, but only temporarily. Now, using the four 10-32 binding head screws, flat washers and lock washers, bolt the power panel hinge, lower section to the rack mount. The power panel should now be aligned in the rack in such a way as to allow it to be folded down on its hinge far enough to allow access to each piece of equipment mounted on the panel. This is done by pivoting the panel down and adjusting the hinge in the rack until the panel can be pivoted out the back of the rack. After finishing this, all the screws should be replaced.

3.4.3 Mounting the Processor

Prior to mounting the processor in the rack, the portion of the rack slide assembly on the processor should be adjusted. First, set the rack slide tilt mechanism arms to the horizontal position. The angle between these extended arms and the rear face of the processor should be 90 degrees. This can be simply checked by laying the edge of the extender board on the rack slide arm and visually checking to see if this is square with respect to the rear cover of the processor. If not, the nut behind the large screw in the middle of the tilt mechanism should be loosened and the arm repositioned until a 90-degree angle is achieved. Repositioning the arm is accomplished by turning the large screw mentioned above. The nut should then be re-tightened. Both tilt mechanisms (left and right) should be checked and adjusted if necessary.

Preferably with two people, set the processor on the floor in front of the rack with the front panel forward. Lift the processor and insert the arms into the slide slots of the rack slides. Slide the processor into the rack until it stops. Now grasp the outer portion of the slides and, while pulling forward slightly, press the release buttons and slide the processor into the rack. The mounting ears should now be flush in the rack. If the processor does not slide freely into the rack (it binds) or the mounting ears do not fit flush in the rack, further adjustment of the rack slide hardware in the rack will be necessary to allow the processor to slide into the rack without binding. First, the mounting of the rack slide hardware in the rack should be adjusted relative to the rack to allow the processor to slide in the rack without binding. Then, if necessary, the rack slide tilt mechanism should be re-adjusted until the front panel fits flush with the rack. Then from the rear of the rack, visually verify that there is at least two inches of clearance between the line box (on the power panel) and the rear cover of the processor.

3.4.4 Control Panel

If the control panel is to be mounted in the front of the rack, it will mount immediately above (or below) the processor. If your rack has untapped mounting holes, it will be necessary for you to use four #10 clip-type speed nuts to mount the control panel. If the processor and power panel have been mounted according to instructions, then above and below both the processor and power panel will be rack holes spaced 3.18 cm. (1-1/4 inches) apart, ready to receive the control panel.

In order to attach the connector to the control panel (Section 3.5.1) it will be necessary to gain access to the rear of the control panel. Therefore, it is recommended that it be mounted temporarily at this time, or connect the cable to the connector now. Four 10-32 oval head screws with cup washers and inserts will be used to mount the control panel. Now, locate the control panel in the position chosen and install the screws through the slots in the panel into the rack. Now adjust the position of the control panel to provide a .19 cm. (1/16th of an inch) clearance between the edge of the control panel and either the processor or the power panel. This is necessary to allow the processor or power panel to move without hindrance.

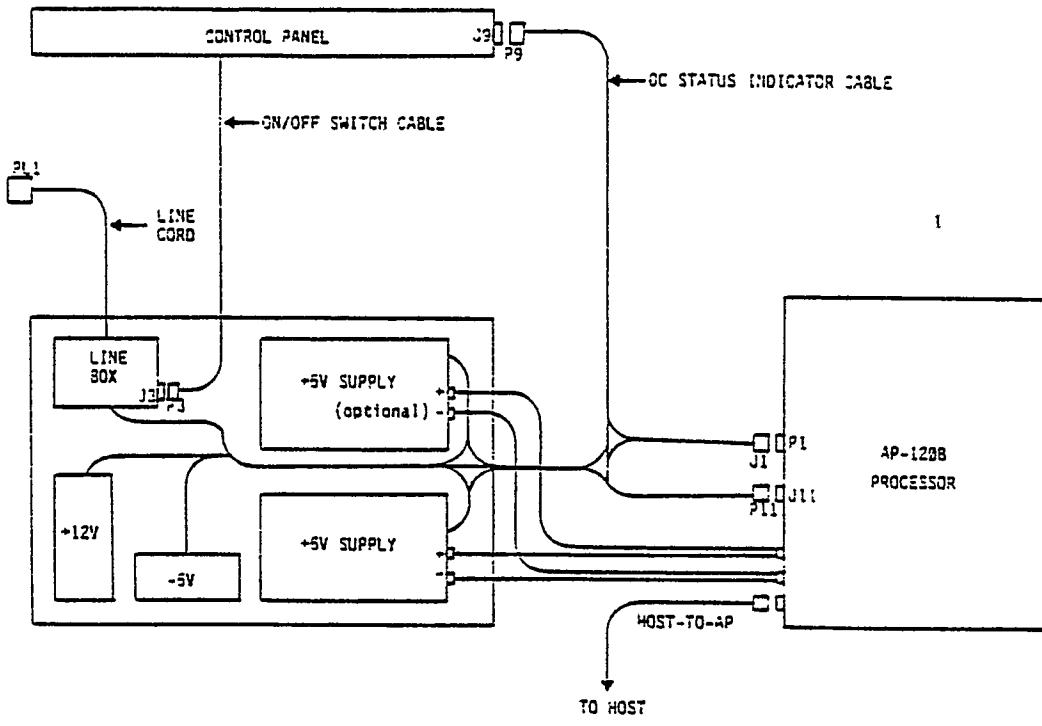
3.5 SYSTEM INTERCONNECTION

The AP has several connections which must be properly made prior to powering up the system. There are sub-system interconnections, a Host-to-AP interconnection, and an electrical service connection. Figure 3-3 shows the necessary connections.

3.5.1 Power Panel to Control Panel

Prior to connecting any of the cabling, the serial number tag on the power panel should be checked to ensure that the power panel received is of the voltage option specified, and that this voltage is compatible with the service voltage supplied to the AP. The fuses and labels (mounted on the line box) should be checked to verify that they are of the proper value as specified in Table 2-1. The control panel serial number tag should then be checked to make sure that its voltage is compatible with the power panel.

While the control panel is out of the rack, feed the nine-pin D sub-miniature female connector P9 through the space left by the removal of the control panel. This connector is labeled P9 on the cable clamp and has six #22 wires coming into it. Now connect plug P9 to socket J9 on the control panel and secure it with the locking screws. After making this connection, take P3 and its cable (originating from the control panel) and drop it through the space in the rack left by the removal of the control panel. Then the control panel should be re-installed as per Section 3.4.4 and the screws tightened. Again make sure that the control panel has .19 cm. (1/16th of an inch) clearance between itself and either the processor or the power panel. Now, from the back of the rack connect P3 (the jones plug) to socket J3 on the line box. Note that if the control panel is mounted above the processor, the two above-mentioned cable runs should be dressed in such a way that they will not hang near the processor's fan filters.



0232

Figure 3-3 AP System Interconnections

3.5.2 Power Panel to Processor

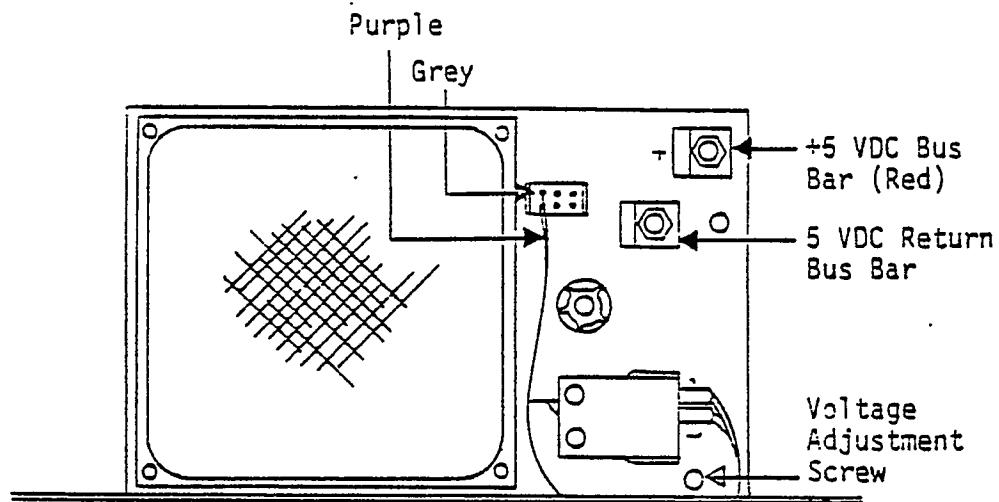
Prior to connecting the cabling from the power panel to the processor, it will be necessary to slide out the processor, remove the front panel and tilt the processor forward on its rack slides. Then remove the front panel by removing the four binding head screws (two on the left side and two on the right side) and lifting the front panel off. Slide the processor forward until the rack slides lock and then tilt the processor, top forward, to an angle of 90 degrees. This will allow easy access to the processor for connecting the cables. The user should exercise caution, now that the front panel has been removed, not to damage the wire-wrap pins on the motherboard.

Now, connect the 25-pin D sub-miniature female connector P11 which comes from the power panel to socket J11 on the I/O bracket at the bottom rear of the processor. From the front of the rack, facing the rear, this will be on the left side of the processor near the fans closest to the rack. After connecting P11, secure it with its locking screws. Next, connect the nine-pin D sub-miniature female connector P1 which originates at the power panel to socket J1 on the processor, and secure it with the locking screws. This socket is just to the right of socket J11.

Next, the host-to-AP cable should be connected to the AP. Due to the number of different types of cabling configurations possible, it is impossible to give specific instructions for this operation. General guidelines will be given, however.

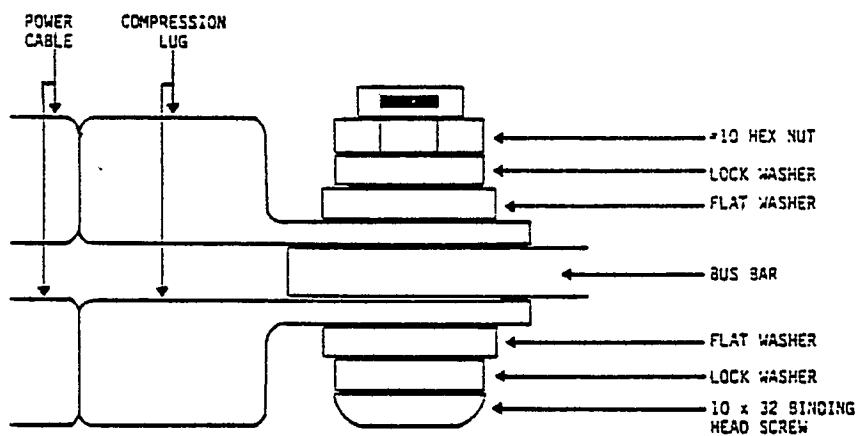
The host-to-AP cabling will connect, on the AP end, to the sockets on the I/O bracket. In most cases, these are keyed to prevent incorrect insertion of the cable. But, if they are not, straightforward markings on the I/O bracket are given (both a connector number and a pin number as a reference). This cabling should be connected to the AP and not connected to the host until after the voltage levels have been checked on the AP as specified in Section 3.6. Further details and directions for the host-to-AP connection are given in Section 3.6.3.

The next connection that must be made is the +5 VDC and the 5 VDC return. With the processor as presently positioned, these four cables will be found on the lower left of the processor nearest the rack. This cabling is properly connected to the processor when shipped and must be connected to the +5VDC/ 150A power supply on the power panel. For this connection, it is recommended that the power panel be pivoted on its hinge out the back of the rack to allow access to the +5 VDC supply. FPS has found it helpful to let the power panel down on a small stool or box. This will support the power panel while the cabling is connected. Now connect these four heavy cables to the main +5 VDC supply. Note that two cables are for the positive terminal and are marked red, and two are for the negative terminal (marked black). With one +5V supply unit, place one cable lug on each side of the appropriate bus-bar and secure it in place with the 10-32 binding head screw, lock washer, flat washer, and nut provided (attached to bus-bar prior to shipment). Place a lock washer and a flat washer between the screw head and the first lug. Insert assembly through the bus-bar. And then place the second lug, flat washer and lock washer over the screw securing it with the hex nut. Place one cable lug on each appropriate bus-bar, and secure it in place as follows. Place a lock washer between the 10/32 binding head screw mounting bracket and secure it with the hex nut. Figure 3-4 shows where the cables are to be mounted on the power supply, and Figure 3-5 shows how they are bolted to the bus-bar. After all cables have been connected, they should be tightened securely and then visually inspected to make sure the positive (+) is not shorted or close to the negative (-) at this connection.



0183

Figure 3-4 +5 VDC Power Supply



0233

Figure 3-5 Bus Bar Bolting Detail

3.6 START-UP

The start-up includes checking the line voltage; applying power to the AP; checking the +5 VDC, -5 VDC, and ± 12 VDC; checking the fans; and re-installing the power panel (returning it to an upright position in the rack).

3.6.1 Line Power

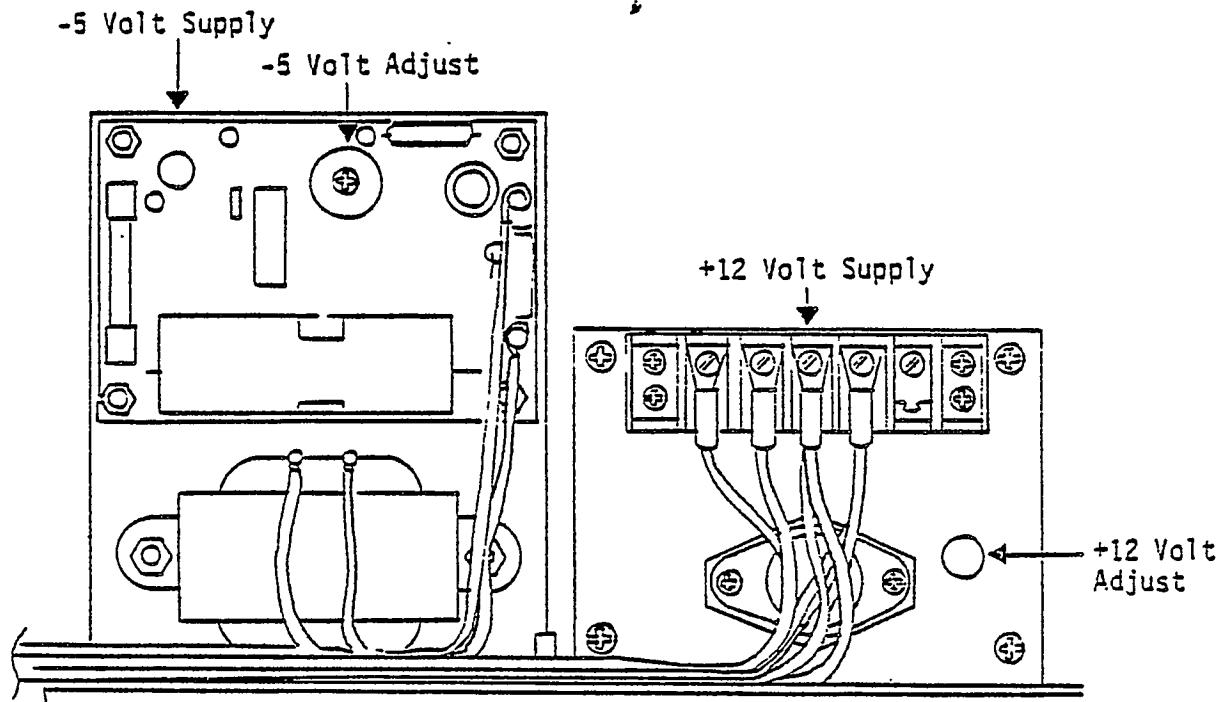
Before plugging in the AP, the line voltage to be used should be checked with a meter to ensure that it falls within the minimum and maximum range specified in the voltage (RMS) column of Table 2-1. The line voltage should also be checked with an oscilloscope to make sure the line voltage is not less than 280 volts peak-to-peak for the 115 VRMS option, 535 volts peak-to-peak for the 208 VRMS option, and 560 volts peak-to-peak for the 230 VRMS option. While checking the line voltage with an oscilloscope, the wave form should be checked to verify that it is really sinusoidal. If the peaks appear flattened, or the peak-to-peak value drops below 280/560 volts when the AP is turned on, then it is possible that a lower impedance power service will be necessary for proper AP operation. (This peak-to-peak value check is detailed in Section 3.6.2. Also, see Table 2-1.)

Place the ON/OFF switch on the control panel in the "OFF" (switch down) position and plug in the power cord. The processor should then be tilted back to its upright position to allow access to the motherboard. The four heavy cables on the motherboard at the bottom of the processor are the +5 VDC and 5 VDC return. The +5 VDC is the uppermost terminal (above the return terminal). A meter which is accurate to a tenth of a volt should be placed on the mother board between the +5 VDC and common. This will be used to monitor the +5 VDC as the power is applied to the unit. Now turn the ON/OFF switch to the "ON" position while watching the meter monitoring the +5 VDC. The fans should come on, the +5 VDC should read slightly over +5 volts, and the supply indicators on the control panel should be lit. If problems occur, the system should be immediately powered down and the line cord unplugged.

3.6.2 DC Power Supplies

After it is possible to power up the AP, the DC voltage supplies should be checked to verify that they are properly adjusted. Again, an accurate voltmeter that can give a tenth of a volt resolution should be used for this verification. The voltmeter should currently be attached to the +5 VDC (from Section 3.6.1) on the motherboard and this voltage should be 5.15 ± 0.05 volts when the ON/OFF switch is in the ON position. If this voltage does not fall within this specified range, it will be necessary to adjust it. There is a screw-type adjustment on the +5 VDC power supply. Figure 3-4 shows this adjusting screw location in the lower right hand corner of the supply. Turning this screw clockwise will increase the voltage of the +5 VDC supply. For a single +5V supply make necessary adjustment and then remove meter leads. Figure 3-4 shows the location of the adjustment screw. Turning the screw clockwise increases the +5 volts. With the power panel folded down the +5V supply on the right has sense wires connected to J1. Refer to Figure 3-4 for J1 location. Adjust the supply for the correct +5V level. (Note the supplies are factory adjusted to be in balance). Disconnect the meter leads.

The -5 volts should then be checked. Directly to the left of the left +5 VDC and down slightly on the motherboard, is a rectangle labeled -5V. The positive lead of the meter should be placed on the AP common (the common of the meter is currently on AP common) and the common of the meter touched to any of the wirewrap pins protruding from the -5V rectangle. This should read $+5.00 \pm 0.05$ volts. If this voltage does not fall within this specification it should be adjusted until it does. This power supply has a screw-type adjustment. Figure 3-6 shows the -5 VDC power supply and adjustment.



SUPPLY	VOLTAGE ADJUSTMENT	TOLERANCE	MEASURING POINT
+5 VDC	+5.15	± 0.05	Backplane
+12 VDC	+12.00	± 0.05	Backplane
-5 VDC	-5.00	± 0.05	Backplane

0183

Figure 3-6 -5 VDC and +12 VDC Power Supplies

Next, the +12 volt supply should be checked to verify that it reads +12.00 + 0.05 volts. The common of the meter should be returned to the common on the AP mother board. After assuring that the meter is to the proper scale, the other probe should be placed on one of the wirewrap pins protruding from the 12V rectangle. This rectangle is in the middle at the top of the mother board. If the voltage does not meet the above specification, it should be adjusted. Figure 3-6 shows the +12 VDC supply and the location of the adjusting screw.

Now that the voltage levels are properly set, visually check to verify that the indicators on the control panel labeled "control power", "+5 power", "+12 power", and "-5 power" are lit. Also, visually check to ensure that all 12 fans are operating. With the oscilloscope again placed across the line voltage (Section 3.6.1), toggle the ON/OFF switch between ON and OFF while monitoring the waveform on the oscilloscope. The line voltage on the scope should not vary more than 35 volts peak-to-peak as the system is switched. If it does, consideration should be given to installing a low impedance line source (see Section 2.3). You may prefer to use a differential oscilloscope input, so the oscilloscope ground can be on the power line safety ground, and the oscilloscope inputs on the true power lines. (Remember, there is a significant voltage drop on both power lines). Hopefully, there will be at least 10 percent more than 280/560 volts peak-to-peak line voltage under load, in order to allow for occasional low input line voltage.

3.6.3 Final Installation

After powering down the AP and the host system, the host-to-AP cabling should be connected to the host in the appropriate manner. Due to the number of different types of cabling configurations (these are host-dependent), only general guidelines will be given. In some cases, such as the FDP-11, this connection requires as little as plugging a connector into a socket on the host. In others, such as the Nova, this requires the insertion of an etch circuit (or wirewrap) card into a slot in the host chassis, and then cabling from this I/O adapter to the AP.

The power panel should then be pivoted back into the rack, the screws replaced and tightened securely. The front panel should then be mounted on the processor with the four screws. Then slide the processor into the rack and secure it with 12 oval headed screws, cup washers, and inserts. Note that if your rack has untapped mounting holes, 12 clip-type speed nuts are supplied.

CHAPTER 4

CHECKOUT AND TEST

4.1 INTRODUCTION

After the AP-120B has been installed into the user's system, the system should be checked to verify that the integration of the AP-120B has not caused interactive-type system problems. Then the diagnostic software provided by FPS should be used to verify the operation of the host-to-AP interface and the AP-120B system.

4.2 HOST SYSTEM

After the power panel and the processor have been installed in the rack, the host system should be turned on. Then the AP-120B should be turned on by placing the ON/OFF switch in the ON position. The user's system should then be checked to verify that the host processor and all peripheral devices function properly. First, the host should be checked to verify that it is able to operate. This can be done by operating some simple program like a bootstrap loader and seeing that it executes properly. Then each peripheral on the system (disk, teletype, terminal, tape drive, etc.) should be checked to see that it functions properly. The diagnostic programs supplied with each of these devices will adequately verify their operation.

4.3 AP-120B DIAGNOSTICS

FPS supplies a diagnostic software package with each AP-120B. This software package consists of four separate test programs. Certain hardware options supplied by FPS, at the customer's request, necessitate the addition of one or more diagnostic programs. If this is the case for your AP-120B, the necessary information will be supplied by FPS on an individual basis. The general diagnostic programs supplied by FPS are called APTEST, APPATH, APARTH, and FIFFT. These four programs not only verify the AP-120B system operation, but are also used in isolating and correcting system deficiencies. This manual will give the user enough information to verify the AP-120B system. If a further description of these programs and their input commands is necessary, FPS Manual #7284 AP-120B Diagnostic Software Manuals should be consulted.

For verifying the operation of the AP-120B, the four diagnostic programs should be run for a short period of time. APTEST should be run first, then APPATH, then APARTH, and then FIFFT. For verification, all four programs use the same input command string. The method of running the test will be to load the test to be run into the host (if the program does not autostart); start the program from the teletype (or terminal); input the command characters RWE; and then input a carriage return. The program will, after a short period of time (this short period of time varies with the different tests, but should not exceed five minutes), return a status character to the teletype (or terminal) indicating proper system action. If an error is detected, the program will inform the user by printing the error message out on the teletype in the appropriate format (defined by FPS Manual #7284).

4.3.1 APTEST

APTEST tests the ability of the host to load and read the various AP-120B registers and memory elements accessible to it through the virtual front panel. APTEST also verifies the DMA-to-DMA transfer capabilities of the system and provides a simple test of the AP-120B formatter. A "T" is typed on the teletype as status to indicate four error-free passes through the program. This test uses random patterns to test the memory elements of the AP-120B and should be run for one hour to verify the system. *Generates T in 55 Sec.*

4.3.2 APPATH

APPATH runs the processor of the AP-120B. Its main function is to verify the various data paths internal to the AP-120B. This test should be run for ten minutes. A "P" is displayed to indicate that the program is running and that the AP-120B is running error-free.

Generates P in 20 Sec.

4.3.3 APARTH

APARTH runs the arithmetic elements of the AP-120B on strings of randomly-selected numbers verifying the operation of these arithmetic elements. The program returns an "A" status to indicate error-free operation. This test should be run for one hour.

4.3.4 FIFFT

FIFFT does forward and inverse Fast Fourier Transforms (FFT) on two different sets of data. First, the program uses data sets with only one non-zero value (an impulse). Then the program uses a randomly-selected data set. After the impulse test is done, the program types "END OF IMPULSE TEST" and then automatically proceeds to compute FFT's on random data. The program then types a "F" as status verifying error-free operation. This test should be run for one hour.

CHAPTER 5

THEORY OF OPERATION

5.1 INTRODUCTION

This chapter provides the detailed theory of operation for the AP-120B and is divided into four major sections: system overview, memory, arithmetic and control.

The system overview (paragraph 5.2) presents a functional system description by delineating the major functional elements, the various data paths, and the flow of data between the AP-120B and the host computer.

The memory description (paragraph 5.3) covers the data pad memory, the main data memory, and the table memory.

The arithmetic description (paragraph 5.4) begins by delineating floating-point numbers and floating-point addition and multiplication. Then, this section covers the floating-point adder (FADD) and floating-point multiplier (FMUL) hardware.

The control description (paragraph 5.5) covers the program source address memory and the scratchpad (SPAD) registers.

It should be noted that the general-purpose interface and the internal interface are both covered in Chapter 6 of this manual.

5.2 SYSTEM OVERVIEW

The architecture of the AP-120B basically consists of a floating-point adder, floating-point multiplier, and a number of memories capable of independent operation. The main system elements are interconnected by multiple buses which also operate independently. This structure allows for the execution of several simultaneous operations without conflict. For example, array indexing, loop counting, and retrieval of data from memory can be performed simultaneously with arithmetic operations on the data. A simplified block diagram of the AP-120B is shown in Figure 5-1.

Before attempting to understand detailed theory of operation of the AP-120B system, it is necessary to have a basic knowledge of the functional elements of the system, as well as to understand how data flows between these elements. The following paragraphs describe the main functional units and the data paths connecting these units. In addition, examples are provided to demonstrate how various types of data flow through the system.

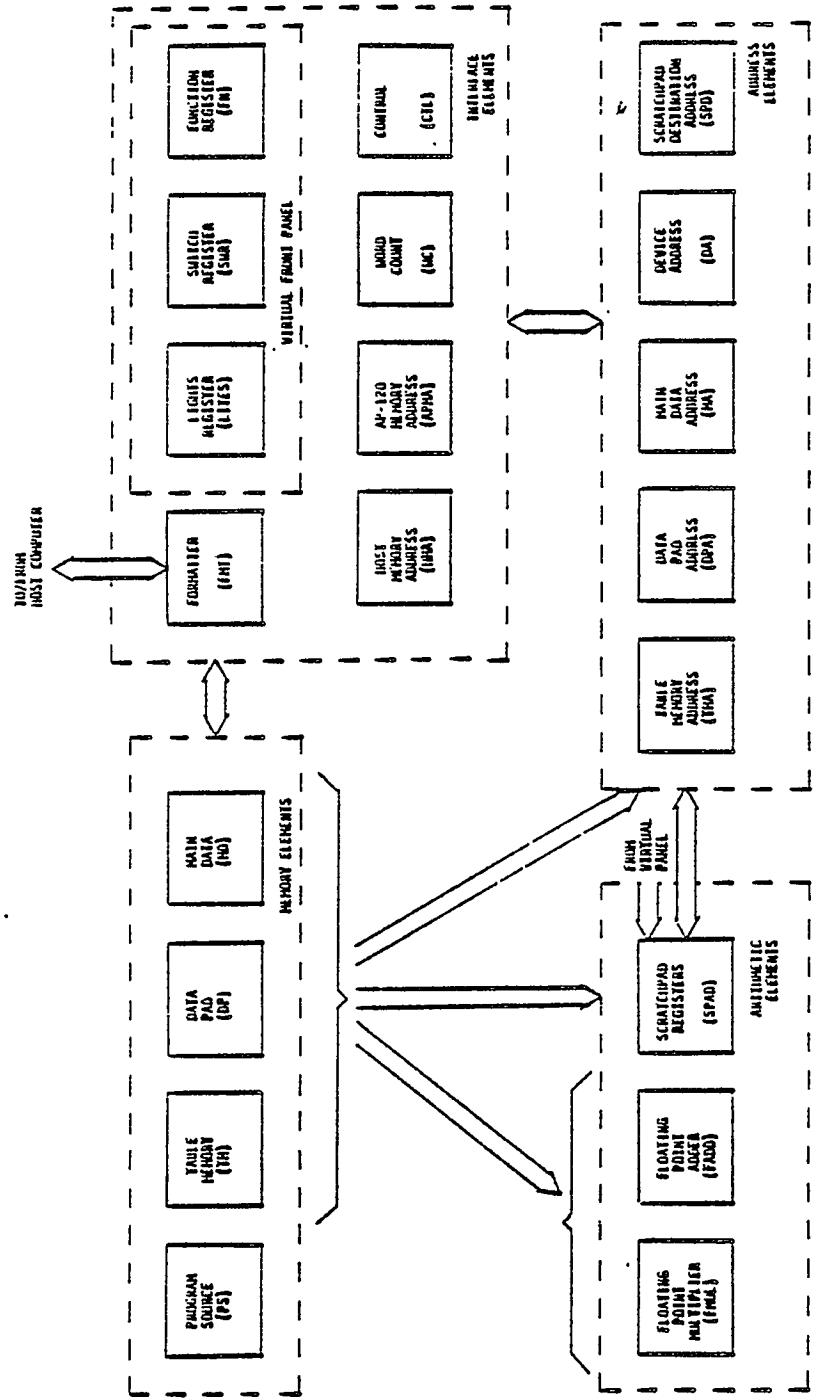


Figure 5-1 AP-120B Simplified Block Diagram

5.2.1 Functional Units

The main units of the AP-120B system can be functionally grouped into five major categories: memory, arithmetic, address, interface and control. Each of these main categories is briefly described in the following paragraphs.

5.2.1.1 Memory Elements

The AP-120B contains four main memory elements which operate independently due to the multiple bus structure. Thus, simultaneous operation of the various memories is possible. The four main memories are:

Program Source (PS)

This memory stores the 64-bit instruction word which controls operation of the AP-120B. The instruction words are retrieved from this memory, decoded, and fed to appropriate logic. A program source address (PSA) register serves as a pointer (or program counter) to the appropriate instruction word.

The PS memory is a bi-polar RAM available in increments of 256 words up to a maximum of 4K words.

Table Memory (TM)

This memory is used for storing frequently-used constants such as sines and cosines applicable to a continuing calculation, complex roots of unity and transcendental values. Because this memory functions independently of other memories, there is no conflict between retrieving data and retrieving constants from storage.

The TM memory is either a ROM or RAM (or both) (ROM 512 words to 64K) and is available in 1K increments up to a maximum of 64K. This memory stores 38-bit words which is the basic data word size of the AP-120B.

Data Pad (DP)

This memory is used for fast access and is divided into two portions: data pad X (DPX) and data pad Y (DPY). Each portion contains 32 accumulators capable of handling 38-bit words. Each portion of the data pad memory is addressed by a combination of an address register and an index field. This index field is a portion of the instruction word.

Main Data (MD)

This is the primary data storage area for the AP-120B which normally handles the main data from the host computer. It is 38 bits wide (10 bits for exponent, 28 bits for mantissa).

The main data memory is available in 8K increments, expandable up to 64K of directly-addressable memory. A page select option is available which permits memory expansion of up to 1.2 megawords.

The MD memory is a MOS memory and is available in either a standard or fast version.

5.2.1.2 Arithmetic Elements

The AP-120B contains three arithmetic elements. Two of these elements are specifically designed for floating-point operations. These two elements, which are the adder (FADD) and the multiplier (FMUL), receive inputs from the various memory elements. The third element, known as the scratchpad (SPAD), is used for integer arithmetic and also performs various overhead functions.

**Floating-point
Adder (FADD)**

This adder performs either addition or subtraction along with other logic and formatting operations on the contents of one or two input registers of the adder. Although each operation requires two machine cycles, a "pipeline" method is used so that a new input may be entered into the pipeline stream every cycle. In effect, it takes 333ns to perform an add operation, but a new add can be started every 167ns.

The adder can receive 38-bit data from the data pad (DP), table (TM) or main data (MD) memories as well as from the output of the multiplier or the adder itself. This latter path is extremely useful for operations requiring an accumulation of the sum. The sum moves directly from the output of the adder to the input without the necessity of first moving into an accumulator.

The adder output may be sent to the multiplier, to the data pad (DP), to main data memory (MD), or back to the input of the adder. Results from the adder are normalized, rounded, error-detected values.

Because of the adder's independent operation and the multiple bus structure of the system, the adder can operate simultaneously with the multiplier and the scratchpad (SPAD) arithmetic units.

Floating-point Multiplier (FMUL)

This multiplier calculates the product of the values in the two input registers of the multiplier. Each multiply operation requires three machine cycles. During the first cycle, the 56-bit product of the two 28-bit fractions is partially completed. During the second cycle, the product of the fractions is completed. During the third and final cycle, the mantissa product is normalized and rounded.

The floating-point multiplier, like the adder, also uses a "pipeline" method, so that a new multiply operation can be initiated every cycle (every 167ns). When a multiply operation begins, the two values to be multiplied are loaded into the input registers and previous multiplier inputs are pushed down the pipeline to buffers. One cycle later, the result from the last buffer is available for storage or use. Thus, although a new operation may begin every 167ns, the result is not ready until 500ns after the inputs have been loaded.

The multiplier can receive 38-bit inputs from the data pad (DP), table (TM), or data (MD) memories, as well as from the output of the adder or the multiplier itself. This latter path is extremely useful for operations requiring an accumulation of products from the multiply operation because the product can be fed back into the multiplier without the necessity of first moving into an accumulator.

The multiplier output may be sent to the adder, the data pad memory (DP), main data memory (MD), or back to the input of the multiplier. Results from the multiplier are normalized, rounded, and error-detected.

Because of the multiplier's independent operation, and the multiple bus structure of the system, the multiplier can operate simultaneously with the adder and the scratchpad (SPAD) arithmetic units.

Scratchpad (SPAD)

The scratchpad arithmetic unit performs the integer address indexing, loop counting, and control functions required by the program. These operations are performed in parallel with operations performed by the other arithmetic units (FADD and FMUL). In effect, the scratchpad performs overhead functions while the other units are performing computational functions.

The scratchpad basically consists of sixteen 16-bit registers, an ALU, and a shifter. In effect, the SPAD may be thought of as a small minicomputer of the PDP-11 or NOVA type.

The scratchpad can receive 16-bit inputs from the host computer (through the virtual panel) or from the data pad bus.

The output of the ALU in the scratchpad can be used as is, shifted left or right once, or shifted right twice. In addition, bit-reversal hardware is included in the SPAD to provide the bit swapping required to access scrambled data after completion of a FFT.

The scratchpad output may be fed back to the specified SPAD destination register or may be sent to: the table memory address register (TMA), data pad address register (DPA), memory address register (MA), device address register (DA), the AP status register (APSTAT), the data pad bus or the panel bus.

5.2.1.3 Address Elements

The AP-120B system contains five registers used for addressing the system's memories, peripheral devices, and scratchpad registers. Address registers associated with DMA transfers between the AP-120B and peripheral devices are covered in paragraph 5.2.1.4. The five address registers are:

Table Memory Address Register (TMA)

This register contains an address which serves as a pointer to the location containing the desired constant in table memory. The address in the register can be changed by incrementing or decrementing the register, or by loading in a new value from the scratchpad unit or the data pad bus.

Data Pad Address Register (DPA)

This register contains an address which is combined with an index value in the instruction word to access a specific location in the data pad. This register may be thought of as a base address register or stack pointer. Four fields in the instruction word permit accessing of either data pad X (DPX) or data pad Y (DPY) for either reading or writing.

Memory Address Register (MA)

This register contains an address which is a pointer to the desired location in the main data (MD) memory. The register may be incremented or decremented. Or a new value may be loaded in from the scratchpad unit or the data pad bus.

Device Address Register (DA)

This register contains the address of the external device that is to be used with the AP-120B. For example, the address in this register might be used to access a disk or tape unit for DMA transfers.

Scratchpad Destination This register contains the address of the scratchpad destination. In other words, the address of the scratchpad memory location or register that is to receive the output of the scratchpad (SPAD).

5.2.1.4 Interface Elements

The interface unit is designed to permit communication between the AP-120B and the host computer. This interface consists of two basic parts: a simulated front panel and a direct memory access (DMA) control.

The simulated front panel (or "virtual" front panel) permits the host computer to examine or modify the internal AP-120B registers. It can be used to load programs as well as to start or stop program execution. The three elements that comprise the simulated front panel are briefly described below:

Switch Register (SWR) This register is used by the host computer to load both addresses and data into the AP-120B. The host computer can either write into or read from this register. The program within the AP-120B can read the register but cannot load it.

This register is similar to a normal computer switch register except that information is loaded or read under computer control rather than entered by means of front panel switches.

Lights Register (LITES) This register simulates the front panel lights in order to display the contents of internal AP-120B registers.

The lights register can be loaded, but not read by the AP-120B. The host computer can only read this register.

Function Register (FN) This register provides the remaining front panel controls required by a computer. Such controls are: stop, start, deposit, examine, step, continue, etc. Each function is selected by setting the appropriate bit in the control register.

The function register can be loaded or read by the host computer and by the AP-120B.

The DMA portion of the interface permits block transfers of data from the host computer to the AP-120B or vice versa. Because data may be received from a variety of host computers, each using a different word length and word format, this portion of the interface includes a formatter which serves as a buffer between the two memories to ensure that data is received in the form required by the AP-120B or the host. The DMA portion of the interface also includes the four registers required for block DMA transfers. It should be noted that on host computers supporting channel type DMA, the WC and HMA registers are not implemented. The formatter and the four registers are briefly described below:

Formatter (FMT)

The formatter converts the input word to the 38-bit format required by the AP-120B, or vice versa. Conversions made by the formatter are dependent on the type of host being used. The appropriate conversion is selected from the host by the control register described below. In most implementation, the interface supports four format types. The four conversions are:

- 32-bit transparent
- 16-bit integer
- host floating-point
- IBM floating-point

Host Memory Address Register (HMA)

This register serves as a pointer to host computer memory locations involved in the DMA transfer. The HMA register always operates in either the auto-increment or auto-decrement mode (as specified by the control register) so that consecutive memory locations are addressed.

Array Processor Memory Address Register (APMA)

This register serves as a pointer to AP-120B main data memory locations involved in the DMA transfer. The APMA register always operates in either the auto-increment or auto-decrement mode (as specified by the control register) so that consecutive memory locations are addressed. It should be noted that DMA transfers can only be used with the main data (MD) memory.

Word Count
Register (WC)

This register keeps track of the number of words transferred during a DMA block transfer. The register is initially loaded with the number of words that are to be transferred. Each time a word is moved, the register is decremented by 1. When the register reaches zero, it indicates that the DMA block transfer of data is complete. A control bit in the AP-120B control register is set when the word count reaches zero. As far as the host is concerned, this is a read-only bit that can be monitored to determine when the DMA transfer is complete.

Control Register (CTL)

The control register controls both the DMA and interrupt functions of the host interface. This register is a combination control and status register. The control bits select both the direction and type of transfer desired (either DMA or program control), select the type of data format to be used, and determine whether the host memory address (HMA) and array processor memory address (APMA) registers are to be auto-incremented or auto-decremented. The status bits are used to provide information pertaining to the transfer. Such bits include word count, data late, and busy.

5.2.2 Data Paths

The AP-120B uses multiple buses to interconnect the various memory, arithmetic, address, interface, and control elements of the system. By using multiple buses, data can be moved around the AP-120B on one specific path while other data can simultaneously be moved around on a completely different path. This structure greatly reduces the time required for processing information because it allows various system elements to handle tasks in parallel, rather than sequentially. For example, a single instruction can be used for loading both inputs into the adder and both inputs into the multiplier. In other words, a particular instruction might apply a value from table memory (TM) and a value from data pad X (DPX) to the multiplier and, at the same time, apply a value from main data memory (MD) and the product from the multiplier (FMUL) to the adder. This simultaneous operation is possible because all but one of the buses are either single-source or single-destination buses. It should be noted that the example given above is not indicative of all instructions. Some instructions may perform even more tasks in a single cycle.

Although a specific bus may have multiple sources or multiple destinations, data can never follow more than one path at any given time. In other words, "data path" is not the same as "bus". Only one input and one output is enabled at any point in time on a specific bus. Data paths are set up by means of drivers and multiplexers. Assume, for instance, that a particular bus has only one source, but two possible destinations. The output side of the bus is normally connected to a multiplexer (a 2:1 multiplexer in this example) which is used to select the desired destination element.

The following paragraphs describe each of the AP-120B buses. These buses are: arithmetic buses, data pad bus, main data buses, panel bus, host data bus, and input/output bus.

5.2.2.1 Arithmetic Buses

There are six separate and independent 38-bit buses that are used to load the floating-point multiplier (FMUL) and the floating-point adder (FADD). The multiplier and adder buses are described separately below.

The floating-point multiplier (FMUL) has two input registers (M1 and M2). Each register can be loaded from either one (but not both) of two buses. Register M1 can be loaded from the M1 bus (M1BS) or the floating-point multiplier output bus (FM). Register M2 can be loaded by the M2 bus (M2BS) or from the floating-point adder output bus (FA).

Multiplier Buses (M1BS and M2BS)

Multiplier bus M1 (M1BS) is a single-destination bus connected through a 2:1 multiplexer to the M1 input register of the multiplier (FMUL). The multiplexer selects data from either the M1BS or the FM bus to be loaded into the multiplier.

Data on the M1BS can come from table memory (TM), data pad X (DPX), or data pad Y (DPY) depending on which of the associated drivers are enabled at the time. Data can never come from more than one source at one time.

Multiplier bus M2 (M2BS) is also a single-destination bus connected through a different 2:1 multiplexer to the M2 register of the multiplier (FMUL). This multiplexer selects data from either the M2BS or the FA bus for loading into the multiplier.

Data on the M2BS can come from data pad X (DPX), data pad Y (DPY), or main data memory (MD) depending on which of the associated drivers are enabled at the time.

Note that both buses connect to the data pad while only the M1 bus connects to table memory and only the M2 bus connects to main data memory. Nevertheless, this bus structure allows access to all data in the memories.

For example, if it is desired to multiply a value from MD with a value from TM, the equation might be:

$$MD \times TM$$

Although it is not possible to feed data from MD into multiplier register M1, nor data from TM into register M2, the equation is equally valid if written:

$$TM \times MD$$

This latter case represents valid data paths because TM can be applied to M1 and MD applied to M2.

The floating-point adder (FADD) also has two input registers (A1 and A2). Each register can be loaded from either one (but not both) of two buses. Register A1 can be loaded from the A1 bus (A1BS) or the floating-point multiplier output bus (FM). Register A2 can be loaded from the A2 bus (A2BS) or from the floating-point adder output bus (FA).

Adder Buses
(A1BS and A2BS)

The combination of the two adder buses permits the floating-point adder (FADD) to be loaded from table memory (TM), data pad memory (DP), or main data memory (MD).

Because both the adder and the multiplier receive data from the same sources, the structure of the adder buses is the same as the structure of the multiplier buses. That is, adder bus A1 (A1BS) is connected to both data pad sections (DPX and DPY) and the table memory (TM), while adder bus A2 (A2BS) is connected to both data pad sections (DPX and DPY) and the main data memory (MD).

It is important to note, however, that an adder bus (either A1BS or A2BS) can receive data from only one source at a time depending on which of the associated drivers are enabled.

Both adder buses are single-destination buses. Bus A1, for instance, can receive data from any one of three sources (DPX, DPY, or TM), but the data is always applied through a 2:1 multiplexer to the A1 input register of the adder. The multiplexer is used to select either the A1BS or the FM bus as the input to register A1.

Bus A2 is connected to a different 2:1 multiplexer that selects either the A2BS or the FA bus as the input to register A2 of the adder.

Although the adder buses are connected in a manner similar to the multiplier buses, it must be remembered that they are separate and independent buses. The purpose of having separate buses with identical paths is to allow data to be routed to the adder input registers independent from and simultaneously with the routing of data to the multiplier input registers.

The floating-point multiplier (FMUL) and the floating-point adder (FADD) each have a separate output bus for routing information to other parts of the system. Both buses are described below:

Multiplier Output
Bus (FM)

This is a single-source, multiple-destination bus. The multiplier output can thus be fed back to the multiplier input (M1 register only), or to the adder input (A1 register only), or to either portion of the data pad (DPX or DPY), or to main data memory (MD).

Although the bus is connected to multiple destinations, data can only follow one path at any given time. This is implemented by using various multiplexers to select only one data path at a time. For example, the 2:1 multiplexer on the multiplier (FMUL) input selects either the data on this bus (FM) or data from the M1BS for loading into register M1. A similar multiplexer on the adder (FADD) input selects either data from this bus (FM) or from the A1BS for loading into register A1. Two 4:1 multiplexers (one is located on the input of data pad memory and the other is located on the input of main data memory) are used to select the appropriate input data for the DP and MD memories.

The structure of the FM bus allows the multiplier product to be fed back to the multiplier or sent to the adder for iterative arithmetic calculations or stored in either the data pad (DP) or main data (MD) memories.

Adder Output Bus (FA)

The adder output bus is also a single-source, multiple-destination bus that follows the same general path as the multiplier output bus. That is, data on the FA bus can be fed back to the adder or multiplier, or stored in the data pad (DP) or main data (MD) memories. The same type of multiplexing scheme is used to ensure that data only follows one path at any point in time.

It should be noted that the adder output bus (FA) is connected to multiplier input register M2 and adder input register A2.

The purpose of having two separate buses with similar paths is to allow output data from the adder to be routed independently and simultaneously with the routing of output data from the multiplier.

5.2.2.2 Data Pad Bus

The data pad bus (DPBS) is the only bus in the AP-120B that has both multiple sources and multiple destinations. It is a 38-bit parallel bus that is used primarily to interconnect the memory elements with other elements within the system. Because data can flow in either direction on the bus, and because the bus connects a number of system elements, each portion of the bus is discussed separately to allow for ease of understanding. It must be remembered, however, that data can flow on only one path at any given point in time.

Memory Outputs

The DPBS receives the output of table memory (TM) through the TM register, the output of data pad X (DPX), the output of data pad Y (DPY), and the output of main data memory (MD) through the MD register.

?

Although the bus can receive data from any of the above memories, it can receive data from only one at a time depending on which drivers and/or registers are enabled.

At this point, data on the bus can be sent to one of five places:

- back to the input side of the memories
- to the address registers (TMA, DPA, etc.)
- to the panel bus (PNL)
- to the interface registers (HMA, WC, etc.)
- to the input/output bus (IOBS)

Each of the five bus destinations mentioned above is discussed separately below, along with the elements that set up the specific data path.

Memory Inputs

Information on the data pad bus can be used as an input for any one of the memory elements.

Data on the bus (DPBS) is applied through a 4:1 multiplexer and the MI register to the input of main data memory (MD).

Data is applied from the DPBS through a 4:1 multiplexer to data pad X (DPX) and data pad Y (DPY). Other inputs to this multiplexer come from the arithmetic output buses (FA and FM).

Data is applied from the DPBS bus to the IOBS and then through the TMI register to table memory (TM).

The various multiplexers and registers are used to select the appropriate data path so that data on the data pad bus (DPBS) is applied to the desired memory element.

Data can never be loaded into more than one memory simultaneously when using the data pad bus (DPBS).

Address Register
Inputs

The information that is on the data pad bus (DPBS) can be used to load any one of various address registers as well as the AP-120B status register. That is, data on the bus loads one of the registers with the required address.

A 2:1 multiplexer is used to select either data from the DPBS or another source (the output of the scratchpad) as the input to the address register. An enable signal on the appropriate register then loads the address from the bus into the register.

The registers that can be loaded from the data pad bus (DPBS) are:

table memory address register (TMA)
data pad address register (DPA)
main data address register (MA)
I/O device address register (DA)
scratchpad destination address
register (SPD)
AP-120B status register (APSTAT)

Interface Register
Inputs

Information on the data pad bus (DPBS) can be used to load the various registers involved in DMA transfers. These registers are:

host memory address register (HMA)
word count register (WC)
AP-120B memory address register
(APMA)
control register (CTL)

Although all of these registers must be loaded in order to perform a DMA transfer, they must be loaded one at a time because data from the bus can only be used with a single destination.

The data on the data pad bus (DPBS) is fed to a 2:1 multiplexer which selects either the DPBS data or host data (HDBS) as the input to the registers. An enable signal on the appropriate register then loads the data from the bus into the register.

Panel Bus Input

Data on the data pad bus (DPBS) can be fed to the panel bus (PNL) in order to load the LITES register which can then be read by the host computer. In effect, this permits the host to read the contents of various AP-120B registers because the AP-120B can use the PNL bus to move the contents of its switch register (SWR) of address registers (TMA, DPA, etc.) into the LITES register.

I/O Bus Input

The data pad bus (DPBS) is connected to the input/output bus (IOBS). Because of this structure, data from the host computer can be used to load any of the elements connected to the data pad bus (DPBS) such as the AP-120B memory elements, interface registers, address registers, etc.

Note, however, that the input to each element is controlled by a multiplexer or enable signal so that there is always a single data path through the system even though the bus itself has multiple source and destination lines.

5.2.2.3 Main Data Buses

The two main data buses (MD₁ bus and MDI bus) are both 38-bit, single-source, single-destination buses.

The main data input bus (MDI) receives data from the formatter (FMT) and applies it through a 2:1 multiplexer to the main data memory (MD). The main data bus (MD) performs the reverse function in that it takes the output of main data memory and applies it through a multiplexer to the formatter.

The multiplexers are used to select the appropriate data path. For example, the input to main data memory (MD) could come from either the MDI bus or from the output of the MI register.

Because of the structure of these two buses, data can be retrieved from main memory and sent to the formatter which can then send it out to the host computer or apply it to the host data bus (HD) in order to load one of the four registers involved with DMA transfers (HMA, WC, APMA, and CTL).

The bus structure also allows data from the host data bus (HD) to be applied through the formatter directly to the input of main data memory. This means that data can be loaded into main data memory from the host computer.

5.2.2.4 Panel Bus

The panel bus (PNLBS) is associated with the AP-120B simulated front panel which is normally referred to as the "virtual" front panel. The bus is a 16-bit single-destination bus. The destination is the LITES register of the virtual front panel.

The panel bus can receive data from any one of the following elements: the data pad bus (DPBS), the switch register (SWR), the output of the scratchpad ALU/shifter function (SPFN) or the output of one of the various address registers (TMA, DPA, MA, DA, etc.). Because data can only be received from one source at any given time, a series of drivers are enabled to gate the appropriate data on to the panel bus.

In effect, the contents of any one of the above registers or the data on the data pad bus (DPBS), can be fed into the LITES register by means of the panel bus. The host computer can then read the LITES register to examine the contents of the appropriate register.

5.2.2.5 Host Data Bus

The host data bus (HD) allows the host computer to control the virtual front panel as well as the registers involved in DMA transfers. By using the host data bus (HD), the host can either load or read the switch register (SWR), the function register (FN) or any of the registers associated with DMA transfers (HMA, WC, APMA, and CTL). In addition, the host can read, but not load the LITES register.

It should be noted that the host can only perform one function at a time. For example, if the host is loading a register, data is applied to the host data bus (HD) which is applied through a 2:1 multiplexer to the four registers used for DMA transfers. This multiplexer selects data from either the host data bus (HD) or the data pad bus (DPBS) as an input to the four registers. However, only the register that has received the appropriate enable signal is loaded. When reading data from a register, the register output passes through a 4:1 multiplexer and some drivers to the host data bus (HD). The multiplexer places the contents of the appropriate register on the bus.

When reading the LITES register, the host data bus is connected directly to the host computer. However, when reading or loading any of the other registers, data on the host data bus always passes through the formatter (FMT). Depending on operation of the formatter, the host data bus is connected to either the host computer or to the main data bus (MD) of the AP-120B.

5.2.2.6 Input/Output Bus

The input/output bus (IOBS) is a 38-bit, multiple-source, single-destination bus that permits communication between the interface portion of the AP-120B and other internal elements. In all instances, data on the bus is fed to the formatter (FMT). The bus receives input data from any one of the four registers used in DMA transfers (HMA, WC, APMA, and CTL) or from the data pad bus (DPBS). The output lines from all four registers are fed to the bus through a 4:1 multiplexer which selects data from the appropriate register and applies it to a series of drivers. When these drivers are enabled, the data is then placed on the IOBS.

Because of the bus structure, data from the memory elements can be placed on the data pad bus (DPBS) which is connected to the IOBS, and then sent through the formatter to the host data bus (HD) in order to load the virtual front panel (SWR and FN registers) or the four registers used in DMA transfers. Note, however, that data can only follow one path at a time which means that only one register can be loaded at a time. The specific path required is selected by means of multiplexers, drivers, and enable signals on the registers as previously described.

Data from the host can be fed through the formatter (FMT) to the input/output bus (IOBS) provided the appropriate enable signals are applied to the formatter. Because the IOBS is connected to the DPBS, this data can be used to load any of the elements connected to the data pad bus (DPBS) such as the AP-120B memory elements, the interface registers, etc.

5.2.3 Examples of Data Flow

In order to understand the AP-120B bus structure, it is helpful to know how data flows through the system when certain tasks are being performed. The following paragraphs present examples of how data flows along the various buses during specific operations. The operations covered are: host to AP-120B, DMA transfer and AP-120B to host DMA transfer.

5.2.3.1 DMA Transfer - Host Computer to AP-120B

The first step in performing any DMA transfer is to set up the appropriate address and word count registers used to establish the parameters of the block transfer. Thus, the following registers must be loaded:

Host Memory Address Register (HMA)	Must contain the starting address of the block of data to be transferred from the host computer.
AP-120B Memory Address Register (APMA)	Must contain the starting address of the AP-120B memory block that is to receive the data from the host.
Word Count Register (WC)	Must contain a value indicating the number of words to be transferred.
Control Register (CTL)	Must contain a value that will set the required control bits such as auto-increment or auto-decrement and start.

The host computer loads each of the above registers by sending the data through the formatter to the host data bus (HD). The registers can be loaded from this bus.

Once the registers have all been loaded, the AP-120B main data memory (MD) must be informed of the starting location where it is to receive the data. Thus, the contents of the APMA register is applied to the input/output bus (IOBS) and fed through the formatter to the MDI bus. In effect, the contents of APMA serves as an address pointer. The contents of the host memory address register (HMA) is fed through the IOBS and the host data bus (HD) to the host computer so that the proper starting address in the host's memory can be selected.

The next step in performing a DMA transfer is to move data from the host to the AP-120B. Data from the host is applied through the formatter to the MDI bus and then loaded into the main data memory (MD) location specified by the APMA register.

The final step in the transfer is to monitor the word count and stop the DMA transfer when the word count reaches zero. This indicates that the desired number of words has been transferred. There are two methods for monitoring the word count.

One method of reading word count is for the host computer to monitor the LITES register. This is possible because the contents of the word count register (WC) are applied through the IOBS and the formatter to the data pad bus (DPBS), then to the panel bus (PNLBS) and finally to the LITES register. Because the output of the LITES register is connected to the host data bus (HD), the host computer can read this register to determine if the word count has reached zero or not.

The second method of reading word count is the one normally used. With this method, the host computer monitors the contents of the CTL register. Whenever the word count register (WC) reaches zero, an appropriate bit is set in the CTL register to indicate that the DMA transfer is complete. The output of the CTL register is connected to the host data bus (HD) and can therefore be read by the host computer.

Once the host computer knows that the word count is zero and the transfer is complete, it sends a new value on to the host data bus (HD) in order to load the CTL register with the appropriate bits to stop the transfer.

5.2.3.2 DMA Transfer - AP-120B to Host Computer

A DMA transfer from the AP-120B to the host computer is identical in concept to a transfer from the host. However, the buses used in this transfer are different.

Because the four registers involved in the transfer all receive inputs from the data pad bus (DPBS), any register can be loaded from any of the AP-120B memory elements. Once the registers are all loaded, different buses are used to send the register contents to appropriate locations.

The AP-120B memory address register (APMA) contents pass through the IOBS and the formatter to the MDI bus so that it can point to the starting address in main data memory (MD).

The contents of the host memory address register (HMA) are fed through the IOBS and the formatter to the host data bus (HD) and out to the host computer so that the proper starting address in the host's memory can be selected.

The contents of the word count register (WC) is applied through the IOBS and the formatter to the data pad bus (DPBS), then to the panel bus (PNL) and finally to the LITES register. Because the output of the LITES register is connected to the host data bus (HD), the host computer can read this register to determine if the word count has reached zero. However, indication of a completed transfer is normally found by reading the CTL register which is also fed to the host data bus (HD). When word count reaches zero, the appropriate bit in the CTL register is set to indicate that the transfer is complete.

The data to be transferred out of the AP-120B is retrieved from main data memory (MD), applied to the MD bus and then sent through the formatter to the host computer on the host data bus (HD).

5.3 MEMORY

The following paragraphs provide detailed descriptions of the various AP-120B memory elements. These memories are: data pad memory, main data memory, and table memory. It should be noted that the program source memory is not described here but is covered in paragraph 5.5.

5.3.1 Data Pad

The data pad system consists of two high speed accumulators (data pad X - DPX and data pad Y - DPY), a data pad address register (DPA) and four indices (XR, YR, XW, and YW). See Figure 5-2 for a block diagram of the data pad system.

The data pads can be used as sources of data for the M1, M2, A1, A2 registers and the data pad bus (DPBS). The data pads can be used as destinations for data from FM, FA, and DPBS.

This discussion of data pad will first treat the hardware used to implement the accumulators and then discuss the addressing logic.

5.3.1.1 Accumulator Hardware

Data pad consists of two high speed accumulators called data pad X (DPX) and data pad Y (DPY). Each accumulator is 38 bits wide and each contains 32 locations. Thus, there are 64 accumulators available for data storage.

The data pads physically reside on two etch circuit cards 200 left and 200 right. The ECBs are identical and may be interchanged. However, these are the only two interchangeable boards in the AP-120B.

Figure 5-3 presents DPX in a block diagram form. The DPY hardware is exactly the same as the DPX hardware with the exception that different control signals (YIA, YIB, YCLKE*, Ynn, and SAMEY) are used on DPY. Figure 5-3 shows that DPX consists of an input selector and latch (XI MUX, X BUF), the memory (HIGH X STACK, LOW X STACK) and an output latch (X REG).

5.3.1.2 Input Hardware

The input hardware to DPX consists of a 4-to-1 multiplexer and a holding register. The 4:1 mux, XI MUX, is controlled by two control signals XIA and XIB. XIA and XIB are generated from the control buffer and are the decode of program source bits 32 and 33. Only three of the four possible input choices to XI MUX are used. The three possible input sources are the output of the floating point adder (FAnn*), the output of the floating point multiplier (FMnn*) and the data pad bus (DPBSnn*).

The three inputs allow the DPX buffer register, X BUF to be loaded from three different sources depending on the state of XIA and XIB. !DPLCLK is a free-running 167nsec clock that is in phase with the system clock. XCLKE* is generated from the control buffer and is the logical "OR" of program source bits 32 and 33. X BUF is loaded when XCLKE* is true and on the low-to-high transition of !DPLCLK. Figure 5-4 shows this relationship between XCLKE* and !DPLCLK.

5.3.1.3 Memory Elements

The memory element used in DPX is a 74S189. This is a bi-polar RAM that is four bits wide by 16 elements deep. In order to make a 32 location by 38-bit wide memory element it was necessary to parallel 10 of these chips to give the appropriate width and then stack two of these rows together to give the 32 locations. The two different rows are shown in Figure 5-3. They are called HIGH X STACK and LOW X STACK.

Figure 5-5 shows an expanded view of the data pad memory element logic. From this diagram it should be noted that the most significant bit of the address (X01) is inverted and used as the memory enable on the HIGH X STACK.

It should be noted that DPX (and DPY) can be read from and written into in the same 167ns instruction cycle. This is accomplished by using the read address ($DPA+XR \rightarrow X_{nn}$) during the first portion of the cycle and the write address ($DPA+XW \rightarrow X_{nn}$) during the second portion of the cycle. In order to read from a register the address must be valid and the write enable (WE) must be true (low). After a short delay (for setup) the data may then be clocked into the XREG. In order to write into a register, the address must be valid, the write enable (WE) must be true (low) and the memory enable (ME) must be true (low). When this happens, the data from the XBUF is written into the register addressed by X_{nn} .

5.3.1.4 Output Hardware

The data pad output hardware consists of the output holding register (X REG and Y REG) and five 2:1 multiplexing drivers. Figure 5-6 presents the data pad output hardware in block diagram form.

The output holding register for DPX is X REG. This latch is a 2:1 multiplexing latch. The inputs to the latch are either the output of the stack registers or the output of the XBUF. If a register is written in one cycle and the next instruction cycle attempts to read the same register, the SAMEX signal goes true (high) and the data in the X BUF is loaded into the X REG. This is a special case. In every other case the data is read from the stack and loaded into the X REG on the low-to-high transition of !DPLCLK*. Note that !DPLCLK* is a free running, 167ns clock that is 180 degrees out-of-phase with the system clock.

The contents of the X REG (Y REG) can be enabled onto DPBS, A1BS, M2BS, M1BS and A2BS. It should also be noted that the drivers are 2:1 multiplexing drivers. Thus, the appropriate input must be selected to the output with Y2DP, Y2A1, Y2M2, Y2M1, and Y2A2, and the data must be enabled onto the appropriate bus with DP2DPE*, DP2A1E*, DP2M2E*, DP2M1E* and DP2A2E*.

5.3.1.5 Addressing

Data pad addressing is accomplished by adding one of four indices to the contents of the data pad address (DPA) register. The index is a three-bit wide biased field in the instruction word. Thus, we can add any value between -4 and +3 to our DPA register. There is a separate index for reading DPX (XR), writing DPX (XW), reading DPY (YR) and writing DPY (YW). Either XR or XW is selected through a 2:1 multiplexer to become the address for DPX depending upon whether DPX is being read or written. Similarly, YR and YW are used for DPY.

5.3.1.6 Address Logic

The data pad address logic, shown in Figure 5-2, is physically located on ECB number 211. The logic can load DPA, set DPA, increment or decrement DPA.

Initially, the data pad address is set by 2:1 multiplexer. This multiplexer may select: SP+DP (SPAD OR Data Pad), or increment or decrement the current address. The output of DPA is incremented or decremented by adder AA. The output of 2:1 multiplexer is also supplied to adders A2 and A3, where the current address is added to XR and YR. The output of adders A2 and A3 are routed to latch A4 and A5 and then to 2:1 multiplexers A6 and A7. The multiplexers (A6 and A7) provide the X read and Y read addresses during the first half of the clock cycle.

The XW and YW (X write and Y write addresses) are coupled to latch B1 and B2 and then to adders B3 and B4. XW and YW are added to DPA by B3 and B4 and then routed to latch B6 and B7, where they are available as outputs.

Latches B6 and B7 delay the XW and YW addresses so they are available during the second half of the clock cycle.

The 2:1 multiplexer (B5) is used with the value field. YW is replaced with XW.

Comparators A8 and B8 detect a read instruction in the next cycle after the write instruction to the same register. If the addresses are the same, SAMEX or SAMEY is generated, thereby inhibiting the output of the stack register and selecting the output of the X buffer as the input to the X register.

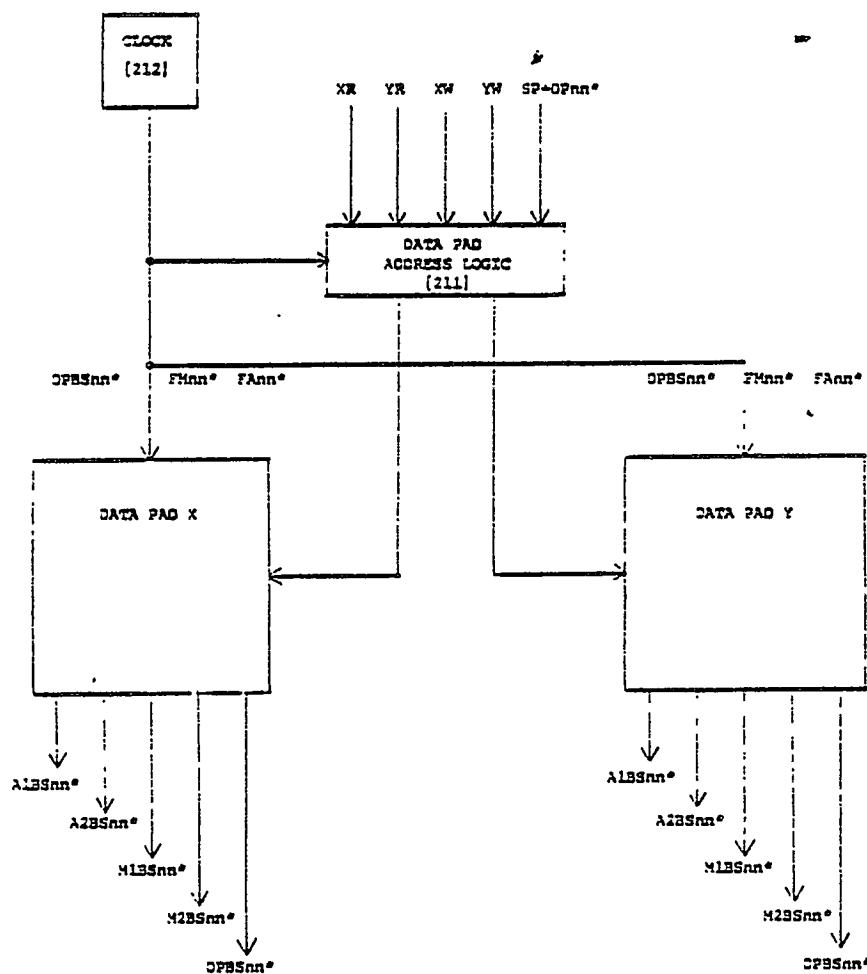


Figure 5-2 Data Pad System Block Diagram

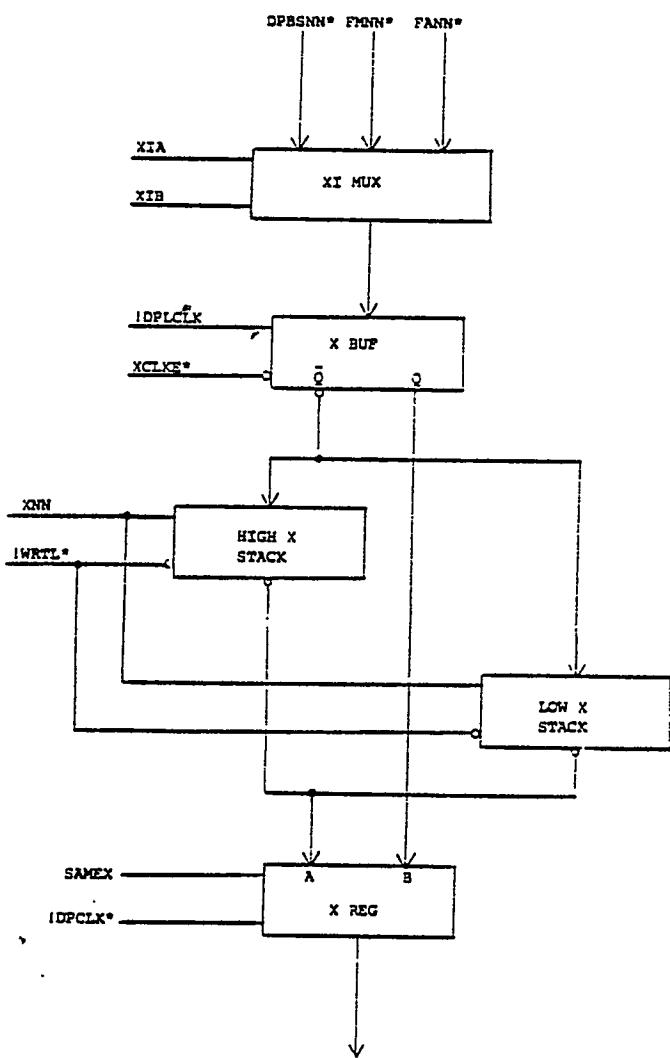


Figure 5-3 Block Diagram of DPX

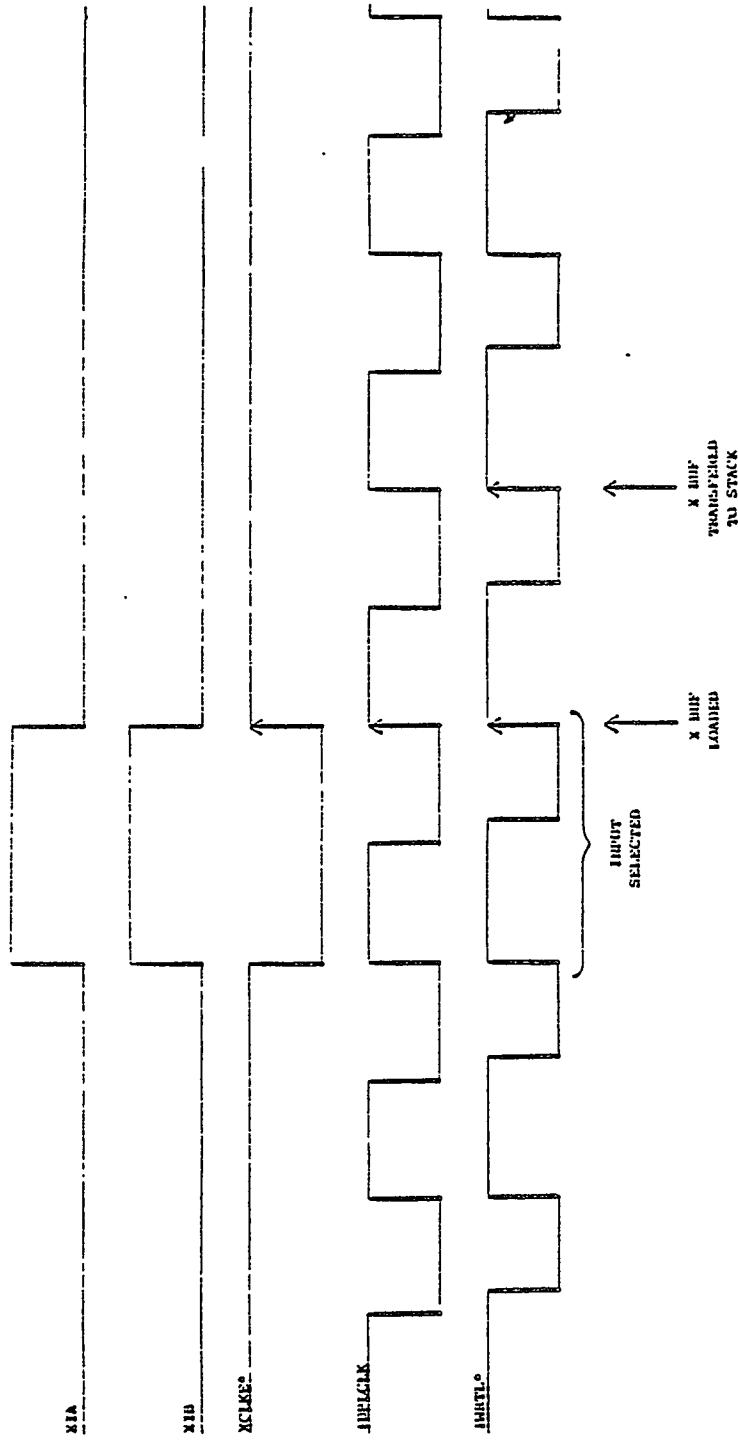


Figure 5-4 Timing Diagram of Writing DPX

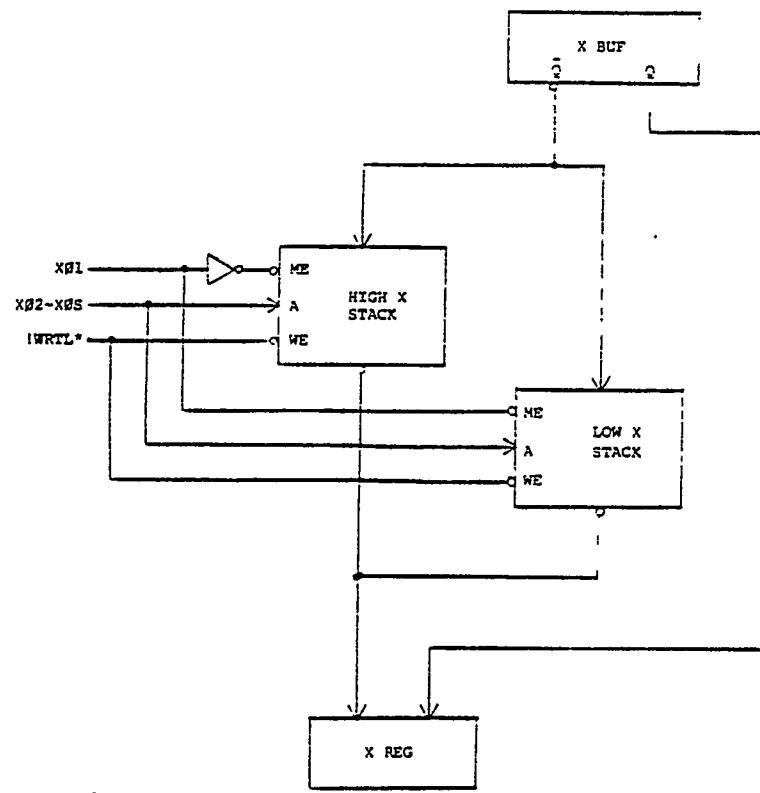


Figure 5-5 Detail Showing Stack Address

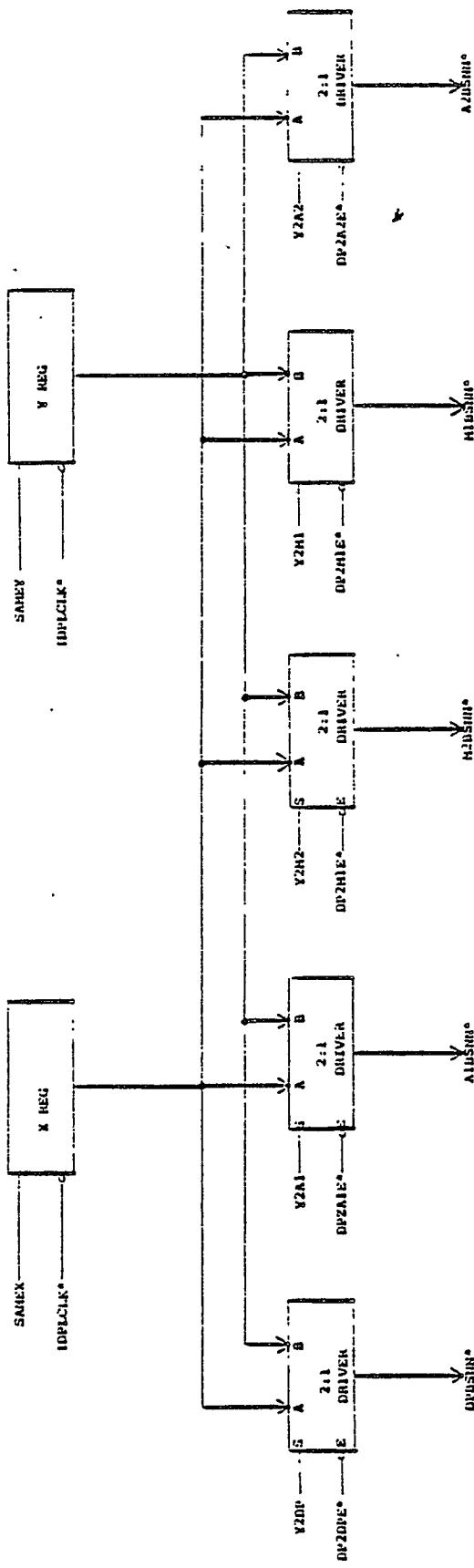


Figure 5-6 Data Pad Output Logic

5.3.2 Main Data

Main data, as shown in Figure 5-7, is the primary storage element for 38-bit data words inside the AP-120B. Main data is available in 8K word increments, up to 64K words directly addressable and up to 1.2M words with the bank address option. Main data is also available as either fast or standard memory. The customer selects, as an option, the speed of the memory best suited to his application. The figure shows how the main data system is interconnected in the AP-120B.

This discussion on MD will discuss the memory input register (MIreg), the memory element (MD), the main data output register (MDreg), the memory addressing (MA) logic and the memory controller.

5.3.2.1 Input Register

The MIreg (main data input register) is shown in Figure 5-8. Three different data paths can use the MIreg as their destinations: FAnn*, FMnn*, and the data pad bus. These three input buses are used as inputs to a 3:1 multiplexer which has selection signals MIA and MIB. MIA and MIB are the decode of program source bits 56 and 57 (the MI field) and select which input is to be loaded into the MIreg. MIreg is loaded when MICLKE* is true (low) and on the low-to-high transition of !MICLK. !MICLK is a free-running 167ns clock that is generated from and in phase with the master system clock. It should be noted that loading data into the MIreg and modifying the memory address in the same instruction cycle, initiates a write cycle into the main data memory element.

The output of the MIreg is enabled onto MDInn and loaded into main data (ECB 215 or 244) when MDCA3 is true (high). It should be noted that MDInn is a tri-state bus and the MIreg or the formatter may enable data onto this data bus based on the signals MDCA3 and MDCA1.

5.3.2.2 Memory Element

The main data memory element, illustrated in Figure 5-9, is physically located on ECB 215 or ECB 244. ECB 215 is composed of 8K words of standard speed memory. ECB 244 can have varying sizes of memory, depending on the integrated circuit used and some strap options and both standard speed or fast memory. ECB 244 can have either 8K, 16K, 32K or 64K words of either standard or fast memory.

Data to the main data memory element (MD) comes from either MI reg or the formatter and is clocked into a latch on the low-to-high transition of MDICLK. MDICLK is derived from MDINA* and the decode of the upper address bits of MAnn*, the bank select.

The memory elements (memory) are dynamic MOS random access memories. In order to write the appropriate ICs into these memory elements, as determined by the address, the chip must be chip-enabled (CE), chip-selected (CS), write-enabled (WE), the address supplied, and the input data supplied. The data is supplied as the output of the latch mentioned in the above paragraph. The CS is supplied as signal derived from the bank select decode done on the ECB, the upper bits of the MAnn*. It should be noted that different portions of the memory elements (EXP, HMAN, LMAN) can be selectively (based on MDEXP, MDHMAN, and MDLMAN) chip-enabled (CE). Thus, the memory element may be written in three bytes. The write-enable (WE) is enabled by MDWRT* and the bank address decode.

The above paragraph describes the steps necessary to read from the memory elements if the references about the WE are deleted. The ICs must be CS, CE, and the address supplied. After the data is valid, the tri-state latch must be clocked and the data enabled onto the MDun* lines. The clock is MDCLK* which is derived from !MCLK and the latched "NOT" of MDWRT*. Thus, if a MD cycle is initiated and it is not a write, MDWRT* is false. Then it is a read by default. The data is enabled onto MDun* by MDENB* which is derived from the bank select decode and MDCLK*.

5.3.2.3 Output Register

The MDreg (main data output register) is shown diagrammatically in Figure 5-10. The input to MDreg is the 38 data word from the memory element. Data is loaded into the MDreg when MDCLKE* is true (low) and on the low-to-high transition of !MDCLK*. It should be noted that !MDCLK* is generated from the master system clock on ADDR (ECB 212) and is 180 degrees out of phase.

The output of the MDreg is bused together and used as the input to three tri-state drivers. Thus, the output of the MDreg may be used as the source of data for the A2BS, M2BS, and the data pad bus. Three separate enables (MD2DP, MD2M2, and MD2A2) are provided and allow the output of the MDreg to be selectively enabled onto the three buses mentioned.

5.3.2.4 Addressing

Because the AP-120B's main data can be written/read from an internal program or from the host via a direct memory access, the address for MD can be sourced from the AP or the host. The host supplies the address to the AP-120B interface (IF) and the IF supplies it to a 2:1 multiplexer that physically resides on ECB 201. The other input to the 2:1 multiplexer is the AP-120B internal memory address (MA) register. MA may be loaded from SPFN or the least significant bits of the data pad bus via the SP+DPnn* bus. The MA may also be incremented or decremented. This is controlled by the decode of program source bits 59 and 58.

Modifying the contents of MA, whether loading, incrementing, or decrementing, causes the signal MDCR3* (main data cycle request 3). The interface requests MD cycles on MDCR1. The 2:1 address multiplexer is selected by MDCAL. Figure 5-11 shows a simplified block diagram of the main data addressing logic.

5.3.2.5 Memory Controller

The memory controller physically resides on CBL (ECB 210). It accepts main data cycle requests (MDCR1 and MDCR2) and does the necessary priority decode to grant cycle acknowledges (MDCA1 and MDCA2). If the interface requests a memory cycle (MDCR1) at the same time the AP-120B requests a memory cycle (MDCR3), the interface gets the cycle and the controller sends MDCA1 to the IF. This also sets up the address appropriately. The MDCR3 will be held and processed (MDCA3) after the IF is done.

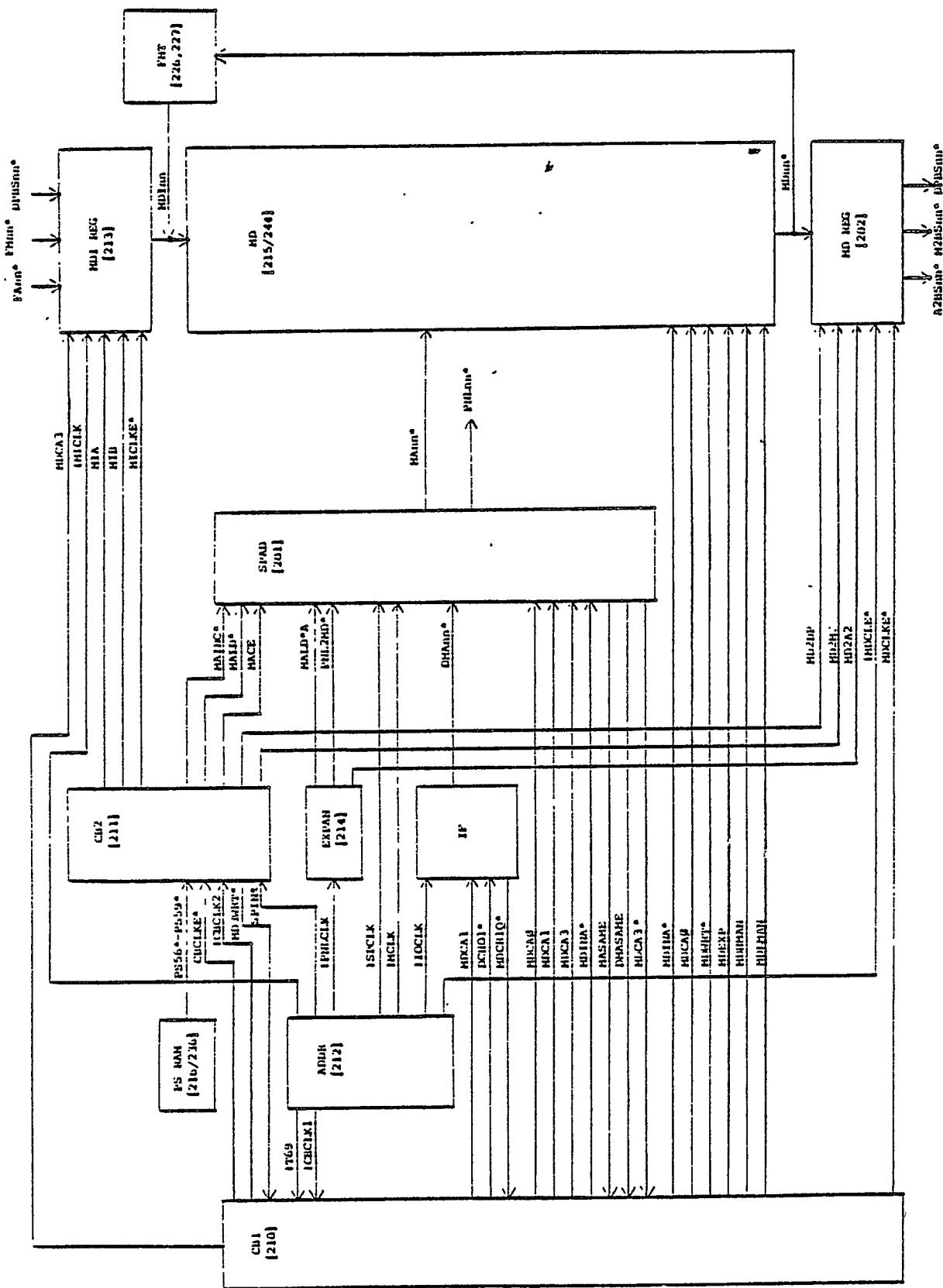


Figure 5-7 Main Data System Interconnection Block Diagram

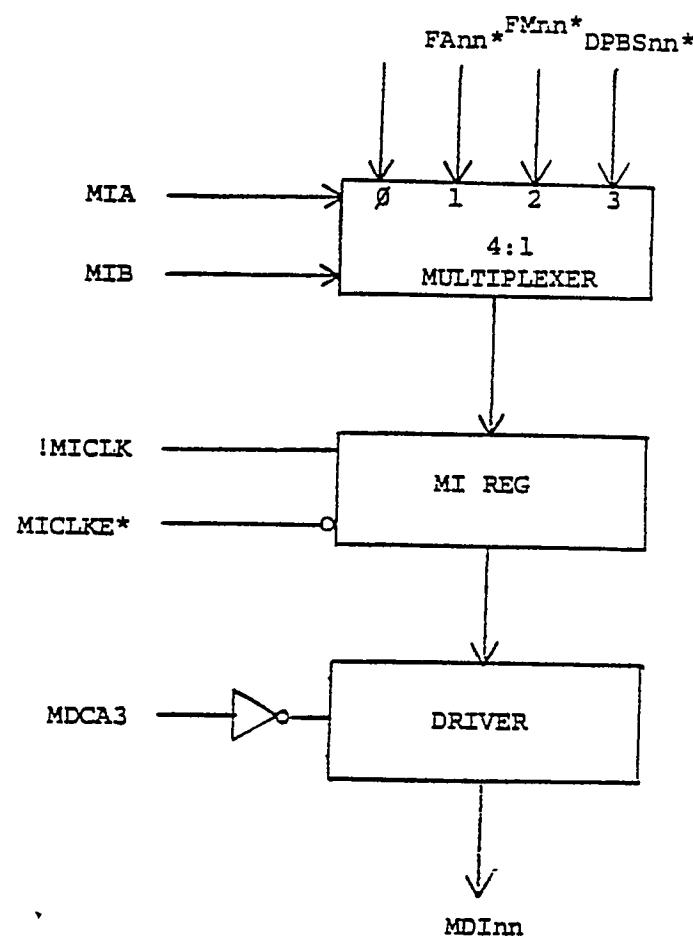


Figure 5-8 MI Reg Block Diagram

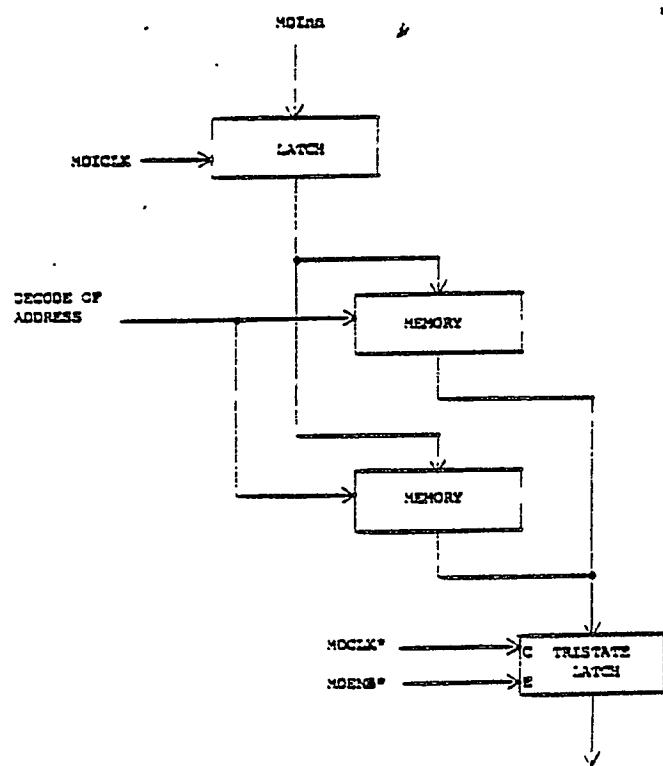


Figure 5-9 Main Data Memory Element Block Diagram

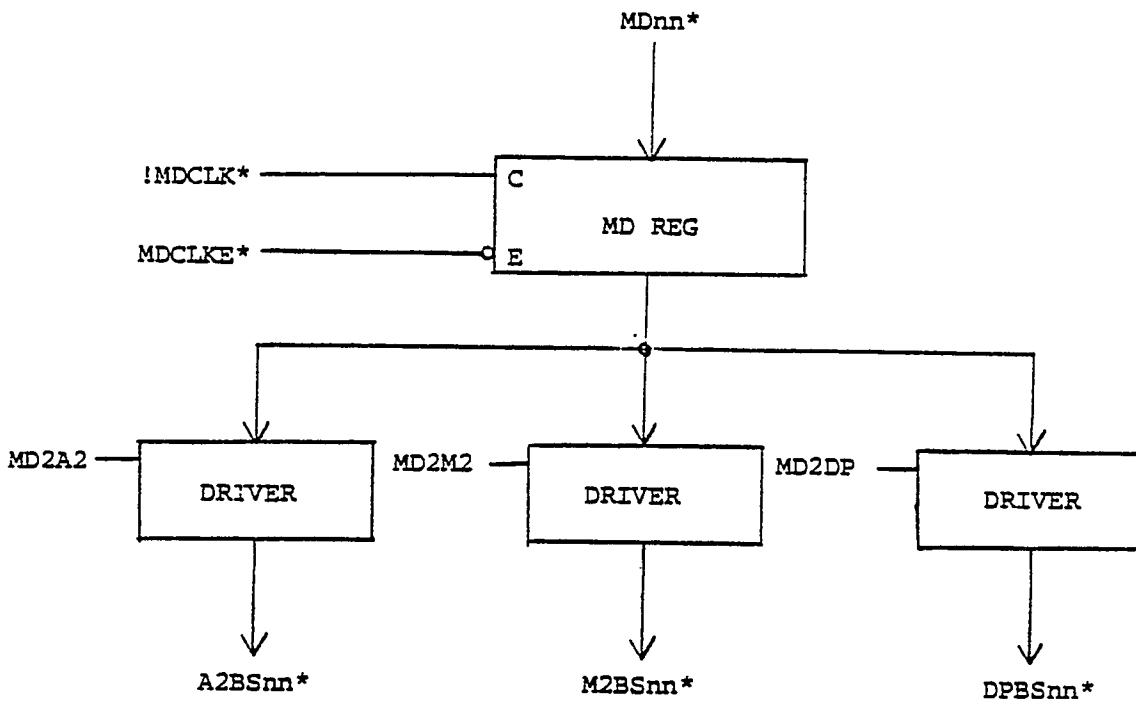


Figure 5-10 Main Data Register Block Diagram

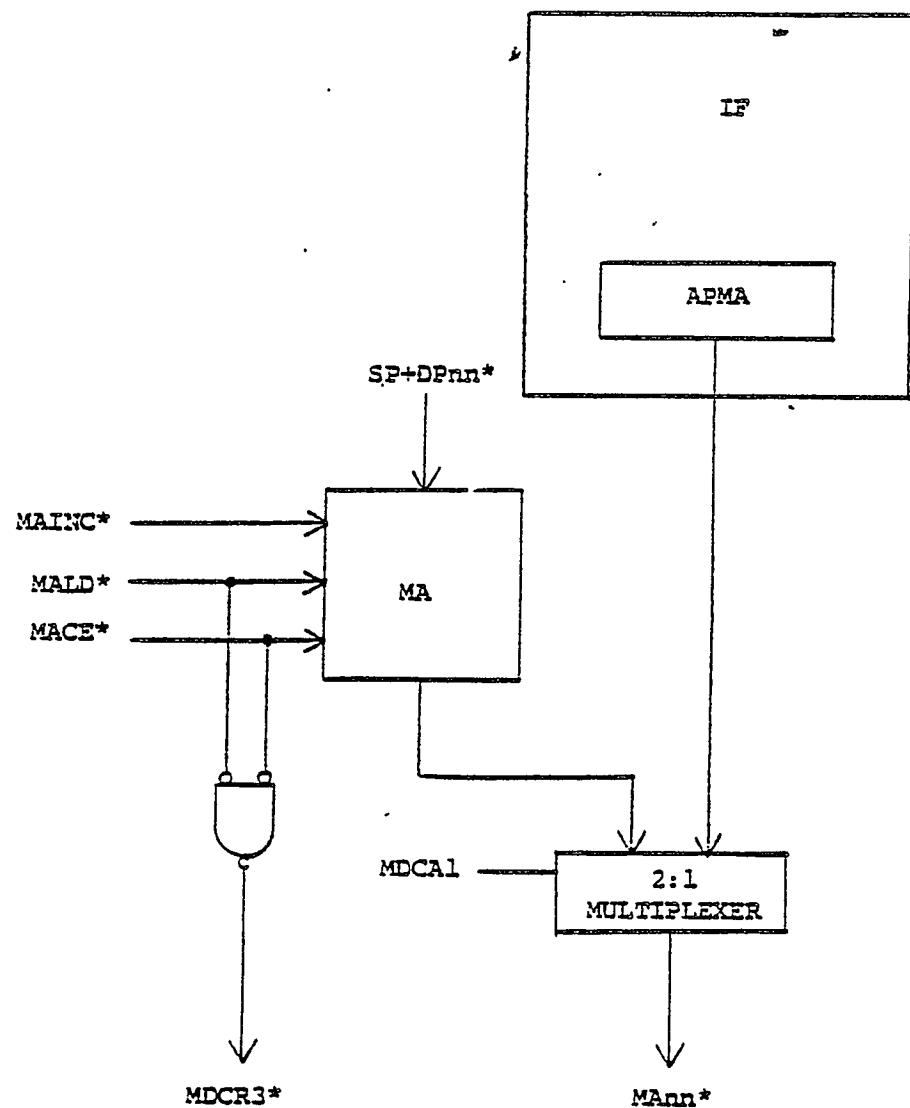


Figure 5-11 Memory Address Simplified Block Diagram

5.3.3 Table Memory

The memory element in the AP-120B used to store standard constants, such as complex roots of unity, is the table memory (TM). Figure 5-12 is a block diagram showing the table memory system. Associated with table memory is the table memory address (TMA) register, table memory read only memory (TMrom) and the table memory register (TMreg). As an option, the customer can also have table memory random access memory (TMram) and the table memory input register (TMIreg).

The output of the table memory is loaded in TMreg and can be enabled onto the M1BS, the A1BS, and the data pad bus. When TMram is present, the input to the TMram is from the IOBS into the TMI register.

Figure 5-13 shows how the table memory is interconnected in the AP-120B system. Program source bits 62 and 63, PS62* and PS63*, are used to control TMA. PS62* and PS63* are decoded on control buffer 2 (CB2, ECB 211) to determine if any change in TMA is called for (TMADEC*, TMACE*, and TMALD*). The table memory address (TMA) register physically resides on ADDR (ECB 212) and TMAnn (TMA00-TMA15) are the outputs of the TMA register. TMrom receives TMAnn and physically resides on ECB 217. Changing the contents of TMA causes TMreg (ECB 209) to be loaded with new data. Two mode control lines (FFTQ and IFFTQ*) comes from the APSTATUS register, which resides on control buffer 1 (CB1, ECB 210), and are used in the TMA register.

The enables, used to enable the data from the TMreg to the various buses, come from CB2 (ECB 211) and EXPAN (ECB 214). These enables are TM2DP, TM2M1, and TM2A1.

This discussion of table memory will first discuss table memory addressing. Then TMrom will be presented, followed by TMreg.

5.3.3.1 Addressing

The table memory address register (TMA) resides on ECB 212 and is shown diagrammatically in Figure 5-14. The TMA is used alternately, as an address pointer and as a quadrant indicator. The address point is used for both TMrom and TMram. The quadrant indicator is used with TMrom and is used to address a cosine table stored in TMrom. One quadrant (90 degrees) of a cosine table is stored in TMrom. However, the TMA may be used to access the cosine table as if it were a full table (360 degrees) of sine and cosine values. The sine and cosine values are primarily used in the forward and inverse fast fourier transform calculations.

The TMA shown in Figure 5-14 is a 74S169 UP-DOWN counter. TMA may be loaded from the output of SPFN or data pad bus. It may also be incremented or decremented. The control signals for TMA (TMADEC*, TMACE*, TMALD) are the decode of program source bits 62 and 63. The decode takes place on CB-2 (ECB 211). Table 5-1 is a chart of the TMAreg control signals and their effects.

Table 5-1 Table Memory Addressing

MNEMONIC	EFFECT	CONTROL SIGNALS
INCTMA	ADD ONE TO PRESENT ADDRESS	TMACE* • !PSACLK†
DECTMA	SUBTRACT ONE FROM PRESENT ADDRESS	TMADEC • TMACE* • !PSACLK†
SETTMA	LOAD SPFN	TMALD* • !PSACLK†
LDTMA	LOAD DATA PAD BUS BITS 12-27	TMALD*A • !PSACLK†

If FFTQ is not true (low) the data loaded into TMA will be used as the address on TMrom (TMAnn*). If bit 15 of the TMA register is a "1" (high) then the 4:1 multiplexer is set to pass input 1 to the output. If not set, input 0 is selected. In either case, the output of TMA is used as TMAnn*.

If the operator desires to use TMrom to access the 360 degrees of cosines and sines, he must turn on the FFTQ bit in the APSTATUS register. Setting FFTQ causes the 4:1 multiplexer to either input 2 or 3 as TMAnn*.

5.3.3.2 ROM

The TMrom physically resides on ECB 217. Figure 5-15 presents a block diagram of the logic on ECB 217.

The read only memories (ROMs) are programmed prior to leaving the FPS factory. Therefore, utilizing TMrom is nothing more than selecting the data in the desired address. The output of the table memory address logic is applied to ECB 217 where it is decoded to enable and address the appropriate location in the appropriate integrated circuit. Applying an address and an enable causes the data that was programmed into the IC to be applied to the output TMnn*.

5.3.3.3 Register

TMreg is physically located on ECB 209. Figure 5-16 is a block diagram of the TMreg hardware. The input to the TMreg, TMnn* is the output of TMrom. It should be noted that the output of the TMrom remains valid after TMA settles and until the next time TMA is notified. !TMCLK is a free-running, 167ns clock that is generated from and in phase with the system clock. As long as the AP-120B is not spinning, TMreg is being loaded.

It should be noted that there is a conditional 2's complementer as a part of TMreg that is capable of complementing TM12*A through TM39*A (the mantissa). This is used with accessing sines and cosines from TM ROM. This complementer allows 90 degrees, 2K cosine table to appear as a 360 degree 8K cosine table. TMNEG* is the signal that causes the mantissa to be complemented. The exponent out of TMreg is fed directly into the drivers.

The output of the table memory can be used as the source for the M1BS, A1BS or the data pad bus. TM2M1, TM2A1, and TM2DP are the enables that drive the output of TM onto the bus(es).

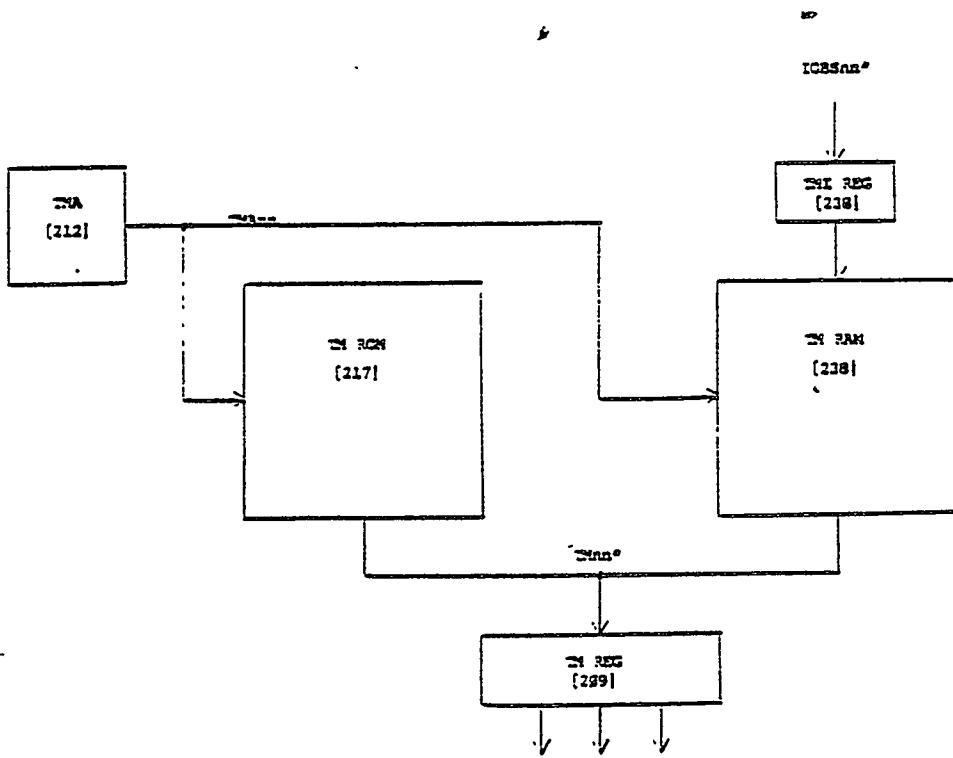


Figure 5-12 Table Memory System Block Diagram

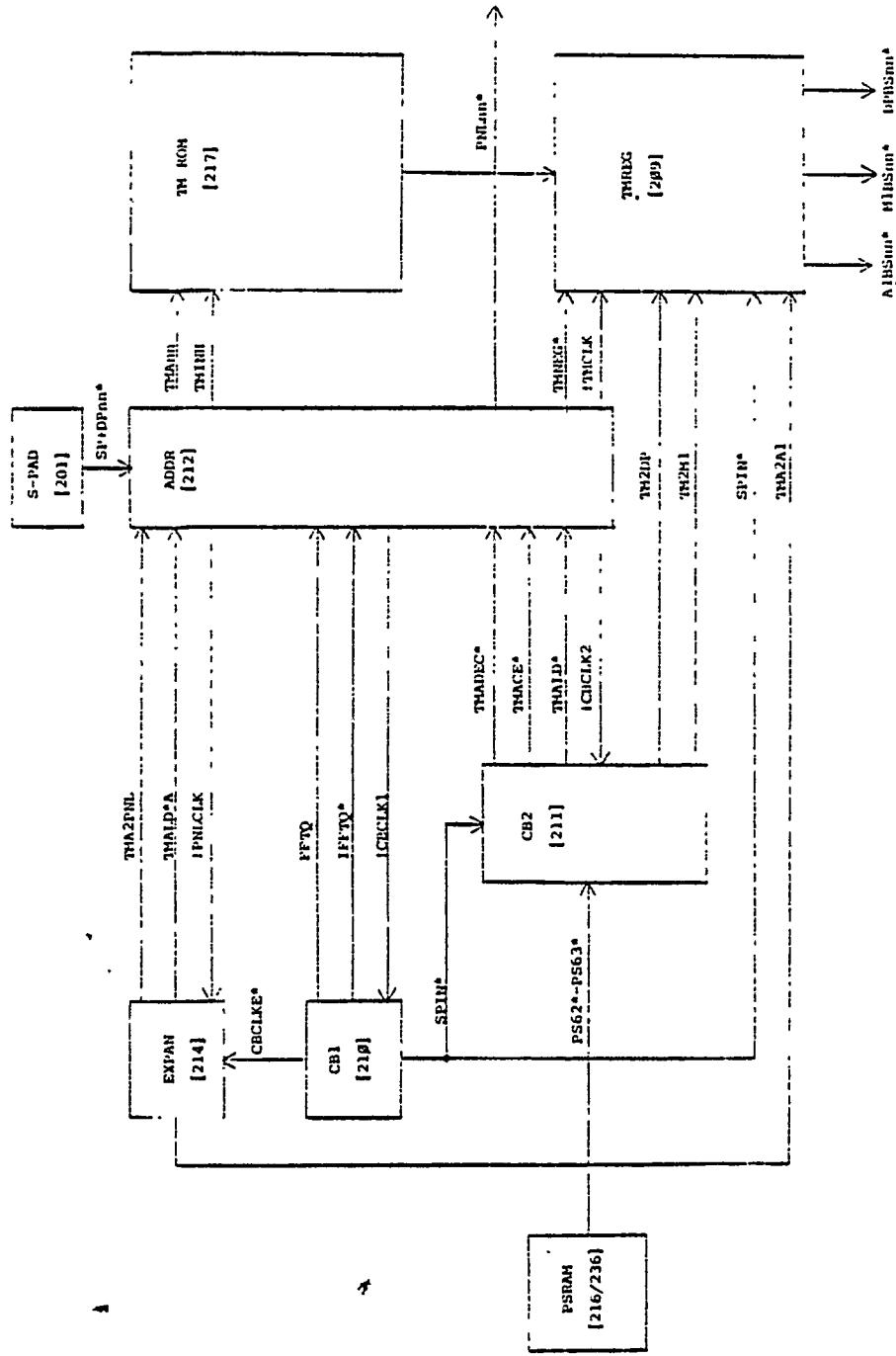


Figure 5-13 Table Memory ROM Interconnection Block Diagram

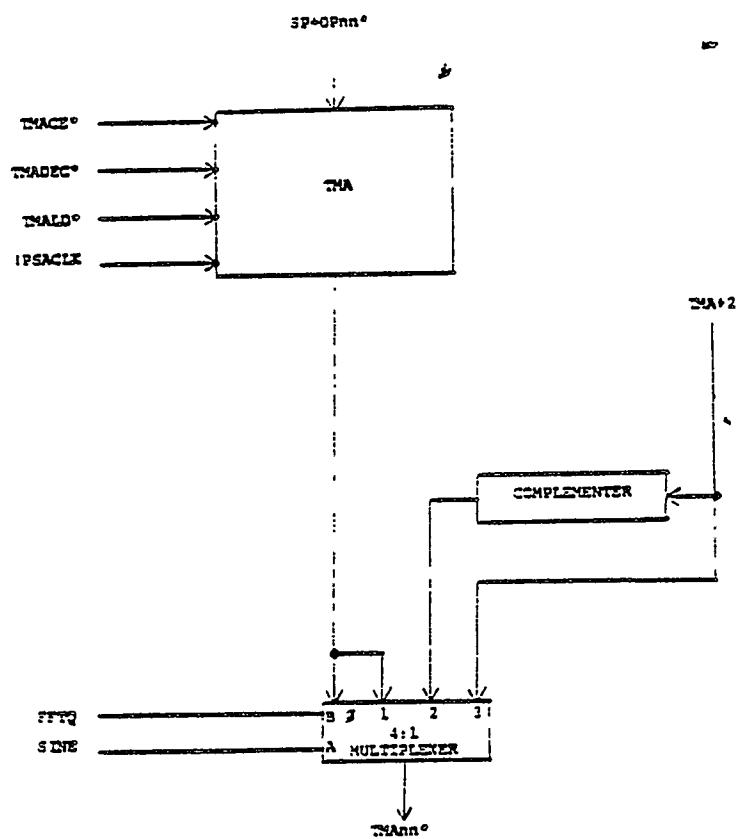


Figure 5-14 Table Memory Address Logic

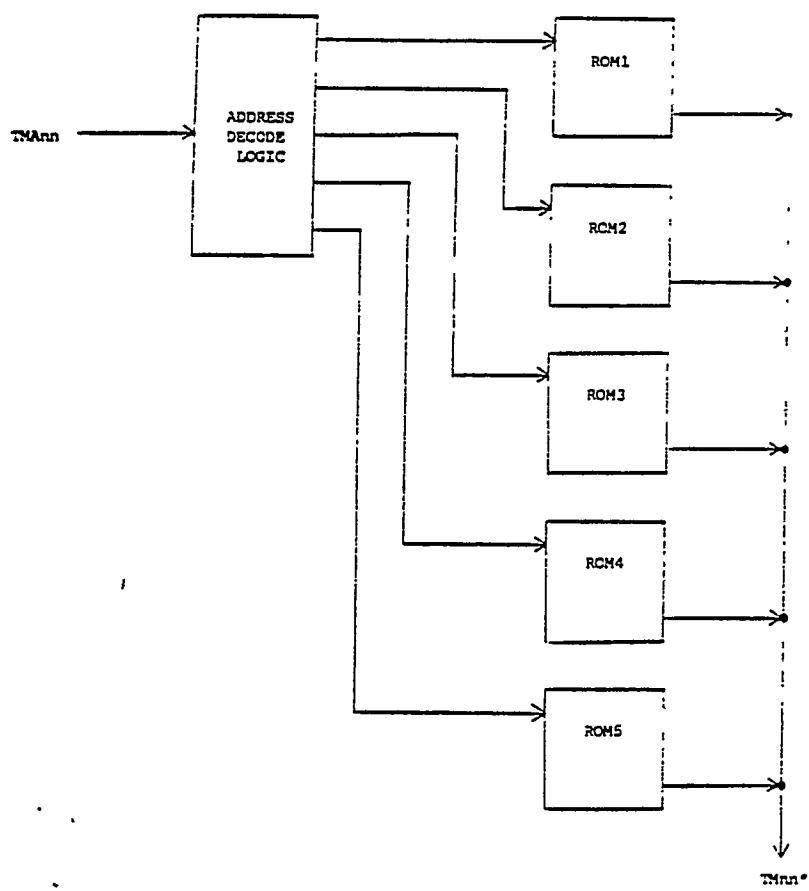


Figure 5-15 Table Memory ROM Block Diagram

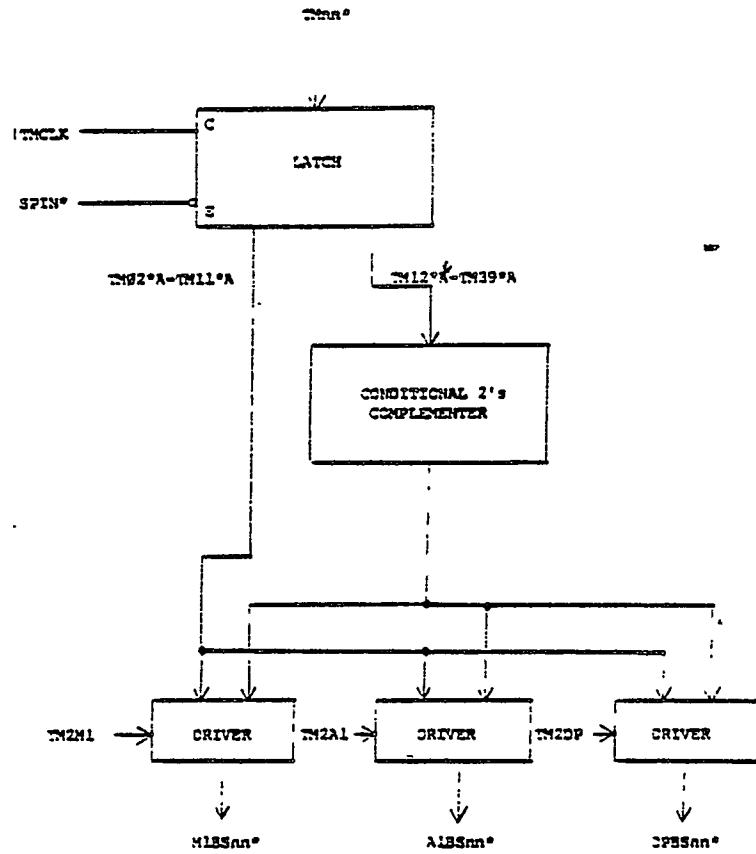


Figure 5-16 Table Memory Register Block Diagram

5.4 ARITHMETIC

The AP-120B floating-point adder (FADD) and floating-point multiplier (FMUL) operate on 38-bit, 2's complement, normalized floating-point numbers. In order to understand operation of the FADD and FMUL hardware, it is necessary to know the fundamentals of floating-point arithmetic. Therefore, this section first presents a discussion of floating-point arithmetic and then describes the FADD and FMUL hardware along with the associated control logic.

The discussion of floating-point arithmetic assumes that the reader has a basic knowledge of the binary number system. The order of presentation is: an explanation of the basic structure of floating-point numbers, a basic introduction into 2's complement arithmetic, a description of the floating-point format used by the AP-120B, and a discussion of the algorithms used for floating-point addition and multiplication including normalization and rounding of the results.

5.4.1 Floating-Point Numbers

Data processed by computers may either be represented by fixed-point numbers or by floating-point numbers. The difference between the two types of numbers is dependent on the placement of the radix point. The radix point is used to separate the whole number (integer) from the fractional portion of the number. In the decimal system, for example, the radix point is referred to as the decimal point. In the number 6.5, the decimal point separates the integer (6) from the fraction (5 or one-half). The general term for the point between the integer and fractional portions of a number in any base (radix) is the radix point. The term radix point is used throughout this manual.

The term "fixed-point" means that the radix point is always in the same position (fixed) within the number. Integers are often referred to as fixed-point numbers because the radix point never varies. It usually indicates the least significant place of the number and is normally not represented. Thus, in the decimal number 314, for example, the decimal point (radix point) is understood to be at the right of the number four (4).

The term "floating-point" means that the radix point is allowed to "float". That is, the radix point can be moved as needed. The use of floating-point numbers can significantly increase the dynamic range of the computer because neither extremely small nor extremely large numbers can be easily represented by the same fixed-point format. For example, in dealing with a four-digit number in a fixed-point format of 00.00, the largest number that can be represented is 99.99 while the smallest number that can be represented is 00.01. In a floating-point format, however, the radix point may be positioned as needed. Thus, with a floating-point format, a four-digit number could represent a value as small as .0001 or as large as 9999.

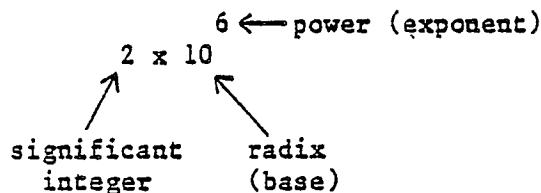
5.4.1.1 Structure

Although most general-purpose computers in use today handle only fixed-point numbers, the AP-120B handles floating-point numbers. This not only increases the accuracy of the system, but also permits both extremely large numbers and extremely small numbers to be represented with the same number of bit positions.

In order to handle floating-point numbers more easily, the number is usually represented by some significant integer which is multiplied by the radix raised to some power. For example, in the decimal system:

$$2,000,000 = 2 \times 10^6$$

The breakdown of 2×10^6 is:



Note that the same number can be represented in many different ways. For example:

$$\begin{aligned} 483 &= 48300 \times 10^{-2} \\ 483 &= 4830 \times 10^{-1} \\ 483 &= 483 \times 10^0 \\ 483 &= 48.3 \times 10^1 \\ 483 &= 4.83 \times 10^2 \end{aligned}$$

Notice in the above example that the position of the radix point is shifted each time the exponent is changed. Because the exponent determines the position of the radix point, it can be said that the exponent serves as a "pointer" to the radix point.

Because integers are normally not used in floating-point formats, the number is always expressed as a fraction such as $0.1 \times 2(7)$ or $0.11001 \times 2(3)$. Thus, all floating-point numbers are divided into two basic parts: the mantissa (which represents the fraction) and the exponent (the power of the base). For example:

0.1101 $\times 2^4$
mantissa exponent

As stated earlier, a floating-point number (FPN) is usually expressed as the product of a signed fraction and a base raised to a power. A general expression for floating-point numbers is:

$$\text{FPN} = \pm M * B^E$$

The above expression can be broken down as follows:

$\pm M$ = the signed fraction which has the radix immediately to the left of its left-most significant bit. This fractional portion of the FPN is called the mantissa. The AP-120B can handle a 28-bit mantissa.

B = the base which is determined by the specific number system used in the floating-point format. Because the AP-120B uses a binary base, B equals 2. It should be noted that B is determined by the particular hardware implementation and is, therefore, constant.

E = the exponent which serves as a pointer to the position of the radix point. The AP-120B uses a 10-bit, base 2 biased exponent.

The term "bias" is explained later in this section. In general, it refers to a number added to the exponent to prevent certain overflow conditions. This number must therefore be subtracted from the exponent in order to determine the exponent's true value.

In summary, it can be seen that there are four essential parameters to every floating-point number:

- a. The sign of the mantissa.
- b. The magnitude of the mantissa.
- c. The bias of the exponent.
- d. The magnitude of the exponent.

5.4.1.2 Normalization

Although it is possible to represent the same floating-point number in a variety of ways (such as 11.1×2^1 or 1.11×2^2), doing so could lead to wasted bit positions and resultant inaccuracy. This can best be shown by first looking at how numbers are handled in fixed-point formats.

When dealing with extremely large numbers in fixed-point arithmetic, the radix point is positioned to the extreme right of the bits of numerical significance and trailing zeros are inserted after the radix point. When dealing with extremely small numbers, the radix point is located to the left of those bits of numerical significance and leading zeros are inserted between the radix point and the first bit of numerical significance. For example, assume that the fixed-point format is used for an eight-bit binary word and that the radix point is used to divide the word into two four-bit segments.

	radix point
large number	1 <u>0 0 0</u> . 0 0 0 0
	trailing zeros
small number	0 0 0 0 . <u>0 0 0 1</u>
	leading zeros

As can be seen in the above example, many bit positions are only used for leading or trailing zeros and are, therefore, wasted as far as being able to represent numerical significance.

In a floating-point number, the fractional part of the number (the mantissa) is determined by the number of bits allocated by the computer. If non-significant leading zeros are retained, the accuracy of the number is decreased. Because of this, a process called "normalization" is used to achieve maximum accuracy by eliminating trailing and leading zeros.

The normalization process consists of shifting the number either left or right until only significant bits are retained and the radix point is to the left of the most significant bit. The exponent must also be adjusted accordingly. Thus, typical normalized numbers in base 2 (binary) are:

7
0.1 x 2
6
0.100001 x 2
0
0.1 x 2
-3
0.1 x 2

Note that in each of the above cases, the most significant bit is to the right of the radix point.

Normalizing a whole number is performed by shifting the number right until the most significant bit is to the right of the radix point. Each shift causes the exponent to be incremented by one. For example:

0
111. x 2 original number (unnormalized)
1
11.1 x 2 after first shift
2
1.11 x 2 after second shift
3
.111 x 2 after third shift

After the third shift in the above example, the radix point is to the left of the most significant digit. Therefore, the number is now normalized.

Normalizing a fractional number is performed by shifting the number left until all leading zeros are eliminated. Each shift causes the exponent to be decremented by one. For example:

7
.0001 x 2 original number (unnormalized)
6
.001 x 2 after first shift
5
.01 x 2 after second shift
4
.1 x 2 after third shift

Again note that after the third shift the radix point is to the left of the most significant digit, indicating that the number has been normalized.

5.4.1.3 Rounding

Although a large number of bit positions may be allocated to represent numbers in order to increase the accuracy of floating-point number calculations, a number may still be larger than can fit into the specified bit positions. Therefore, numbers exceeding the allotted bit positions are usually rounded to preserve accuracy.

In the AP-120B, three extra bits of significance to the right of the mantissa are used for rounding. These bits (called the "residue") are tested to determine if this residue is greater than one half of the least significant bit. If the residue is greater, then the result is rounded. If the residue is the same or less, the result is not rounded. This process causes the cumulative rounding error to converge toward zero on repeated calculations.

5.4.2 Arithmetic

When implementing arithmetic operations in a computer, the numbering system that is used must be capable of expressing both positive and negative quantities (numbers) in order to be useful in arithmetic operations. In addition, each element within the computer system must also have the capability of handling signed quantities. Although a variety of formats are available for handling signed binary numbers, the format used in the AP-120B system is known as "2's complement notation."

Although 2's complement arithmetic has a number of inherent advantages, there can be a problem of overflow when going from the least negative number to zero. This problem can be eliminated by "biasing" the number system. A "bias" is simply a constant value that is added to the number at the beginning of the calculation and then subtracted from the result.

The purpose of this section is to illustrate, in a general way, how a computer performs arithmetic calculations by using 2's complement notation. In order to understand 2's complement notation, it is first necessary to understand the concept of complementary numbers. Therefore, this section begins with a short discussion of complementary numbers and is then followed with discussions of: 2's complement notation, 2's complement arithmetic and bias.

5.4.2.1 Complementary Numbers

When implementing arithmetic operations in a computer, it must be remembered that a computer is only capable of adding, not subtracting. Although it is normal to think of addition and subtraction as different operations, it is also possible to think of subtraction as nothing more than the addition of signed numbers, such as adding +5 to a -3. However, this still poses a problem because the computer can only add the numerals. If a computer were to add a +5 and a -3, for example, it would add the two numeric values and then assign the appropriate sign. The result in this case would be a -8, which is obviously incorrect. Therefore, when using signed numbers in a computer, it is necessary to convert each negative value to an equivalent positive value prior to the addition. This is accomplished by using the "complement" of the negative number.

In order to understand the concept of complementary numbers, it is easier to begin by using an example in the decimal (base 10) system. Assume, for the sake of this example, that the computer can handle only one digit position and that the problem is as follows:

$$\begin{array}{r} +5 \\ -3 \\ \hline \end{array}$$

As stated before, the computer can only add. Therefore, it is necessary to convert the -3 to its complement (its positive equivalent). This is accomplished by subtracting the number (which is 3) from the base (radix) of the number system being used. Because decimal is a base 10 system, 3 is subtracted from 10 which gives a result of 7. Thus, +7 is the complement of -3. At this point the problem can be set up as follows:

$$\begin{array}{r} +5 \\ +7 \text{ (complement of -3)} \\ \hline \end{array}$$

When these two numbers are added (as shown below), the result is a 2 with a carry of 1. However, because the computer can only handle one digit position, the carry is lost. Therefore, the result is 2 which is the same answer that would have resulted had 3 been subtracted from 5.

$$\begin{array}{r} +5 \\ +7 \text{ (complement of -3)} \\ \hline 1 \ 2 \\ \text{carry} \end{array}$$

There are actually two methods that can be used to find the complement of a number. The method used above is known as the 10's complement because the number was subtracted from the base of 10 in order to find the complement. Another method, known as the 9's complement in the decimal system, is to subtract the number from the highest integer in the base. Thus, 3 can be subtracted from 9 (the highest integer in the base 10 system) to provide 6. Because only the 10's complement can be used as described above, a 1 must be added to the 9's complement number. Thus, 6 plus 1 gives the result of 7 which is the complement of -3. Either method (10's complement or 9's complement plus one) produces a +7 as the complement of -3.

At this point, it might be valuable to look at the base 8 (octal) number system. For the purpose of explanation, the octal number is shown in both binary and octal form. Assume again, for the sake of example, that the problem is to subtract octal 3 from octal 5. The notation is:

Binary	Octal
1 0 1	5
0 1 1	3

There is a problem however. Because unsigned numbers are used in the above example, it is necessary to convert octal 3 (binary 0 1 1) to its complementary form in order to perform the subtraction.

One method of finding the complement of octal 3 is to subtract the number from the highest integer in the base and then add 1. The highest integer in the octal number system is 7. Therefore, 7 minus 3 is 4 (or binary 1 0 0). Notice what has happened at this point:

	Octal	Binary
Number to be complemented	3	0 1 1
Result of subtraction (7 minus 3)	4	1 0 0

In effect, rather than actually subtracting numbers, the state of each binary bit was simply reversed. That is, all 1's were changed to 0's and vice versa.

In order to find the complement, a one must now be added. Thus:

Octal	Binary
4	1 0 0
1	0 0 1
	<hr/>
5	1 0 1

Thus, the complement of octal 3 is octal 5. Note that the complement was found by reversing the state of each binary bit and then adding 1. No subtraction was involved in determining the complement.

When the two numbers are added (as shown below), the result is binary 0 1 0 with a carry (that is, binary 1 0 1 0). However, because the computer is only dealing with three bit positions, the carry is lost. Therefore, the result is binary 0 1 0 (octal 2) which is the same result that would have been obtained had octal 3 (binary 0 1 1) been subtracted from octal 5 (binary 1 0 1).

Octal	Binary
5	1 0 1
5	1 0 1 (complement of 3)
	<hr/>

1 2 1 0 1 0
 ↑ →
 carry is "lost" in
 both cases

In summary, a computer performs addition operations only. Subtraction (or addition of negative numbers) is accomplished by first converting the negative number to its complementary form and then adding. Finding the complement of a negative number is accomplished by one of two methods:

- a. Subtracting the negative number from the base
- b. Subtracting the negative number from the highest integer in the base and then adding 1

When using the base to determine the complement, the name of the base is used. Thus, it is possible to have 10's complement (decimal), 8's complement (octal) and 2's complement (binary).

When using the highest number in the base to determine the complement, the name of that integer is used. Thus, it is possible to have 9's complement (decimal), 7's complement (octal) and 1's complement (binary).

Because the binary number system is used in the AP-120B, both 1's and 2's complement notation will be covered in the next paragraph.

5.4.2.2 Two's Complement Notation

Any numbering system that is to be useful in arithmetic computations must be capable of expressing both positive and negative quantities (numbers). Each element of the system must also have the capability of handling a signed quantity. This requirement for signed numbers has created a variety of formats for the binary number system. The three most common methods of ascribing signs to binary numbers are: the sign-magnitude format, the 1's complement format and the 2's complement format.

Although both sign-magnitude and 1's complement methods are briefly described in this section, the major emphasis is given to 2's complement notation because it is the method used by the AP-120B system.

The sign-magnitude format uses an extra bit known as the "sign" bit. One state of this bit (usually 0) signifies a positive quantity while the other state (usually 1) signifies a negative quantity. Generally, this sign bit is placed to the left of the significant bits that are used to specify the magnitude of the number. Some examples of binary numbers in sign-magnitude format are:

Sign Bit	Magnitude	Decimal Value Represented
0	0 0 1 0 0 1	+9
0	0 0 0 1 1 1	+7
0	0 0 0 0 1 0	+2
0	0 0 0 0 0 0	+0
1	0 0 0 0 0 0	-0
1	0 0 0 1 0 1	-5
1	0 0 1 0 0 0	-8
1	0 0 1 1 0 0	-12

It should be noted in the previous example that there are two representations for zero when using the sign-magnitude format.

The 1's complement format also uses a "sign" bit which is placed to the left of the significant bits that are used to specify the magnitude of the number. However, implementation of 1's complement numbers is slightly different than that used for sign-magnitude numbers.

When using 1's complement notation, the sign bit is 0 when the number is positive. However, a negative number is created by complementing both the magnitude and the sign of the number. The number is complemented by simply reversing the state of each bit in the number (changing all 1's to 0's and all 0's to 1's). For example:

	Decimal Value	Binary Value
	Sign	Magnitude
original number	+100	0 1 1 0 0 1 0 0
1's complement	-100	1 0 0 1 1 0 1 1

Both the sign-magnitude format and 1's complementation notation are identical when representing positive numbers, as shown below. However, although both methods have representations for zero, the representations for negative zero are different. In addition, the method of handling negative numbers is completely different as shown below:

Decimal Value	Sign-Magnitude Form		1's Complement Form	
	Sign	Magnitude	Sign	Magnitude
+103	0	1 1 0 0 1 1 1	0	1 1 0 0 1 1 1
+12	0	0 0 0 1 1 0 0	0	0 0 0 1 1 0 0
+7	0	0 0 0 0 1 1 1	0	0 0 0 0 1 1 1
+0	0	0 0 0 0 0 0 0	0	0 0 0 0 0 0 0
-0	1	0 0 0 0 0 0 0	1	1 1 1 1 1 1 1
-7	1	0 0 0 0 1 1 1	1	1 1 1 1 0 0 0
-12	1	0 0 0 1 1 0 0	1	1 1 1 0 0 1 1
-103	1	1 1 0 0 1 1 1	1	0 0 1 1 0 0 0

In 2's complement notation, the sign is also at the left of the significant bits of the number. A positive number is expressed in an identical manner to sign-magnitude representation. That is, the sign bit is 0 and the magnitude is represented in the normal manner. Thus +7, would be represented as 0 111.

In order to express a negative number in 2's complement notation, the entire number, including the sign bit, is complemented by reversing the state of each bit. This, in effect, gives the 1's complement of the number. To form the 2's complement, a binary 1 is added to the 1's complement number. For example:

	Sign	Magnitude
Positive number	0	0 1 1 0 1
1's complement (bit states reversed)	1	1 0 0 1 0
2's complement (binary 1 added)	1	1 0 0 1 1

If, for example, it were decided to change the positive binary representation of +100 decimal to its negative equivalent, it could be accomplished by 2's complementing (negating) the positive number. Thus:

Decimal Value	Binary Equivalent
+100	0 1 1 0 0 1 0 0
-100	1. 0 0 1 1 1 0 0 (above number complemented and incremented)

Note that whenever a positive number is complemented, the sign is changed. Therefore, in 2's complement notation, a 0 always indicates a positive value and a 1 always indicates a negative value.

It should be noted that this negation process is symmetric because the positive equivalent of a negative number can be obtained by the same process. That is, the negative number is converted to the 2's complement form in order (complemented and incremented) to obtain the positive equivalent.

Therefore, in order to find the absolute magnitude of a negative 2's complement number, it is simply a matter of complementing the number and adding binary 1 as shown in the following example:

Negative 2's complement number	1	0 1 1	-5
Complemented	0	1 0 0	
Binary 1 added	0	0 0 1	
Result	0	1 0 1	+5

On the other hand, a negative 2's complement number can be obtained by complementing the positive representation and adding binary 1 as shown in the following example:

Positive binary representation	0	0 1 1	+3
Complemented	1	1 0 0	
Binary 1 added	0	0 0 1	
Result (in 2's complement form)	1	1 0 1	-3

There is one possible point of confusion when dealing with 2's complement negative numbers. This problem has to do with the method used to view the number and can be clarified by looking at an example. Assume that it is desired to subtract binary 3 from binary 5 as shown below:

1	0	1	5
0	1	1	3

In order to perform the subtraction, the number 3 must be negated (2's complemented). Thus, the 3 is complemented and incremented as shown below:

1	0	1	5
1	0	1	2's complement form of 3

Now, when looking at the 2's complement form of 3, it can be analyzed in three different ways. It can be read as an unsigned binary 5, it can be read as -1 if the most significant bit is considered to be a sign bit, or it can be read as -3 because it is the complement of +3. The number remains the same regardless of how it is interpreted. However, for the sake of consistency, the method used throughout this manual is to view the number as the negative representation of the positive number. Thus, in the above case, the number would be considered to be a -3.

A full definition of a 4-bit, 2's complement, binary number system is presented in Table 5-2 as follows:

Table 5-2 4-Bit, 2's Complement, Binary Number System

2's Complement Representation				Equivalent Signed Decimal Number
Sign				
3	3	1	0	
-2	2	2	2	
(-8)	(4)	(2)	(1)	
---	---	---	---	
0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	0
1	1	1	1	-1
1	1	1	0	-2
1	1	0	1	-3
1	1	0	0	-4
1	0	1	1	-5
1	0	1	0	-6
1	0	0	1	-7
1	0	0	0	-8

In the above table note that the 2's complement format has only one representation for zero. Also, note that there is one negative number that has no positive counterpart. This is the number 1 0 0 0 or -8. This number is one larger in magnitude than the 2's complement of the largest positive number which is 0 1 1 1 or +7.

If N is used to represent the number of bit positions in a 2's complement number, d0 is used to represent the 0th bit position and di is used to represent the digit in the ith bit position, then the value of any 2's complement number can be calculated by using the following equation:

$$\left\{ TV = (-d_0 * 2^{N-1}) + (d_i * 2^{N-(i+1)}) \right\}$$

In the above equation:

i = 1

TV = true value

d0 = digit in the 0th position (sign bit)

di = digit in the ith position

N = number of digits in the data word

The 2's complement format was selected for use by the AP-120B because there is only one representation for zero and because 2's complement numbers can be added without being concerned about the sign of each number.

Now that 2's complement notation has been explained, the next section presents a discussion of 2's complement arithmetic.

5.4.2.3 Two's Complement Arithmetic

In the 2's complement number system, the operation of addition is simply the bit-by-bit binary addition of the two 2's complement numbers with the result being a 2's complement representation of the sum. The two examples below are given to demonstrate addition with 2's complement numbers.

0100 +4	1100	-4
0011 +3	0011	+3
-----	-----	-----
0111 +7	1111	-1

Note that the ordinary rules of binary addition apply to the sign bit as well as to the rest of the number.

Subtraction in 2's complement format is accomplished by first negating the number being subtracted (subtrahend) and then adding the two numbers. Negating the subtrahend is accomplished by taking its 2's complement. After the subtrahend is 2's complemented it is then added to the minuend and the result is a 2's complement representation of the difference. The two examples below are given to illustrate 2's complement subtraction.

Minuend	0100	+4	1100	-4
Subtrahend	0011	+3	1101	-3
Negate subtrahend	1101	-3	0011	+3
	---	---	---	---
Result	0001	+1	1111	-1

Note that the addition of two 4-bit, 2's complement numbers can give a 5-bit sum and all 5-bits are necessary to specify the correct sum. For example:

$$\begin{array}{r}
 0111 \quad +7 \\
 + 0111 \quad +7 \\
 \hline
 1110 \quad +14
 \end{array}$$

Unfortunately, the answer shown above is incorrect. When a 4-bit, 2's complement system is used (as in the above example), 1110 is a -2, not a +14. Therefore, the addition shown in the above example would result in an overflow condition. However, this overflow can be correctly handled in a N-bit machine by providing one extra bit in the critical arithmetic portions of the machine and by sign-extending the input arguments by one bit. Therefore, with an extra sign bit implemented, the two +7's in the previous example can be correctly added. This addition is shown below along with an example of adding two -7's.

$$\begin{array}{ccc}
 & \text{Sign bit} & \\
 & \downarrow & \\
 \text{Sign Extension} \rightarrow &
 \begin{array}{ll}
 00111 & +7 \\
 00111 & +7
 \end{array} &
 \begin{array}{ll}
 11001 & -7 \\
 11001 & -7
 \end{array} \\
 & 01110 & +14 \qquad \qquad \qquad 10010 & -14
 \end{array}$$

Note that the extra bit provides for the true sign of the result. Thus, overflow conditions no longer occur.

5.4.2.4 Bias

Any floating-point number consists of a mantissa (the fractional portion of the number) and an exponent (the base raised to some power). Thus, a typical floating-point number is:

$$0.100011 \times 2^5$$

The magnitude of numbers that can be handled by any floating-point processor is dependent on the number of bits that have been allocated to express the exponent. In the AP-120B, 10 bits are used to represent the exponent. One bit is used as a sign bit and the remaining nine are used for the number.

These 10 bits of exponent provide a range of exponents from -512 to +511 as illustrated in Table 5-3.

Table 5-3 Range of Exponent Values

Bit Positions	Sign									
	512	256	128	64	32	16	8	4	2	1
Most positive	0	1	1	1	1	1	1	1	1	1 = 511
Least positive	0	0	0	0	0	0	0	0	0	0 = 0
Least negative	1	1	1	1	1	1	1	1	1	1 = 1023
Most negative	1	0	0	0	0	0	0	0	0	0 = 512

Unfortunately, to go from the least negative value to zero is accomplished by adding 1 to the least negative value which results in an overflow condition. In addition, care must be taken to distinguish between positive and negative values. To correct this problem, the AP-120B uses a "bias" of 512. This simply means that the value 512 is added to all exponents. The result of adding a 512 bias to the values shown in Table 5-4 is illustrated below:

Table 5-4 Adding a 512 Bias

most positive	511	+512	=	1023
least positive	0	+512	=	512
least negative	-1023	+512	=	511
most negative	-512	+512	=	0

Notice that the biasing has changed the range of exponents so that the range is now from 0 to 1023. There is no need to be concerned with the sign of the value, nor is there any overflow problem. Of course, when performing arithmetic computations, the bias must be taken into account and removed from the final result in order to obtain the correct answer.

Because each exponent has 512 added to it, the exponent can be said to have an "excess 512." Because of this, bias is often referred to as "excess 512 notation."

5.4.3 Data Format

The AP-120B floating-point processor represents internal data in a 38-bit floating-point format. This format basically divides the number into two parts: a 10-bit exponent and a 28-bit mantissa.

Although floating-point numbers are fractional values multiplied by the power of a base, such as the number 0.11×2^3 , the base is a constant and does not need to be represented in the data word. Thus, the above number could simply be represented as a mantissa of 0.11 and an exponent of 3.

Figure 5-17 is an illustration of the bit position assignments in the 38-bit word. Because the bias bit (bit 0) is in the 29 position of the exponent, setting this bit adds 512 to the exponent in order to provide the proper bias. (Bias is described in paragraph 4.2.2.4.) The exponent itself is represented by bits 1 through 9 (28 through 20).

The mantissa portion of the word uses bit 10 as the sign bit and bits 11 through 37 to represent the numerical value of the mantissa. The last three bits (38 through 40) are not considered to be part of the 38-bit floating-point number. However, they are shown here because they serve as "guard" bits which are used to round the final bit (least significant bit) of the mantissa in order to preserve accuracy.

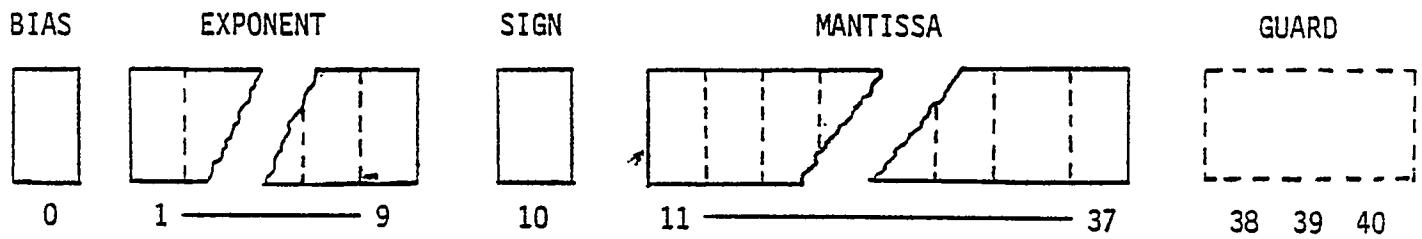


Figure 5-17 Internal Floating-Point Format

At times, a floating-point number is referred to as a floating-point number expressed in scientific notation. Table 5-5 provides four examples of decimal floating-point numbers as they are expressed in scientific notation and as they are represented in a Fortran expression.

Table 5-5 Examples of Decimal Floating-Point Numbers

Number	Scientific Notation	Fortran Expression
Large positive	$+ .9999 * 10^{99}$	$+ .9999E99$
Large negative	$- .9999 * 10^{-99}$	$- .9999E-99$
Small positive	$+ .9999 * 10^{-99}$	$+ .9999E-99$
Small negative	$- .9999 * 10^{-99}$	$- .9999E-99$

Table 5-6 provides examples of binary floating-point numbers that might be seen inside the AP-120B. In the examples given in the table, the first 10 bits are the biased exponent (these bits are labeled MD02 through MD11) with the most significant bit (MSB) being the bias bit. The next bit (labeled MD12) is the sign of the mantissa. The final 27 bits (labeled MD13 through MD39) are the significant bits of the mantissa.

Table 5-6 Examples of AP-120B Internal Floating-Point Numbers

	<u>Exponent</u>	<u>Sign</u>	<u>Mantissa</u>
Zero	0 000 000 000	0	000 000 000 000 000 000 000 000 000 000 000 000
1.0	1 000 000 001	0	100 000 000 000 000 000 000 000 000 000 000 000
-1.0	1 000 000 001	1	100 000 000 000 000 000 000 000 000 000 000 000
-1.0	1 000 000 000	1	000 000 000 000 000 000 000 000 000 000 000 000
AP Max	1 111 111 111	0	111 111 111 111 111 111 111 111 111 111 111 111
-AP Max	1 111 111 111	1	000 000 000 000 000 000 000 000 000 000 000 001
AP Nmax	1 111 111 111	1	000 000 000 000 000 000 000 000 000 000 000 000
AP Min	0 000 000 000	0	100 000 000 000 000 000 000 000 000 000 000 000
AP Nmin	0 000 000 000	1	100 000 000 000 000 000 000 000 000 000 000 000

As stated in a previous section, biasing is used in the AP-120B to add a constant to the true exponent. The net effect of adding a bias is to shift the entire number system up by the amount of the bias, thereby eliminating the need for a sign bit in the exponent. The AP-120B adds a constant of 512 to its exponents. This 512 is the bias value.

Unfortunately, the bias used in the AP-120B is usually different from the bias used for floating-point numbers in the host computer. Therefore, whenever a floating-point number is transmitted from the host to the AP-120B, the bias must be changed. For example, the floating-point format for the PDP-11 minicomputer uses an exponent bias of 128. Therefore, when a floating-point number from a PDP-11 enters the AP-120B formatter, the bias is effectively shifted two positions to the left in order to multiply it by four. This changes the PDP-11 bias of 128 to the bias of 512 required by the AP-120B.

When a 4-bit, 2's complement, binary number system was described in the section entitled "Two's Complement Notation" (paragraph 4.2.2.2), it was mentioned that there was a large negative number (-8) with no positive counterpart. This characteristic of 2's complement numbers shows up in Table 5-6 as two equivalent representations for -1.0. Because the largest positive mantissa is all 1's (approximately .999999 decimal) and its 2's complement is the mantissa shown in the table for the "-APMax" example, the maximum negative mantissa has a value of -1.0. Thus, -1.0 can be represented as either $-0.5 * 2^{**1}$ or as $-1.0 * 2^{**0}$.

When discussing normalized numbers, it was stated that a floating-point number is considered to be normalized if there are no leading zeros in the mantissa. Thus, .1790E4 is an example of a normalized decimal floating-point number, while .0179E5 is an unnormalized floating-point number (note that both of these numbers have the same numerical value).

The process called normalization takes an unnormalized mantissa, adjusts the radix point until the most significant digit of the mantissa is a non-zero value and then increments or decrements the exponent to properly reflect the change in the radix point position.

Assume, for example, that it is desired to normalize the number .0179E5. In order to normalize this number, the fraction is shifted one digit to the left, relative to the radix point. Thus, the mantissa becomes 0.1790. If, as with most computers, the word size is limited (in this example a four-digit word size is used), then the mantissa becomes .1790. Shifting the mantissa one digit to the left means that the exponent must be decremented by one to retain the proper numerical value of the number. Thus, the normalized floating-point number becomes .1790E4.

Internally, the AP-120B operates on 2's complement binary numbers. In a binary 2's complement format, a number is normalized when the sign and the most significant bit of the mantissa are different. Thus, as shown in Table 4-4, the number 1.0 is a normalized floating-point number while 0 is an unnormalized number. Again, notice that there is one legal unnormalized mantissa in the AP-120B format ($\pm 1/2$). This can be seen in the table as the first representation of -1.0.

An example of how the AP-120B normalizes numbers is shown below:

Result of a FADD operation before normalization:	Exponent	Sign	Mantissa
1 000 111 111	1 111 101 000	100 000 000 000 000 000	000 000 000 000 000 000
After normalization:	1 000 111 011	1 010 001 000	000 000 000 000 000 000

As can be seen in the above example, normalization was accomplished by shifting the mantissa left four places and then decrementing the exponent by four.

5.4.4 Floating-Point Addition

This section describes the method used to add binary floating-point numbers. All arithmetic operations, when reduced to their simplest terms, consist of nothing more than addition. Thus, subtraction consists of adding one number to the 2's complement of another number. Multiplication consists of adding the same number to itself a specified amount of times. Division consists of adding the 2's complement of one number to another number a specified amount of times. (In other words, subtracting one number from another as many times as specified.) Therefore, understanding how the floating-point addition process functions can provide the basic knowledge needed to understand all floating-point arithmetic operations.

There are four steps necessary in order to add two floating-point numbers. Each one of these steps is discussed separately in subsequent paragraphs. The four steps are:

- a. aligning the exponents
- b. adding the mantissas
- c. normalizing the sum
- d. rounding the normalized sum

5.4.4.1 Exponent Alignment

When initially dealing with floating-point numbers, it must be assumed that the numbers are unnormalized. Thus, the two numbers to be added most probably have their radix points in different positions.

The first step in addition, then, is to align the radix points so that the two numbers can be properly added. If the significant bits of the smaller floating-point number are shifted right and the exponent incremented with each shift, then the two radix points will be aligned in the same position once the two exponents are equal. In other words, making the two exponents equal effectively shifted the mantissa of one number so that its radix point lined up with the radix point of the other number. This process is referred to as "exponent alignment".

In order to perform exponent alignment, the two exponents must first be compared and then the mantissa of the smaller floating-point number must be shifted accordingly. This exponent comparison selects the smaller of the two floating-point numbers to be shifted and determines the number of bit positions it just shifted. The number of bit positions to be shifted is equal to the magnitude of the positive difference between the two exponents. In order to clarify this process, an example of exponent alignment with two decimal numbers is given below:

Problem: add X1 and X2

$$\begin{aligned} X1 &= 165.2 \text{ (or } .1652 \times 10^3 \text{)} \\ X2 &= 21.00 \text{ (or } .2100 \times 10^2 \text{)} \end{aligned}$$

Floating-Point Method

	Fixed-Point Method	Before Alignment	After Alignment
X1	165.20	$.1652 \times 10^3$	$.1652 \times 10^3$
X2	21.00	$.2100 \times 10^2$	$.0210 \times 10^3$
	-----	-----	-----
	186.20		$.1862 \times 10^3$

In the above example, X2 was selected as the smaller floating-point number. The difference between the two exponents was calculated ($3-2=1$) and the result (1) used to determine how many digits the mantissa of X2 should be shifted to the right. Once the exponents were aligned, the mantissas were added.

It should be noted that the AP-120B handles exponent alignment if the difference between the two exponents does not exceed 31. If the positive difference between the two exponents is greater than 31, then the AP-120B aborts the exponent alignment process and adds zero to the mantissa of the larger floating-point number.

5.4.4.2 Addition of the Mantissas

Once the exponents of the two floating-point numbers have been properly aligned, then the selected arithmetic operation is performed bit-by-bit on the two mantissas. In an add operation, for example, this means that the aligned 2's complement mantissas are added algebraically. In a subtraction operation, the mantissa of the subtrahend is 2's complemented and then algebraically added to the mantissa of the minuend.

The AP-120B actually performs six addition type operations on floating-point numbers. Although these operations are not described here, they are listed below for reference:

FADD	add: (A1) + (A2)
FSUB	subtract: (A1) - (A2)
FSUBR	subtract: (A2) - (A1)
FEQV	logical equivalence: (A1) XOR (A2)
FAND	logical and: (A1) AND (A2)
FOR	logical or: (A1) or (A2)

When using a fixed-word length machine (such as the AP-120B) to perform arithmetic operations on floating-point numbers, two possible error conditions can occur: exponent overflow and exponent underflow.

An exponent overflow condition arises when the value of the exponent of the data word is a greater positive number than can be expressed by using the allocated exponent word size of the machine. For example, if the exponent word size of a machine is two decimal digits, then an exponent of 109 would cause an exponent overflow. As stated previously, the exponent word size of the AP-120B is 10 binary digits (including the bit used for the bias value).

An exponent underflow condition arises when the value of the data word's exponent is a larger negative number than can be expressed by the exponent word size of the machine.

When adding mantissas, however, an overflow in the result is not an error condition. If mantissa overflow occurs, the machine simply shifts the mantissa to the right one bit position and then increments the exponent. The following example shows how mantissa overflow might occur and how it is handled.

Problem: add X1 and X2

$$\begin{array}{r} 9 \\ X1 = .9 \times 10 \\ 9 \\ X2 = .9 \times 10 \end{array}$$

Addition:

$$\begin{array}{r} 9 \\ X1 \quad .9 \times 10 \\ 9 \\ X2 \quad .9 \times 10 \\ \hline 10 \\ \text{sum } 1.8 \times 10 \end{array}$$

$$\begin{array}{r} 10 \\ \text{corrected } .18 \times 10 \quad (\text{mantissa right shifted; exponent incremented}) \end{array}$$

It should be noted that mantissa overflow only occurs if a digit of numerical significance carries to the left of the radix point during the addition. For example:

$$\begin{array}{r} 9 \\ 1.8 \times 10 \quad (\text{mantissa overflow}) \\ 9 \\ 0.8 \times 10 \quad (\text{no mantissa overflow}) \end{array}$$

Up to this point, two of the four processes required to add two floating-point numbers have been completed: exponent alignment and addition of the mantissas. The remaining two processes, normalization and rounding, are covered in subsequent paragraphs.

5.4.4.3 Normalization

Once the arithmetic operation is completed (by aligning exponents and adding mantissas), the preliminary result must be normalized in order to preserve the accuracy of the result. At this point, it might be helpful to look at the difference between a normalized and an unnormalized number.

4	
0.111 x 2	(normalized)
3	
1.110 x 2	(unnormalized)
5	
0.0111 x 2	(unnormalized)

From the above example, it can be seen that a number is considered to be normalized only if the two most significant bits are different. Notice that this does not necessarily mean that the most significant bit must be a zero. If the above numbers were to be complemented, the rule would still apply. For example:

-4	
1.000 x 2	(normalized)
-3	
0.001 x 2	(unnormalized)
-5	
1.1000 x 2	(unnormalized)

Normalization is achieved by shifting the mantissa until the two most significant bits are different. However, the shift element used for normalization must be capable of three operations if it is to take care of all possible cases that can occur. These three cases are:

- mantissa overflow** - If a mantissa overflow occurs, then the shift element must arithmetically shift right one place. For example, 1.8×10^x is an overflow condition that is corrected by a right shift ($.18 \times 10^5$).
- normalized result** - If the result of the operation is already normalized, then no shift in either direction is required.

unnormalized result - If the result is unnormalized, then the shift element must arithmetically shift left from 1 to the number of digits in the mantissa (there are 27 digits in the AP-120B mantissa).

When normalizing mantissas, it is possible to create an exponent overflow or underflow error condition. This can best be illustrated by examples.

The following is an example of exponent overflow which resulted from the normalization process. In this example, the size of the exponent is limited to two digits.

Problem: add X1 and X2

$$\begin{array}{r} 99 \\ X1 = .9 \times 10 \\ 99 \\ X1 = .9 \times 10 \end{array}$$

Addition:

$$\begin{array}{r} 99 \\ X1 .9 \times 10 \\ 99 \\ X2 .9 \times 10 \\ \hline 99 \\ \text{sum } 1.8 \times 10 \end{array}$$

Note that after the addition, there is a digit of significance to the left of the radix point. This indicates that mantissa overflow has occurred. However, mantissa overflow is not an error and can be corrected by normalization. The number is normalized by shifting right one place and incrementing the exponent as follows:

$$\begin{array}{r} 100 \\ .18 \times 10 \end{array}$$

4

However, as mentioned in the statement of the problem, there are only two digits of exponent. Therefore, the normalization process has caused an exponent overflow. That is, the true exponent is larger than can be displayed by the machine. When this occurs in the AP-120B, the system forces the result to be the largest positive number that can be represented by the hardware. In this example, the result would be:

$$\begin{array}{r} 99 \\ .99 \times 10 \end{array}$$

The following is an example of exponent underflow which resulted from the normalization process. In this example, the size of the exponent is again limited to two digits.

Problem: subtract X2 from X1

$$\begin{array}{r} -99 \\ X1 = .90 \times 10 \\ -99 \\ X2 = .89 \times 10 \end{array}$$

Subtraction:

$$\begin{array}{r} -99 \\ X1 \quad .90 \times 10 \\ -99 \\ -X2 \quad .89 \times 10 \\ \hline \\ -99 \\ \text{sum} \quad .01 \times 10 \end{array}$$

Note that after the subtraction, there is a leading zero after the radix point that must be eliminated. In other words, the number must be normalized. This is accomplished by shifting the mantissa left one place and decrementing the exponent as follows:

$$\begin{array}{r} -100 \\ 0.1 \times 10 \end{array}$$

However, the exponent of -100 is larger than can be displayed in two digits. Thus, the normalization process caused the exponent to underflow. That is, the exponent of the result is a larger negative number than can be displayed in the machine. When an underflow condition occurs in the AP-120B, the system forces the mantissa to zero and the exponent to its largest negative value. In this example, the result would be:

$$\begin{array}{r} -99 \\ .00 \times 10 \end{array}$$

5.4.4.4 Rounding

When shifting significant bits of the mantissa to the right in order to align the exponents of the floating-point numbers prior to addition, it is possible that the least significant bits of the number could be lost. This decreases the accuracy of the number, and ultimately, the accuracy of the calculation. To prevent this loss of accuracy, the AP-120B has three extra bits to the right of the mantissa. These bits, known as "guard" bits, preserve the information shifted right during exponent alignment. These bits are first used in the arithmetic operation itself and are later used in rounding the result.

When making the rounding decision, these three bits (called the "residue") are tested to determine if this residue is greater than one-half of the least significant bit of the mantissa. If it is greater, then the result is rounded -- if not, the result is not rounded. This process causes the cumulative rounding error to converge toward zero on repeated calculations.

Rounding can only be performed on a normalized number. Therefore, the result of an arithmetic operation is first normalized. Once normalized, the result can then be rounded.

Problem: add X1 and X2, normalize the sum, round the sum

$$\begin{array}{r} 5 \\ X1 = .9999 \times 10 \\ 1 \\ X2 = .5100 \times 10 \end{array}$$

Note that the word size of the mantissa in this example is four digits. The first step of addition is to align the exponents. The positive difference between exponents is 4. Because X2 is smaller than X1, the mantissa of X2 is shifted right four places. Thus, X2 is now .000051 x 10⁵ (remember, the exponent had to be incremented once for each shift). Now the numbers can be added as follows:

$$\begin{array}{r} 5 \\ X1 = .9999 \times 10 \\ 5 \\ X2 = .000051 \times 10 \\ \hline \\ 5 \\ \text{sum} = .999951 \times 10 \end{array}$$

At this point, the problem states that the sum is to be rounded. Because, in this example, the residue of 5 is greater than half of the least significant digit, which is 9, the number must be rounded. Rounding is accomplished by adding a rounding constant of .0000499 to the result of the calculation as shown below:

$$\begin{array}{r} & & 5 \\ \text{sum of } X_1 + X_2 & .999951 & \times 10 \\ \text{rounding constant} & .0000499 \\ \hline & & 5 \\ \text{result} & 1.0000000 & \times 10 \end{array}$$

Notice that this operation has resulted in a mantissa overflow condition. In order to obtain the proper result, the mantissa is shifted right one place and the exponent is incremented by one. Thus, the correct result of the calculation (add, normalize and round) is:

$$\begin{array}{r} & & 6 \\ & .10000 & \times 10 \end{array}$$

5.4.5 Floating-Point Multiplication

This section describes the method used to multiply floating-point numbers. Multiplication may be defined as adding a number to itself a specified number of times. For example, multiplying 2786 x 4 simply means that the number 2786 is to be added to itself four times. This definition of multiplication is valid for the binary system as well as for other number systems. This process is particularly appropriate for use with binary numbers because all hardware implementation can be based on add or no-add decisions.

Although multiplication by a computer may appear to be straightforward (that is, a series of additions), actual hardware implementation uses a specific algorithm in order to increase the speed of the multiplication. Therefore, this section is divided into four basic parts: binary multiplication, floating-point number multiplication, Booth's algorithm and a description of the specific integrated circuit that implements the multiply function in the AP-120B.

5.4.5.1 Multiplication of Binary Numbers

When multiplying binary numbers, the two numbers being multiplied are referred to as the "multiplier" (the number that multiplies) and the "multiplicand" (the number to be multiplied). The result of each step in the process is known as a "partial product" and the final result is called the "product."

The example below shows how binary multiplication can be performed with a series of additions. Each bit of the multiplier, starting with the least significant bit (LSB), is used to multiply the multiplicand. As each partial product is produced, it is entered in the appropriate bit weight column. In effect, each partial product is shifted one bit to the left. Finally, all bits are added to form the product.

1 1 0 0	(12)	multiplicand
1 1 0 1	(13)	multiplier

1 1 0 0		1st partial product
0 0 0 0		2nd partial product
1 1 0 0		3rd partial product
1 1 0 0		4th partial product

1 0 0 1 1 1 0 0	(156)	final product

Two significant facts should be noted from the above example as they can be helpful when reading the description of the multiplication algorithm presented later in this section. The first fact is that binary multiplication is actually an add or no-add decision. That is, if the multiplier bit is a 1, the entire multiplicand is added to the partial product. If the multiplier bit is a 0, then nothing is added (or all zeros are added) to the partial product. The second fact is that, regardless of the add or no-add decision, each partial product is shifted left one place from the previous partial product.

Rather than listing each partial product as it occurs and then adding all of them to produce the final product, the partial products could be added in partial product adders. This process is shown below:

1 1 0 0	(12)	multiplicand
1 1 0 1	(13)	multiplier

1 1 0 0		1st partial product
0 0 0 0		2nd partial product

0 1 1 0 0		sum of products 1 and 2
1 1 0 0		3rd partial product

1 1 1 1 0 0		sum of product 3 and previous sum
1 1 0 1		4th partial product

1 0 0 1 1 1 0 0	(156)	final product (sum of product 4 and previous sum)

The method shown above is usually implemented in the multiplier hardware. An increase in multiplier speed is achieved by performing these partial product additions in parallel.

Another means of decreasing the multiplication time is to ignore the addition whenever the multiplier bit is a 0. However, even if the 0 multiplier bit is ignored, the shift must be accounted for during the next operation as shown below:

O's added	O's ignored
1 1 0 0	1 1 0 0
1 1 0 1	1 1 0 1
-----	-----
1 1 0 0	1 1 0 0
0 0 0 0	
1 1 0 0	1 1 0 0
1 1 0 0	1 1 0 0
-----	-----
1 0 0 1 1 1 0 0	1 0 0 1 1 1 0 0

← shift two places

5.4.5.2 Multiplication of Floating-Point Numbers

Whenever two numbers with exponents are multiplied, the numerical values are multiplied and the exponents are added. Thus, $32 \times 23 = 65$. This same rule applies to floating-point numbers because every floating-point number consists of a numerical value (mantissa) and an exponent. Therefore, when multiplying two floating-point numbers, the mantissas are multiplied and the exponents are added.

When the AP-120B performs floating-point multiplication, it also rounds the product of the mantissas, adds a shift count from the mantissa to the exponent, and then checks to see if there is a resultant overflow or underflow error condition.

The example below shows how two positive floating-point numbers are multiplied. Notice that the rules of binary arithmetic are used to multiply the mantissas and add the exponents. Although the AP-120B operates on a 38-bit word (28-bit mantissa and 10-bit exponent), the example below uses a 7-bit word for simplicity.

Exponent	Mantissa	Decimal Equivalent
1 0 1	0 1 0 1	5×2
0 0 1	0 1 1 0	6×2
-----	-----	
	0 0 0 0	
	0 1 0 1	
	0 1 0 1	
-----	-----	
<u>1 1 0</u>	<u>0 0 1 1 1 1 0</u>	30×2^6
sum	product	

The following example shows how a positive number is multiplied by a negative number. In this instance, the negative number must be 2's complemented prior to performing the multiplication. Because a positive number multiplied by a negative number produces a negative product, the resulting products of the multiplication must also be 2's complemented. Notice however, that only the mantissa portion of the product is complemented.

	Exponent	Mantissa	Decimal Equivalent
Original problem	1 1 0	0 1 0 1	5×2^6
	1 0 1	1 1 0 1	-3×2^5
	-----	-----	
Problem after negative number complemented	1 1 0	0 1 0 1	5×2^6
	1 0 1	0 0 1 1	$+3 \times 2^5$
	-----	-----	
		0 1 0 1	
		0 1 0 1	
		0 0 0 0	
		0 0 0 0	
	-----	-----	
Product	1 0 1 1	0 0 0 1 1 1 1	$+15 \times 2^{11}$
Product complemented (mantissa only)	1 0 1 1	1 1 1 0 0 0 1	-15×2^{11}

The final example in this section shows how two negative numbers are multiplied. Both numbers must be complemented prior to the multiplication. Because a negative multiplied by a negative produces a negative product, the final answer must also be complemented.

	Exponent	Mantissa	Decimal Equivalent
Original problem	1 0 1	1 0 1 1	-5×2^5
	0 1 0	1 0 1 1	-5×2^2
	-----	-----	
Problem after negative numbers complemented	1 0 1	0 1 0 1	$+5 \times 2^5$
	0 1 0	0 1 0 1	$+5 \times 2^2$
	-----	-----	
		0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0	
	-----	-----	
Product	1 1 1	0 0 1 1 0 0 1	$+25 \times 2^7$
Product complemented	1 1 1	1 1 0 0 1 1 1	-25×2^7

5.4.5.3 Booth's Algorithm

The purpose of Booth's algorithm is to increase multiplication speeds by providing a method that the hardware can use to implement the multiplication process more efficiently. Thus, when Booth's algorithm is implemented, the multiplication process takes fewer steps than normal.

When describing the multiplication process, it was stated that if a multiplier bit were a 1, the multiplicand was added to the existing partial product. While if the multiplier bit were a 0, the multiplicand could be ignored provided the required shift was accounted for in the next partial product. In order to clarify this point, assume that the following two numbers are to be multiplied:

1 0 1 1 1 0 1

1 0 0 0 0 0 1

The example below shows how these two numbers would be multiplied in the conventional way and how they would be multiplied if all 0 multiplier bits were ignored.

Conventional Method	Faster Method
1 0 1 1 1 0 1	1 0 1 1 1 0 1
1 0 0 0 0 0 1	1 0 0 0 0 0 1
-----	-----
1 0 1 1 1 0 1	1 0 1 1 1 0 1
0 0 0 0 0 0 0	
0 0 0 0 0 0 0	
0 0 0 0 0 0 0	
0 0 0 0 0 0 0	
0 0 0 0 0 0 0	
1 0 1 1 1 0 1	1 0 1 1 1 0 1
-----	-----
1 0 1 1 1 1 0 0 1 1 1 0 1	1 0 1 1 1 1 0 0 1 1 1 0 1

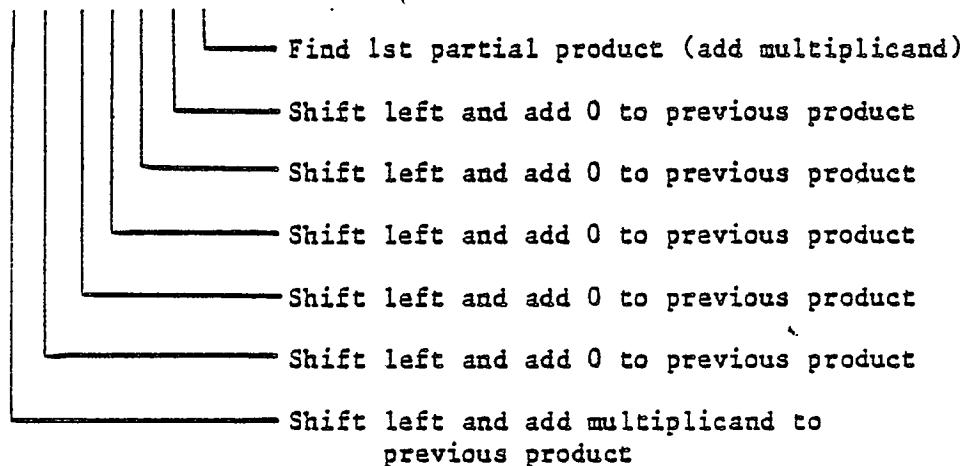
{ all 5 zeros in multiplier
ignored; next partial
product therefore shifted
6 places to the left, 5
for the zeros and 1 for
the MSB.

At this point, it might be beneficial to see how these two different methods might be implemented in a computer.

Conventional Method

1 0 1 1 1 0 1

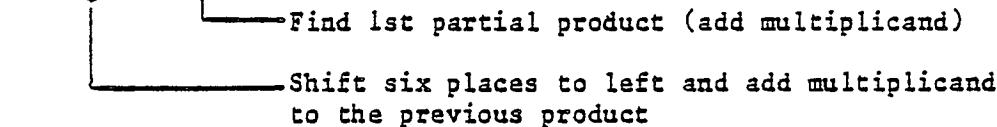
1 0 0 0 0 0 1



Faster Method

1 0 1 1 1 0 1

1 0 0 0 0 0 1



As can be seen in the above example, the conventional method requires seven add operations and six separate shift operations to perform the calculation. The faster method, however, requires only two add operations and a single shift operation.

In order to use the faster method, however, the computer must have the ability to see what bit is coming up next. In this way, it can keep track of the number of shifts that must eventually be performed. If the computer did not have this "look ahead" capability, it would shift each time it recognized a zero. And although unnecessary additions would be eliminated, the hardware would not be as efficient as it should be.

When one of the numbers to be multiplied is a negative number, it normally must be 2's complemented prior to being multiplied. This additional step is not required when using Booth's algorithm.

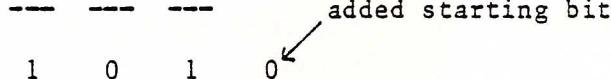
Booth's algorithm, then, can be said to provide faster multiplication because it provides a "look ahead" capability and because it eliminates the need for complementing the multiplier or multiplicand. The "look ahead" capability allows the machine to anticipate long strings of 1's or 0's, thereby eliminating many of the addition and shift operations normally associated with the multiplication process.

Before providing an example of multiplication which is performed according to Booth's algorithm, it is necessary to explain the basic concept of the algorithm.

Basically, Booth's algorithm compares two bits of the multiplier at a time and based on a table of rules, performs a specific operation depending on the results of the comparison. In order to provide a starting place for the comparisons, a 0 is added to the right of the least significant digit of the multiplier. For example, assume that the multiplier consists of the three digits 1 0 1. These digits would actually be written as:

Bit	Bit	Bit
0	1	2
---	---	---
1	0	1

added starting bit



When the multiplier bits are compared, the sequence is: the added bit is first compared with bit 2, then bit 2 is compared with bit 1, and then bit 1 is compared with bit 0. In other words, regardless of how many bits are in the multiplier, the process begins by comparing the added bit with the least significant bit and then continues by comparing each subsequent bit with the preceding bit. The result of each comparison determines the operation that is to be performed based on a table of rules. This table of rules is given in Table 5-7.

Table 5-7 Table of Rules for Booth's Algorithm

Bits Being Compared		Operation	Remarks	Shorthand Notation
Left	Right			
0	0	Do nothing	Write down all 0's for the partial product	$K + 0$
0	1	Add multiplicand	Use the multiplicand for partial product	$K + X$
1	0	Subtract multiplicand	Use the 2's complement of the multiplicand as the partial product	$K - X$
1	1	Do nothing	Write down all 0's for the partial product	$K + 0$

NOTES: Left bit means the more significant of the two bits being compared; right bit means the less significant

K = indicates the partial product; initially, $K = 0$

X = indicates the multiplicand

Multiplication according to Booth's algorithm is accomplished by using the following procedure:

- A. Write down the multiplier and the multiplicand, including the sign bits.
- B. Place a 0 to the right of the least significant bit in the multiplier.
- C. Begin with the added 0 and the least significant bit of the multiplier to compare the two bits.
- D. Based on the result of the comparison, perform the operation indicated in the table of rules (Table 4-5).
- E. Continue comparing bits and performing the required operations for all bits in the multiplier.
- F. Sign extend all partial products.
- G. Add all partial products to find the final product.
- H. Disregard any overflow bits in the final product.

As an example of this process, assume that 0110 is to be multiplied by 1011. The procedure is as follows:

Write down multiplier and multiplicands, including sign bits

$$\begin{array}{r} 0110 \\ \underline{1011} \end{array}$$

6
-5

Add 0 to the right of the multiplier

$$\begin{array}{r} 0110 \\ \underline{10110} \end{array}$$

Compare first two bits to determine operation which is subtract (1 0) at this point

$$\begin{array}{r} 0110 \\ \underline{10110} \end{array}$$

Subtract by using 2's complement of multiplicand as partial product

$$\begin{array}{r} 0110 \\ 10110 \\ \hline 1111010 \end{array}$$

1st partial product
sign 2's complement
extend

Compare next two bits.
Operation indicated here is to do nothing (1 1)

$$\begin{array}{r} 0110 \\ 10110 \\ \hline 1111010 \\ 0000000 \end{array}$$

2nd partial product

Compare next two bits.
Operation indicated here is to add multiplicand (0 1)

$$\begin{array}{r} 0110 \\ 10110 \\ \hline 1111010 \\ 0000000 \\ 00110 \end{array}$$

3rd partial product

sign
extend

Compare last two bits.
Operation indicated here is to subtract (1 0) by using 2's complement of the multiplicand as the partial product

$$\begin{array}{r} 0110 \\ 10110 \\ \hline 1111010 \\ 0000000 \\ 00110 \\ 1010 \end{array}$$

4th partial product

Add all partial products to find the final product.
Ignore overflow bits.

$$\begin{array}{r} 0110 \\ 10110 \\ \hline 1111010 \\ 0000000 \\ 00110 \\ 1010 \\ \hline 1000010 \end{array}$$

final product

ignore
overflow

Note that the result should be -30. If +30 is 2's complemented, the result is 1 0 0 0 0 1 0 which is the result obtained above.

There is a variation to this method that might be easier to use when attempting to do multiplication by using Booth's algorithm. In this variation, each of the two bits are listed to the left of the partial products. The table can then be used to write down the appropriate shorthand notation for the operation. Once this has been done, the multiplication can be performed. For example:

comparisons	notation	0 1 1 0 . 6
		1 0 1 1 0 -5
1 0	K-X	-----
1 1	K+0	
0 1	K+X	
1 0	K-X	

Once this has been done, the multiplication can then be performed according to the operations listed at the left. Thus:

1 0	K-X	0 1 1 0 . 6
1 1	K+0	1 0 1 1 0 -5
0 1	K+X	-----
1 0	K-X	1 1 1 1 0 1 0
		0 0 0 0 0 0
		0 0 1 1 0

		1 1 0 0 0 1 0 -30

When Booth's algorithm is implemented in a computer, the net effect is that the computer adds the multiplicand to the running product whenever a multiplier bit is 1 and simply shifts left corresponding to the number of zeros encountered in any given string. For example, assume that 00011 (+3) is to be multiplied by 01001 (+9). By following the rules for Booth's algorithm, the computer would effectively perform the following process:

0 0 0 1 1 +3

0 1 0 0 1 +9

0 0 0 0 0 0 0 1 1

Because first multiplier bit is 1, add multiplicand to running partial product (which is initially 0) and sign extend.

0 0 0 0 0 1 1

Shift left the number of bit positions required by the string of 0's then add multiplicand to running partial product when a 1 is encountered. This provides the final product.

0 0 0 0 0 1 1 0 1 1 +27

In summary, Booth's algorithm provides a look-ahead capability that permits long strings of 0's to be ignored, thereby increasing the speed of the multiplication. Another advantage of this algorithm is that negative numbers do not have to be complemented prior to the multiplication.

5.4.5.4. Multiplier Integrated Circuit

The multiply function in the AP-120B is implemented by using the AM25S05 2's complement digital multiplier integrated circuit. This chip is a special purpose adder that implements a 3-bit Booth's algorithm. In other words, the circuit can compare three bits at once rather than just two as described in the previous section. The table of rules for a 3-bit algorithm is given in Table 5-8.

Table 5-8 Table of Rules of 3-Bit Booth's Algorithm

LSB+1	LSB	LSB-1	Operation	Notation
0	0	0	Do nothing	$K + 0$
0	0	1	Add X	$K + X$
0	1	0	Add X	$K + X$
0	1	1	Add 2X	$K + 2X$
1	1	1	Subtract 2X	$K - 2X$
1	0	1	Subtract X	$K - X$
1	1	1	Do nothing	$K + 0$

NOTES: LSB = least significant bit

LSB-1 = bit to right of least significant bit

LSB+1 = bit to left of least significant bit

K = partial product (intiitially K=0)

X = multiplicand

The multiplier chip includes: a multiplier decoder, a shifting array, a completer, a high-speed adder and a control for sign and overflow. The Y inputs to the multiplier chip determine the function to be performed (such as $K+X$, $K-2X$, etc. as shown in Table 5-8).

A number of these multiplier chips are used in the AP-120B. The chips are divided into two separate arrays in order to increase multiplication speeds by providing parallel processing of data.

The entire 28-bit mantissa multiplicand is supplied to both arrays. The 28-bit multiplier is divided into two 14-bit portions, one portion being applied to one array and the second portion being applied to the other array. Both arrays operate simultaneously. The partial product from the first array is then added to the partial product from the second array in order to produce the final product of the multiplication.

In summary, the AM25S05 multiplier chip permits high-speed multiplications to be performed because the chip implements a 3-bit algorithm and because it is divided into two separate arrays to permit parallel processing.

5.4.6 FADD Hardware

This section describes the overall operation of the floating-point adder (FADD), the instruction set used with the adder and a detailed description of FADD hardware implementation.

5.4.6.1 Overall Operation

The floating-point adder (FADD) used in the AP-120B is a two-stage adder. During the first stage, the two input fractions are aligned and added. During the second stage, the result of the addition is normalized, rounded and checked for errors. Although a complete add operation requires two machine cycles (one cycle per stage), a "pipeline" method is used so that new inputs may be entered into the pipeline stream every cycle. In other words, new values can be entered on each cycle and a result can be extracted on that same cycle. Thus, once the pipeline is "primed" with its first inputs, a result can be obtained every 167ns (one cycle time), although the add operation itself requires 333ns. It should be noted, however, that although it takes two cycles for the add operation, the result is not available at the end of the second cycle. But, rather, it is available at the start of the third cycle.

The pipeline operation of the adder is illustrated in Figure 5-18. As shown in the figure, input values X₁ and Y₁ are loaded into the first stage and added during the first machine cycle. During the second cycle, values X₂ and Y₂ are loaded and added. At the same time, the second stage completes the normalization and rounding operations so that the sum of X₁ and Y₁ is placed in the output register. This sum is available at the start of the third cycle. Notice that at any given cycle, two new values are added by the first stage while the second stage completes processing of the two previously loaded values.

Figure 5-19 illustrates how a set of values is "pushed" through the pipeline. In other words, the figure shows how the sum can be obtained without continually feeding new values into the adder.

As shown in Figure 5-19, the FADD X₁, Y₁ instruction loads X₁ and Y₁ into the adder and the addition is performed during the first stage. A "dummy" FADD instruction is now used (that is, a FADD with no operands) to produce another machine cycle. This effectively "pushes" X₁ and Y₁ down the pipeline so that the second stage can perform the normalization and rounding operations. The result is then available at the start of the third machine cycle. It should be noted that when the values are pushed down the pipeline, the input registers retain their original values and a new X₁ and Y₁ are moved into the first stage. Thus, successive dummy FADD instructions will result in the same sum being produced on each subsequent cycle.

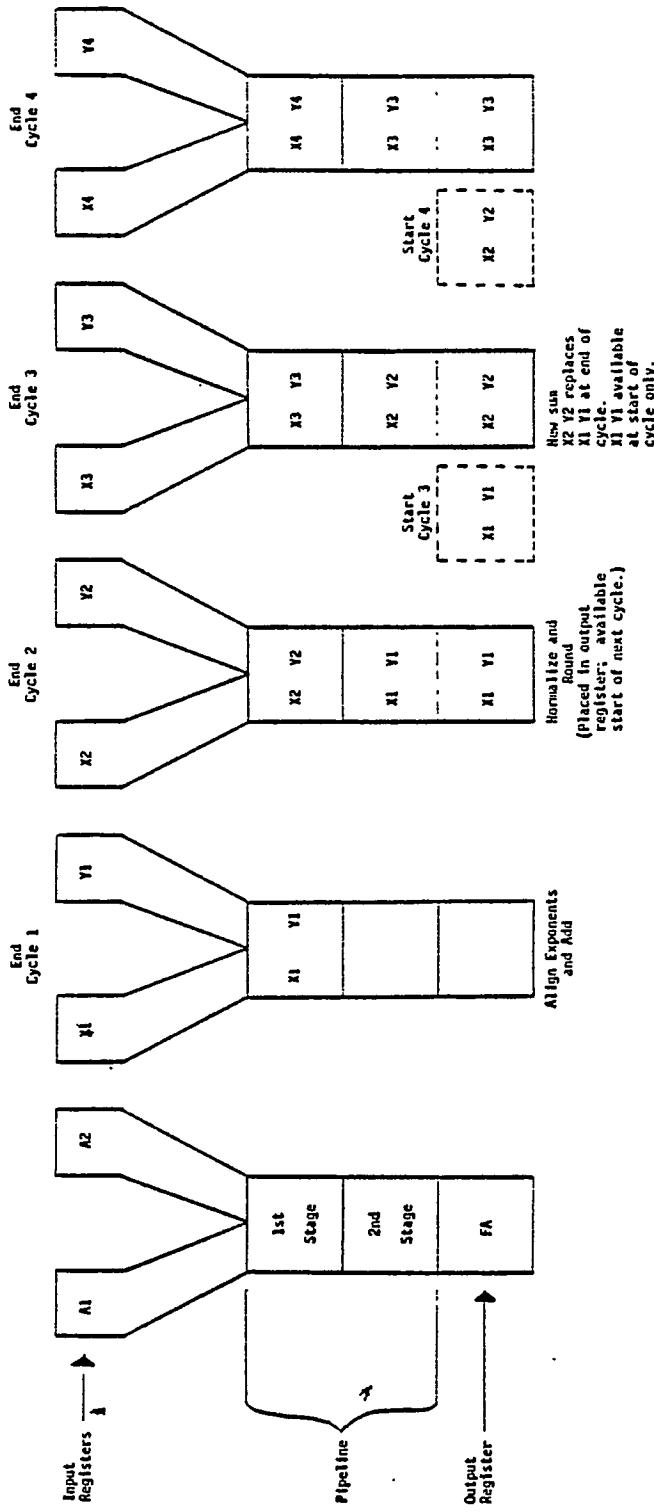


Figure 5-18 Adder (FADD) Pipeline Operation

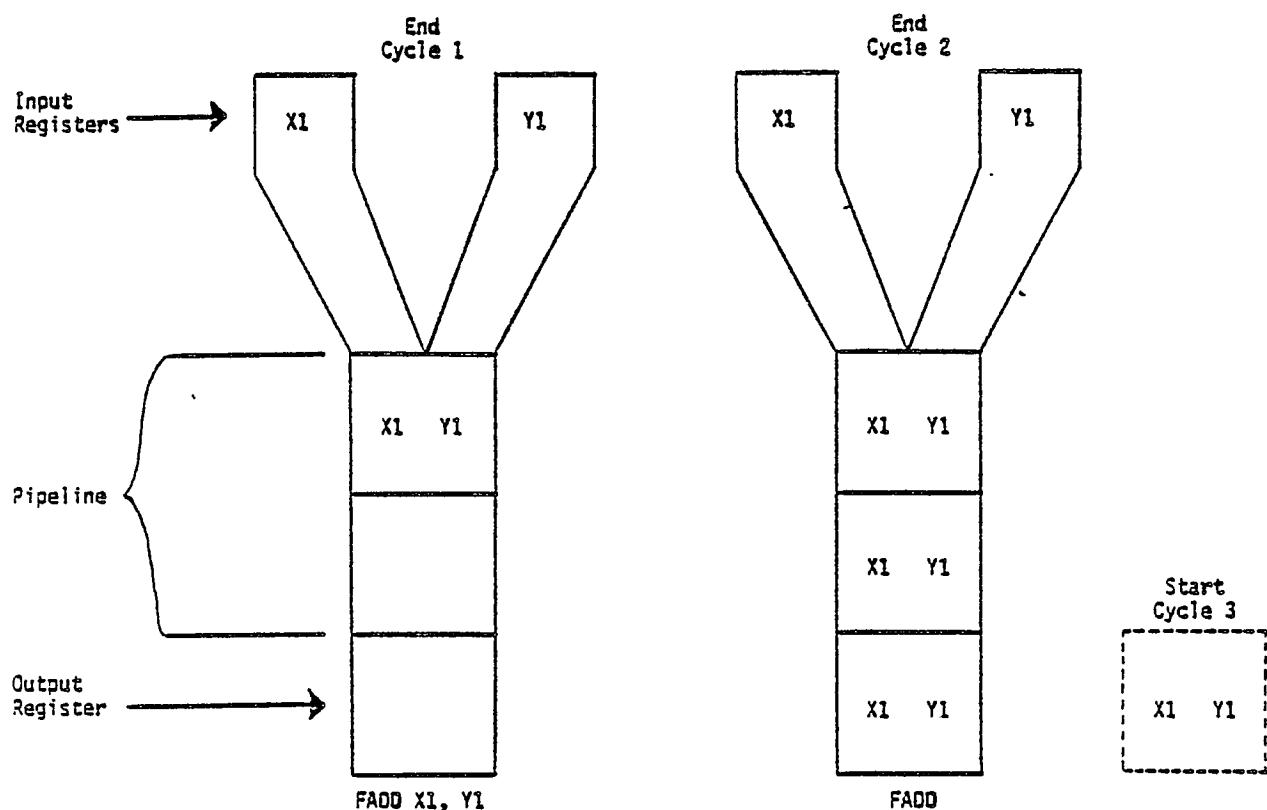
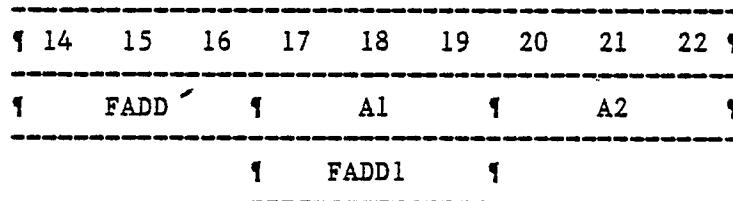


Figure 5-19 Pushing Values Through the Adder (FADD)

5.4.6.2 FADD Instructions

The floating-point addition element in the AP-120B (FADD) can selectively perform one of 14 different instructions in any one processor instruction cycle. The following is a visual presentation of the FADD control field of the 64-bit instruction word. Note that the FADD1 sub-field is also shown.



In every case but one, PS14 through PS16 of the 64-bit instruction word are decoded to determine the instruction the FADDER is to perform. The decode of the FADD field (PS14 through PS16) determines which of the two sub-fields are to be accessed or which of the six instructions in the field is to be executed. If the decode of the FADD field is 0, the instruction to be executed by the FADDER is determined by the decode of the FADD1 field. If the decode of the FADD field is 7 (seven), the instruction to be executed is an I/O-class instruction and the FADD hardware will not be used during the execution of this instruction cycle. An explanation of the I/O-class instruction will not be given here, but will be discussed in the section on the interface. Following is a table relating the octal decode of the FADD and FADD1 fields to the performed instruction.

Table 5.9 Octal Decode of the FADD and FADD1 Fields

FADD		FADD1	
Octal Code	Definition	Octal Code	Definition
0	Go to FADD1 field	0	No Operation
1	FSUBR	1	FIX
2	FSUB	2	FIXT
3	FADD	3	FSCLT
4	FEQV	4	FSM2C
5	FAND	5	F2CSM
6	FOR	6	FSCALE
7	Go to I/O field	7	FABS

The floating-point input arguments to FADD are held in two holding registers called A1 and A2. Single operand instructions (all those in the FADD1 sub-field) use A2 as the holding register for their floating-point input argument. The registers A1 and A2 are physically located on the FADD ECBs and are classified as part of the FADD hardware. A1 can be loaded from five possible sources and A2 can be loaded from seven possible sources. The following is a table relating the octal decode of the A1 field (PS17 through PS19) and the A2 field (PS20 through PS22) to its selected input source.

Table 5.10 Octal Decode of the A1 and A2 Fields

A1		A2	
Octal Code	Definition	Octal Code	Definition
0	No Change	0	No Change
1	FM	1	FA
2	DPX	2	DPX
3	DPY	3	DPY
4	TM	4	MD
5	Zero	5	Zero
6	Not Implemented	6	MDPX
7	Not Implemented	7	EDPX

The floating-point adder (FADDer) physically resides on three ECBs inside the AP-120B (boards 203, 204, and 205). Figure 5-20 is a block diagram showing the FADD system interconnections. FA, FM, TMreg, MDreg, DPX and DPY provide the floating-point input arguments to A1 and A2. Thus, A1BSnn*, A2BSnn*, FMnn*, and FAnn* are each 38-bit wide data buses. The portion of the 64-bit instruction word that controls the FADD hardware (PS14 through PS22) comes from program source \$216/236† and is decoded on EXPAN \$214†. The decoded instruction generates control signals that are routed directly to FADD and also two enable signals that are sent to the AP-120B system clock on ADDR \$212† to generate the clocks for the FADDer. The FADDer sends the feedback signal SELA2 to EXPAN \$214†. The FADDer sends four status signals to the AP status register which is located on CB1 \$210†. Three system signals are necessary for FADD operation. !PNLCLK (generated on ADDR \$212†) is used to load PS14 through PS22 into the control buffer. CBCLKE* is the signal used to enable the control buffer to be loaded. And SPIN* is the signal that is used to suspend the execution of an instruction.

The three ECBs that make up FADD (board nos. 203, 204, and 205) are called FADD1 \$203†, FADD2 \$204†, and FADD3 \$205†. All of the FADDer manipulation done on the mantissa during any of the operations performed by the FADDer is done on the FADD1 and FADD2 ECBs. FADD3 handles the exponent and the overflow/underflow detection. On FADD1 are two 28-bit wide holding registers (A1M and A2M), two two-to-one selectors (multiplexers), the exponent alignment shifter and the arithmetic and logic unit. The pipeline latch, the normalization hardware, the rounding hardware, and the special case selector physically reside on FADD2. Figure 5-21 presents the FADDer hardware in logical block diagram form. Figure 5-22 is a flow chart of the AP-120B FADD logic and presents in a flow chart form the description of section 5.4.6.2.

5.4.6.3 Input Latches

The two input latches A1 and A2 are fed from two 2:1 multiplexers that are part of the FADD hardware. The inputs to these multiplexers are FMnn* and A1BSnn* in the A1 case, and FA nn* and A2BSnn* in the A2 case. The output bus of FMUL (FADD), titled FM (FA), is not enabled onto the A1BS (A2BS) and then clocked into the A1 (A2) latch primarily because the setup time for the bus is not met by the output of FM (FA). A special "fast" path, with added selection logic, is necessary to implement this possibility. BS2A1 (bus to A1 register) is the signal that selects whether FM or A1BS is to be applied to the input of the A1 latch. BS2A2 selects whether FA or A2BS is to be applied to the input of the A2 latch. After the appropriate inputs have been enabled to A1 and A2, the signals !A1CLK and !A2CLK load the A1 and A2 registers respectively. It should be noted that the above description is logically true for both the exponent and mantissa portions of the A1 and A2 registers. But the hardware implementation of the exponent portion of A1 and A2 physically use an IC 2:1 multiplexer and an IC latch, while the mantissa portion of A1 and A2 use an IC 2:1 multiplexing latch. Thus, the input latch section of FADD diagrammatically looks like Figure 5-23. The exponent arrangement is dictated by the need to have both true and complemented outputs available from the latch.

5.4.6.4 Exponent Comparison Logic

After the input arguments have been latched, there is a 167 nsec time period in which the exponent alignment and the arithmetic or logic function must be completed. Then, this result is available to be latched in the pipeline latch. In order to assure that data is not lost, the worst case of propagation delay has been calculated to assure appropriate settling and setup time through this logic to the pipeline latch. Prior to beginning the exponent alignment, the smaller of the two exponents and the shift count must be determined. Propagation delay considerations necessitated that redundant hardware be utilized in the exponent comparison logic so that the shift count could be determined as early in the cycle as possible.

Figure 5-24 shows, in block diagram form, the exponent input latches and the exponent comparison logic. Both ADDER1 and ADDER2 are hard-wired to perform an A+B function and the carry is hard-wired to force a carry-in. Thus, by providing A2 and the complement of A1 to the inputs of ADDER1, the result (IM1*) will be the difference between A2 and A1 (A2-A1). Similarly, by providing A1 and the complement of A2 to ADDER2, the result (IM2*) will be the difference between A1 and A2 (A1-A2). Control signals SELA1 (select A1) and SELA2 are also generated as outputs of ADDER1 and ADDER2.

SEL_{A1} is the sign of the difference between A₁ and A₂. If the difference between A₁ and A₂ (A₁-A₂) is positive, then SEL_{A1} will be high and IM_{2*} will be high, and IM_{2*} will be selected as the output of MUX3 (the 2:1 multiplexer) and will be called DE (delta exponent). DE is the shift count and goes from FADD1 to FADD3 where the exponent alignment shifting is actually performed on the mantissa. If SEL_{A1} is low, the sign of A₁-A₂ is negative and IM_{1*} is selected as the positive difference between the exponents and used as the shift count.

SEL_{A2} is the sign of the difference between A₂ and A₁ (A₂-A₁), and is used to select which of the operands' mantissas is to be right-shifted. SEL_{A2} also informs the FADDer control logic which operand is applied to which input of the arithmetic and logic unit (ALU). This status is used when a subtraction operation has been selected.

SCIN (SCalar INhibit) is the logical OR of DE02, DE03, DE04, DE05, and DE06. If the magnitude of the exponent difference is greater than 31 decimal (37 octal), SCIN goes high and disables the exponent alignment shifter. Attempting to right-shift the 28-bit mantissa of the smaller operand more than 31 positions will shift the number off the end of the precision of the machine. So, rather than shift the number, the hardware inhibits the shift and forces a zero as the output of the shifter.

5.4.6.5 Exponent Alignment

The exponent alignment logic is presented in Figure 5-25. Notice that in this hardware implementation, there is only one shifter. While it is only necessary to have one (only one operand is to be shifted per instruction at this stage), it is necessary that the machine have the ability to shift the contents of either the A₁ or the A₂ register. Thus, MUX4 and MUX5 were added to the logic. The inputs to MUX4 are the contents of A₁ and A₂, and the output (Dnn*) is conditioned by SEL_{A2} as specified above. Again, the smaller of the two numbers (the one with the smaller exponent) will become Dnn*, the number to be shifted.

The exponent alignment hardware conditionally shifts D_{nn^*} in two stages. The first stage conditionally right-shifts D_{nn^*} from 0 to 7 positions. The second stage of the shifter then conditionally right-shifts the output of the stage 1 shifter 0 (zero)⁻ positions, 8 positions, 16 positions or 24 positions. As discussed above, the shift is conditioned by the positive magnitude difference of the exponents DE. If the magnitude of the shift is greater than the number of shift positions (31), the signal SCIN disables the output of the shifter, thus supplying a zero to the A inputs of the arithmetic and logic unit. It should be noted that a 28-bit mantissa is input to the shifter and a 31-bit argument is supplied at the output of the shifter. These extra three-bits of word size will be carried throughout the adder and eventually used in the rounding decision. Further, the bit bucket circuit was added. This circuit remembers if any bit shifted off the end of the shifter was a 1 (one) and if a 1 (one) was shifted into the bit bucket, then it is OR'd with bit 30 before the arithmetic operation is performed by the ALU.

5.4.6.6 Arithmetic and Logic Unit

The arithmetic and logic unit is fed from the exponent alignment shifter and a selector (MUX5). The selector's input is from A1M or A2M, depending upon which is smaller.

The arithmetic and logic unit physically resides on FADD1 §203-6† and consists of eight IC ALU chips (74S381) and three look-ahead carry generators (74S182). The ALU performs six operations depending on the state of the function select lines (FAS0, FAS1, and FAS2) of the ALUs. The actual operations are: B-A, A-B, A+B, A 'OE' B, A 'OR' B, A 'AND' B.

It should be noted that a sign extension takes place at this point. The sign-out of the exponent alignment logic and the selector is added together twice in the leftmost ALU. This is done to take care of the possible overflow case that can occur when adding two's complement numbers. (See the floating-point arithmetic summary section.)

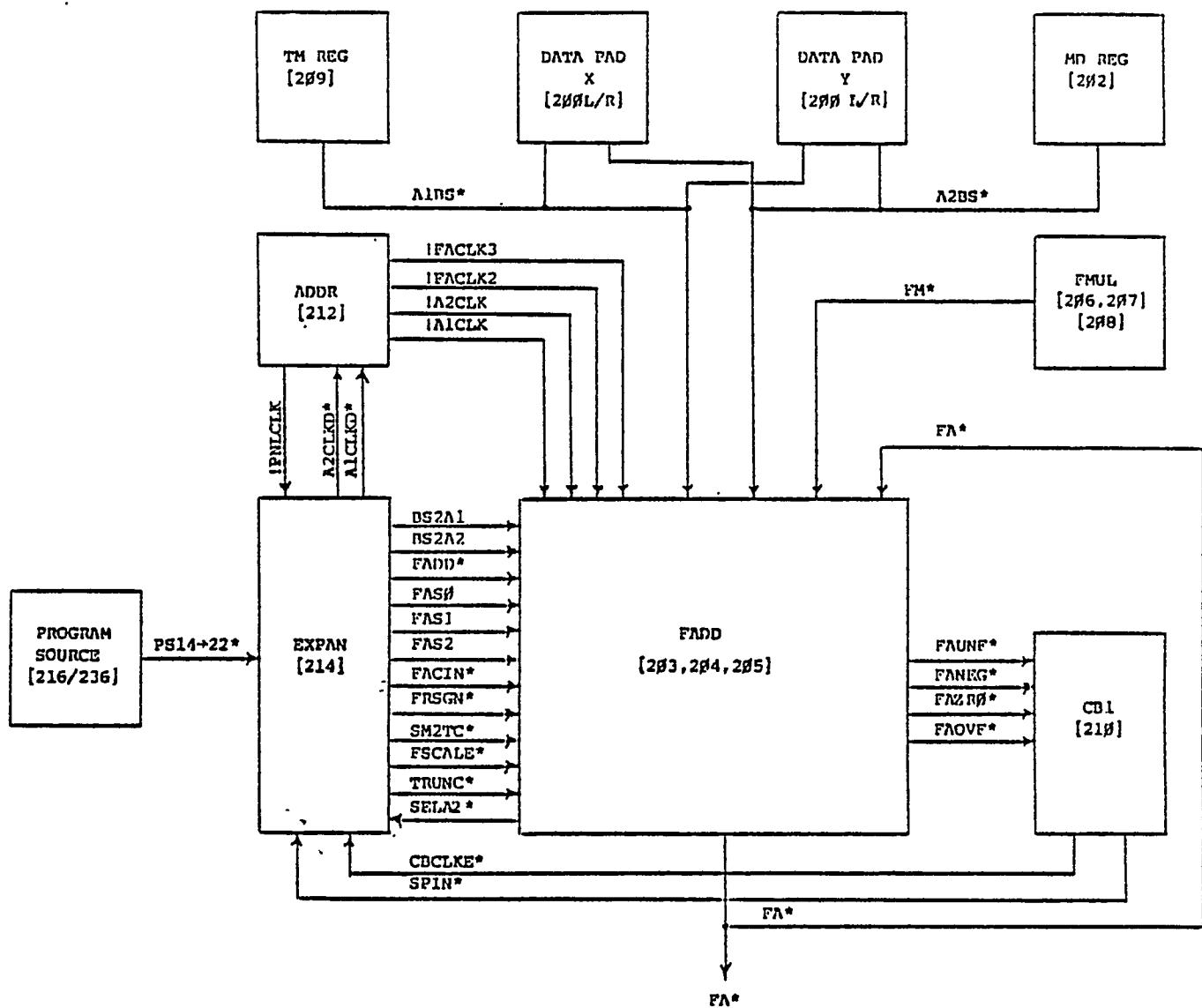


Figure 5-20 Interconnection Block Diagram

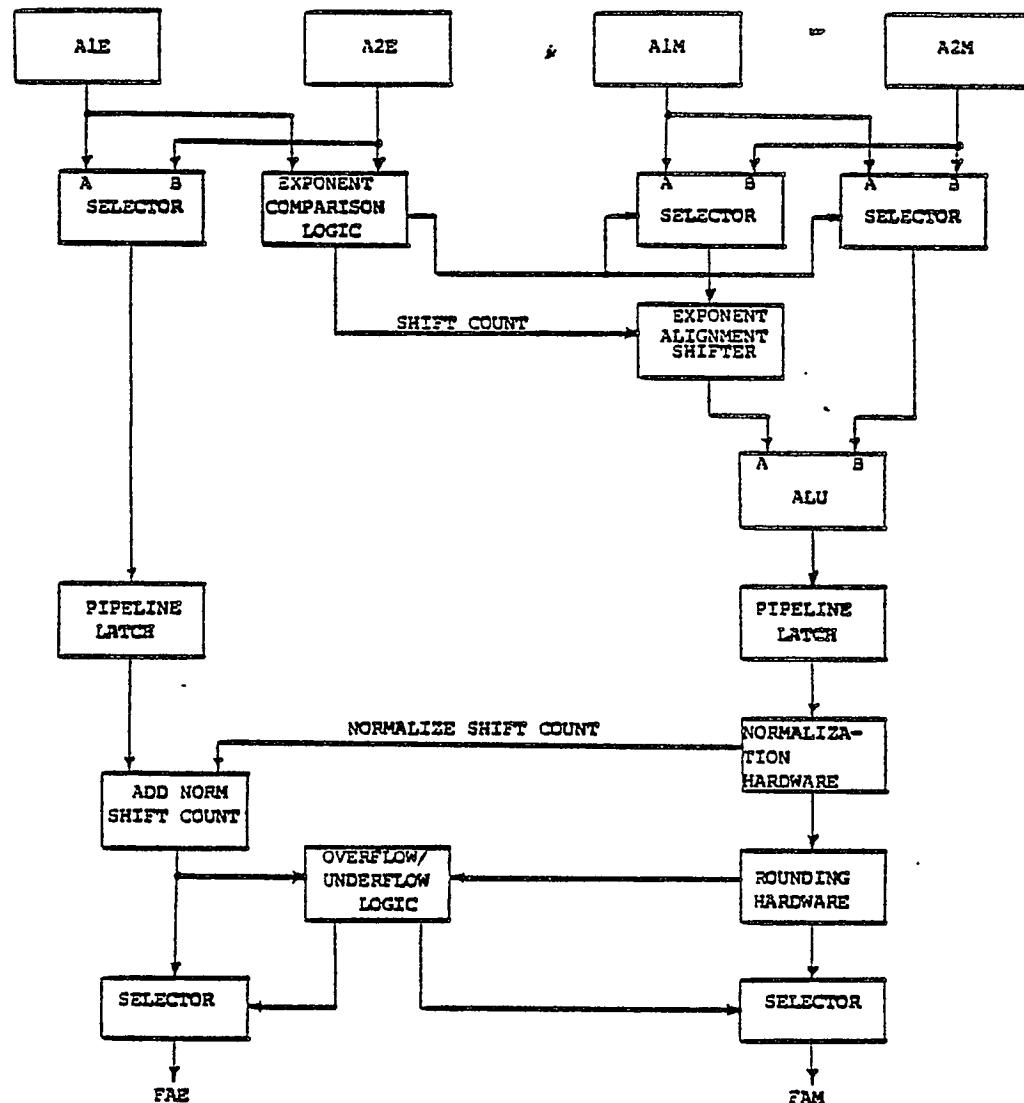


Figure 5-21 FADD Hardware Block Diagram

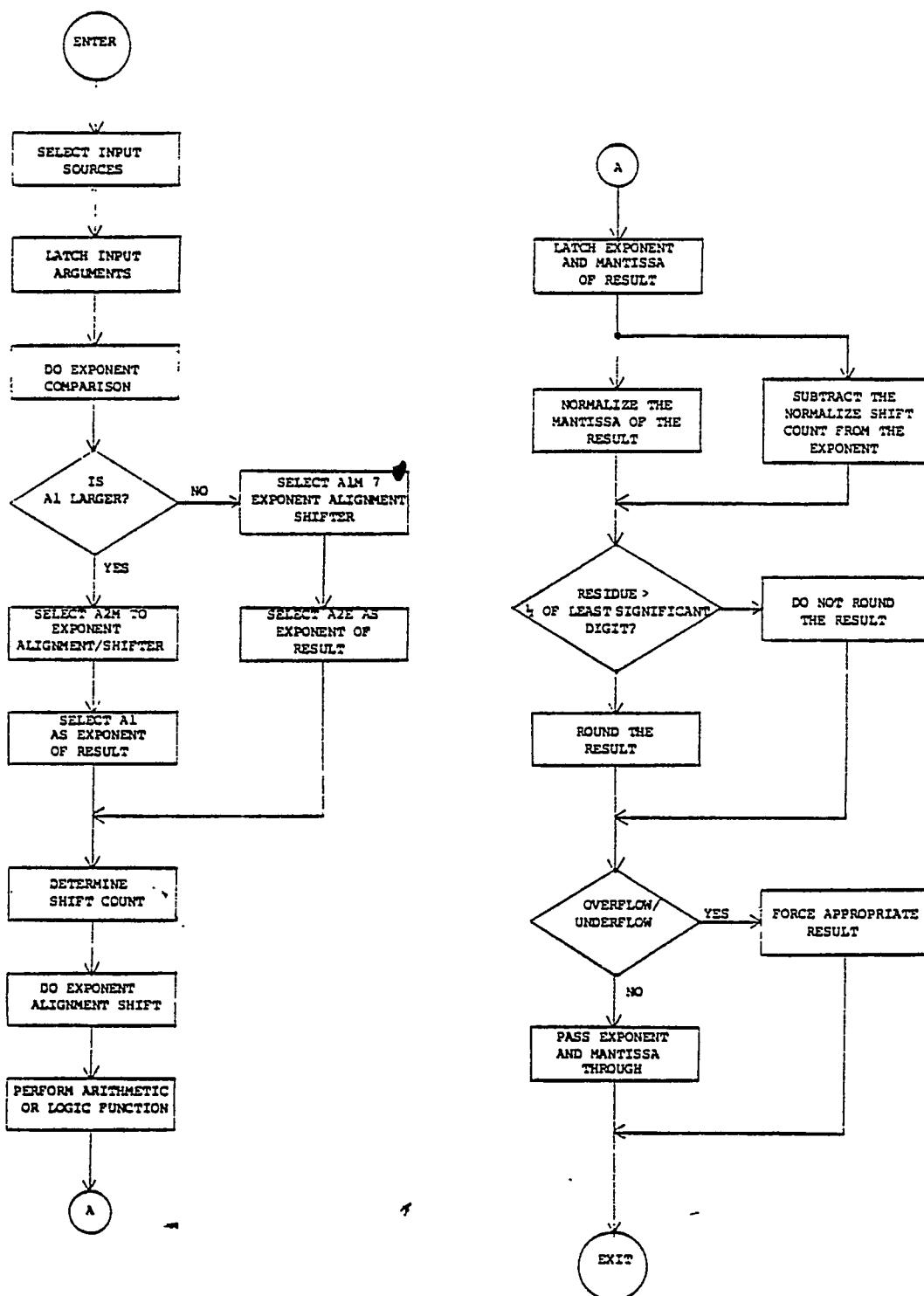


Figure 5-22 Flow Chart AP-120B Floating Adder Logic

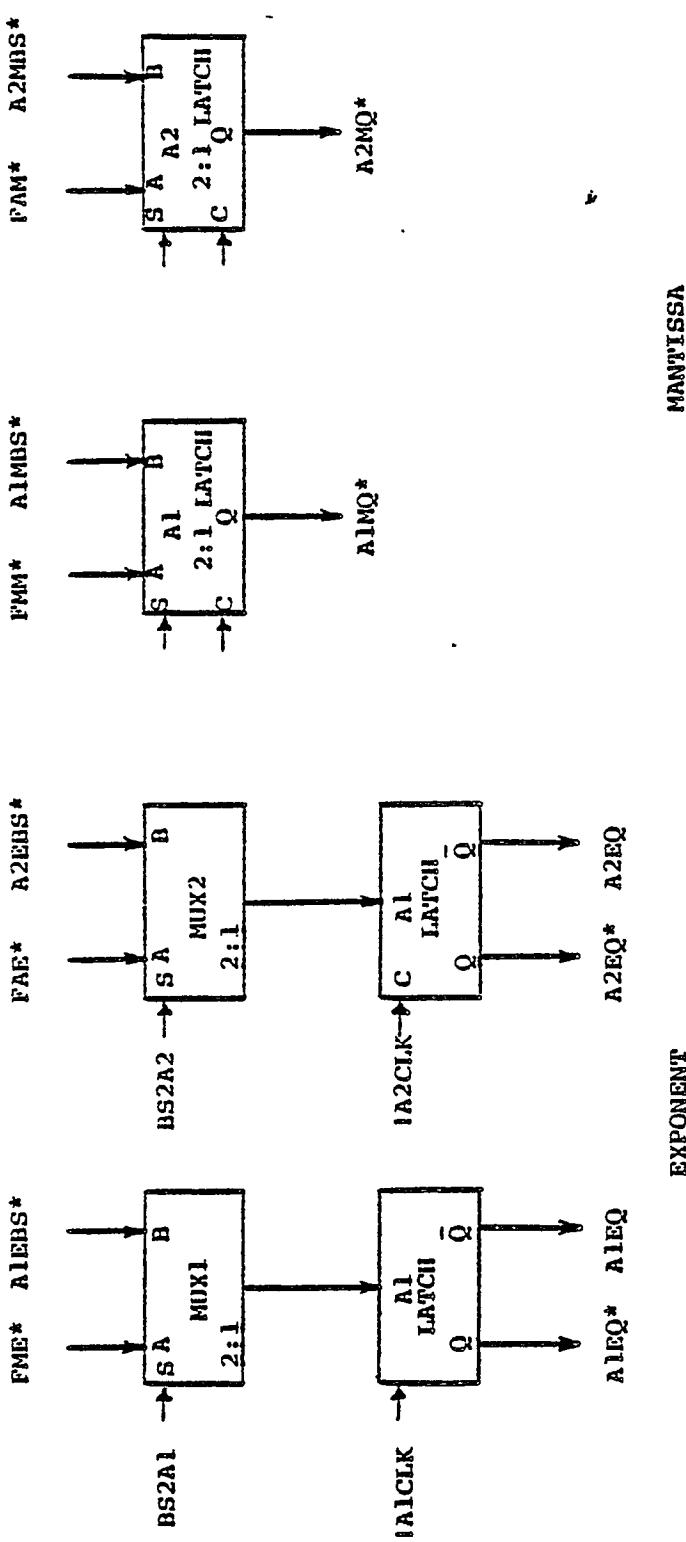


Figure 5-23 Floating Adder Input Latches

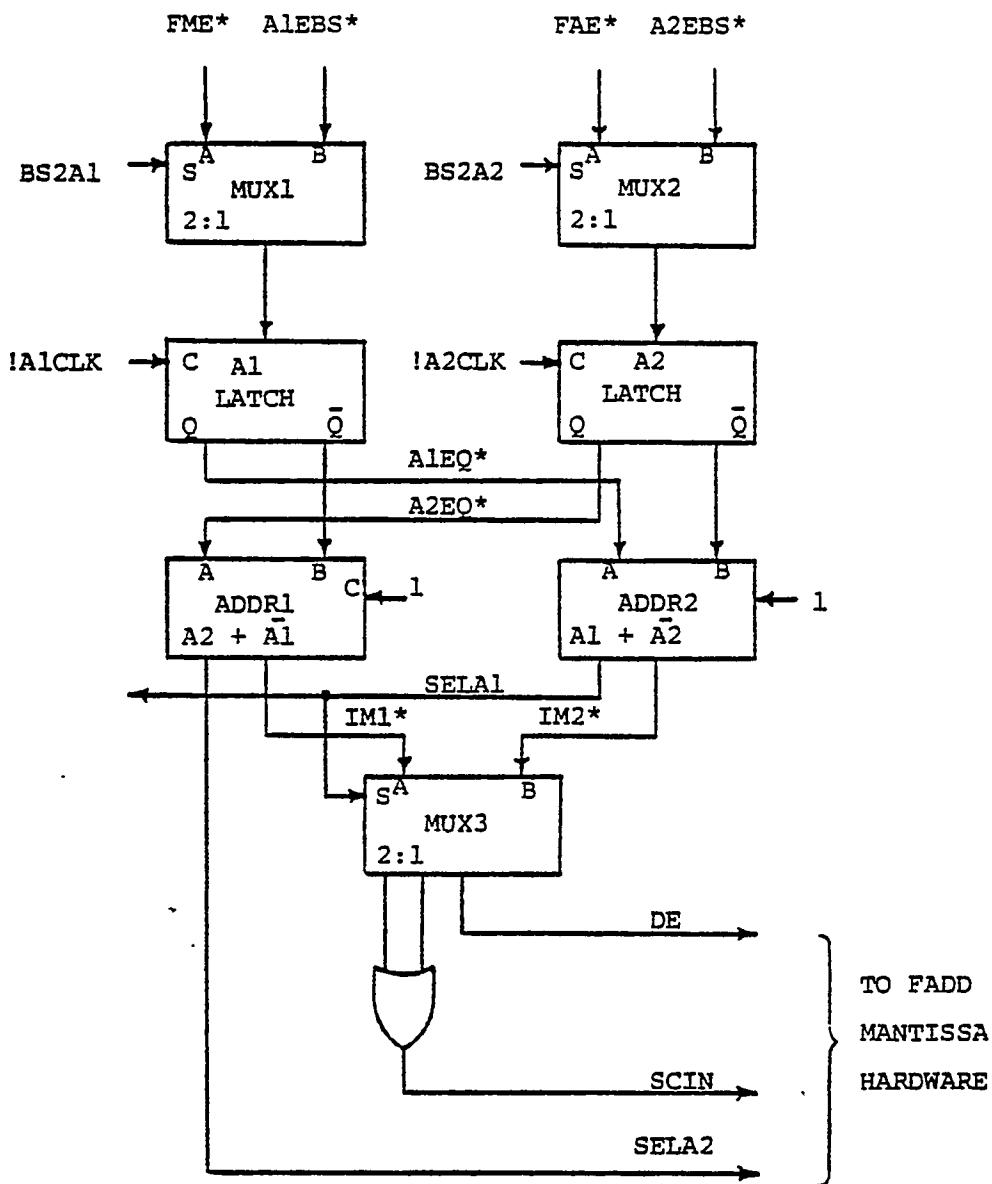


Figure 5-24 Input Latches and Exponent Comparison Logic

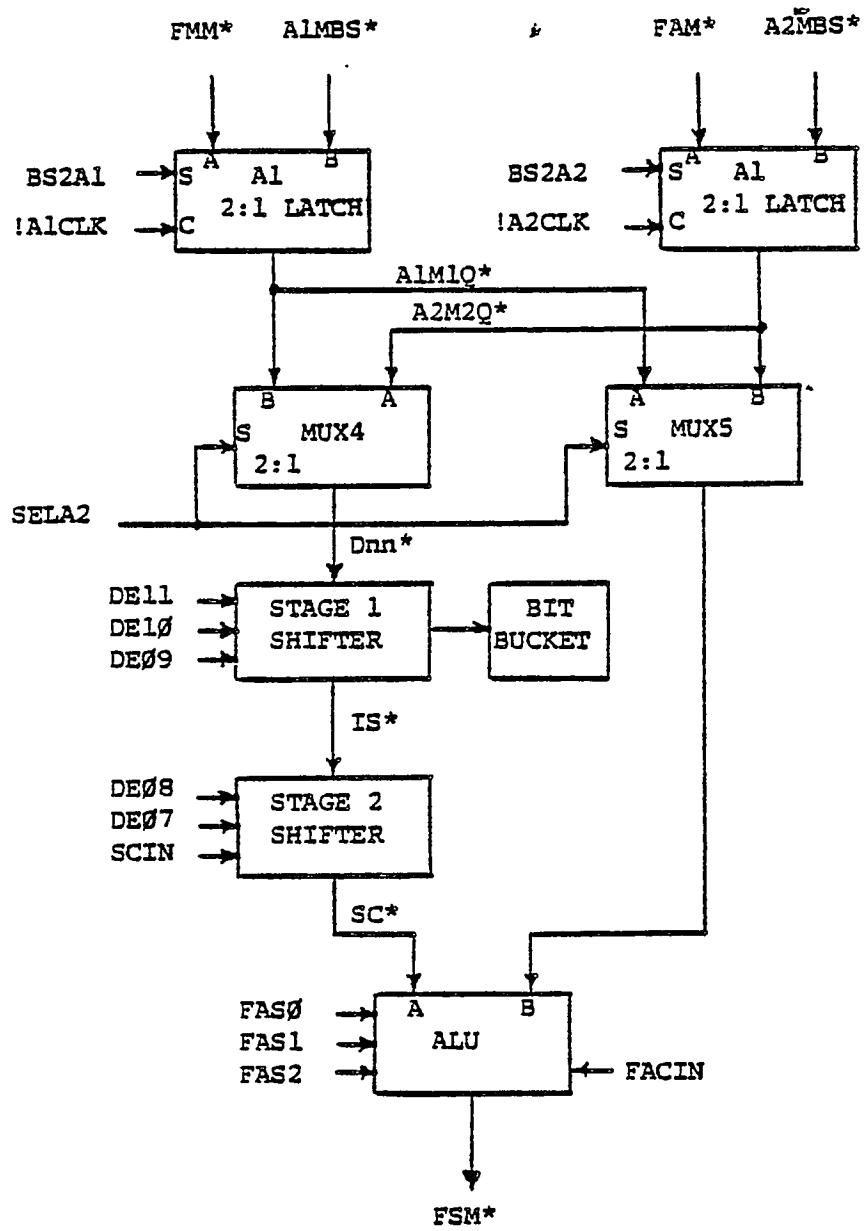


Figure 5-25 Exponent Alignment Logic

5.4.7 FMUL Hardware

The AP-120B has an arithmetic element that is dedicated to multiplying floating-point numbers. This arithmetic element is called the floating-point multiplier (FMUL). Figure 5-26 shows how the floating-point multiplier (FMUL) interconnects in the AP-120B system. Input operands to FMUL may come from data pad X (DPX), data pad Y (DPY) or the table memory (TM) via the M1BS; the output of FMUL (FM); data pad X (DPX) data pad Y (DPY) or main data (MD) via the M2BS; and from the output of FADD (FA). The output of FMUL is FM. FM may be used as a source for data into data pad X (DPX), data pad Y (DPY) or main data (MIreg).

Bits 51 through 55 of the AP-120B's 64-bit control word stored in program source (PSRAM), are used to control the floating point multiplier (FMUL). These bits (PS51* through PS55*) are decoded on control buffer 2 (CB2, ECB 211). The decode of these bits selectively enables the proper data onto the M1BS and the M2BS, selects the path the data is to be loaded from (BS2A1 and BS2A2), enables the data to be loaded (FMUL*A and FMUL*) and enables the clocks that cause the data to be loaded (!M1CLK and !M2CLK).

The AP-120B system clock is generated on ADDR (ECB 212) as are the FMUL system clocks. The FMUL system clocks, !FMCLKA, !FMCLKB and !FMCLKC are used to clock the data internal to the FMUL.

FMUL creates two status signals, FMOUTF* and FMUNFL*. These signals are sent to the AP-120B internal status register on control buffer 1 (CB1, ECB 210).

This discussion of the FMUL hardware will first give a brief discussion of the multiplier operation. Secondly, the multiplier exponent hardware will be discussed. Finally, the mantissa hardware will be presented.

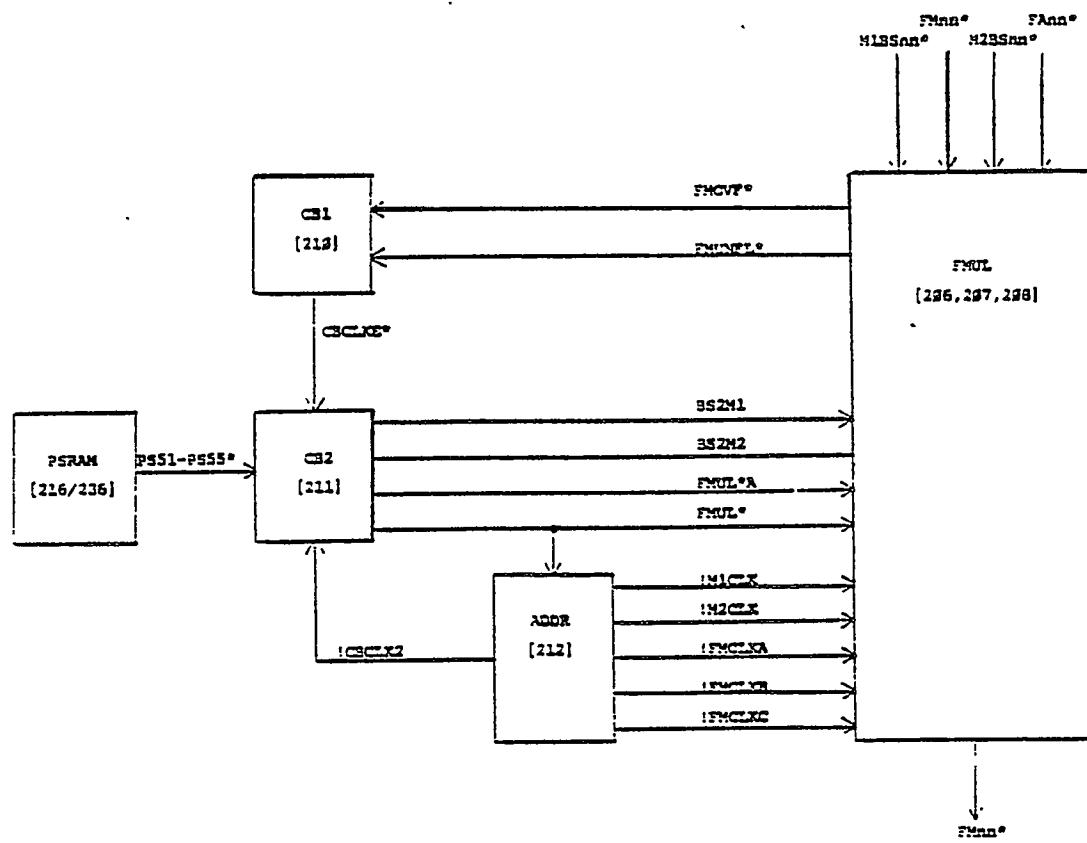


Figure 5-26 Floating Multiplier Interconnection Block Diagram

5.4.7.1 FMUL Overview

The floating-point multiplier (FMUL) used in the AP-120B is a three-stage multiplier. During the first stage, multiplication of the mantissas is started and the exponents are aligned and added. During the second stage, multiplication of the mantissas is completed. During the third stage, the result of the multiplication is normalized, rounded and error-checked. Although a complete multiply operation requires three machine cycles (one cycle per stage), a "pipeline" method is used so that new inputs may be entered into the pipeline stream every cycle. In other words, new values can be entered on each cycle and, as a result can be extracted on that same cycle. Thus, once the pipeline is "primed" with sufficient inputs, a result can be obtained every 167ns (one cycle time), although the multiply operation itself requires 500ns. It should be noted, however, that although it takes three cycles for the multiply operation, the result is not available at the end of the third cycle, but rather, is available at the start of the fourth cycle.

The pipeline operation of the multiplier is illustrated in Figure 5-27. As shown in the figure, input values X1 and Y1 are loaded into the first stage and multiplication begins during the first machine cycle. During the second cycle, values X2 and Y2 are loaded and multiplication of the mantissas started. At the same time, the second stage completes multiplication of the mantissas of X1 and Y1. During the third cycle, the first stage starts multiplication of X3 and Y3. The second stage completes multiplication of X2 and Y2. And the third stage completes the normalization, rounding, and error-checking of X1 and Y1. At this point, the product of X1 and Y1 is placed in the output register where it is available at the start of the next cycle (cycle 4). Notice that at any given cycle, the first stage starts multiplication of two new input values. The second stage completes multiplication of the previous values. And the third stage completes processing of the values that were loaded two cycles previously.

Figure 5-28 illustrates how a set of values is "pushed" through the pipeline. In other words, the figure shows how the product of two values can be obtained without continually feeding new values into the multiplier.

As shown in Figure 5-28, the FMUL X1, Y1 instruction loads X1 and Y1 into the multiplier which begins the multiplication of these values during the first cycle. A "dummy" FMUL instruction is now used (that is, a FMUL with no operands) to produce another machine cycle. This effectively "pushes" X1 and Y1 down the pipeline so that the second stage can complete the multiplication. Another "dummy" FMUL instruction pushes the values down to the third stage so that the product can be normalized, rounded and error-detected. This product is then available at the start of the third machine cycle.

It should be noted that when values are pushed down the pipeline, operation is different than that of the FADD (adder) pipeline. In the case of the multiplier (FMUL), each "push" down the pipeline enters all zeros into the input register. Thus, as shown on the figure, at the end of the third cycle, stages one and two both contain all zeros, while stage three contains the required product.

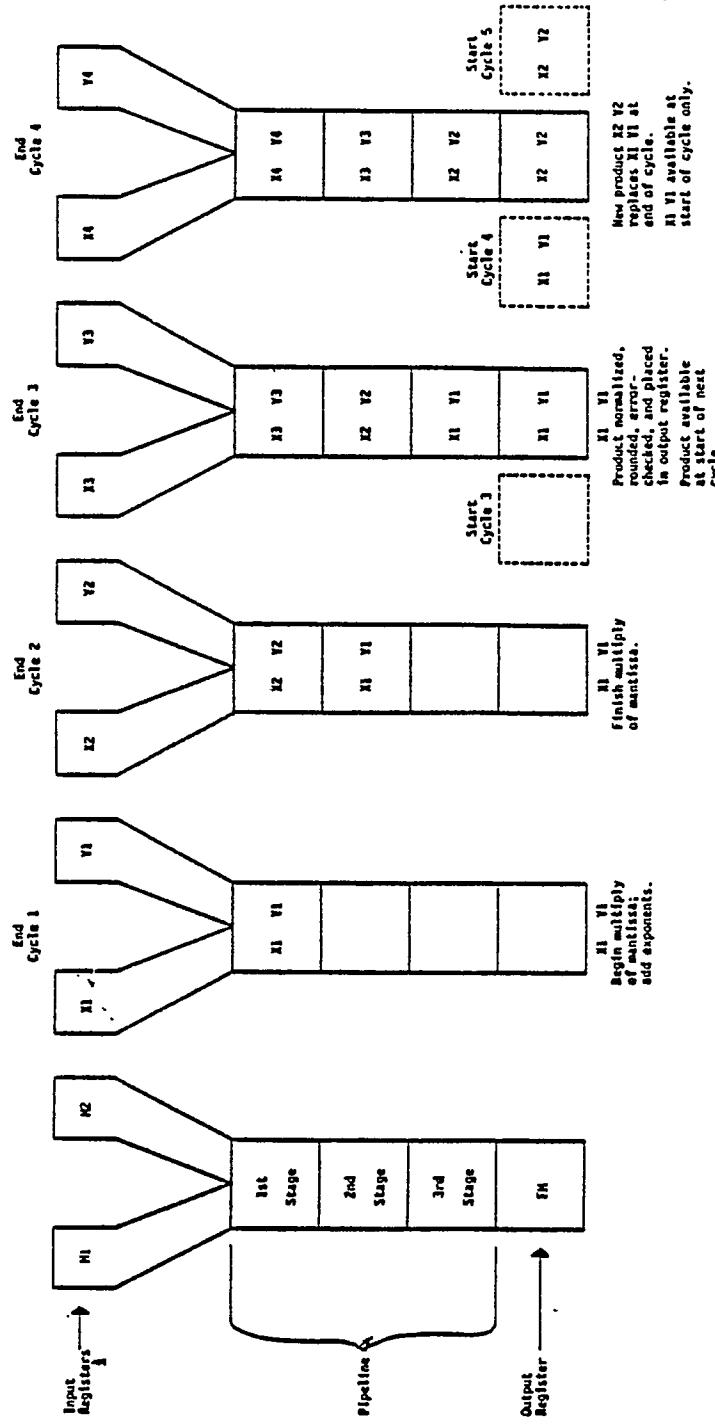


Figure 5-27 Multiplier (FMUL) Pipeline Operation

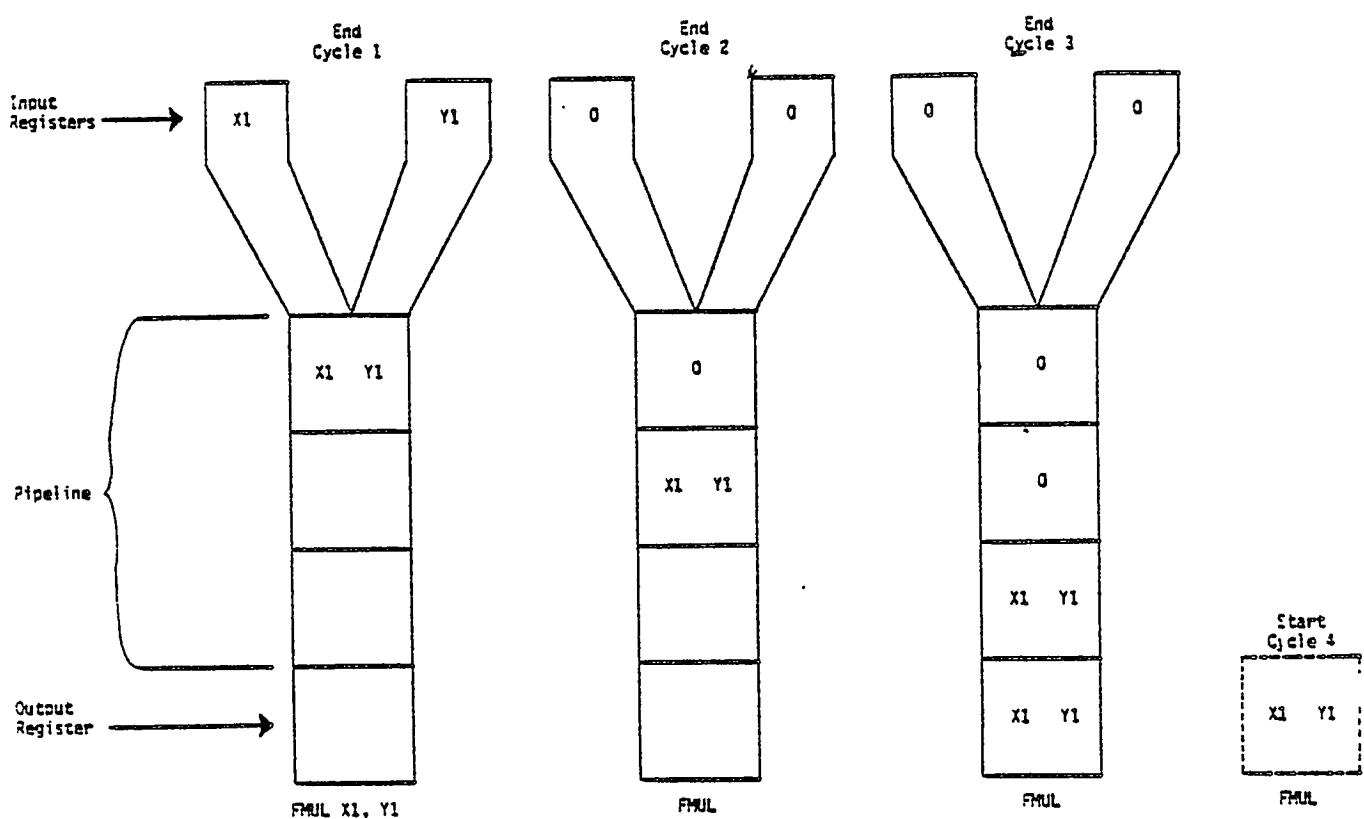


Figure 5-28 Pushing Values through the Multiplier (FMUL)

5.4.7.2 Exponent

Figure 5-29 shows the exponent portion of the floating multiplier (FMUL). The logic shown in Figure 5-29 is physically located on ECB 208. The exponent logic adds the exponent portions of the two input arguments in the M1 and M2 registers. The input to M1 may be data pad X, data pad Y, table memory or floating multiplier. The input to M2 may be floating adder, data pad X, data pad Y or main data.

The inputs to the M1 and M2 registers are selected by the current value of the M1 and M2 field of the instruction word. BS2M1 selects between the FM lines and the M1 bus for the input operand to the M1 register and on the low-to-high transition of !M1CLK, the selected input argument is loaded into the M1 register. BS2M2 selects between the FA lines and the M2 bus for the input operand to the M2 register. The outputs of the exponent portions of the M1 and M2 registers with the most significant bit of the M2 exponent register inverted, are then used as inputs to an integrated circuit adder. Inverting the most significant bit of the M2 register effectively subtracts +512 from the M2 argument.

The integrated circuit adder referred to above is wired to add M1 to M2 and add +1 to the sum ($M1+M2+1$).

The subtraction of +512 from the M2 arguments is done, so that when the exponents are added, the result is offset by only 512 (the normal bias for AP-120B floating point numbers).

The effect of adding one (1) to the exponents is the same as shifting the mantissa right once. (By adding one (1) to the exponent, the hardware that is needed to shift the mantissa for normalization will need only to shift left.) This completes stage one of the exponent portion of the multiplier.

Latch number 1, in Figure 5-29, forms the second stage of the exponent multiplier pipeline. When the next FMUL instruction is executed, the results from stage one are set in the latch and held until the next FMUL instruction.

Stage three of the multiplier (exponent) includes latch number 2, A+B adder and the 4:1 multiplexer. At FMUL* and !FMCLKC, the information from stage two is enabled into latch 2, then to the adder, and then to the multiplexer. The A+B adder adds the normalized shift count and MANOV (from the mantissa hardware) to the exponent. The adjusted output may be selected as FMEnn* if MPA and MPB are both false. MPA and MPB come from the overflow/underflow detection logic. The 4:1 multiplexer is capable of placing two other hard-wired answers onto the FMEnn* lines, which are the outputs of exponent logic based on the condition of the overflow/underflow logic.

5.4.7.3 Mantissa

The mantissa portion of the floating multiplier is shown diagrammatically in Figure 5-30. The multiplier is organized in a three-stage pipeline. Stage one does a partial multiply. Stage two completes the multiply. And stage three normalizes and rounds the result. Each stage takes one machine cycle (167ns) to complete. However, a multiply may be initiated every cycle with the results available three cycles later. The mantissa hardware is located on ECB's 206, 207 and 208.

The M1 input may be loaded from data pad X (DPX), data pad Y (DPY), the floating multiplier (FM) or the table memory (TM). M2 is loaded from data pad X (DPX), data pad Y (DPY), the adder (FA) or main data (MD).

The input to the M1 and M2 registers are selected by the M1 and M2 fields of the current instruction word. A FMUL instruction loads the M1 and M2 registers !M1CLK and !M2CLK. The multiplier is loaded into M2. The multiplication is done in two arrays. Each of the multiplier arrays multiply 14 bits of the multiplier by 28 bits of the multiplicand. The entire multiplicand is supplied to both arrays. But multiplier A receives 14 bits of the multiplier and multiplier B receives the remaining 14 bits of the multiplier. The most significant bits from the output of the multiplier arrays are loaded into a latch.

The least significant partial product bits are provided to the A+B adder in stage one. If the sum of the partial products produce a carry, it is supplied to the latch in stage two. Additionally, the adder makes a preliminary rounding decision to determine if the discarded bits are greater than one half of the least significant bit. The rounding decision is made during stage three of the multiplier.

The multiplier arrays are implemented with Amm25S05 four-bit by two-bit 2's complement multiplier ICs. The AM25S05 chip performs a Booth's algorithm multiplier. Booth's algorithm is an addition scheme that looks at more than one bit at a time to make a decision. The AM25S05 looks at three bits at a time. The propagation delay of the AM25S05 makes it impossible to complete a 28-bit by 28-bit multiply in 167mns. So the multiply is done in two stages. Stage one does a partial multiply and the results are latched and then completed in the next cycle.

The latch in stage two serves as a holding register for stage one partial products. A FMUL instruction loads the most significant bits into the stage two multiplier array where the multiplication is completed. MPSCVQ is loaded into the adder and MPS31Q* is loaded into the OR gate. The A+B adder adds the partial products form the multiplier array and MPSCVQ is added as a carry-in if it's true (high). The OR gate detects a low (true) from either MPS31Q* and the low order bit from the adder.

The latch in stage three serves as a holding register for the multiplier results. A FMUL instruction enables the results from stage two to the left-shifter. The left-shifter shifts the mantissa until a normalized condition is reached. The number of shifts (shift count) is supplied to the exponent logic.

The output of the shifter is applied to the rounder. The rounder is a 74S181 ALU wired for A+B with a carry-out. The ALU tests the residue bit from the left-shifter. If the discarded bits are greater than one half of the least significant bit of the shifted mantissa, the ALU rounds up, thereby increasing the mantissa by one. A carry-out of the rounder is provided to the exponent for further correction.

The output of the rounder is applied to the 4:1 multiplexer. The output may be positive maximum, negative maximum, zero or the result of the round.

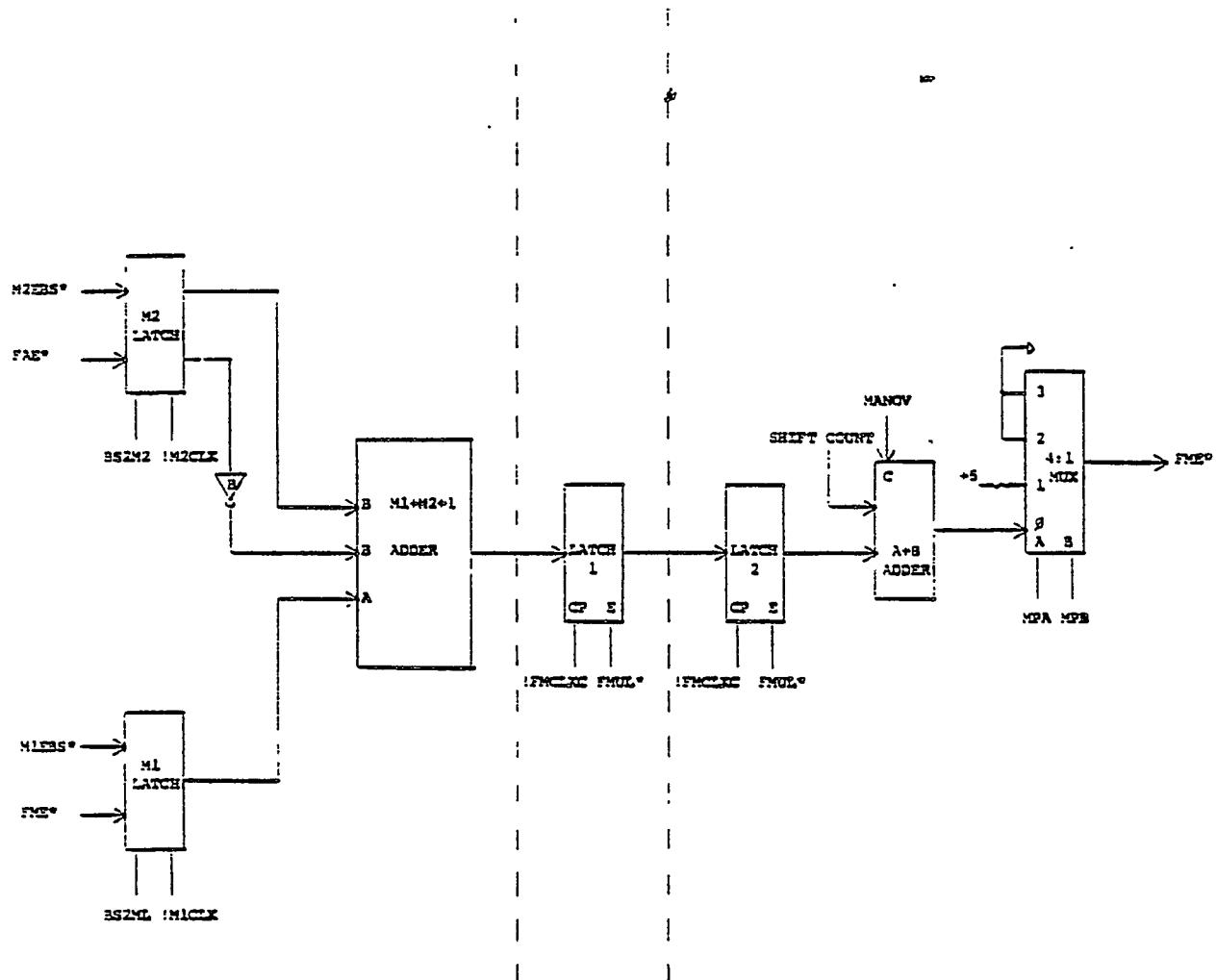


Figure 5-29 Floating Multiplier Exponent Logic

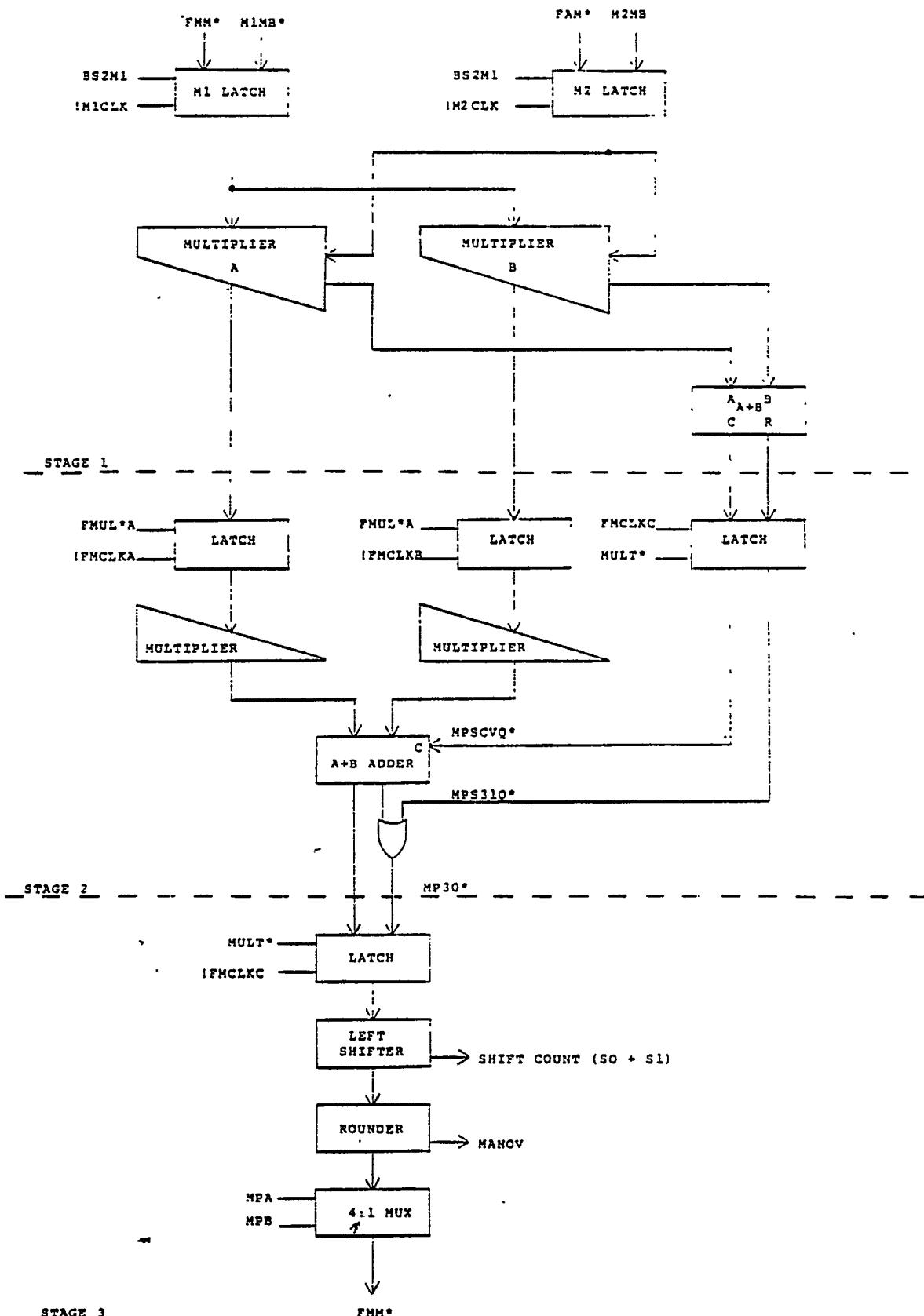


Figure 5-30 Floating Multiplier Mantissa Logic

5.5 CONTROL

The control portion of the AP-120B, which is covered in this section, consists of two main elements: program source address and the scratchpad (SPAD).

The program source address description includes a discussion of the memory used to store the program instructions, the instruction register and decoding logic, and the program counter.

The scratchpad description covers the 16 addressable registers used to perform "overhead" operations in parallel with arithmetic functions such as FADD and FMUL.

5.5.1 Program Source Address

This discussion of the AP-120B processor will discuss the program store, the instruction register and the program counter.

Like most programmable processors, the AP-120B has a memory in which the program is stored (program source, PS RAM), an instruction register (control buffer, CB), instruction decode logic, and a program counter (program source address, PSA). Figure 5-31 presents an interconnection block diagram of the AP-120B processor.

5.5.1.1 Program Source Memory

The program memory (PS RAM) is physically located on ECB 216 or ECB 236. ECB 216 can have up to 512 words of 64-bit wide program source instruction words. ECB 236 can have up to 2048 words of 64-bit wide program source instruction words.

Programs may be loaded into PS RAM from either the PNL bus or the data pad bus. The PNL bus is a 16-bit wide bus. Thus, in order to load one 64-bit instruction word into PS RAM, it must be loaded in four 16-bit bytes. Instructions are provided to allow PS RAM to be written in four 16-bit bytes. The data pad bus is a 38-bit wide bus. Thus, two 32-bit byte words can be loaded into PS RAM to make one 64-bit instruction word. Instructions are provided to allow PS RAM to be written in two 32-bit bytes.

PS RAM is implemented by utilizing bi-polar random access memory (RAM) integrated circuits. These were chosen for this application because of their fast access time. In order to write into these RAMs, the location to be written must be selected (the appropriate PSA applied) and the chip must be supplied with the write enable (WE). The write enables come from PSOWRT through PS3WRT. In order to read from these RAMs, the appropriate address must be applied to the chip (PSA).

5.5.1.2 Control Buffer

In the AP-120B, the instruction register and the instruction decode logic is called the control buffer (CB). The control buffer is physically implemented on three etch circuit boards: ECB 210, 211 and 214. ECB 210 is called control buffer one (CB1). ECB 214 is called EXPANSion (EXPAN). And ECB 211 is called control buffer 2 (CB2). CB1 latches and decodes the portion of the instruction (PS00*-PS13*) that controls the SPAD hardware. CB1 also decodes the portion of the instruction word (PS23*-PS26*) that controls the branch decision.

EXPAN latches and decodes that portion of the 64-bit instruction word (PS14*-PS22*) which controls the floating-point adder and I/O devices.

CB2 latches and decodes that portion of the instruction word which controls the data pads (PS32*-PS50*), the floating-point multiplier (PS51*-PS55*), the main data input register (PS56*-PS57*), and the address registers (PS58*-PS63*).

The clocks necessary to latch the data into the control buffer (!CBCLK1, !CBCLK2, and !PNLCLK) are generated from the master system clock on ADDRess (ECB 212). The system signals, which allow the control buffer to be loaded (SPIN*, CBCLKE*, and PSACLKE*), are generated on CB1 (ECB 210).

5.5.1.3 Program Source Address Logic

The AP-120B program counter, called program source address (PSA), physically resides on ADDRess (ECB 212). Figure 5-32 presents the PSA logic in block diagram form.

This diagram shows that PSA is not the output of a register, but an 8:1 multiplexer. In order to fetch an instruction from PS RAM every 167ns, it is necessary to process all possible next addresses from which the next instruction is to be fetched and then select the one to be used. In effect, all the addresses are processed in parallel with the decision of which is to be used. The decision of which is to be used comes from the decode of the branch decision logic which is done on CB1 (ECB 210). PSAAD, PSABD, and PSACD are the result of this decode and select one of the eight addresses. It should be noted that the input addresses to the 8:1 multiplexer, in every case, is the output of a register.

The eight possible addresses are PSA+1, PSA Jump, SRS, PSA, JMPA, JMP, TMA, and PNL. PSA+1 is the current program source address, plus one. PSA Jump is a short relative jump that adds PS27*-PS31* to the current program source address. This allows a branch range of +15 locations to -16 locations. SRS is the branch at the top element of a 16-element hardware subroutine return stack (SRS). PSA is the address of the current program source address. This allows an instruction to be re-fetched and is used to implement the step and continue functions. JMPA is an absolute jump to any location in program source. This operation is implemented by using PS52* - PS63* as the next address. JMP is a jump that makes it possible to branch to any location in program source, relative to the current program source address. This operation is implemented by adding PS52* - PS63* to the current program source address. The output table memory address register can be used as the next program source address. This is a handy register that can be set by the programmer. It is commonly used as the program source pointer when loading programs into program source. The PNL is commonly used to start executing the AP-120B from an address specified by the host computer.

The hardware subroutine return stack (SRS) is a 16-element last-in-first-out (LIFO) stack. Instructions are provided (JSR, JSRA, JSRT, and JSRP) allowing the current contents of the PSA+1 to be loaded onto the stack. The next PSA to be used is specified by PS52*-PS63*, PSA+(PS52*-PS63*), TMA or the PNL. A further instruction (RETURN) is provided to allow the top element of the stack to be used as the next program source address.

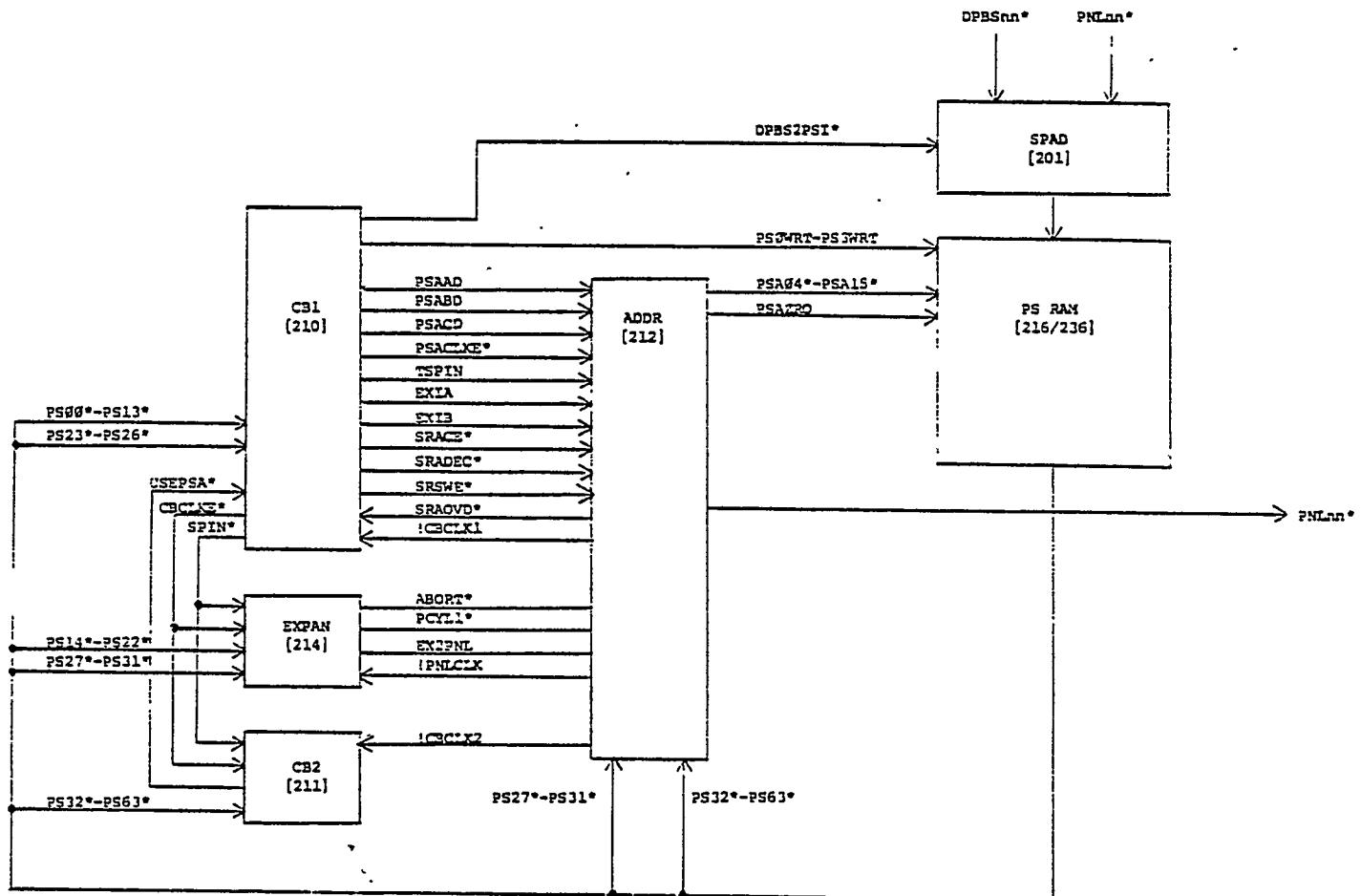


Figure 5-31 AP-120B Block Diagram

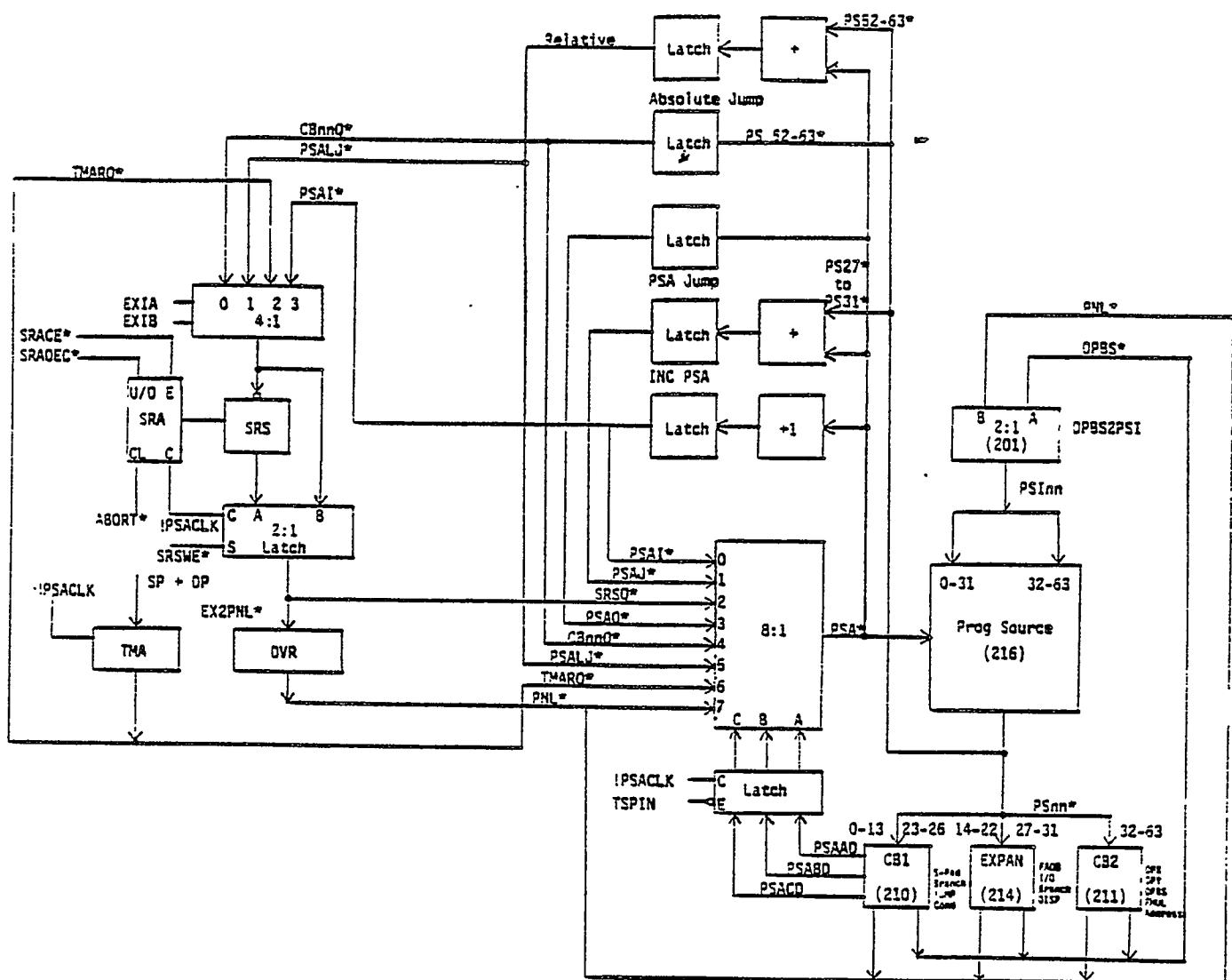


Figure 5-32 Program Source Address Logic

5.5.2 SPAD

Inside the AP-120B, SPAD performs operations normally thought of as "overhead" in parallel with arithmetic operations such as FADD and/or FMUL. The SPAD can be thought of a standard minicomputer with a limited memory size (16 elements). The SPAD normally performs address indexing, loop counting and other control functions for the executing program. Associated with SPAD is a 16-bit wide, 16-element register file and an integer arithmetic and logic unit.

Figure 5-33 presents, in block diagram form, how the SPAD interconnects in the AP-120B system. The control for the SPAD comes from program source (PS00* through PS13*). These bits of the control word are decoded on control buffer 1 (CB1, ECB 210). The SPAD hardware returns three status signals (SPFNCRY, SPZED and SPFN00) to CB1.

The SPAD registers can be externally loaded from the data pad bus or the panel bus. The clock for the SPAD, !SPCLK, is generated from the master system clock on ADDR (ECB 212). The output of the SPAD can be enabled onto the data pad bus, the panel bus, or SP+DP.

SPAD has a 16-bit wide register file with 16 directly addressable registers. (See Figure 5-34.) It is used to perform integer addressing and loop counting operations. These operations can be done in parallel with other machine operations. Thus, no computing time will be lost.

SPAD is capable of performing either single-operand or double-operand arithmetic or logic operations. Examples of single-operand operations are increment, decrement, and clear. Examples of double-operand operations are add, subtract, move, and, and or.

The operands are called SPAD source (SPspd) and SPAD destination (SPspd). A SPAD operation can address up to any two of the 16 registers in SPAD. The SPAD ALU then performs the indicated operation. The result of the ALU can then be shifted, if desired. The shifter is capable of performing a left-shift, a right-shift, a double right-shift, or no shift. The result is called the SPAD function (SPFN00-15*). The SPFNnn* lines are then loaded back into SPspd. The contents of SPspd are then lost. There is, however, a no-load capability which will inhibit the loading of SPFNnn* into SPspd.

Another capability of the SPAD is a bit reverse operation. This is used only in a fast fourier transform (FFT) operation. The bit reverse is used to access an array of data in a scrambled order. When the bit reverse is used, the contents of SPspd (bits 0-14) are bit reversed about bit 7. This can then be right shifted from 0 to 7 places depending upon the size of the array of data to be accessed. The shift count comes from bits 13, 14 and 15 of the AP status register. This must be programmed into this register according to the size of the array of data which the FFT is to operate on. The result of the bit reversed data goes to a 2:1 multiplexer. This multiplexer can choose from either the bit reversed data or the non-bit reversed data.

The control for the SPAD comes from program source (PS 00-13*). These are decoded on ECB 210. PS00* is termed decimate which controls the bit reverse. This actually controls the 2:1 multiplexer (which selects between the bit reversed or non-bit reversed data). The decode of PS01-03* controls which arithmetic or logic operation is to be performed. The decode of PS04-05* controls the shift count of the SPAD shifter. The decode of PS06-09* is used as the address of the source operand. And PS10-13* is used as the address of the destination operand. The no-load is controlled from the branch group PS23-26*.

SPAD initially may be loaded from one of four inputs through the 4:1 multiplexer. These inputs are the panel bus (PNL00-15*) data pad exponent bus (BPEBS02-11*), data pad mantissa bus (DPMBS12-27*), and data pad mantissa bus (DPMBS02-08).

The output of SPAD (SPFN00-15*), besides being loaded back into SPAD, can be enabled onto the SPAD or data pad lines (SP+DP00-15) through a 2:1 multiplexer. The other input to this 2:1 multiplexer is the data pad bus (DPMBS 12-27). The output of the 2:1 multiplexer is available to be loaded into the SPAD destination address register (SPD), the table memory address register (TMA), the main data address register (MA), the data pad address register (DPA), the AP status register (APSTAT), or the device address register (DA). Also the SPFNnn* can be loaded onto the PNL or DPMBS. The control is decoded from the I/O field (PS 14-22*). This decode is done on ECB 214 and ECB 211.

There are three conditions on which SPAD may be tested. These are the carry bit, the sign bit, and the condition in which SPAD is zero. These signals are sent to ECB 210 where the branch decision decode logic is located.

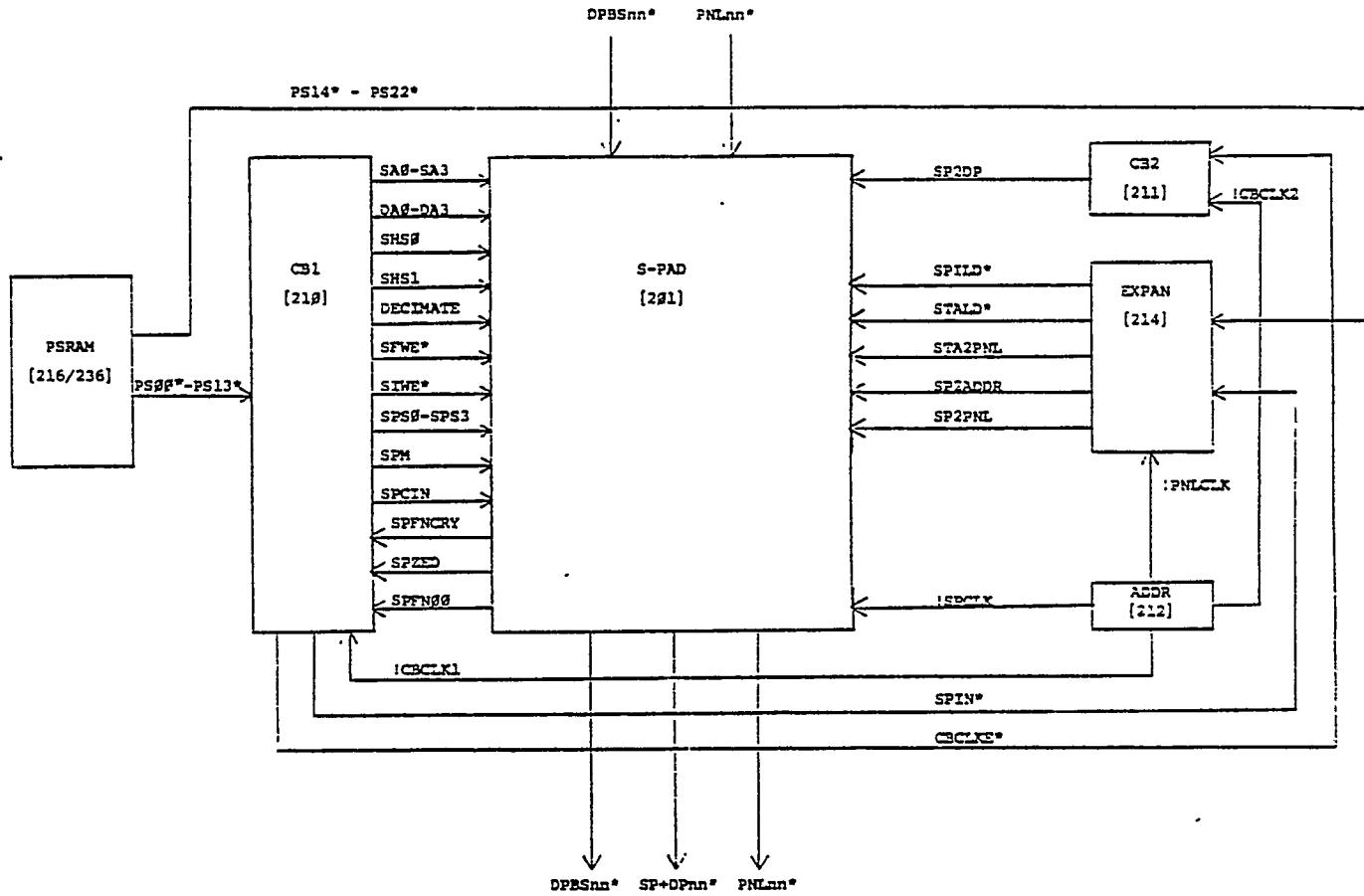


Figure 5-33 SPAD Interconnection Block Diagram

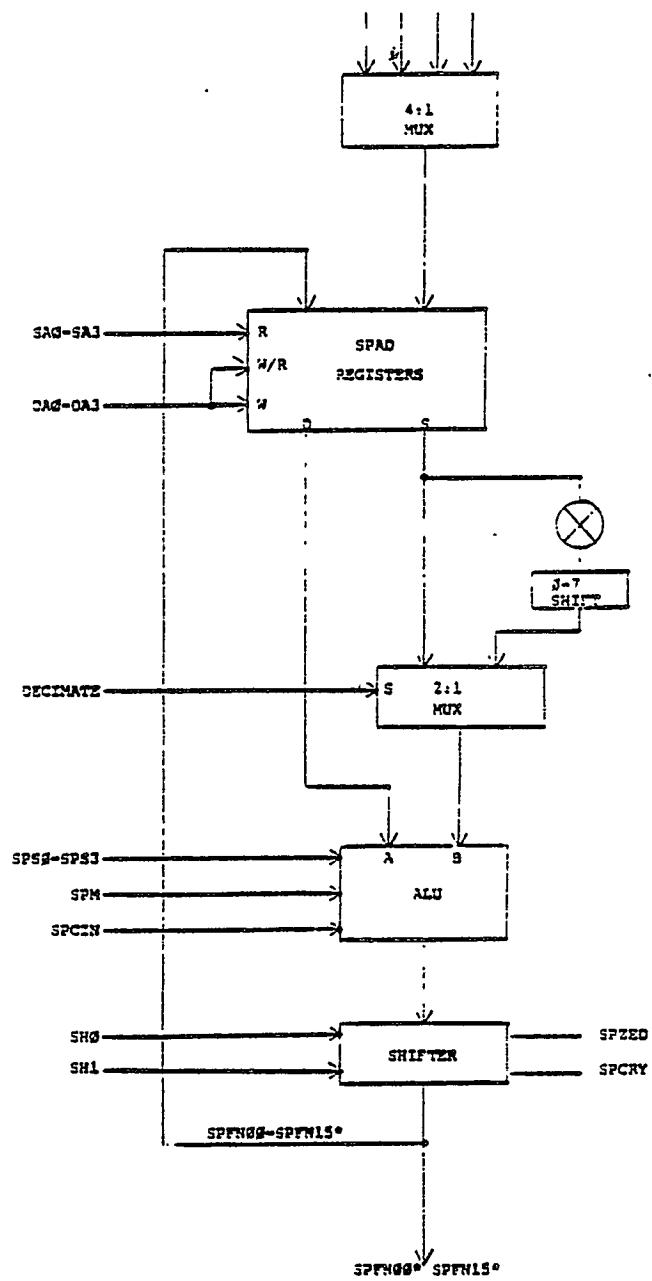


Figure 5-34 SPAD Block Diagram

CHAPTER 6

INTERFACES

6.1 INTRODUCTION

Because the array processor can be used with any one of a variety of host computers, an interface must be provided between the AP-120B and the host.

There are actually two interfaces that are part of the AP-120B. The first is a general-purpose interface containing the various registers and the bus required for communication between the AP-120B and the host. The second interface is a special-purpose interface designed for one specific host computer.

This chapter is divided into two basic parts: general-purpose interface and special-purpose (internal) interface.

The first part describes operation of the general-purpose interface registers and bus. The second part describes the timing and signal characteristics that are necessary for proper programming of special-purpose interfaces.

6.2 GENERAL-PURPOSE INTERFACE

All communication between the AP-120B and a host computer takes place through eight registers that make up the interface of the AP. The interface registers are organized around a 16-bit, tri-state, bi-directional bus called the host data bus (HD). The high-true host data bus is the only path over which data transfers can take place. The output of each register is driven onto the HD bus by 8095, 8096, 8097, or 8098 tri-state bus drivers. The enable inputs of the drivers common to any register are wired together to form an enable signal for that register. When all register output enable signals are false, the HD bus floats since all driver outputs are in a high-impedance state. Holding an enable signal true causes the output of the drivers to follow the contents of the register to which the driver is connected. The enable should be held true until the data has been latched. All register to HD signals should be left false when not needed to clear the bus for other transfers.

The interface registers are located on three separate cards in the AP chassis. The panel registers are on the EXPAN (214) board, the DMA control registers are on the INTERFACE (219) board, and the 32-bit format register is on the FMT (222) card. The registers within each group are loaded and read in a similar manner.

The switch (SW), function (FN), and panel display (LITES) registers form the electronic front panel. The inputs to the SR and FN are tied directly to the HD bus. Both registers are loaded with the data on the HD bus by pulsing the appropriate clock. Note the LITES register cannot be loaded by the host. The output of the panel registers are buffered onto the HD bus through tri-state bus drivers. Holding the enable of the appropriate buffer low forces the contents of that register onto the HD lines.

Four registers, the host memory address (HMA), AP memory address (APMA), control (CTL), and word count (WC) registers control direct memory access (DMA) to the AP. The inputs of the DMA registers are tied in common to the output of a 2:1 multiplexer. One input to the multiplexer is wired to the HD bus, the other to a data path internal to the AP. Data on the HD bus is enabled to the inputs of the DMA registers by holding HD2REG high. HD2REG should be left low when input to these registers is not needed by the host to enable the internal bus access to the registers. Pulsing one of the register clocks while holding HD2REG high causes data on the HD bus to be stored in the clocked register. Pulsing the clock on the HMA, APMA, or WC registers while HD2REG is low causes the clocked register to increment or decrement, according to the contents of the CTL register. (See Section 4.2 of the "Processor Handbook".) The outputs of the four registers are connected to a 4:1 multiplexer. The output of the multiplexer is buffered to the HD lines by a tri-state bus driver. Two signals, HOSTRS0 and HOSTRS1, are driven to select the output of one of the DMA registers to the bus driver. Holding REG2HD high drives the contents of the selected register onto the HD lines.

The format conversion register (FMT) is organized as two 16-bit bytes for I/O purposes. The most significant byte (FMTH) contains the sign, exponent, and high mantissa of the 32-bit word. The least significant byte (FMTL) contains the low mantissa. The HD bus is wired in parallel to the two bytes through a tri-state buffer. The enable to the buffer is controlled by bit 10 of the CTL register. CTL 10 must be cleared to allow the host to load the FMT through the HD bus. The leading (high-to-low) edge of one of the two FMT input clocks, B0CLK* or B2CLK*, stores the contents of the HD bus in FMTH or FMTL, respectively, if CTL 10 is cleared. The output of the FMT register is buffered to the HD bus through a tri-state bus driver. Holding BH2HD* or BL2HD* low drives FMTH or FMTL, respectively, onto the HD bus.

Reading or loading the interface registers should be done only while the AP is halted. The AP has access to these registers only as a default condition when an interface controller is not using the input channels. More importantly, the interface control signals should be left in the false state when the controller is idle to prevent interference with AP microcode execution.

The following sections provide a more specific description of the use of the AP interface. Signal names are always underlined. An asterisk (*) following a signal name indicates that signal is low true. Signal names with no asterisk are high true. A signal is considered to be high if it lies between +2.5 to +5.5V dc. Low signals are restricted to 0 to 0.4V dc. Delay times given consider only delay through any levels of logic between source and destination in the AP chassis. Additional time should be allowed for data to settle on the HD bus and propagation delays through the interconnecting cables. For example, a 10-foot ribbon cable requires 60ns settling time and a 30ns propagation delay.

All signals to the AP can be driven with standard TTL logic.

6.2.1 Programmed I/O

Simple I/O to the AP is accomplished by loading and reading the interface registers. The eight interface registers are directly accessible by the host through the HD bus, while the AP internal registers are available through the electronic front panel. The following discussion describes how to load and read the interface registers. This discussion also describes the system reset, run and interrupt signals.

SWITCH REGISTER (SW)

- To load: Data on the HD bus is stored in the SW on the high-to-low transition of LDSR*.
- To read: The contents of the SW are driven onto the HD bus by SR2HD. Data is true on the HD bus no later than 50ns after SR2HD goes high.

FUNCTION REGISTER (FN)

- To load: Data on the HD bus is stored in the FN on the high-to-low transition LDFN*.
- To read: The contents of the FN are driven onto the HD bus by FN2HD. Data is true on the HD lines no later than 50ns after FN2HD goes high.

LITES REGISTER (LITES)

- To load: The lights register cannot be loaded by the host.
- To read: The contents of the LITES are driven onto the HD bus by LT2HD. Data is true on the HD lines no later than 50ns after LT2HD goes high.

HOST MEMORY ADDRESS REGISTER (HMA)

- To load: Data on the HD bus is enabled to the input of the HMA by driving HD2REG high. The data is true at the input no later than 15us after HD2REG goes high. Data at the input is stored in the HMA on the high-to-low transition of HMACLK*.
- To read: The output of the HMA is selected to a common driver by driving HOSTRS0 low and HOSTRS1 high. Data is true at the input to the driver no later than 18ns after HOSTRS0 and HOSTRS1 are in the proper state. The output of the HMA is then driven onto the HD bus by driving REG2HD high. Data is true on the HD lines no later than 40ns after REG2HD goes true. REG2HD must be held true as long as the data is needed on the HD lines.

AP MEMORY ADDRESS REGISTER (APMA)

- To load: Data on the HD bus is enabled to the input of the APMA by holding HD2REG high. Data at the input is stored in the APMA on the high-to-low transition of HDMACLK*.
- To read: The output of the APMA is selected to a common driver by holding HOSTRS0 high and HOSTRS1 high. The output of the APMA is then driven onto the HD bus by driving REG2HD high. See also "HMA read" above.

WORD COUNT REGISTER (WC)

- To load: Data on the HD bus is enabled to the input of the WC by holding HD2REG high. Data at the input is stored in the WC on the high-to-low transition of HWCLK*.
- To read: The output of the WC is selected to a common driver by holding HOSTRS0 low and HOSTRS1 low. The output of the WC is then driven onto the HD bus by driving REG2HD high. See also "HMA read" above.

CONTROL REGISTER (CTL)

- To load: Data on the HD bus is enabled to the input of the CTL by holding HD2REG high. Data at the input is stored in the CTL on the high-to-low transition of HCTLCLK*.
- To read: The output of the CTL is selected to a common driver by holding HOSTRS0 high and HOSTRS1 low. The output of the CTL is then driven onto the HD bus by holding REG2HD high. See also "HMA read" above.

FORMAT CONVERSION REGISTER (FMT)

- To load: High (sign, exp, high mantissa) - Data on the HD bus is loaded into the high FMT on the high-to-low transition of BOCLK*.
- Low (low mantissa) - Data on the HD bus is loaded into the low FMT on the high-to-low transition of B2CLK*.
- To read: High - The contents of the high format buffer are driven onto the HD lines by holding BH2HD* low. Data is true on the HD lines no later than 40ns after BH2HD* goes true.

Low - The contents of the low format buffer are driven onto the HD lines by holding BL2HD* low. Data is true on the HD lines no later than 40ns after BL2HD* goes true.

In addition to the signals used for loading and reading the interface registers, two other signals may be used for programmed I/O. These signals are:

- | | |
|-------|---|
| SYRT* | System Reset - Holding SYRT* low for at least 50ns clears the AP-120B interface to a ready state. True SYRT* will unconditionally stop any DMA transfer in progress and reset all timing. SYRT* does not clear any registers. |
| RUN | AP Running Indicator - RUN is held true while the AP is executing microcode. Run is usually used to drive a status sign back to the host computer to indicate the AP is busy. |

Interrupt capability in the AP is described in the "Processor Handbook." Two signals are provided to the interface controller to enable processing of interrupts.

- | | |
|--------|--|
| APINTR | APINTR high indicates the AP has reached a state where the CTRL register directs an interrupt should be issued to the host. APINTR will remain true until cleared by the host. |
| CLINT* | Driving CLINT* low for a minimum of 50ns clears only interrupt condition enabled by CTRL bit 5. Interrupt on AP halted and interrupt on WC=0 are cleared by clearing CTRL bits 3 or 4, respectively. |

6.2.2 Direct Memory Access

The mode, length, and direction of DMA transfers to or from the AP are controlled entirely by the four DMA registers (see figure 6-1). DMA transfers are initiated by setting control register bit 15 under program control. The DMA transfer starts immediately. Two signals from the interface controller DCHIN and DCHOUT*, synchronize DMA transfers with the host. A flip-flop triggered by these signals automatically selects the high or low 16-bit byte of the 32-bit word transfer according to the state of the CTRL register. If CTRL bits 13, 14 specify the 16-bit integer mode, the low byte of the format register is always selected. When the CTRL register specifies one of the 32-bit formats the byte select flip-flop steps through the high and low byte for each word as each 16-bit byte is transferred. At the start of a DMA transfer, if CTRL bit 12 is not set (i.e., increment HMA) the select flip-flop starts off with the high byte (sign, exp, high mantissa). If bit 12 is set, the transfer will decrement through host memory. Here the flip-flop is initialized to the low byte so data will sit in host memory in the proper sequence.

DCHIN governs DMA input to the host. True DCHIN causes either BH2HD* or BL2HD* to drive the format buffer onto the HD lines according to the byte select flip-flop. Data is present on the HD lines no later than 65ns after the leading (low-to-high) edge of DCHIN is received at the 219 board. DCHIN should be held high until the data is latched by the host. The trailing (high-to-low) edge of DCHIN removes the data from the HD bus and toggles the byte select flip-flop. If the last byte of a word has been transmitted, the DMA control logic initiates another AP memory cycle and loads the next word into the format register. The next word is available a maximum of 333ns after the trailing edge of DCHIN. The minimum cycle time for DCHIN is 1.50usec.

DMA input to the AP is controlled by DCHOUT*. Data present on the HD bus is stored in FMT on the leading (high-to-low) edge of DCHOUT*. There is a maximum 40ns propagation delay from the time DCHOUT* reaches the 219 board until the data is actually latched. Since edge triggered flip-flops make up the format register, DCHOUT* reaches the 219 board until the data is actually latched. Since edge triggered flip-flops make up the format register, DCHOUT* need only be a pulse and should be returned high when not in use. Holding DCHOUT* true disables the FMT input clocks, making it impossible to load the format buffer by other means. The trailing edge (low-to-high) toggles the byte selection flip-flop. If the last byte of a data word was just loaded into the format register the DMA control logic initiates another MD cycle to store the word.

While the DMA control logic automatically increments and decrements the APMA register, the interface controller must clock the HMA and WC registers at a time appropriate to the host. The leading (high-to-low) edge of HMACLK* increments or decrements the HMA register, depending on the state of the CTRL register. The leading edge of HWCCCLK* decrements the WC register. In either register, the count settles a maximum of 15us after the clock.

A separate path to the host is provided for the contents of the HMA register to simplify driving data addresses to the host. The HMA bus always displays the current contents of the HMA register. The HMA lines are high true and are driven directly from the HMA register by 7408 TTL gates.

Four status signals from the AP to the controller provide additional information on the DMA transfer in progress.

- WC=10* Indicates the word count register has been counted to zero. Intended to be used to terminate DMA transfers.
- CTL08* CTL08* is held true (low) when CTL bit 8 is set, indicating DMA transfers should take place on consecutive host memory cycles.
- CTL10 CTL10 is held true (high) when the direction of transfer is from the AP to the host. CTL10 false (low) indicates data is transferred from the host to the AP.
- SETREQ* SETREQ* goes true as a result of setting CTL bit 15 to indicate the AP is ready to begin the transfer. SETREQ* is a pulse approximately 60ns wide. SETREQ* is intended to be used by the controller to request host memory access.

The controller should drive the REQ line high when requesting memory cycles from the host to indicate to the AP CTL register the interface is doing DMA transfer. This signal is also necessary to enable the DMA indicator light on the AP front panel.

The WC register is decremented at SETREQ* time by the AP internal DMA control. Preadvancing the word count gives the DMA controller sufficient time to terminate the transfer on WC=0 before the next cycle. At the start of each transfer the word count indicates one transfer greater than the actual number of words transferred.

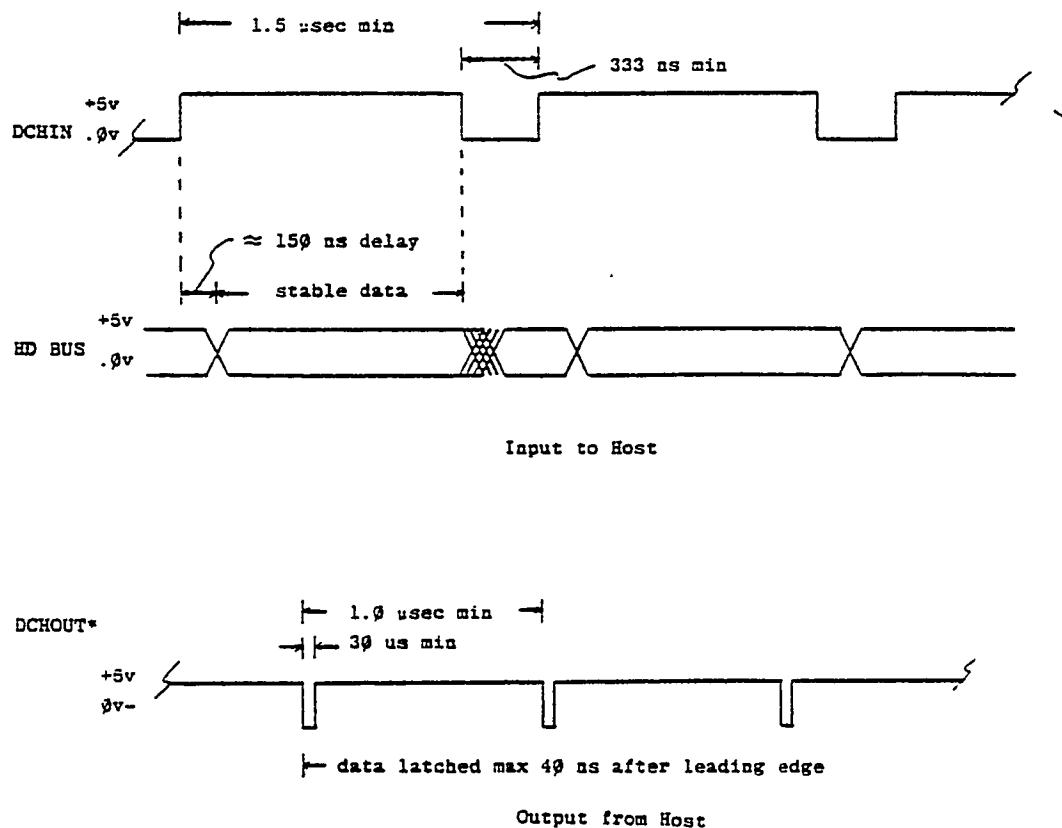


Figure 6-1 Timing Diagram

Table 6-1 Controller to AP Signals

Signal	Backplane Con. Board Pin #	Jumper Con.	Signal Type ++	Load Type *	Description (see also test)	
B0CLK*	219	A94	J1-30	Pulse	A	Loads FMTH from HD bus
B2CLK*	219	A96	J1-40	Pulse	A	Loads FMTL from HD bus
BH2HD*	219	B06	J1-36	Level	A	Drives FMTH to HD bus
BL2HD*	219	B08	J1-44	Level	A	Drives FMFL to HD bus
CLINT*	219	B23	J1-20	Pulse	D	Clears AP interrupt to host
DCHIN	219	B07	J1-06	Level	A	Drive DMA input to host
DCHOUT*	219	B35	J1-04	Pulse	C	Latch DMA output from host
FN2HD	219	B84	J1-47	Level	A	Drive FN to HD bus
HCTLCLK*	219	A73	J1-42	Pulse	B	Clock CTL reg
HD2REG	219	A95	J1-34	Level	A	Enable HD bus to inputs of HMA, WC, APMA, CTL REG
HDMACLK*	219	A75	J1-38	Pulse	B	Clock APMA reg
HMACLK*	219	A71	J1-24	Pulse	B	Clock HMA reg
HOSTRS0	219	A77	J1-49	Level	A	HMA, APMA, WC, or CTL
HOSTRS1	219	A79	J1-50	Level	A	Select to HD
HWCCLK*	219	A47	J1-26	Pulse	B	Clock WC reg
LDFN*	214	A27	J1-28	Pulse	D	Load FN from HD bus
LDSR*	214	A16	J1-32	Pulse	D	Load SR from HD bus
LT2HD	219	B66	J1-48	Level	A	Drive LITES to HD bus
REG2HD	219	B05	J1-46	Level	A	Drive HMA, WC, APMA, CTL reg to HD bus
REQ	219	B37	J1-02	SS	A	Indicates controller is requesting DMA access to host
SR2HD	219	B86	J1-45	Level	A	Drive SR to HD bus
SYRT*	219	B63	J1-22	Pulse	B	Reset AP

Table 6-2 AP to Controller Signals

Signal	Backplane Con. Board	Pin #	Jumper Con.	Signal Type ⁺⁺	Load Drive	Description
APINTR	219	B29	J1-98	SS	A	Interrupt generated by AP
CTL08*	219	B27	J1-18	SS	A	Indicates consecutive cycle DMA transfer
CTL10	219	B25	J1-16	SS	A	High = write to host; Low = host to AP
RUN	219	B31	J1-14	SS	A	AP running indicator
SETREQ*	219	B22	J1-19	Pulse	lk pull- up	Set DMA request to host
WC=0*	219	B65	J1-12	SS	A	Indicates WC reg has reached 0

⁺⁺ Signal Types:

Pulse Return false max 100 ns after going true.

Level Usually an enable that should be held until data is latched.

SS A steady state signal that will remain in one state until reset by the host.

Table 6-3 Host Data Bus

Signal	Backplane Con. Board	Pin #	Jumper Connection	
HD00	219	A74	J2-43	
HD01	219	A76	J2-45	
HD02	219	A78	J2-47	Tri-state, bi-directional 16-bit data bus
HD03	219	A80	J2-49	
HD04	219	B24	J2-50	Terminated 1k to Vcc and ground on 219 board
HD05	219	B26	J2-48	Drive with 8095 or equivalent
HD06	219	B28	J2-44	Should be terminated with 1k to Vcc at host end of cable.
HD07	219	B30	J2-46	
HD08	219	B36	J2-42	
HD09	219	B38	J2-40	
HD10	219	B40	J2-06	
HD11	219	B42	J2-04	
HD12	219	B58	J2-02	
HD13	219	B60	J2-01	
HD14	219	B62	J2-03	
HD15	219	B64	J2-05	

Table 6-4 Host Memory Address Lines

Signal	Backplane Con. Board	Pin #	Jumper Con.	
HMA00	219	A61	J2-38	
HMA01	219	A63	J2-36	
HMA02	219	A67	J2-34	The HMA lines always display the current contents of the HMA register
HMA03	219	A69	J2-32	
HMA04	219	B09	J2-30	
HMA05	219	B11	J2-28	The ground pins of the four IC's that drive the HMA lines are brought out to the backplane at board 219 pins A65
HMA06	219	B15	J2-26	B13
HMA07	219	B17	J2-24	B51
HMA08	219	B47	J2-22	B91
HMA09	219	B49	J2-20	and are tied directly to the 18 grounds of jumper B
HMA10	219	B53	J2-18	
HMA11	219	B55	J2-16	
HMA12	219	B87	J2-14	
HMA13	219	B89	J2-12	
HMA14	219	B93	J2-10	
HMA15	219	B95	J2-08	
GND	219	A65 B13 B51 B91		

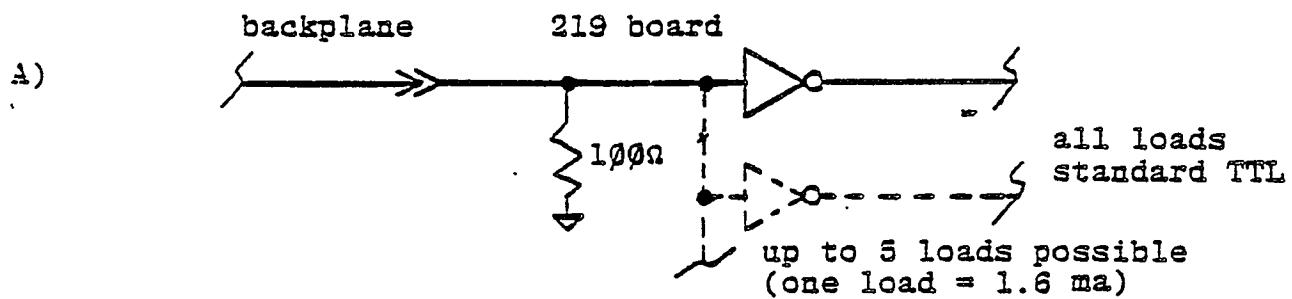


Figure 1

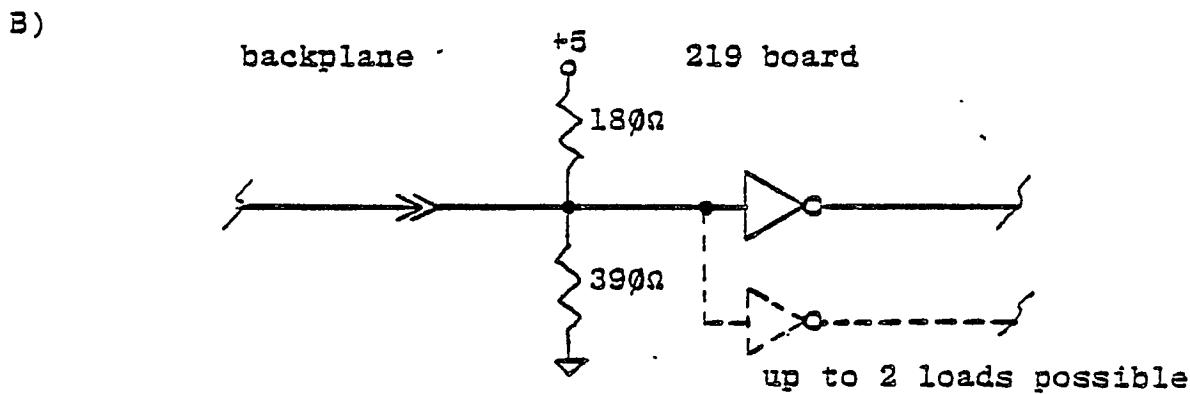


Figure 2

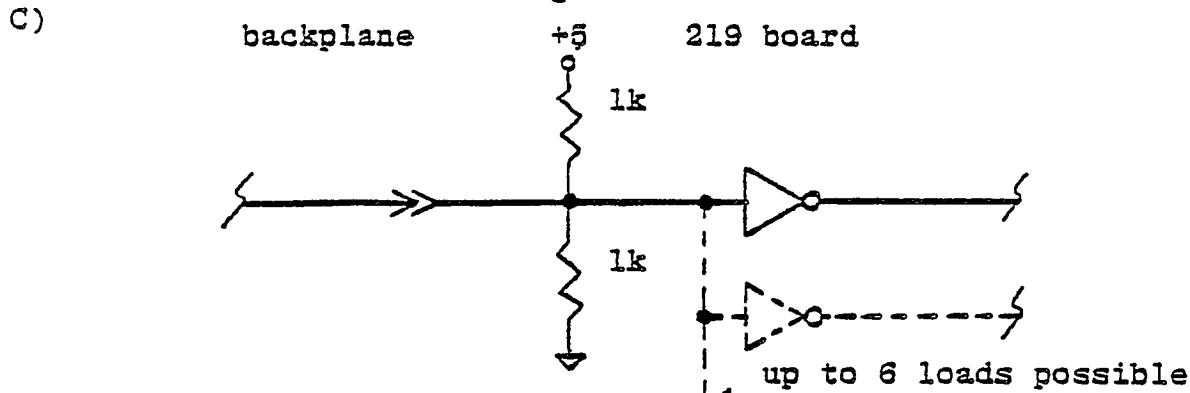


Figure 3

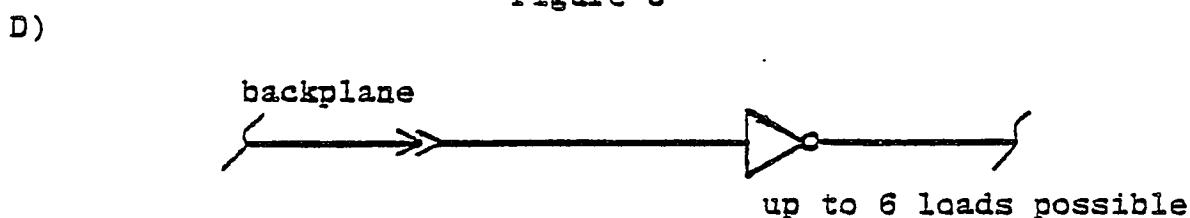


Figure 6-2 Signal Load Variations

6.3 AP-120B SPECIAL-PURPOSE (INTERNAL) INTERFACE

This section describes the timing and signal characteristics of the AP-120B internal input/output and direct memory access buses and their interfaces.

6.3.1 Programming Considerations

Due to the fast cycle time of the AP-120B processor, the programming of I/O devices on the AP-120B is somewhat different than a typical host CPU. The AP-120B has I/O input and output instructions. However, in place of the customary device code field in the I/O instruction, the AP-120B has a device address (DA) register which is typically loaded in a separate instruction from the actual I/O instruction. Provision is made in the AP-120B instruction set to perform I/O input and output, and to update the DA register in the same instruction. The programmer is advised to exercise caution, however, when using these I/O and DA load instructions. He should be sure that he is dealing with a device that is fast enough to respond at the AP-120B clock rate of 167ns. This is especially true when the I/O spins are being used (SPININ, SPINOUT, SPINA, etc.) since these instructions require that the device respond on the signal line IODRDY* within 50ns of clock in order for a correct SPIN (AP-120B processor hang) decision to be made. Note that the control registers in the host interface (HMA, WC, CTRL, APMA) are in the "fast" category in that they can be loaded in succeeding AP-120B instructions. Whereas, the FMT register is in the slow category if it is being used in the SPIN mode and there are other devices in the system that can drive IODRDY*.

The following examples illustrate the use of the AP-120B I/O instructions in the initialization and control of the host interface to the AP-120B.

LAPAL REV 2

PASS 1

PASS 2

\$TITLE APAL1
\$ENTRY APAL1
"PROGRAMMING EXAMPLE
"AFDMA TO HOST DMA
"WITH AP WC = 0 INTERRUPT ENABLED.
"APMA = 0
"HMA = 100
"WC = 20
"CTL = 20345 = IAPWC CC, APDMA WRT HOST FMT = 2 HDMA
"START
"

000000	001603	APAL1: LDDA; LDSPI 0; DB=0	"SET DA AND SP(0)
	107000		
	002000		
	000000		
000001	001103	- OUTDA; INC 0; DB=20	"SET WC = 20
	142000		
	002000		
	000020		
000002	001103	OUTDA; INC 0 DB=100	"DA < 1 "HMA = 100
	142000		
	002000		
	000100		
000003	000003	LDDA; DB=3	"DA < 3
	107000		
	002000		
	000003		
000004	000003	OUT DB=0	"APMA = 0
	140000		
	002000		
	000000		
000005	000003	LDDA; DB=2	"DA < 2
	107000		
	002000		
	000002		
000006	001103	OUTDA INC 0 DB=20345	"CTL = 20345
	142000		
	002000		
	020345		

\$END

**** 0 ERRORS ****

SYMBOL	VALUE
APAL1	000000 ENT

IAPAL REV 2

PASS 1
PASS 2

\$TITLE APAL2
\$ENTRY APAL2
"WHEN THE
"APPROPRIATE TIME ARRIVES TO CHECK FOR DMA TRANSFER
"COMPLETION THE PROGRAMMER NEEDS TO INSERT THE
"FOLLOWING CODE:

000000	000000	APAL2	BINTR Q DONE	"IF INTERRUPT GO
			000162	
			000000	
			000000	
000001	000000		BR .-1	"TO SERVICE ROUTINE "IF NOT BRANCH BACK OR
			000117	
			000000	
			000000	
000002	000000	DONE	NOP	"DO OTHER USEFUL THINGS. "INTERRUPT SERVICE
			000000	
			000000	
			000000	
				"ROUTINE GOES HERE
		\$END		

**** 0 ERRORS ****

SYMBOL	VALUE
APAL2	000000 ENT
DONE	000002

The above two examples are intended to indicate some of the different characteristics of AP-120B I/O programming. They are not indicative of efficient coding. The first example demonstrates the use of the I/O output and load DA instructions (OUTDA used to load WC or HMA from DB, and DA from SPFN) and the simple program control method of testing for completion (read CTL and test the least significant bit).

The second example illustrates the use of a SPAD register as a counter to control the transfer. This method results in a very tight loop for the output of the data to the host. Note that in both of the examples the SPIN I/O instructions (SPININ, SPINDA, SPINOUT) hold up the execution of the rest of the instruction until the addressed device responds on IODRDY*. Thus, in the examples, the INCDPA field does not execute until after DPY(0) has been written or read.

The following example illustrates the use of interrupt programming. This example assumes that the host interface is the only one with its interrupt enabled. Note that the significant advantage of this type of coding lies in the fact that it requires only the condition and branch fields in an otherwise useful instruction to test the interrupt.

Note that if more than one device is interrupting, then the AP-120B would have to execute an INTA instruction to identify the device with highest priority. That device will respond by placing its device address on the IOBS (INBS). If the host interface has highest priority, it responds with device address zero.

IAPAL REV 2

PASS 1
PASS 2

\$TITLE APAL3

\$ENTRY APAL3

"AP-120B INITIALIZATION OF HOST INTERFACE
"HOST DMA TO AP-120B PROGRAM CONTROL
"THIS PROGRAM TRANSFERS 16 32-BIT WORDS
"FROM HOST MEMORY LOCATION 100 INTO DPX AND DPY
"

000000	001677	APAL3	LDDA DB=0; 107000	LDSPI 17	"SET DA = 0 (WC) "SET SP(17) = 0
			002000		
			000000		
000001	001177		OUTDA; DB=20; 142000	INC 17	"SET WC = 20 "ADVANCE DA TO 1
			002000		
			000020		
000002	001177		OUTDA DB=100; 142000	INC 17	"SET HMA = 100 "ADVANCE DA TO 2
			002000		
			000100		
000003	000003		OUT DB=201		"SET CTL = 201
			140000		
			002000		
			000201		
000004	000003		LDDA; DB=4		"CONSECUTIVE CYCLE "START DMA "AP PROGRAM CONTROL "32-BIT INTEGER FORMAT "SET DA = 4(FMT)
			107000		
			002000		
			000004		
000005	000000		NOP		"NOP TO WAIT FOR
			000000		
			000000		
			000000		

000006	000003 145000 041004 000000	LOOP SPININ; DB=INBS DPY(0)<DB	"OTHER DEVICES "WAIT FOR DMA "TO LOAD FMT
000007	041777 147000 011000 100004	SPINDA; DB=INBS; DPY(0)<DB;INCDA; MOV 17,17	"AND THEN READ FMT "TO DPBS. "GET NEXT WORD "FROM FMT TO DPY "ADVANCE DPA
000010	001673 144000 001000 000000	IN; DB=INBS; LDSPI 16	"SET DA TO 2 "READ CTL TO SP(16) "TO TEST FOR DONE
000011	047673 107000 002000 000004	MOVR 16 16 LDDA; DB=4	"SET UP TEST OF LSB "RESET DA TO 4
000012	010010 000014 000000 000000	ENC LOOP	"DMA DONE?
000013	000003 170000 000000 000000	HALT	

SEND

**** 0 ERRORS ****

SYMBOL	VALUE
APAL3	000000 ENT
LOOP	000006

6.3.2 Hardware Interfacing Considerations

Tables 6-5 and 6-6 and Figures 6-3 and 6-5 describe the I/O and DMA interface signals and their timing relationships. Tables 6-7 and 6-8 give driver recommendations and loading rules for these signals. Table 6-9 gives a list of recommended pinouts for an interface board. Use of these pinouts will result in a board that can be installed in a slot that is wired for board 215 (main data) with minimum length connections to board 219 (general purpose host interface) and board 226 (format mantissa: has all 38 IOBS lines). Tables 6-10 and 6-11 list the pinouts for boards 219 and 226 so that the necessary backplane wire list can be generated. Figure 6-6 shows the physical dimensions of an AP-120B printed circuit board and Figure 6-8 shows the layout of the AP-120B breadboard board (board 221). Finally, Figures 6-7 and 6-9 show examples of the control logic for I/O and DMA interfaces.

6.3.3 I/O Interface

Refer to Figures 6-3, 6-4 and 6-9. The signal !IOCLK is provided to all the AP-120B internal I/O and DMA interfaces in order to allow them to synchronize their activities with those of the AP-120B processor. !IOCLK is a nearly square, 6MHz clock that is in phase with the clocks that drive the AP-120B processor (to within line length delays). Figure 6-3 illustrates the various propagation delays and setup time requirements for the I/O bus signals with respect to !IOCLK.

Figure 6-3 illustrates the typical timing to be expected for a fast device; i.e., one that either does not use IODRDY* or is able to decode DAnn* in time to meet the IODRDY* set-up-time requirements. Typical devices that use IODRDY* will drive it when their device-dependent condition and their device address is true. They may include IN, SNSA, or SNSB in the logic term that generates IODRDY* but should not include OUT* since OUT* may be conditional on the receipt of IODRDY*.

For I/O input, the IOnn* lines should be driven by the addressed device when IN is true and be stable 100ns prior to the next low-to-high transition of !IOCLK. This allows for approximately 45ns delay from the receipt of IN to the stabilization of data on IOnn*. Note here that if the designer wishes to restrict the programmer to a subset of the possible DPBS destinations, (no input to S-PAD or program source memory) he may relax this 100ns time constraint to approximately 60ns. Another alternative is to require the program to execute two consecutive IN instructions in a row using the data on the second IN.

For I/O output, the IOnn* are guaranteed to be valid 30ns prior to !IOCLK. If the interface uses IODRDY, then the OUT* and IOACK* low-to-high transition can be used to clock the data into the desired device register. Note that this register must have a fast input set-up time in order to accept data. FPS-designed interfaces typically use Schottky registers with input setup times of 5 to 7ns.

Interfaces that do not use IODRDY* during I/O output are slightly more complicated, since IOACK* will not be generated if IODRDY* is not driven. For these interfaces a clock switching technique similar to that shown in Figure 6-9 is recommended. In this technique a self-clearing 74S74 is used as a fast, edge-triggered one-shot to produce the desired clock.

Figure 6-4 illustrates a typical I/O sequence for a slow device that uses IODRDY*. In this sequence, the AP-120B is assumed to be executing either a SPININ or a SPINOUT instruction. During this type of instruction, the AP-120B processor hangs without completing execution of the instruction until the device responds on IODRDY*. Note that if for some reason an addressed device does not respond on IODRDY* during an I/O SPIN type instruction, the AP-120B will hang indefinitely and a panel RESET operation will be required to stop the AP-120B.

Referring back to Figure 6-4, note in particular that OUT* may come true and then go false before coming true again to indicate that the instruction has executed. The concurrence of OUT* and the low-to-high transition of either !IOCLK or IOACK* should be used as the appropriate point to sample the data on the IOm* lines. Gating of !IOCLK with OUT* is not recommended.

When using the basic control parts of a single device address interface, the user should try to minimize the device address decode propagation delays. Thus a two-level decode of DA is represented by the 74S04 and 74S30 in the upper left hand corner of Figure 6-9. An !IOCLK can be used to synchronize the switched clock to the device register.

The example in Figure 6-9 also illustrates the driving of IODRDY* and the use of SNSA to test for some device condition (device condition 'A'). If the propagation delay from DAn* to IODRDY can be kept to four or fewer Schottky delays, then it will probably be possible to run the device as a "fast" device (i.e., one for which the LDDA instruction can immediately precede an I/O instruction that SPINS on IODRDY). However, to be safe, it is good practice to insert an instruction between the LDDA and the SPIN and to insert two instructions between the LDDA and a branch on IODRDY.

The lower half of Figure 6-9 illustrates a typical interrupt interface. !IOCLK is used to synchronize interrupt requests and the gating of the serial priority line INTPOUT. If this device is causing INTR* it inhibits INTPOUT going to the next lower priority device. When the AP-120B executes an INTA instruction, the device that is generating INTR* and also has INTPIN true will place its device address on the IOm* lines (right justified) and synchronously clear its request at the end of the INTA instruction.

6.3.4 DMA Interface

Figure 6-5 illustrates the DMA timing while Figure 6-7 contains a schematic of the control section for a typical DMA interface. The DMA logic in the AP-120B will support an asynchronous request on MDCR2*. However, if more than one device is to use DMA level 2 (refresh is at priority level 0, the host interface is at priority level 1 and the AP-120B processor is at priority level 3, the lowest priority), then the request should be synchronized with IOCLK so that the data channel priority (DCHPOUT) logic will have time to settle properly to avoid conflicts.

Following the request on MDCR2*, there will be a one clock minimum delay (longer if there are higher priority requests queued up) before the memory timing logic on board 210 issues the acknowledge signal MDCA2, the DMA interface must drive the desired memory address into MA00* (MSB) to MA15* (LSB) and drive MDInn and MDWRT* if it is a memory write cycle. If it is a read cycle, the device should wait until the second clock period following MDCA2 and then use DCHO2* (low-to-high transition) to clock the data on MDnn* into the device register. Note that DCHO2* is not conditioned by MDWRT*. Thus, it is necessary to condition it in the DMA interface.

6.3.5 Signal Nomenclature

- 1) * An asterisk following a signal name indicates that it is low true.
- 2) ! An exclamation mark preceding a signal name indicates that it is to be wired as a twisted pair on the backplane.

Table 6-5 I/O Signal Names and Functions

<u>NAME</u>	<u>SOURCE</u>	<u>FUNCTION</u>
DA08* to DA15* (msb to lsb)	Bd 219 or Host Interface	Device Address lines.
IN	Bd 214 (B58)	I/O INput strobe. Device gates data to IOBS. True during IN instruction.
INTA	Bd 214 (B60)	INTerrupt Acknowledge. Interrupting device gates its DA to IOBS.
INTR*	Device (dest Bd 214 B67)	INTerrupt Request.
IOACK*	Bd 214 (B63)	I/O ACKnowledge, indicates receipt of IODRDY*.
IO92* to IO39* (msb to lsb)	Bd 226, 227 or Device	I/O Bus. DPBS gated to IOBS by OUT*. IOBS gated to DPBS by DB=INBS (signal HD2DP).
!IOCLK	Bd 212 (A24)	I/O ClocK. Used to synchronize our watches. (6MHz) clock in sync with AP-120B system clocks. (Exclamation mark indicates a twisted pair.)
IODRDY*	Device (dest B214 B66)	I/O Data ReaDY. Used by device to indicate readiness to receive or transmit data on IOnn*.
IORST*	Bd 214 (B62)	I/O ReSeT.
OUT*	Bd 214 (B59)	I/O OUTput strobe. Device accepts data from IOnn* at OUT* AND !IOCLK low-to-high transition. OUT* true during execution of OUT instruction.
SNSA	Bd 214 (B57)	Sense 'A'. True during SNSA instruction. Function is device dependent.
SNSB	Bd 214 (B56)	Sense 'B' true during SNSB instruction. Function is device dependent.
INTPIN	Next interface	INTerrupt Priority INput from next higher priority device.
INTPOUT	This interface	INTerrupt Priority OUTput to next lower priority device.
ABORT*	Bd 214 (A39)	System Reset: power-on or Host Reset.

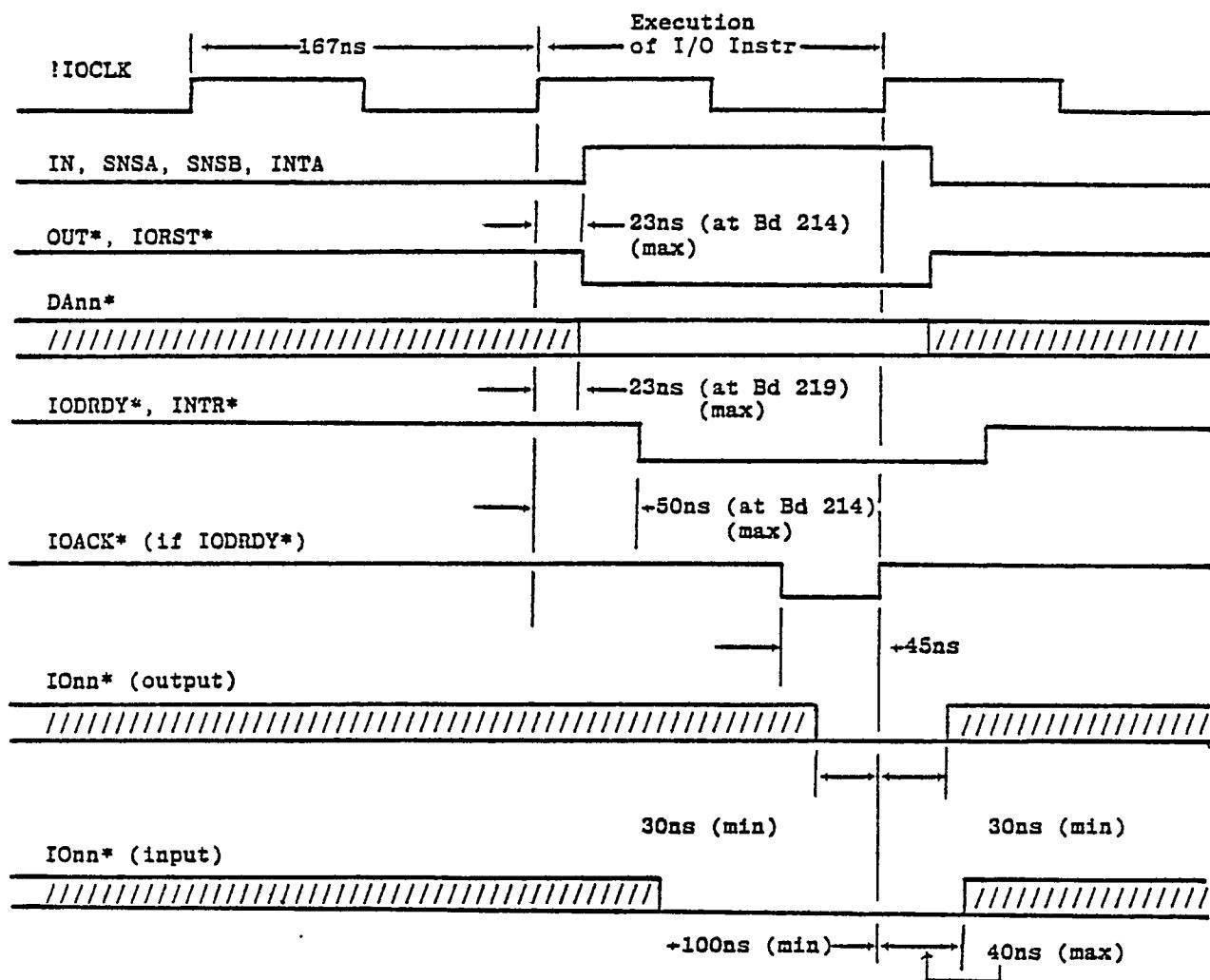
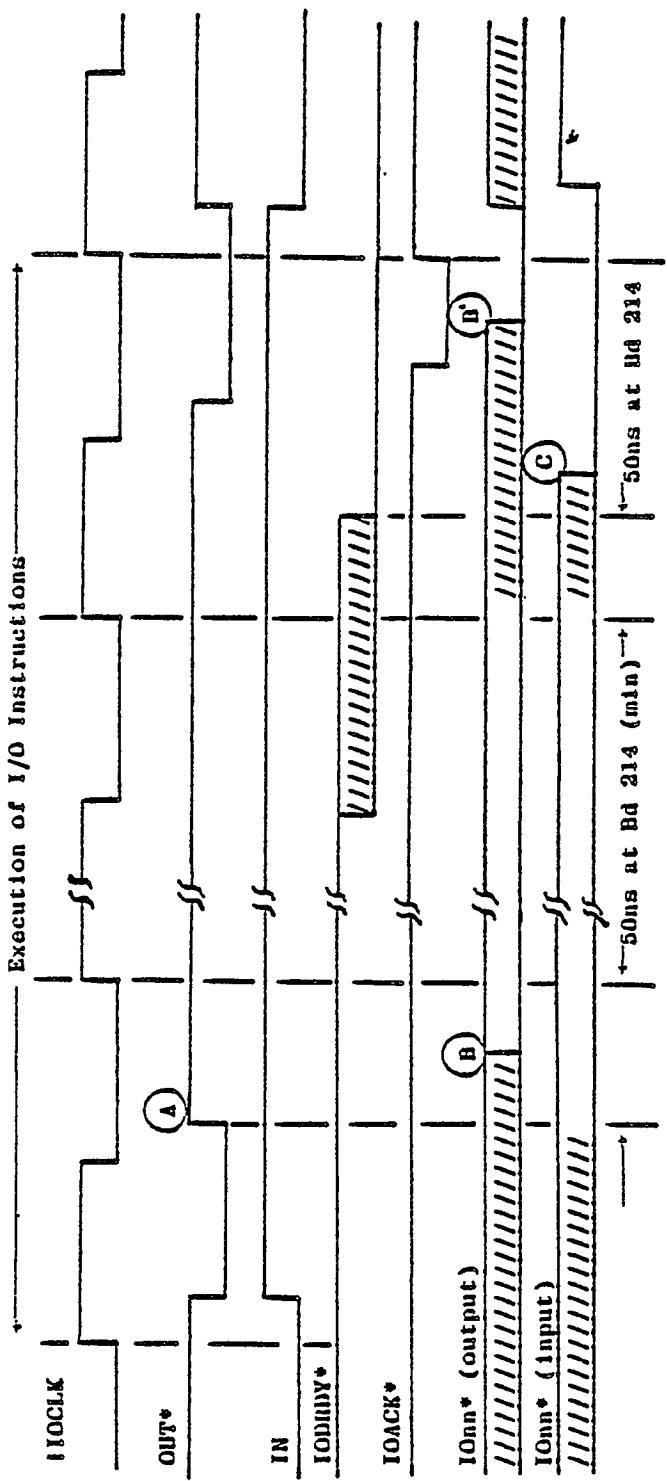


Figure 6-3 AP-120B I/O Bus Timing (Fast Device)



NOTES: (A) If IODRDY* not received, OUT* will be turned off until after IODRDY* is received and synchronized at Bd 214.

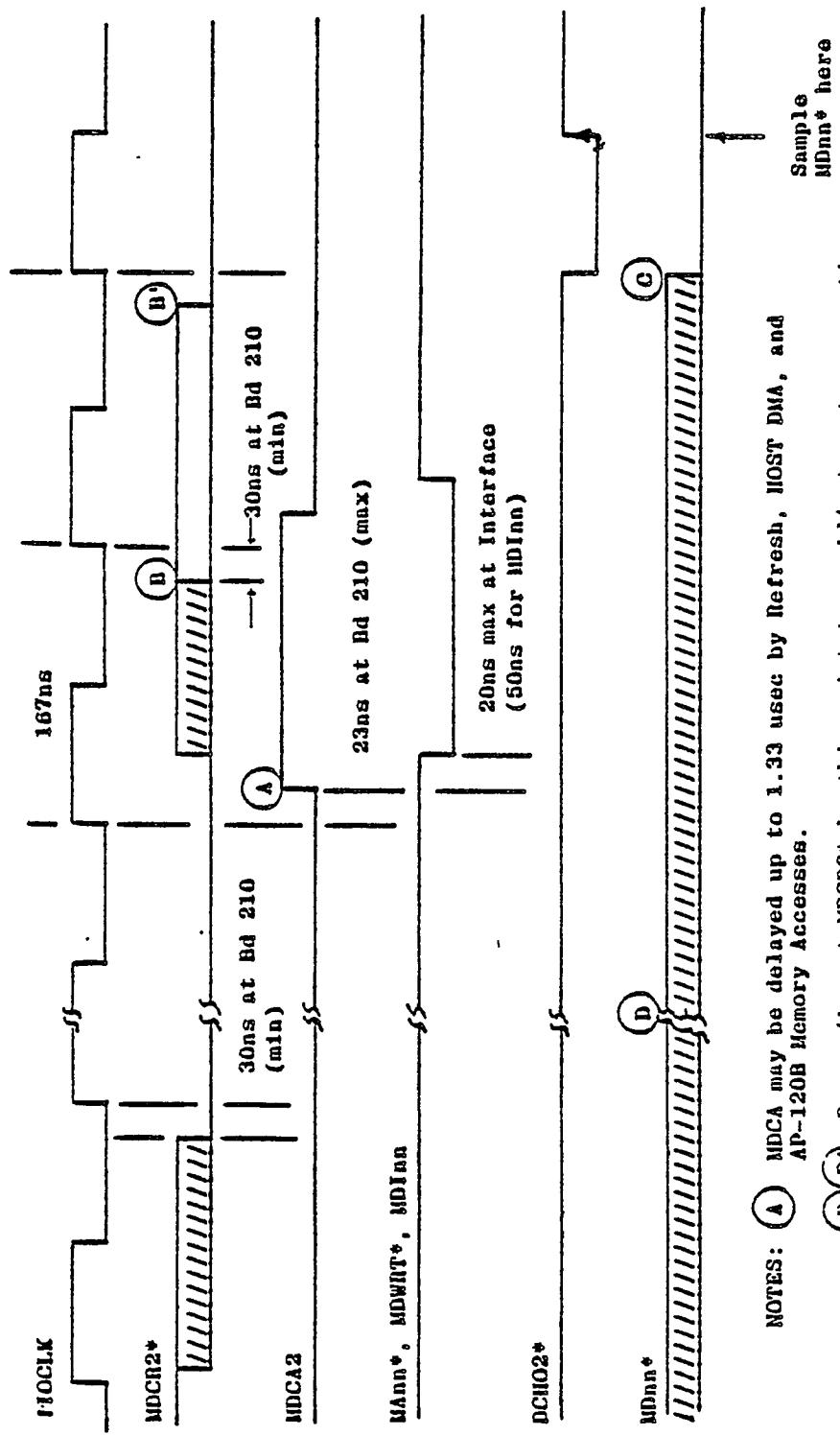
(B) (B) Depending on the source, IOnn* will be valid during the entire period or may be held off until the specified 30ns prior to the low-to-high transition of IOACK*.

(C) IOnn* setup time of 100ns required.

Figure 6-4 I/O Bus Timing with Spin

Table 6-6 DMA Signal Names and Functions

<u>NAME</u>	<u>SOURCE</u>	<u>FUNCTION</u>
DCH02*	Bd 210 (B93)	Data CHannel Out level 2. Indicates presence of valid output data from Main Data memory during a read cycle. (Also present during a write cycle.)
MA00* to MA15* (msb to lsb)	Device or Bd 201	Memory Address lines to be driven by the device during cycle acknowledge (MDCA2).
MD02* to MD39* (msb to lsb)	Bd 215	Main Data output bus.
MDCA2	Bd 210 (B83)	Main Data Cycle Acknowledge level 2. Device places address on MAnn* and data on MDInn (write only) during MDCA2.
MDCR2*	Device (dest Bd 210 B79)	Main Data Cycle Request level 2.
MDI02 to MDI39	Device or Bd 213 or Bds 226, 227	Main Data Input bus.
MDWRT*	Device or Bd 210 or Host Interface	Main Data WRiTe enable. Causes memory write cycle when driven during MDCA2.
DCHPIN	Next interface	Data CHannel Priority INput from next higher priority device on DMA level 2.
DCHPOUT	This interface	Data CHannel Priority OUTput to next lower priority device on DMA level 2.



NOTES: (A) MDCA may be delayed up to 1.33 usec by Refresh, HOST DMA, and AP-120B Memory Accesses.

(B) Remove/Assert MDCLR2* by this point to avoid/get next consecutive Memory Cycle. **B** = 333ns, **B** = 500ns Cycle Memory. Consecutive cycles will occur at memory cycle time.

(C) MDnn* guaranteed for only one clock period (167ns).

(D) Multiple 167ns delays depending on Memory Activity. 167ns minimum delay.

Figure 6-5 AP-120B DMA Bus Timing

Table 6-7 Recommended Drivers

<u>I/O SIGNALS</u>	<u>DRIVER</u>
INTR*	Open collector ¹ 40 ma drive
IO02* to IO39*	Tri-state ² 16 ma drive
IODRDY*	Open collector ¹ 40 ma drive
INTPOUT	Schottky ³ totem pole 20 ma drive

<u>DMA SIGNALS</u>	<u>DRIVER</u>
MDCR2*	Open collector ¹ 40 ma drive
MDI02 to MDI39	Tri-state ² 16 ma drive
MDWRT*	Open collector ¹ 40 ma drive
DCHPOUT	Schottky ³ totem pole 20 ma drive
MA00* to MA15*	Schottky tri-state ⁴ 20 ma drive

¹ = 7438 or equivalent

² = 8095/8096/8097/8098 or equivalent (National device numbers)

³ = 74S08 or equivalent

⁴ = 74S257 or equivalent

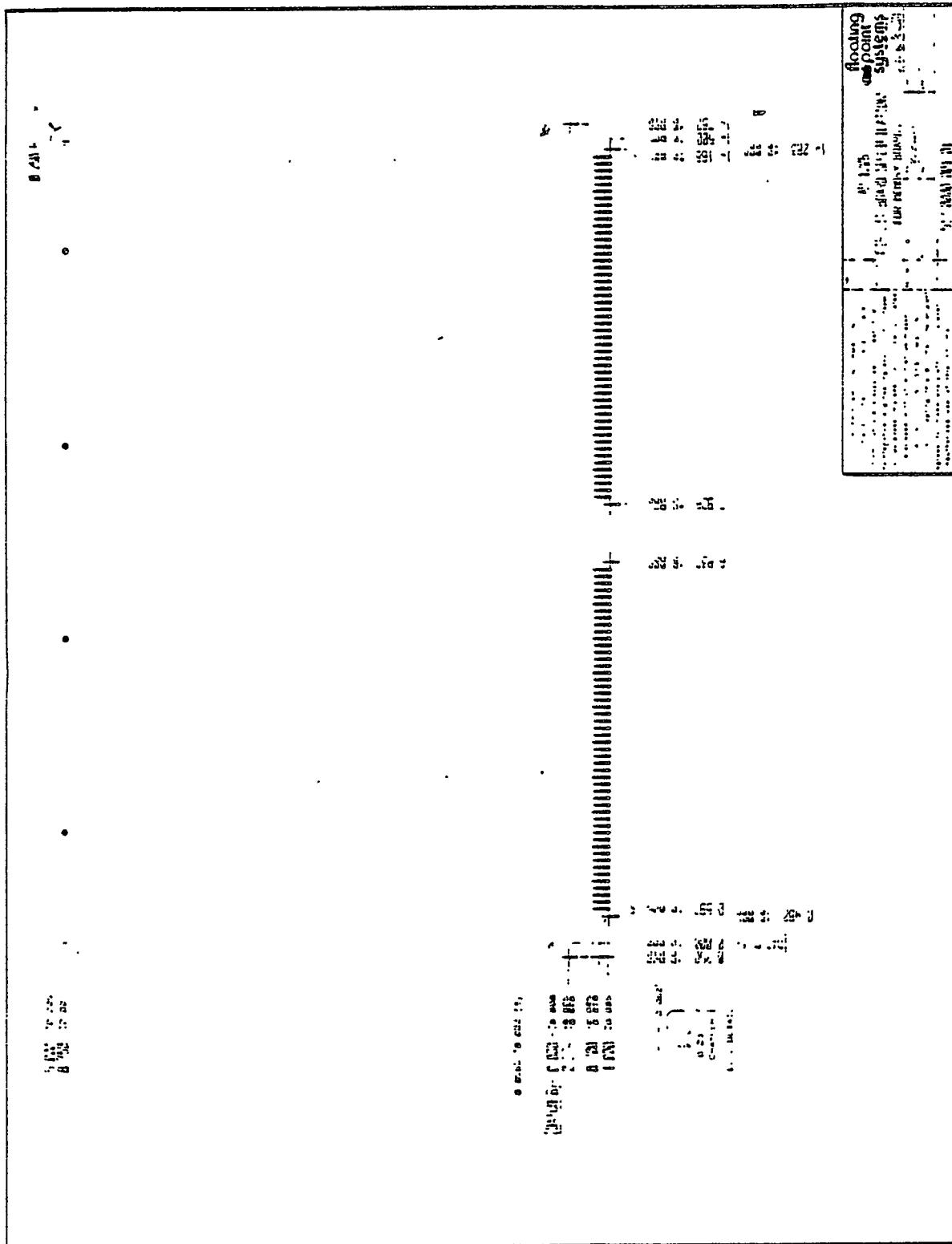


Figure 6-6 Circuit Board Specifications

Figure 3 DNA INTERCAGE

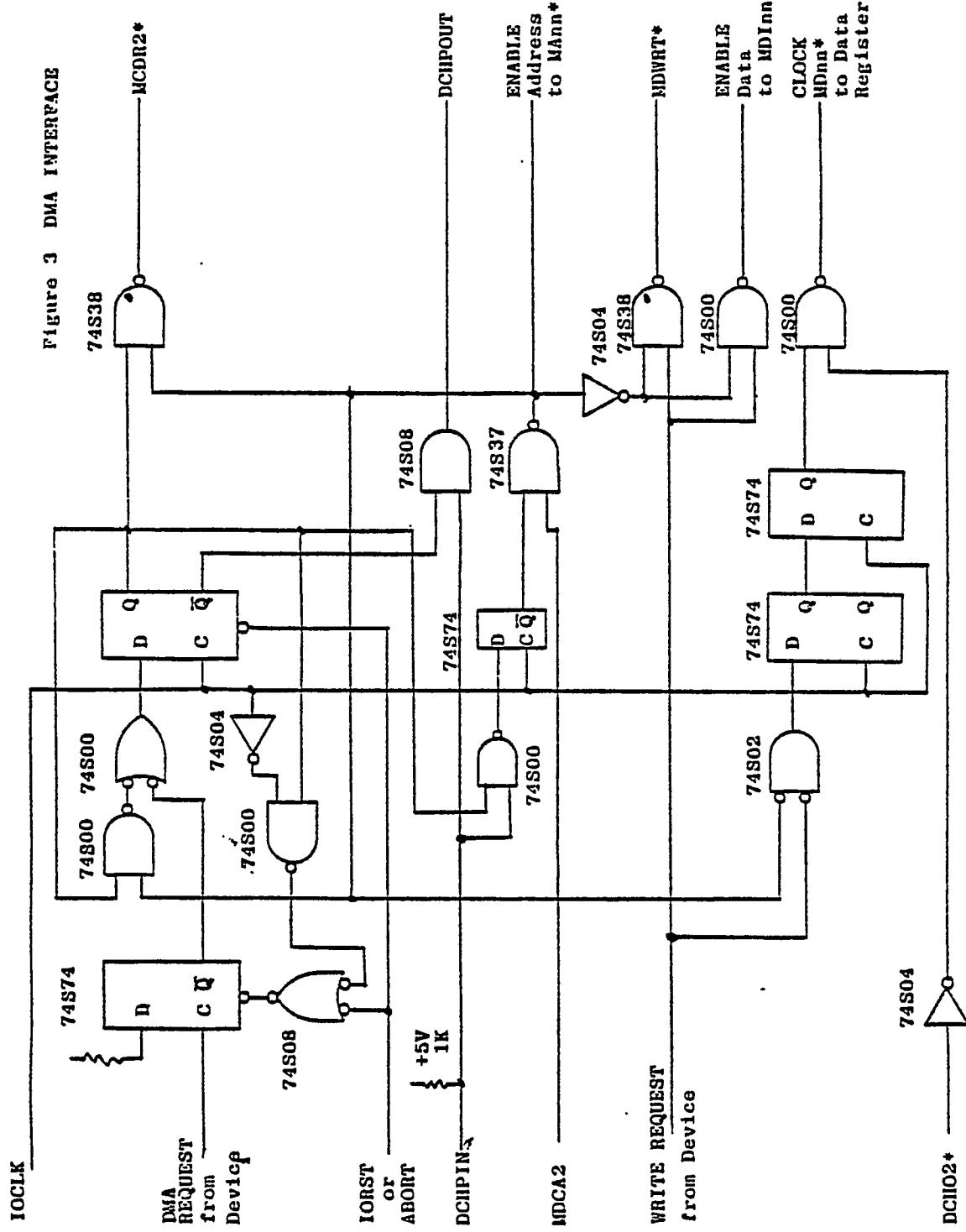


Figure 6-7 DMA Interface

Table 6-8 Loading Rules

<u>SIGNAL</u>	<u>LOAD</u>	<u>STUB LENGTH</u>
!IOCLK	2 ma	2" (max)
INTPIN	20 ma	4" to 8" (max)
All other I/O Signals	2 ma	4" to 8" (max)
MDCA2	4 ma	4" to 8" (max)
DCHPIN	20 ma	4" to 8" (max)
All other DMA Signals	2 ma	4" to 8" (max)

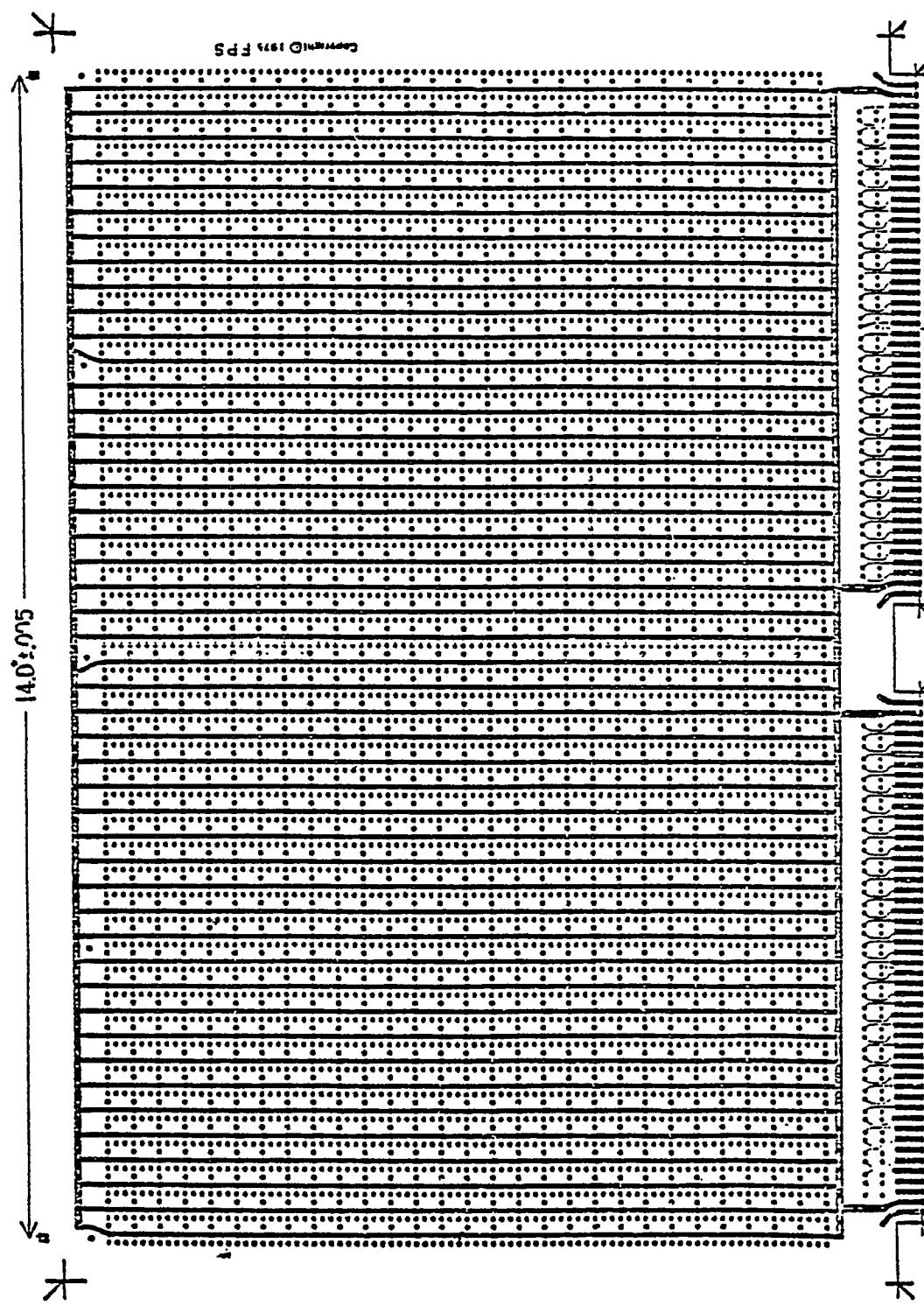


Figure 6-8 Bread Board 221

Table 6-9 Recommended Interface Pinouts

A01	VCC	A50	MD24*
2	VCC	1	GND
3	GND	2	IOACK*
4	GND	3	MDI11
5	N/A	4	INTA
6	N/A	5	MDI10
7	+12V	6	INTPIN
8	+12V	7	MDI09
9	MD03*	8	INTPOUT
		9	MDI06
A10	MD07*	A60	IORT*
1	MD02*	1	MDI07
2	MD06*	2	IO02*
3	MD01*	3	MDI08
4	MD05*	4	IO03*
5	MD00*	5	IO04*
6	MD04*	6	IO05*
7	GND	7	MDI29
8	! IOCLK	8	MA15*
9	MA00*	9	MDI24
A20	DA08*	A70	N/A
1	MA01*	1	MDI25
2	DA09*	2	N/A
3	MA02*	3	MDI28
4	DA10*	4	IO06
5	MDI00	5	MDI26
6	DA11	6	IO07*
7	MDI01	7	MDI27
8	DA12*	8	IO08*
9	MDI02	9	IO09*
A30	DA13*	A80	IO10*
1	MDI03	1	GND
2	DA14*	2	IO11*
3	MDI04	3	MD31*
4	DA15*	4	MD35*
5	MDI05	5	MD30*
6	IODRDY*	6	MD34*
7	GND	7	MD29*
8	IN	8	MD33*
9	INTR*	9	MD28*
A40	SNSA	A90	MD32*
1	SNSB	1	IO12*
2	OUT*	2	IO13*
3	MD11*	3	GND
4	MD27*	4	IO14*
5	MD10*	5	N/A
6	MD26*	6	IO15*
7	MD09*	7	GND
8	MD25*	8	GND
9	MD08*	9	VCC
		A100	VCC

N/A = Not Available

Table 6-9 Recommended Interface Pinouts (cont.)

B01	VCC	B50	IO31*
2	VCC	1	MDI38
3	GND	2	IO32*
4	GND	3	MDI39
5	ABORT*	4	IO33*
6	IO16*	5	MDI12
7	MDI30	6	IO34*
8	IO17*	7	MDI13
9	MDI31	8	IO35*
		9	GND
B10	IO18*	B60	IO36*
1	MDI32	1	MD19*
2	IO19*	2	MD23*
3	MDI35	3	MD18*
4		4	MD22*
5	MDI34	5	MD17*
6	MDI33	6	MD21*
7	GND	7	MD16*
8	IO20*	8	MD20*
9	MD39*	9	IO37*
B20	MD15*	B70	IO38*
1	MD38*	1	IO39*
2	MD14*	2	
3	MD37*	3	GND
4	MD13*	4	MDCR2*
5	IO21*	5	MDI17
6	MD12*	6	MDCA2
7	GND	7	MDI18
8	MD36*	8	MDI19
9	MA13*	9	MDI14
B30	IO22*	B80	DCHO2*
1	MA09*	1	MDI15
2	IO23*	2	DCHPIN
3	MA11*	3	MDI16
4	IO24*	4	DCHPOUT
5	MA12*	5	MDI23
6	IO25*	6	MDI21
7	MA10*	7	MDI20
8	MA14*	8	MDI22
9	MDWRT*	9	MA03*
B40	IO26*	B90	MA08*
1	N/A	1	MA04*
2	IO27*	2	MA07*
3	N/A	3	MA06*
4	IO28*	4	MA05*
5	N/A	5	-5V
6	IO29*	6	-5V
7	MDI36	7	GND
8	IO30*	8	GND
9	MDI37	9	VCC
		B100	VCC

N/A = Not Available

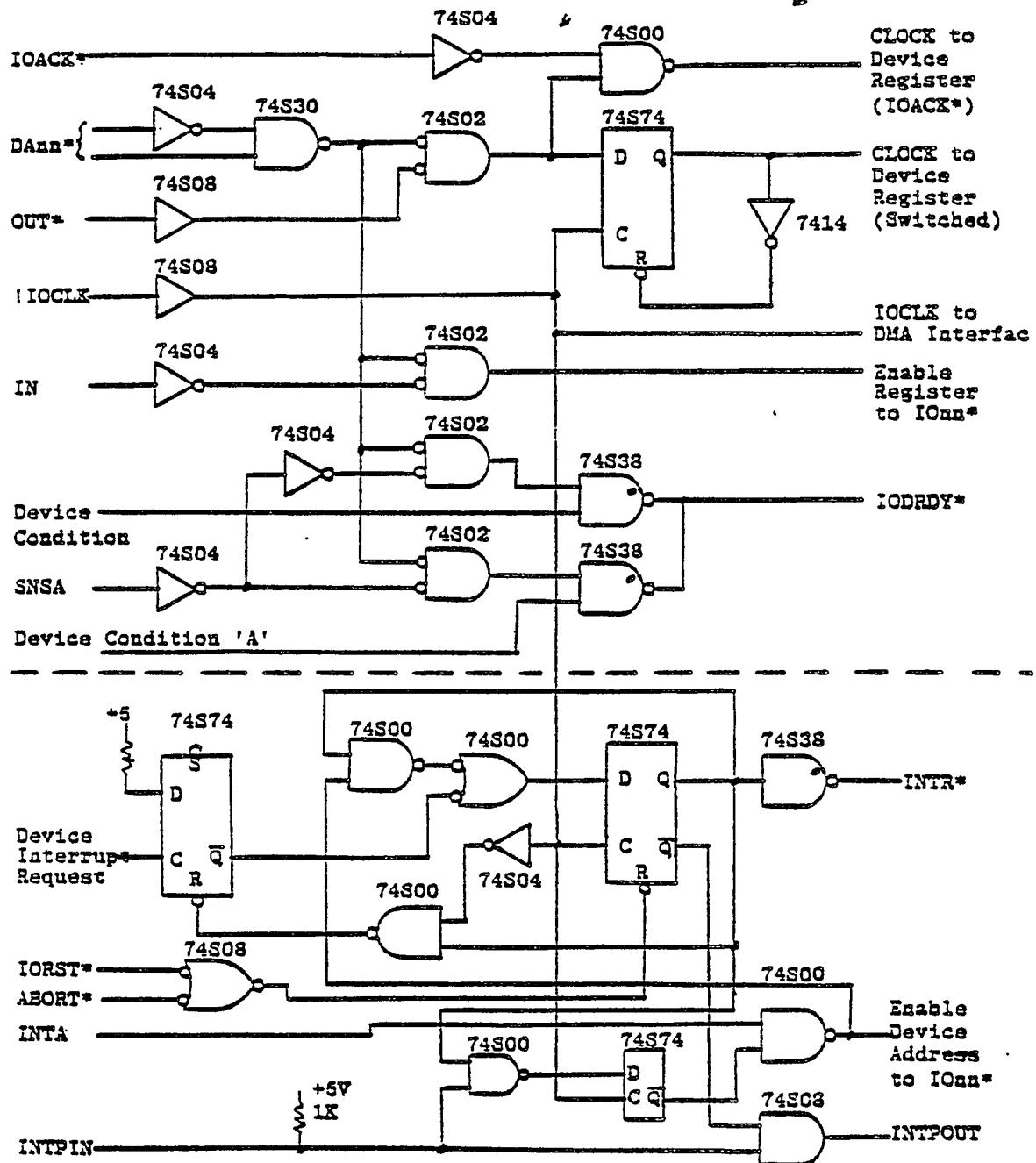


Figure 6-9 Programmed I/O Interface

Table 6-10 Board 219 (UNIV) Pinouts

A03 "GND	A64 DMA01*	B32 B1CLK	B92 SR2HD*
A05 HHD2DIN	A65 "3ND	B33 B3CLK	B93 HMA14
A06 !I OCLK	A66 DMA02*	B34 HDSELA	B94 BL2HD
A07 DALD*	A67 HMA02	B35 DCHOUT*	B95 HMA15
A08 "GND	A68 DMA03*	B36 HD08	B96 BH2HD
A09 DA2PNL	A69 HMA03	B37 REQ	*
A10 I ODRDY*	A70 OVFL*	B38 HD09	
A11 PNL08*	A71 HMACLK*	B39 DMA08*	
A12 DA08*	A72 UNFL*	B40 HD10	
A13 PNL09*	A73 HC1LCLK*	B41 DMA09*	
A14 DA09*	A74 HD00	B42 HD11	
A15 PNL10*	A75 HDMACLK*	B43 DMA10*	
A16 DA10*	A76 HD01	B44 I+H09	
A17 PNL11*	A77 HOSTRS0	B45 DMA11*	
A18 DA11*	A78 HD02	B46 I+H10	
A19 SP+DP08*	A79 HOSTRS1	B47 HMA08	
A20 SP+DP09*	A80 HD03	B48 HDSEL*	
A21 SP+DP10*	A81 I 028*	B49 HMA09	
A22 SP+DP11*	A82 DPMBS16*	B50 H2HD	
A23 PNL12*	A83 I 029*	B51 "GND	
A24 DA12*	A84 DPYRS17*	B52 HRSET*	
A25 PNL13*	A85 I 030*	B53 HMA10	
A26 DA13*	A86 DPMBS18*	B54 HD2DIN	
A27 PNL14*	A87 I 031*	B55 HMA11	
A28 DA14*	A88 DPYBS19*	B56 BDCHIN	
A29 PNL15*	A89 I 032*	B57 H2HD*	
A30 DA15*	A90 DP.BS20*	B58 HD12	
A31 SP+DP12*	A91 I 033*	B59 MMD2HD	
A32 SP+DP13*	A92 DPMBS21*	B60 HD13	
A33 SP+DP14*	A93 RUN*	B61 RAY2HD	
A34 SP+DP15*	A94 B0CLK*	B62 HD14	
A35 MDCA1	A95 HD2REG	B63 SYRT*	
A36 B0CLK	A96 B2CLK*	B64 HD15	
A37 MDVRT*	B05 REG2HD	B65 WC=0*	
A38 IN	B06 BH2HD*	B66 LT2HD	
A39 INTR*	B07 DCHIN	B67 I 034*	
A40 SNSA	B08 BL2HD*	B68 DPMBS22*	
A41 B2MDI	B09 HMA04	B69 I 035*	
A42 B210	B10 DMA04*	B70 DPMBS23*	
A43 OUT*	B11 HMA05	B71 I 036*	
A44 INTEN	B12 DMA05*	B72 DPMBS24*	
A45 B2CLK	B13 "GND	B73 I 037*	
A46 IOACK*	B14 DMA06*	B74 DPMBS25*	
A47 HWCCCLK*	B15 HMA06	B75 I 038*	
A48 BUF2CLK	B16 DMA07*	B76 DPMBS26*	
A49 HD2DP	B17 HMA07	B77 I 039*	
A50 CTLCLK	B18 DMAIND	B78 DPMBS27*	
A51 I 024*	B19 H2FMT	B79 DMA12*	
A52 DPMBS12*	B20 RUNIND	B80 I+H13	
A53 I 025*	B21 E3CLK*	B81 DMA13*	
A54 DPMBS13*	B22 SETREQ*	B82 I+H14	
A55 I 026*	B23 CLINT*	B83 DMA14*	
A56 DPMBS14*	B24 HD04	B84 FN2HD	
A57 I 027*	B25 CTL10	B85 DMA15*	
A58 DPMBS15*	B26 HD05	B86 SR2HD	
A59 DCH01*	B27 CTL08*	B87 HMA12	
A60 MDCR19*	B28 HD06	B88 LT2HD*	
A61 HMA00	B29 APINTR	B89 HMA13	
A62 DMA00*	B30 HD07	B90 FN2HD*	
A63 HMA01	B31 RUN	B91 "GND	

Table 6-11 Board 226 (FMTM) Pinouts

A5 DPEBS02*	A64 HD05	B32 HD00	B92 MD39*
A6 I002*	A65 MD13*	B33 HD01	B93 "3 ND
A7 DPEBS03*	A66 HD06	B34 HD02	B94 SGNQ*
A8 I003*	A67 I014*	B35 HD03	B95 B3CLK
A9 DPEBS04*	A68 HD07	B36 MD24*	B96 BUF2CLK
A10 I004*	A69 MD14*	B37 I024*	*
A11 DPEBS05*	A70 MDI12	B38 MD25*	
A12 I005*	A71 I015*	B39 I025*	
A13 DPEBS06*	A72 MDI13	B40 MD26*	
A14 OUT*	A73 MD15*	B41 I026*	
A15 DPEBS07*	A74 MDI14	B42 MD27*	
A16 I006*	A75 I016*	B43 I027*	
A17 DPEBS08*	A76 MDI15	B44 MD28*	
A18 I007*	A77 MD16*	B45 I028*	
A19 DPEBS09*	A78 MDI16	B46 MD29*	
A20 I008*	A79 I017*	B47 I029*	
A21 DPEBS10*	A80 MDI17	B48 MD30*	
A22 I009*	A81 MD17*	B49 I030*	
A23 DPEBS11*	A82 FMTB	B50 MD31*	
A24 I010*	A83 I018*	B51 I031*	
A25 DPMBS00*	A84 FMTA4	B52 HD04	
A26 I011*	A85 MD18*	B53 HD05	
A27 DPMBS01*	A86 B210	B54 HD06	
A28 FPDP	A87 I019*	B55 HD07	
A29 DPMBS02*	A88 B2MDI	B56 HD08	
A30 SHIE*	A89 MD19*	B57 HD09	
A31 DPMBS03*	A90 B2ZR0*	B58 MD128	
A32 SD05*	A91 BUF16C*	B59 MD129	
A33 DPMBS04*	A92 BUF17Q*	B60 MD130	
A34 SD06*	A93 BUF18Q*	B61 MD131	
A35 DPMBS05*	A94 BUF19Q*	B62 MD132	
A36 SD07*	A95 HD2BUF*	B63 MD133	
A37 DPMBS06*	A96 B1ZR0*	B64 B3ZR0*	
A38 SD08*	B5 HD09	B65 I032*	
A39 DPMBS07*	B6 MDI18	B66 MD32*	
A40 SD09*	B7 HD10	B67 I033*	
A41 DPMBS08*	B8 MDI19	B68 MD33*	
A42 SD10*	B9 HD11	B69 I034*	
A43 DPMBS09*	B10 MDI20	B70 MD34*	
A44 B(-1)ZR0*	B11 HD12	B71 I035*	
A45 DPMBS10*	B12 MDI21	B72 MD35*	
A46 MIN*	B13 HD13	B73 HD10	
A47 DPMBS11*	B14 I020*	B74 HD11	
A48 OVFL*	B15 MD20*	B75 HD12	
A49 SCN1	B16 I021*	B76 HD13	
A50 MCV*	B17 MD21*	B77 HD14	
A51 SCN2	B18 I022*	B78 HD15	
A52 BH2HD	B19 MD22*	B79 MDI34	
A52 BH2HD	B20 I023*	B80 MDI35	
A53 SLOE*	B21 MD23*	B81 MDI36	
A54 HD2DP	B22 MDI22	B82 MDI37	
A55 BL2HD	B23 MDI23	B83 MDI38	
A56 B1CLK	B24 MDI24	B84 MDI39	
A57 IHD2BUF*	B25 MDI25	B85 I036*	
A58 I02BUF*	B26 MDI26	B86 MD36*	
A59 I012*	B27 MDI27	B87 I037*	
A60 AP2BUF*	B28 HD08	B88 MD37*	
A61 MD12*	B29 HD14	B89 I038*	
A62 HD04	B30 B2CLK	B90 MD38*	
A63 I013*	B31 HD15	B91 I039*	

Notice to the Reader

- Help us improve the quality and usefulness of this manual.

Your comments and answers to the following READERS COMMENT form would be appreciated.

- To mail: fold the form in three parts so that Floating-Point Systems' mailing address is visible for the post office carrier; seal.

Thank You

READERS COMMENT FORM

Document Title _____

Your comments and answers will help us improve the quality and usefulness of our publications. If your answers require qualification or additional explanation, please comment in the space provided below.

How did you use this manual?

- () AS AN INTRODUCTION TO THE SUBJECT
- () AS AN AID FOR ADVANCED TRAINING
- () TO LEARN OF OPERATING PROCEDURES
- () TO INSTRUCT A CLASS
- () AS A STUDENT IN A CLASS
- () AS A REFERENCE MANUAL
- () OTHER _____

Did you find this material . . .

- | | YES | NO |
|-----------------------|-----|-----|
| • USEFUL? | () | () |
| • COMPLETE? | () | () |
| • ACCURATE? | () | () |
| • WELL ORGANIZED? | () | () |
| • WELL ILLUSTRATED? | () | () |
| • WELL INDEXED? | () | () |
| • EASY TO READ? | () | () |
| • EASY TO UNDERSTAND? | () | () |

Please indicate below whether your comment pertains to an addition, deletion, change or error; and, where applicable, please refer to specific page numbers.

Page	Description of error or deficiency

From:

Name _____
Firm _____
Address _____
Telephone _____

Title _____
Department _____
City, State _____
Date _____

First Class
Permit No. A-737
Portland,
Oregon

BUSINESS REPLY

No postage stamp necessary if mailed in the United States

Postage will be paid by:

FLOATING POINT SYSTEMS, INC.

P. O. Box 23489

Portland, Oregon 97223

Attention: Technical Publications

**PROCESSOR
HANDBOOK**

FAST RECONSTRUCT PROCESSOR

AP120B

by FPS Technical Publications Staff

**Processor
Handbook
for
General
Electric**
860-7419-000

Publication No. FPS 860-7419-000
February, 1979

NOTICE

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

Floating Point Systems, Inc. assumes no responsibility for any errors which may appear in this publication.

PROPRIETARY INFORMATION

This document contains proprietary information and is supplied for identification, maintenance, engineering evaluation or inspection purposes only and shall not be duplicated or disclosed without written permission of **FLOATING POINT SYSTEMS, INC.**

By accepting this document the recipient agrees to make every effort to prevent unauthorized use of this information.

Copyright © 1979 by Floating Point Systems, Inc.
Beaverton, Oregon 97005

All rights reserved. No part of this publication may be reproduced in any form or by any means without permission in writing from the publisher.

Printed in USA

CONTENTS

	Page
PREFACE	P-1
CHAPTER 1 GENERAL INFORMATION	
1.1	1-1
1.2	1-3
1.3	1-5
1.4	1-6
1.4.1	General
1.4.2	Forward Unit
1.4.3	Rear Unit
1.4.4	Power, Controls, and Indicators
1.4.5	Serial Numbers
1.5	SOFTWARE
1.5.1	APEX (AP Executive)
1.5.2	APMATH (AP Math Library)
1.5.3	Program Development Package
1.5.4	APTEST (AP Test Programs)
CHAPTER 2 FUNCTIONAL DESCRIPTION	
2.1	INTRODUCTION
2.2	CONTROL UNIT
2.3	S-PAD UNIT
2.4	FLOATING-POINT ADDER UNIT
2.5	FLOATING-POINT MULTIPLIER UNIT
2.6	DATA PAD UNIT
2.7	DATA MEMORY UNIT
2.8	TABLE MEMORY UNIT
2.9	INTERNAL FLOATING-POINT FORMAT

	Page
CHAPTER 3	PROGRAMMING CONSIDERATIONS
3.1	INTRODUCTION
3.2	FLOATING-POINT ADDER
3.2.1	Floating Adder Operations
3.2.2	Adder Pipeline
3.2.3	An Example
3.2.4	Floating Adder Tests
3.2.5	Floating-Point Logical Operations
3.3	FLOATING-POINT MULTIPLIER
3.3.1	Multiply Instruction
3.3.2	Multiplier Pipeline
3.3.3	An Example
3.3.4	Multiply-Adds
3.4	DATA PAD
3.4.1	Data Pad Addressing
3.4.2	Writing Into Data Pad
3.4.3	Data Pad Bus
3.5	DATA MEMORY
3.5.1	Memory Addressing
3.5.2	Data Memory Reads
3.5.3	An Example
3.5.4	Data Memory Writes
3.5.5	An Example
3.5.6	Memory Interleave
3.5.7	Memory Lockout
3.6	TABLE MEMORY
3.6.1	Table Memory Addressing
3.6.2	An Example
3.6.3	A Complex Multiply
3.7	S-PAD
3.7.1	Single Operand Instructions
3.7.2	Double Operand Instructions
3.7.3	S-Pad Test
3.7.4	An Example

	Page	
CHAPTER 4	INTERFACE	
4.1	INTRODUCTION	4-1
4.2	FRONT PANEL	4-1
4.2.1	Switch Register	4-3
4.2.2	Lights Register	4-3
4.2.3	Function Register	4-3
4.3	NOTES ON THE USE OF THE FRONT PANEL AND BREAKPOINT	4-7
4.3.1	Where Does The AP Stop on a Breakpoint?	4-7
4.3.2	Does the Instruction on Which the AP Stops Execute?	4-7
4.3.3	What About MD Timing and Lockout on a Breakpoint in the Middle of an MD Memory Cycle?	4-8
4.3.4	Summary of the Rule for Proceeding from Breakpoint	4-8
4.3.5	What About Stepping the AP?	4-8
4.3.6	What Other Pitfalls Are There in the Use of the Virtual Front Panel?	4-9
4.4	DIRECT MEMORY ACCESS	4-11
4.4.1	Host Memory Address Register	4-11
4.4.2	Word Count Register	4-11
4.4.3	AP Direct Memory Address Register	4-12
4.4.4	Control Register	4-12
4.5	FORMAT CONVERSION REGISTER	4-15
4.6	AP INTERNAL INTERFACE TO HOST INTERFACE	4-16
4.7	AN EXAMPLE OF LOADING PROGRAMS INTO THE AP	4-18
APPENDIX A	AP REGISTERS/DATA PATH NAMES	
APPENDIX B	INSTRUCTION SUMMARY	
APPENDIX C	DATA GENERAL ECLIPSE INTERFACE	

ILLUSTRATIONS

Figure No.	Title	Page
1-1	General AP Block Diagram	1-4
1-2	AP Physical Configuration	1-8
2-1	Control Unit	2-3
2-2	S-Pad Unit	2-5
2-3	Floating-Point Adder Unit	2-8
2-4	Floating Multiplier	2-10
2-5	Data Pad	2-12
2-6	Data Memory Unit	2-14
2-7	Table Memory	2-16
3-1	Data Pad Address	3-15
4-1	AP Panel and Host Interface	4-2
4-2	Panel Function Register Format	4-3
4-3	DMA Control Register Format	4-12
A-1	AP Functional Units	A-5
C-1	Bit Map for Page Select Registers	C-3

TABLES

Table No.	Title	Page
1-1	Floating-Point Arithmetic Times	1-12
1-2	Basic Scalar Functions	1-13
1-3	Summary of AP FORTRAN Callable Routines	1-14
1-4	Convolution (Correlation)	1-23
1-5	Fast Fourier Transforms	1-23
3-1	Floating Adder Tests	3-6
3-2	Memory Interleave Sequence	3-25
3-3	Single Operand Instructions	3-32
3-4	Double Operand Instructions	3-33
4-1	Function Register Bits	4-4
4-2	Bits 8-9	4-5
4-3	Bits 10-11	4-5
4-4	Octal Values	4-6
4-5	DMA Control Register Description	4-13
4-6	Bits 13-14	4-14
4-7	CTL Register Bits 9-10	4-15
4-8	AP Device Address for Host Interface Registers	4-16
A-1	Registers and Data Paths	A-1
A-2	AP Internal Status Register	A-2
A-3	AP Instruction Summary	A-6
A-4	SPEC Fields	A-7
A-5	I/O Fields	A-7
B-1	AP Instruction Field Layout	B-2
B-2	S-pad Group	B-3
B-3	Special Operations Group	B-5
B-4	Floating Adder Group	B-13
B-5	I/O Group	B-15
B-6	Branch Group	B-19
B-7	Data Pad Group	B-20
B-8	Floating Multiplier Group	B-22
B-9	Memory Group	B-23

PREFACE

Historically, array transform processors have been largely integer-arithmetic devices, since the slower processing rate of floating-point arithmetic was undesirable when working with large arrays of data. However, integer methods have problems which make programming awkward due to the limited dynamic range of integer arithmetic. Array scaling and block floating-point techniques either allowed human and other errors to creep into the results or were costly and time consuming. Further, as processing became more sophisticated, even 16-bit integer data words were insufficiently precise for preserving the accuracy of simple 8-bit analog-to-digital converted input data. This is because the many multiplications and additions in typical cascaded array processing can cause the propagation of truncation errors.

NOTE

A 16-bit integer multiplied by a 16-bit integer results in a 32-bit product. If the result is truncated to the 16 most significant bits, then half the time the resultant's least significant bit (LSB) is wrong since it should have been rounded up. Now the product of two of these potentially wrong LSB numbers results in the next LSB being wrong part of the time; thus cascaded operations propagate the errors leftward toward the most significant bits.

With the advent of faster digital logic, many users realized that floating-point processing makes programming easier, virtually eliminates dynamic range problems, greatly alleviates the precision problem, and is potentially as fast as the last generation of integer processors. Floating Point Systems, Inc., recognized this trend in 1970 and was formed to specialize in floating-point processors.

The rush to floating-point processing was not a smooth one. Many floating-point formats sprang up and Floating Point Systems became expert in format converting on-the-fly so processing time would not be lost during a format conversion. Why convert formats? Simple. Not all formats are mathematically clean. For example, it is unwise to use a hexadecimal-exponent format for serious number crunching because a hexadecimal normalization can cause as many as three leading zeros between the binary point of the mantissa and the first significant bit. This means that as many as three least-significant bits may be lost, due to right-shifting the mantissa past the available word length (truncation) when an extreme hexadecimal normalization occurs (about 25 percent of the time), and, of course, 2, 1, or no bits may be lost (with equal probability) for other possible hexadecimal cases. Cascaded calculations can quickly cause the low-resolution three-leading-zero data words to contaminate a data base.

The FPS solution is to use a true 10-bit binary exponent, which has more dynamic range than the standard 7-bit hexadecimal or 8-bit binary exponent. FPS then uses a 28-bit mantissa, plus three guard bits in the adder and a double mantissa at the multiplier output, which provides enough bits to not only allow for hexadecimal in/out formats, but also to carry enough information to permit post-normalization and convergent-rounding after each arithmetic operation. Thus, FPS can receive any reasonable floating-point format that is desired as the input format, convert it on-the-fly to the FPS format, process it in FPS format with minimal truncation error propagation, and then convert it on-the-fly to the desired output format. This procedure allows transparent no penalty operation on the data, thus preserving the integrity of the input data.

In addition to the well chosen floating-point format, the AP has a general-purpose, multi-bus oriented architecture for the arithmetic units. This allows great flexibility in that operands and resultants can be moved simultaneously from almost any register in the AP to any other. This rather generalized structure of the AP allows it to execute specialized algorithms, such as the FFT, in times comparable to those achieved by hardwired special-purpose processors and also makes the AP well suited to less highly organized computations.

In the matter of software, note that this machine is a synchronous monolithic multiprocessor, as opposed to an asynchronous multiprocessor. The practical significance of this is that programming by the user and/or FPS (Standard Algorithms, System and Test Software) is tremendously simplified due to the predictability of data flow and timing considerations. There is no need for internal hand-shaking between arithmetic units, memories, and microprocessor; data and results are available at precisely determined times. The synchronous approach not only allows a non-stochastic simulator to be written for easy program debugging, but in addition, programs may be single-stepped in the real processor, with execution identical to free-running programs. A further bonus of the synchronous design is the easy producibility, maintainability, interchangeability and reliability (there is no need to explore an infinite number of possible timing conditions as one clock phases by another, as happens in an asynchronous machine). Convenient and rapid data-dependent branching, simple overlapping of data input, arithmetic processing, and data output are further examples of the care taken to assure a fast, accurate, convenient, and reliable array processor.

CHAPTER 1

GENERAL INFORMATION

1.1 INTRODUCTION

The AP is a high-speed (167ns cycle time) peripheral floating-point arithmetic array processor (AP), which is intended to work in parallel with a host computer.

The AP's internal organization is particularly well suited to performing the large numbers of reiterative multiplications and additions required in digital signal processing, matrix arithmetic, statistical analysis, and numerical simulation.

The highly-parallel structure of the AP allows the overhead of array indexing, loop counting, and data fetching from memory to be performed simultaneously with arithmetic operations on the data. This allows much faster execution than on a typical general-purpose computer where each of the above operations must occur sequentially.

The AP achieves its high speed through the use of fast commercial integrated circuit elements and an architecture that permits each logical unit of the machine to operate independently and at maximum speed.

Specifically:

- Programs, constants, and data each reside in separate, independent memories to eliminate memory accessing conflicts.
- Independent floating-point multiply and adder units allow both arithmetic operations to be initiated every 167ns.
- Two large (32 locations each) blocks of floating-point accumulators are available for temporary storage of intermediate results from the multiplier, adder, or memory.
- Address indexing and counting functions are performed by an independent integer arithmetic unit that includes 16-integer accumulators.

In a typical application, such as a fast fourier transform (FFT), the above features allow nearly the entire computation to be overlapped with data memory access time.

Effective processing precision is enhanced by 38-bit internal data words, an internal floating-point format with optimum numerical properties, and a convergent rounding algorithm.

1.2 SYSTEM OVERVIEW

A general block diagram of AP arithmetic paths appears in Figure 1-1.

Connection is made to the host in a manner that permits data transfers to occur under control of either the host computer or the AP. For most host computers, this means that the AP is interfaced to both the programmed I/O and DMA channels.

The system elements are interconnected with multiple parallel paths so that transfers can occur in parallel. All internal floating-point data paths are 38 bits wide (10-bit biased binary exponent and 28-bit 2's complement mantissa).

Main data memory (MD) is organized in 8K-word modules of 38-bit words expandable up to 64K words in the main chassis. The effective memory cycle time (interleaved) is 333ns.

Table memory (TM) is used for storage of constants (FFT constants) and is tied to a separate data path so as not to interfere with data memory. It is bi-polar 167ns read-only memory and is organized in 512-word, 38-bit increments.

Data pad X (DPX) and data pad Y (DPY) are two blocks of 32 floating accumulators. Each is a two-part register block, wherein one register may be read and another written from each block in one instruction cycle.

The floating adder (FA) consists of two input registers (A1 and A2) and a two-stage pipeline which performs the operations and convergently rounds the normalized result.

The floating multiplier (FM) consists of input registers (M1 and M2) and a three-stage pipeline which performs the multiply operation. Products are normalized and convergently rounded 38-bit numbers.

The s-pad consists of 16 integer registers and an integer arithmetic unit which is used to form operand addresses and to perform integer arithmetic.

Chapter 2 contains a more detailed description of each of the functional elements. Chapter 3 describes programming considerations.

Chapter 4 describes in detail the host computer interface, which Floating Point Systems, Inc., supplies. A number of off-the-shelf interfaces are available.

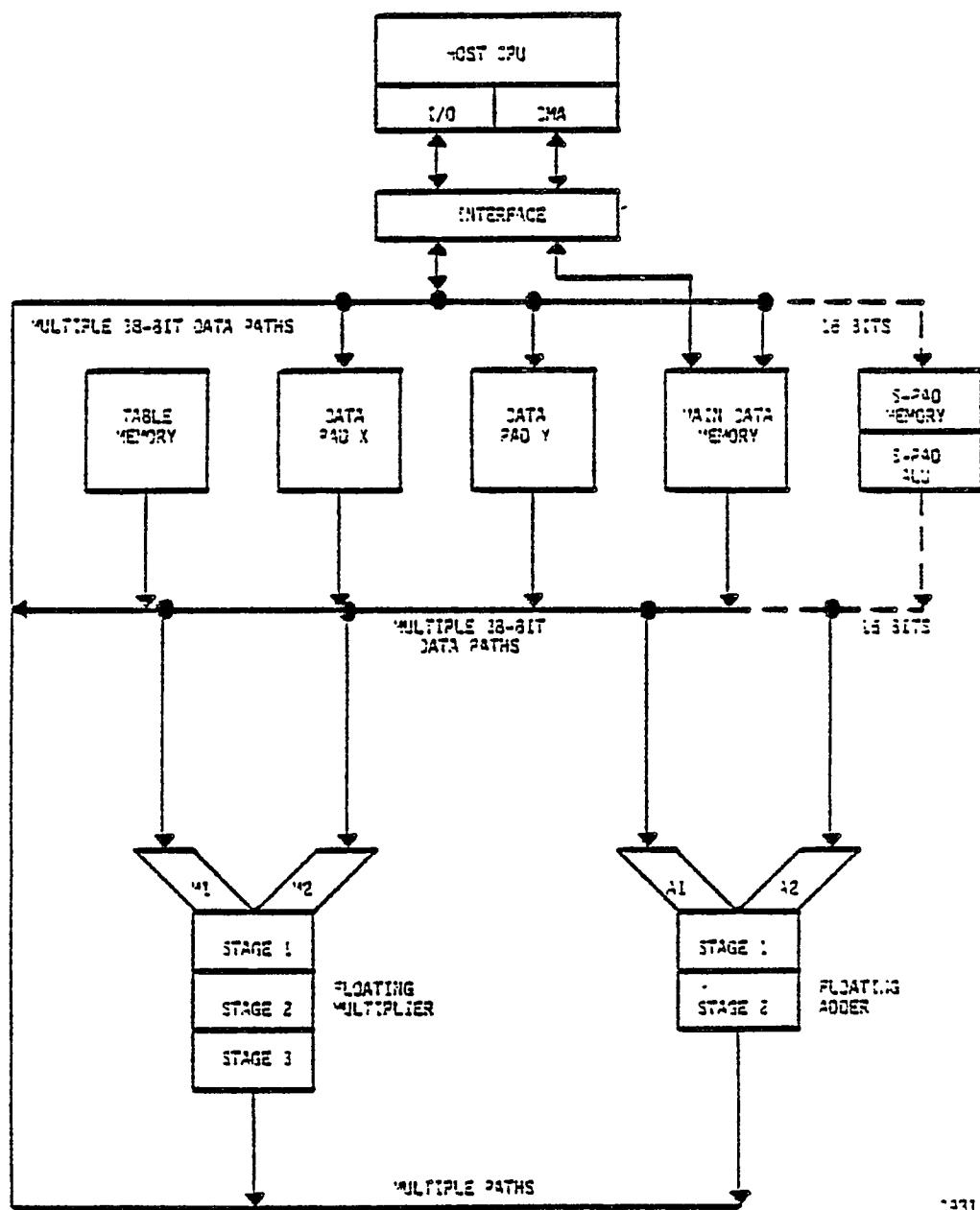


Figure 1-1 General AP Block Diagram

1.3 EXAMPLE AP APPLICATION

A simple FFT processing sequence goes as follows:

Initial conditions are that the FFT program is resident in program source memory internal to the AP, the array to be transformed is resident in host memory, and the host CPU has initiated the AP processor with an I/O instruction.

1. The AP requests host DMA cycles to transfer the array from host memory to internal data memory. Data is converted from host floating-point format to internal AP floating-point format on-the-fly.
2. The FFT algorithm is performed with data remaining in internal AP format. This yields the benefit of 38-bit precision and convergent rounding during the critical phases of processing.
3. The frequency domain array is transferred back to host memory by requesting host DMA cycles. Data is converted from internal format to host format on-the-fly.
4. The AP proceeds to another process or stops executing, depending on previously established conditions. An interrupt to the host can be issued.

The AP is most efficiently used when a sequence of operations is performed on one or more sets of data which reside in internal data memory. This reduces data transfer overhead and retains maximum numerical precision. For example, a reasonable sequence would be to transfer a trace and a filter, FFT both, array multiply, inverse FFT, and transfer the result back to host memory.

The AP data memory has DMA capability. That is to say, MD cycles can be stolen from the AP microprocessor by the interface. This capability allows host computer DMA to AP DMA data transfers to occur, thereby minimizing both host CPU and AP overhead.

The AP is designed with enough flexibility built in so that its power can be harnessed in a variety of ways. Subsequent sections describe its use in detail.

1.4 PHYSICAL DESCRIPTION

The following sections describe the AP hardware.

1.4.1 GENERAL

The AP is available in rack configuration. Mounting is as a standard 19-inch EIA rack-mounted unit requiring 24-1/2 inches of vertical space. The unit is equipped with rack slides permitting easy access to the etched and/or wire-wrapped circuitry with the chassis mounted on the forward portion of the unit. The power panel is mounted at the rear. One and three-quarter inches of space should be available above and below the 24-1/2 inches of the processor. This is for proper intake and exhaust of air through the processor. The control panel (refer to section 1.4.4) and/or blank panels may be used for proper spacing if the customer's equipment mounted above and below the processor does not have the proper free-air space built into it. Intake air should be between 10 and 40 degrees centigrade.

1.4.2 FORWARD UNIT

The forward unit contains all AP circuitry except the power supply. There is provision for up to 31 15-by-10-inch etched-circuit boards (ECB). The ECBs plug into a mother board. The ECBs are arranged in a vertical plane (chimney style) with push/pull fans to assure adequate upwards air circulation even in the event of a fan failure. The I/O cable exits at the bottom rear (the exact configuration is computer dependent). This unit is called the processor.

1.4.3 REAR UNIT

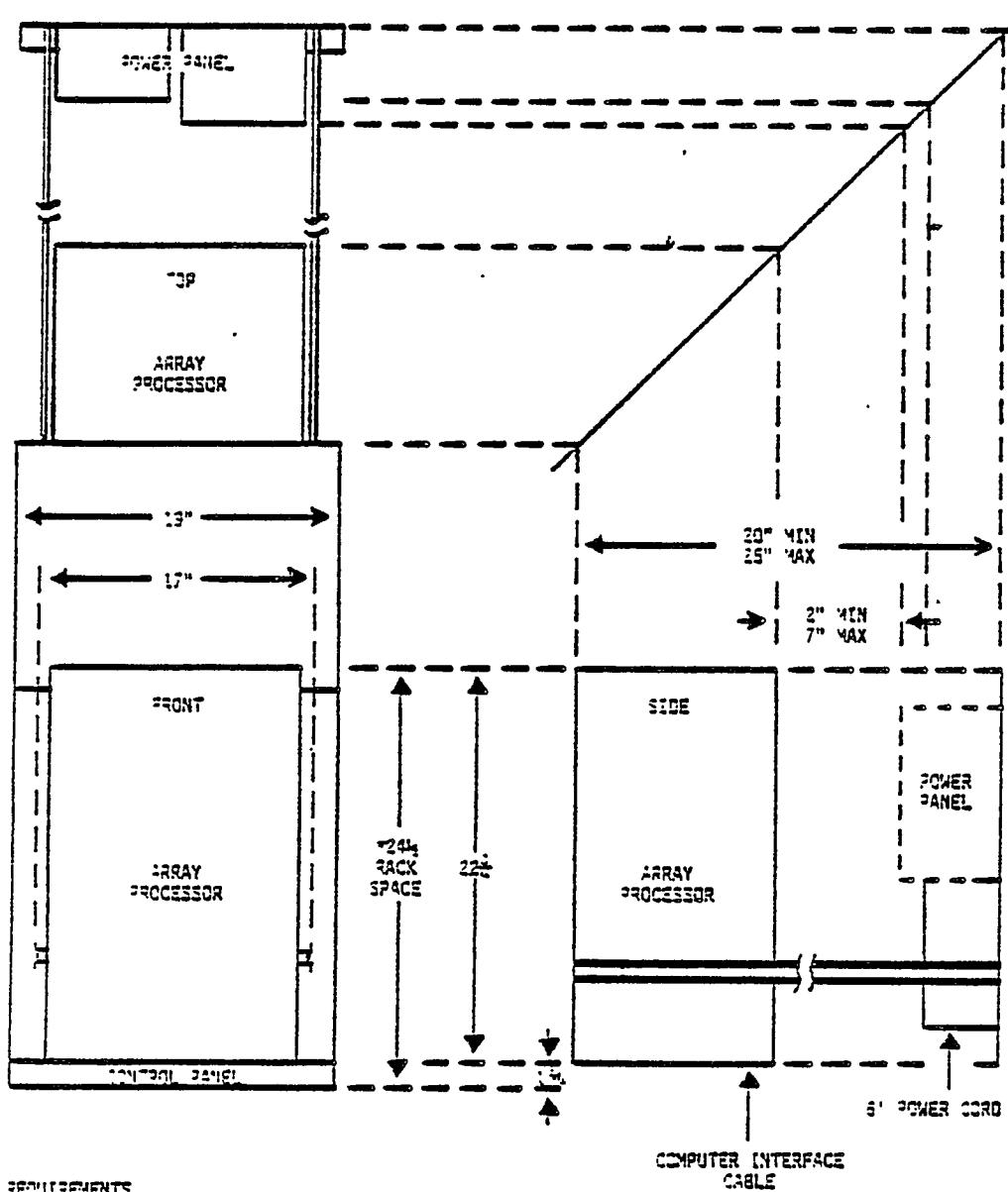
The power supply consists of three assemblies. The first is the main +5 volt supply and is capable of 100 amperes output. The other smaller supplies are -5 and +12 volts. The power supplies have forced convection cooling. All supplies are rear-mounted, along with the line box (containing line filters and contactor), on the power panel.

1.4.4 POWER, CONTROLS, AND INDICATORS

The AP is expected to be normally powered up and down with the rest of the system. The AP switch and indicators are on a control panel. There is a single power cord (US standard three-wire with ground) which must be connected to 105 to 125 volts, 50 to 60 hertz. The service should be rated for 20 amperes (10 amperes in the case of the higher ranges) in order to provide a low-impedance source (power required is approximately 1200 volt-amps). The control panel may be mounted above or below either the processor or the power panel. Availability of line power is indicated by a neon LINE VOLTAGE indicator. If the ON/OFF switch is on, then power supplies should come on. There are two operation indicators: one shows array processor action and the other shows DMA transfers. The three individual power supplies have separate indicators (electroluminescent diodes). There are no external adjustments. The internal adjustments are the three power supply setting potentiometers on the power panel.

1.4.5 SERIAL NUMBERS

The processor has a serial number tag on its starboard side near the top and forward ending in A. The power panel tag, ending in B, is located inside and near the top. The control panel has its tag ending in C, also inside.



REQUIREMENTS

- 1) ENVIRONMENT: 0 - 40°C 3 0 - 90% RELATIVE HUMIDITY.
(DEGRADE 1°C PER 2500 FT. (762 M) ABOVE SEA LEVEL, 5°C FOR 50 HERTZ OPERATION.)
- 2) POWER CONSUMPTION = 1200 W; SERVICE:
A. 125 - 125 VOLTS, 50 - 60 HERTZ 3 20 AMPS. (VOLTAGE OPTION "A" HAS A WHITE WIRE IN THE FAN POWER CABLE.)
B. 138 - 223 VOLTS, 50 - 60 HERTZ 3 10 AMPS. (VOLTAGE OPTION "B" HAS A BLUE WIRE IN THE FAN POWER CABLE.)
C. 220 - 250 VOLTS, 50 - 60 HERTZ 3 10 AMPS. (VOLTAGE OPTION "C" HAS A RED WIRE IN THE FAN POWER CABLE.)
D. LOW IMPEDANCE SERVICE ADVISED.
- 3) SPACE:
WEIGHT: WITH CONTROL PANEL AT THE FRONT: 244" (62.23 KG).
WITH CONTROL PANEL AT THE REAR: 22 1/2" (57.73 KG).
DEPTH: 20" - 25" (50.30 - 63.50 CM).
DEPTH: 20" - 25" (50.30 - 63.50 CM).

CAUTION: ALLOW AT LEAST 1.75" OF FREE AIR SPACE ABOVE THE AP IF USED AS SHOWN. IF THE CONTROL PANEL IS MOVED,
ALLOW 1.75" OF FREE AIR SPACE BELOW THE AP.

NOTE: THE POWER PANEL TO AP POWER CABLE IS LOCATED ON THE LOWER RIGHT SIDE (NOT SHOWN).

3982

Figure 1-2 AP Physical Configuration

1.5 SOFTWARE

Four software packages can be supplied with the AP which assist the user toward the solution of the particular processing task.

1.5.1 APEX (AP EXECUTIVE)

APEX is a mechanism for communicating with the AP via a series of FORTRAN or machine language subroutine calls. The executive driver routine interprets the particular user call and directs the AP to perform the specified action. For example, in FORTRAN, to load an array A containing N real data points into the AP and perform a real fast fourier transform upon that data:

```
    .
    .
    .
    IA=0
    CALL APPUT (A,IA,N,2)
    CALL RFFT (IA,N,1)
    .
    .
    .
```

Both the standard applications subroutines described below and user-developed AP programs may be called from the host computer using APEX.

1.5.2 APMATH (AP MATH LIBRARY)

There are 239 subroutines written in AP assembly language. They are callable from the host computer FORTRAN or machine language using APEX. They are listed in Table 1-3.

1.5.3 PROGRAM DEVELOPMENT PACKAGE

Six FORTRAN IV programs, which are compiled on the host computer during installation, aid user program development.

These are:

APAL	AP assembly language. Cross-assembler which provides a two-pass assembly of symbolic coding into an object module. APAL generates detailed error diagnostics.
APLOAD	AP loader. Links and relocates separate APAL and AP-FORTRAN object modules together. It produces a load module and a host FORTRAN subroutine which transfers the load module to the AP.
APDEBUG	AP debugger. Interactive debugging program. The user may selectively set breakpoints, examine and change memory, and register contents and run program segments.
APSIM	AP simulator. Called by APDEBUG, APSIM provides a programmed simulation of the various hardware elements of the AP. All timing characteristics of the AP are emulated and the floating-point arithmetic is simulated (including rounding) to the least significant bit. APSIM is a convenient tool in bringing up new AP programs off-line without interfering with production runs.
VFC	Vector Function Chainer. A translator to convert VFC syntax to AP assembly language (APAL). It consolidates multiple CALLS to the AP from the host computer into one CALL whenever possible.

AP-FORTRAN Array processor FORTRAN. A compiler which allows FORTRAN subprograms to execute on the AP. The compiler produces object modules which are used as input to the AP loader (APLOAD).

1.5.4 APTEST (AP TEST PROGRAMS)

APTEST is a collection of interactive diagnostic tests and verify programs which aid in isolation of hardware faults.

These are:

APTEST AP tester. Exercises the panel, DMA interface, and various internal registers and memories. Tests main data memory with simple patterns and then with random numbers. Board-level diagnostic indicators are provided.

APPATH AP path tester. Tests the various internal data paths and gives board level diagnostics.

APARTH AP arithmetic test. Tests the floating-point adder, multiplier, and s-pad arithmetic unit with pseudorandom number and operation sequences.

FIFFT Forward/inverse FFT test. Verifies the correct operation of the AP as a complete unit by doing forward/inverse FFT transforms on both spikes and random number sequences.

Table 1-1 Floating-Point Arithmetic Times

OPERATION	TRAVEL TIME	Pipeline Interval
Add/Subtract	0.333 us	0.157 us
Multiply	0.500 us	0.157 us
Multiply-Add	0.333 us	0.157 us
Complex Add/Subtract	0.500 us	0.333 us
Complex Multiply	1.333 us	0.667 us
Complex Multisly-Add	1.667 us	0.667 us

3983

Travel time is the total time required to get from the data source to the destination including the full transport through the arithmetic units. Pipeline interval is the time between successively available resultants. The former is important when the successive arguments of a computation depend on previous calculations. The latter is indicative of the maximum throughput rate available for successively independent calculators.

Table 1-2 Basic Scalar Functions

OPERATION	TYPICAL EXECUTION TIME/LOOP (ns)		PROGRAM SIZE (AP PS WORDS)	
	167 ns	333 ns	167 ns	333 ns
Divide	3.8	3.8	28	28
Square Root	3.8	3.8	28	28
Exponential	4.2	4.2	28	23
Natural Logarithm	4.0	4.0	37	37
Base 10 Logarithm	4.7	4.7	37	37
Sine	4.9	4.9	35	35
Cosine	5.4	5.4	35	35
Arctangent	8.7	8.7	74	74
Arctangent of (Y/X)	13.3	13.3	74	74

3984

These functions take arguments from data pad and return full-word accuracy results to data pad. Full-precision polynomial coefficients for these functions are contained on the standard 512 words of table memory.

Table 1-3 Summary of AP FORTRAN Callable Routines

Name	Operation	Typical Execution Time/Loop (us)	Program Size (AP PS words)
		167 333	167 333

DATA TRANSFER AND CONTROL OPERATIONS (APEX)

APPUT	PUT DATA INTO THE AP	#.#	#.#	0	0
APGET	GET DATA FROM THE AP	#.#	#.#	0	0
APCLR	INITIALIZE THE AP	#.#	#.#	0	0
APWD	WAIT FOR AP DATA TRANSFER	#.#	#.#	0	0
APWR	WAIT FOR AP PROGRAM EXECUTION	#.#	#.#	0	0
APWAIT	WAIT FOR AP	#.#	#.#	0	0
APGSP	READ AN AP S-PAD REGISTER	#.#	#.#	0	0
APCHK	CHECK AP PROGRAM ERROR CONDITION	#.#	#.#	0	0
APSTAT	GET AP HARDWARE STATUS	#.#	#.#	0	0

BASIC VECTOR ARITHMETIC

VCLR	VECTOR CLEAR	0.2	0.3	16	4
VMOV	VECTOR MOVE	0.5	0.8	16	6
VSWAP	VECTOR SWAP	1.2	1.5	21	12
VFILL	VECTOR FILL	0.3	0.3	5	5
VRAMP	VECTOR RAMP	0.3	0.3	12	12
VNEG	VECTOR NEGATE	0.5	0.8	18	7
VADD	VECTOR ADD	0.8	1.3	20	8
VSUB	VECTOR SUBTRACT	0.8	1.3	20	8
VMUL	VECTOR MULTIPLY	0.8	1.3	20	11
VDIV	VECTOR DIVIDE	1.7	1.7	75	75
VSADD	VECTOR SCALAR ADD	0.5	0.8	19	8
VSMUL	VECTOR SCALAR MULTIPLY	0.5	0.8	20	9
VTSADD	VECTOR TABLE SCALAR ADD	0.5	0.8	8	8
VTSMUL	VECTOR TABLE SCALAR MULTIPLY	0.5	0.8	8	8
VSQ	VECTOR SQUARE	0.5	0.8	9	9
VSSQ	VECTOR SIGNED SQUARE	0.5	0.8	21	9
VABS	VECTOR ABSOLUTE VALUE	0.5	0.8	17	7
VSQRT	VECTOR SQUARE ROOT	1.8	1.8	79	79
VLOG	VECTOR LOGARITHM (BASE 10)	2.7	2.7	54	58
VLN	VECTOR NATURAL LOGARITHM	2.7	2.7	42	42

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)	Program Size (AP PS words)
		167 333	167 333
VALOG	VECTOR ANTILOGARITHM (BASE 10)	2.3	58
VEXP	VECTOR EXPONENTIAL	2.3	55
VSIN	VECTOR SINE	1.3	34
VCOS	VECTOR COSINE	1.3	34
VATAN	VECTOR ARCTANGENT	9.7	87
VATN2	VECTOR ARCTANGENT OF Y/X	14.2	88
VRAND	VECTOR RANDOM NUMBERS	1.2	16
VMSA	VECTOR MULTIPLY AND SCALAR ADD	0.8	23
VSMA	VECTOR SCALAR MULTIPLY AND ADD	0.8	21
VSMSB	VECTOR SCALAR MULTIPLY AND SUBTRACT	0.8	21
VMA	VECTOR MULTIPLY AND ADD	1.2	23
VMSB	VECTOR MULTIPLY AND SUBTRACT	1.2	23
VAM	VECTOR ADD AND MULTIPLY	1.2	23
VSBM	VECTOR SUBTRACT AND MULTIPLY	1.2	23
VMSA	VECTOR SCALAR MULTIPLY AND SCALAR ADD	0.5	23
VMMA	VECTOR MULTIPLY, MULTIPLY, AND ADD	1.5	27
VMMB	VECTOR MULTIPLY MULTIPLY AND SUBTRACT	1.5	27
VAAM	VECTOR ADD, ADD, AND MULTIPLY	1.5	13
VSBMB	VECTOR SUBTRACT SUBTRACT AND MULTIPLY	1.5	13
VAND	VECTOR LOGICAL AND	0.8	20
VEQV	VECTOR LOGICAL EQUIVALENCE	0.8	20
VOR	VECTOR LOGICAL OR	0.8	20
VFRAC	VECTOR TRUNCATE TO FRACTION	0.7	13
VINT	VECTOR TRUNCATE TO INTEGER	0.5	9
VINDEX	VECTOR INDEX	0.8	26

VECTOR-TO-SCALAR OPERATIONS

SVE	SUM OF VECTOR ELEMENTS	0.3	0.3	7	7
SVEMG	SUM OF VECTOR ELEMENT MAGNITUDES	0.3	0.3	10	10
SVESQ	SUM OF VECTOR ELEMENT SQUARES	0.3	0.3	10	10
SVS	SUM OF VECTOR SIGNED SQUARES	0.3	0.3	11	11
DOTPR	DOT PRODUCT	0.5	0.8	21	9
MAXV	MAXIMUM ELEMENT IN VECTOR	0.3	0.3	19	19
MINV	MINIMUM ELEMENT IN VECTOR	0.3	0.3	19	19
MAXMGV	MAXIMUM MAGNITUDE ELEMENT IN VECTOR	0.3	0.3	19	19
MINMGV	MINIMUM MAGNITUDE ELEMENT IN VECTOR	0.3	0.3	19	19

Table I-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)	Program Size (AP PS words)
		167 333	167 333
MEANV	MEAN VALUE OF VECTOR ELEMENTS	0.3	49 49
MEAMGV	MEAN OF VECTOR ELEMENT MAGNITUDES	0.3	52 52
MEASQV	MEAN OF VECTOR ELEMENT SQUARES	0.3	52 52
RMSQV	ROOT-MEAN-SQUARE OF VECTOR ELEMENTS	0.3	81 81

VECTOR COMPARISON OPERATIONS

VMAX	VECTOR MAXIMUM	0.8	1.3	22	13
VMIN	VECTOR MINIMUM	0.8	1.3	22	13
VMAXMG	VECTOR MAXIMUM MAGNITUDE	0.8	1.3	14	14
VMINMG	VECTOR MINIMUM MAGNITUDE	0.8	1.3	14	14
VCLIP	VECTOR CLIP	0.5	0.8	16	16
VICLIP	VECTOR INVERTED CLIP	0.7	0.8	19	19
VLIM	VECTOR LIMIT	0.5	0.8	14	14
LVGT	LOGICAL VECTOR GREATER THAN	0.8	1.3	23	13
LVGE	LOGICAL VECTOR GREATER THAN OR EQUAL	0.8	1.3	23	13
LVEQ	LOGICAL VECTOR EQUAL	0.8	1.3	23	13
LVNE	LOGICAL VECTOR NOT EQUAL	0.8	1.3	23	13
LVNOT	LOGICAL VECTOR NOT	0.5	0.8	21	12
VLMERG	VECTOR LOGICAL MERGE	0.8	1.5	23	16

COMPLEX VECTOR ARITHMETIC

CVMOV	COMPLEX VECTOR MOVE	0.8	1.3	9	9
CVFILL	COMPLEX VECTOR FILL	0.5	0.7	8	8
CVCOMB	COMPLEX VECTOR COMBINE	1.1	1.7	10	10
CVREAL	FORM COMPLEX VECTOR OF REALS	0.8	1.2	9	9
VREAL	EXTRACT REALS OF COMPLEX VECTOR	0.5	0.8	17	7
VIMAG	EXTRACT IMAGINARIES OF COMPLEX VECTOR	0.5	0.8	18	8
CVNEG	COMPLEX VECTOR NEGATE	0.8	1.3	11	11
CVCONJ	COMPLEX VECTOR CONJUGATE	0.7	1.3	10	12
CVADD	COMPLEX VECTOR ADD	1.0	2.0	13	12
CVSUB	COMPLEX VECTOR SUBTRACT	1.0	2.0	13	12

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)	Program Size (AP PS words)
		167 333	167 333
CVMUL	COMPLEX VECTOR MULTIPLY	1.0	25 26
CVSMUL	COMPLEX VECTOR SCALAR MULTIPLY	0.8	12 12
CVRCIP	COMPLEX VECTOR RECIPROCAL	5.2	50 50
CRVADD	COMPLEX AND REAL VECTOR ADD	1.3	14 14
CRVSUB	COMPLEX AND REAL VECTOR SUBTRACT	1.3	14 14
CRMUL	COMPLEX AND REAL VECTOR MULTIPLY	1.3	14 14
CRVDIV	COMPLEX AND REAL VECTOR DIVIDE	3.3	92 92
CVMA	COMPLEX VECTOR MULTIPLY AND ADD	1.3	29 30
CVMAGS	COMPLEX VECTOR MAGNITUDE SQUARED	0.7	13 18
SCJMA	SELF-CONJUGATE MULTIPLY AND ADD	0.8	14 15
POLAR	RECTANGULAR TO POLAR CONVERSION	19.5	120 120
RECT	POLAR TO RECTANGULAR CONVERSION	2.3	49 49
CVEXP	COMPLEX VECTOR EXPONENTIAL	2.0	43 43
CVMEXP	VECTOR MULTIPLY COMPLEX EXPONENTIAL	2.3	48 48
CDOTPR	COMPLEX DOT PRODUCT	0.7	15 16

DATA FORMATTING OPERATIONS

VFLT	VECTOR INTEGER FLOAT	0.5	0.8	13	11
VFIX	VECTOR INTEGER FIX	0.7	0.8	18	7
VSMAFX	VECTOR SCALAR MULTIPLY, ADD, AND FIX	0.7	0.8	14	13
VSCALE	VECTOR SCALE (POWER 2) AND FIX	0.7	0.8	12	12
VSCSCL	VECTOR SCAN, SCALE (POWER 2) AND FIX	1.5	1.7	19	19
VSHFX	VECTOR SHIFT AND FIX	0.7	0.8	9	9
VUP8	VECTOR 8-BIT BYTE UNPACK	0.5	0.5	71	71
VUPS8	VECTOR 8-BIT SIGNED BYTE UNPACK	0.9	0.9	107	107
VPK8	VECTOR 8-BIT BYTE PACK	0.9	0.9	65	65
VUP16	VECTOR 16-BIT BYTE UNPACK	0.8	0.8	61	61
VUPS16	VECTOR 16-BIT SIGNED BYTE UNPACK	1.3	1.3	58	58
VPK16	VECTOR 16-BIT BYTE PACK	0.8	0.8	46	46
VFLT32	VECTOR 32-BIT INTEGER FLOAT	1.7	1.7	65	65
VFIX32	VECTOR 32-BIT INTEGER FIX	1.2	1.2	33	33
VSEFLT	VECTOR SIGN EXTEND AND FLOAT	0.8	0.8	15	15

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)	Program Size (AP PS words)
		167 333	167 333

MATRIX OPERATIONS

MTRANS	MATRIX TRANSPOSE	0.5 0.9	18 22
MMUL	MATRIX MULTIPLY	0.62* 0.83	59 59
MMUL32	MATRIX MULTIPLY (DIMENSION <=32)	0.50* 0.73	27 27
MATINV	MATRIX INVERSE	1.6 * 2.1	160 160
SOLVEQ	LINEAR EQUATION SOLVER	0.7 * 0.9	216 222
MVML3	MATRIX VECTOR MULTIPLY (3X3)	2.0 * 2.2	30 30
MVML4	MATRIX VECTOR MULTIPLY (4X4)	3.3 * 3.8	39 39
CTRN3	3-DIMENSION COORDINATE TRANSFORMATION	2.3 * 2.5	37 37
FMM	FAST MEMORY MATRIX MULTIPLY	0.43*	61
FMM32	FAST MEMORY MATRIX MULTIPLY (<=32)	0.41*	33

FFT OPERATIONS

CFFT	COMPLEX TO COMPLEX FFT (IN PLACE)	0.28* 0.40	186 184
CFFTB	COMPLEX TO COMPLEX FFT (NOT IN PLACE)	0.20* 0.28	189 189
RFFT	REAL TO COMPLEX FFT (IN PLACE)	0.18* 0.27	253 251
RFFTB	REAL TO COMPLEX FFT (NOT IN PLACE)	0.14* 0.20	252 252
CFFTSC	COMPLEX FFT SCALE	0.8 1.3	42 42
RFFTSC	REAL FFT SCALE AND FORMAT	0.7 0.8	59 59
CFFT2D	COMPLEX TO COMPLEX 2-DIMENSIONAL FFT	0.5 * 0.5	274 274
RFFT2D	REAL TO COMPLEX 2-DIMENSIONAL FFT	0.4 * 0.4	585 585

AUXILIARY OPERATIONS

CONV	CONVOLUTION (CORRELATION)	0.28* 0.28	106 106
DEQ22	DIFFERENCE EQUATION, 2 POLES, 2 ZEROS	0.8 0.8	25 25
VPOLY	VECTOR POLYNOMIAL EVALUATION	1.0 * 1.2	41 41
VSUM	VECTOR SUM OF ELEMENTS INTEGRATION	0.7 0.8	13 13

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)	Program Size (AP PS words)
		167 333	167 333
VTRAPZ	VECTOR TRAPEZOIDAL RULE INTEGRATION	0.7	0.8 16 16
VSIMPS	VECTOR SIMPSONS 1/3 RULE INTEGRATION	0.7	0.8 25 25
WIENER	WIENER LEVINSON ALGORITHM	0.50*	0.65 100 100

SIGNAL PROCESSING OPERATIONS (optional)

HIST	HISTOGRAM	1.3	1.4	71	71
HANN	HANNING WINDOW MULTIPLY	0.7	0.8	41	41
ASPEC	ACCUMULATING AUTO-SPECTRUM	0.8	1.5	21	22
CSPEC	ACCUMULATING CROSS-SPECTRUM	1.3	2.7	39	40
VAVLIN	VECTOR LINEAR AVERAGING	0.8	1.3	54	46
VAVEXP	VECTOR EXPONENTIAL AVERAGING	0.8	1.3	55	46
VDBPWR	VECTOR CONVERSION TO DB (POWER)	1.2	1.3	75	75
TRANS	TRANSFER FUNCTION	3.3	3.3	100	100
COHER	COHERENCE FUNCTION	4.0	4.5	109	114
ACORT	AUTO-CORRELATION (TIME-DOMAIN)	0.29*	0.29	121	121
ACORF	AUTO-CORRELATION (FREQUENCY-DOMAIN)	1.80*	2.70	501	489
CCORT	CROSS-CORRELATION (TIME-DOMAIN)	0.29*	0.29	121	121
CCORF	CROSS-CORRELATION (FREQUENCY-DOMAIN)	2.58*	3.93	526	510
TCONV	POSTTAPERED CONVOLUTION (CORRELATION)	0.30*	0.30	112	112

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)	Program Size (AP PS words)
		167 333	167 333

TABLE MEMORY OPERATIONS (optional)

MTMOV	VECTOR MOVE (MD TO TM)	0.2	0.3	6	7
TMMOV	VECTOR MOVE (TM TO MD)	0.2	0.3	5	5
MTIMOV	VECTOR MOVE WITH INCREMENT (MD TO TM)	0.5	0.5	7	7
TMIMOV	VECTOR MOVE WITH INCREMENT (TM TO MD)	0.3	0.3	15	15
TTIMOV	VECTOR MOVE WITH INCREMENT (TM TO TM)	0.5	0.5	7	7
MMTADD	VECTOR ADD (MD+MD TO TM)	0.7	0.8	20	13
MMTSUB	VECTOR SUBTRACT (MD-MD TO TM)	0.7	0.8	20	13
MMTMUL	VECTOR MULTIPLY (MD*MD TO TM)	0.7	0.8	20	13
MIMADD	VECTOR ADD (MD+TM TO MD)	0.5	0.3	20	9
MTMSUB	VECTOR SUBTRACT (MD-TM TO MD)	0.5	0.8	20	9
TMMSUB	VECTOR SUBTRACT (TM-MD TO MD)	0.5	0.8	20	9
MTMMUL	VECTOR MULTIPLY (MD*TM TO MD)	0.5	0.8	20	9
MTTADD	VECTOR ADD (MD+TM TO TM)	0.5	0.5	20	20
MTTSUB	VECTOR SUBTRACT (MD-TM TO TM)	0.5	0.5	20	20
TMTSUB	VECTOR SUBTRACT (TM-MD TO TM)	0.5	0.5	20	20
MTTMUL	VECTOR MULTIPLY (MD*TM TO TM)	0.5	0.5	20	20
TTIMADD	VECTOR ADD (TM+TM TO MD)	0.5	0.5	20	20
TTMSUB	VECTOR SUBTRACT (TM-TM TO MD)	0.5	0.5	20	20
TTMMUL	VECTOR MULTIPLY (TM*TM TO MD)	0.5	0.5	20	20
TTTADD	VECTOR ADD (TM+TM TO TM)	0.7	0.7	9	9
TTTSUB	VECTOR SUBTRACT (TM-TM TO TM)	0.7	0.7	9	9
TTTMUL	VECTOR MULTIPLY (TM*TM TO TM)	0.7	0.7	10	10

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)	Program Size (AP PS words)
		167 333	167 333

APAL-CALLABLE UTILITY OPERATIONS

DIV	SCALAR DIVIDE	3.8 @ 3.8	28	28
SQRT	SCALAR SQUARE ROOT	3.8 @ 3.8	28	28
LOG	SCALAR LOGARITHM (BASE 10)	4.7 @ 4.7	37	37
LN	SCALAR NATURAL LOGARITHM	4.0 @ 4.0	37	37
EXP	SCALAR EXPONENTIAL	4.2 @ 4.2	28	28
SIN	SCALAR SINE	4.9 @ 4.9	35	35
COS	SCALAR COSINE	5.4 @ 5.4	35	35
ATAN	SCALAR ARCTANGENT	8.7 @ 8.7	74	74
ATN2	SCALAR ARCTANGENT OF Y/X	13.8 @ 13.8	74	74
SPFLT	FLOAT S-PAD INTEGER	0.8 @ 0.8	5	5
SPUFLT	S-PAD UNSIGNED FLOAT	0.8 @ 0.8	8	8
SPNEG	S-PAD NEGATE	0.3 @ 0.3	2	2
SPADD	S-PAD ADD	0.2 @ 0.2	1	1
SPSUB	S-PAD SUBTRACT	0.2 @ 0.2	1	1
SPMUL	S-PAD MULTIPLY	2.3 @ 2.3	14	14
SPDIV	S-PAD DIVIDE	6.2 @ 6.2	43	43
SPRS	S-PAD RIGHT SHIFT	0.3 * 0.3	5	5
SPLS	S-PAD LEFT SHIFT	0.3 * 0.3	5	5
SPAND	S-PAD AND	0.2 @ 0.2	1	1
SPOR	S-PAD OR	0.2 @ 0.2	1	1
SPNOT	S-PAD NOT	0.2 @ 0.2	1	1
SAVESP	SAVE S-PAD INTO PROGRAM MEMORY	0.8 * 0.8	18	18
SAVSP0	SAVE S-PAD 0 INTO PROGRAM MEMORY	2.0 * 2.0	11	11
SETSP	LOAD S-PADS FROM PROGRAM MEMORY	2.3 * 2.3	33	33
SET2SP	LOAD 2 S-PADS FROM PROGRAM MEMORY	5.7 @ 5.7	33	33
MDCOM	MAIN DATA COMPARE AND SET S-PAD	1.8 @ 2.0	11	11
ZMD	CLEAR ALL PAGES OF MAIN DATA MEMORY	0.2 0.3	29	29
RDC5	READ CONTROL BIT 5 INTERRUPT	1.5 @ 1.5	9	9
SETCS	SET CONTROL BIT 5 INTERRUPT	0.2 @ 0.2	1	1
DAREAD	READ DEVICE ADDRESS REGISTER	0.3 @ 0.3	2	2
DAWRIT	WRITE DEVICE ADDRESS REGISTER	0.3 @ 0.3	2	2
VFC11	VECTOR FUNCTION CALLER (1 ARGUMENT)	0.8 1.0	10	10
VFC12	VECTOR FUNCTION CALLER (2 ARGUMENT)	1.0 1.0	11	11
BITREV	COMPLEX VECTOR BIT REVERSE ORDERING	0.9 1.4	45	43
REALTR	REAL FFT UNRAVEL AND FINAL PASS	0.4 0.7	68	68
FFT2	RADIX 2 FFT FIRST PASS	1.3 2.7	16	16

Table I-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)	Program Size (AP PS words)
		167 333	167 333
FFT4	RADIX 4 FFT PASS	3.7	5.3
FFT2B	RADIX 2 FFT FIRST PASS + BIT REVERSE	1.3	2.7
FFT4B	RADIX 4 FFT FIRST PASS + BIT REVERSE	2.7	5.3
STSTAT	SET FFT MODE STATUS BITS	5.0 @ 5.0	19 19
CLSTAT	CLEAR FFT MODE STATUS BITS	0.5 @ 0.5	19 19
ILOG2	LOGARITHM (BASE 2)	4.0 @ 4.0	19 19
ADV2	ADVANCE POINTERS AFTER RADIX 2 FFT	0.7 @ 0.7	7 7
ADV4	ADVANCE POINTERS AFTER RADIX 4 FFT	0.7 @ 0.7	7 7
SET24B	SETUP FOR FFT2B AND FFT4B	1.2 @ 1.2	8 8
KCFFT	EXPANDED COMPLEX FFT	0.32* 0.42	187 187
XRFFT	EXPANDED REAL FFT	0.19* 0.28	256 256
XBITRE	EXPANDED BIT REVERSE	3.7 3.7	44 44
XREALT	EXPANDED REAL FFT FINAL PASS	0.4 0.7	71 71
PCFFT	PARTIAL COMPLEX FFT	1.05* 1.50	117 117
KFFT4	EXPANDED RADIX 4 FFT PASS	3.7 5.3	79 79
CTOR	COMPLEX TO REAL FFT UNSCRAMBLE	0.13* 0.13	80 80
RTOC	REAL TO COMPLEX FFT SCRAMBLE	0.19* 0.09	143 143
SSDA	SINGLE + SINGLE TO DOUBLE ADD	1.5 @ 1.5	10 10
SSDM	SINGLE * SINGLE TO DOUBLE MULTIPLY	11.5 @ 11.5	81 81
SDDA	SINGLE + DOUBLE TO DOUBLE ADD	4.5 @ 4.5	28 28
DDDA	DOUBLE + DOUBLE TO DOUBLE ADD	7.5 @ 7.5	48 48
DDDM	DOUBLE * DOUBLE TO DOUBLE MULTIPLY	18.5 @ 18.5	117 117

NOTE

#.# Timing host system dependent

* Refer to description of routine for explanation of timing

@ Total execution time

Table 1-4 Convolution (Correlation)

ELEMENT COUNTS		TYPICAL EXECUTION TIME/LOOP μ sec	
4	4	167ns	333ns
8	128	0.28	0.28
32	128	0.83	0.53
128	128	3.0	3.0
6	1024	2.3	2.3
32	1024	6.6	6.6
128	1024	24.0	24.0
1024	1024	186.2	186.2

0985

Table 1-5 Fast Fourier Transforms

POINTS	RFFT		RFFT8		CFFT		CFFT8	
	167ns	333ns	167ns	333ns	167ns	333ns	167ns	333ns
64	0.18	0.27	0.14	0.20	0.29	0.40	0.29	0.28
128	0.35	0.50	0.27	0.38	0.62	0.95	0.47	0.72
256	0.74	1.13	0.58	0.90	1.28	1.36	0.97	1.41
512	1.50	2.22	1.20	1.76	2.86	4.38	2.25	3.43
1024	3.30	5.03	2.70	4.13	5.95	8.73	4.75	6.33
2048	6.81	10.12	5.61	9.32	13.32	20.10	10.33	16.60
4096	14.95	22.96	12.56	19.37	27.44	40.33	22.66	33.16
8192	30.38	45.26	25.09	38.69	60.33	91.56	50.75	77.31
16384	67.19	102.70	57.63	88.36	124.70	183.27	105.58	154.59
32768	138.42	205.35	119.30	176.58	--	--	--	--

0986

CHAPTER 2

FUNCTIONAL DESCRIPTION

2.1 INTRODUCTION

The hardware of the AP is composed of the following three types of functional elements:

- logical and control elements
 - control unit
 - s-pad unit
- floating-point arithmetic elements
 - floating-point adder
 - floating-point multiplier
- memory elements
 - data pad unit
 - main data memory unit
 - table memory unit

Each of these functional units is independent and thus can independently perform the programmed operations for which it was designed in parallel with the other functional units.

2.2 CONTROL UNIT

The control unit, as illustrated by Figure 2-1, consists of:

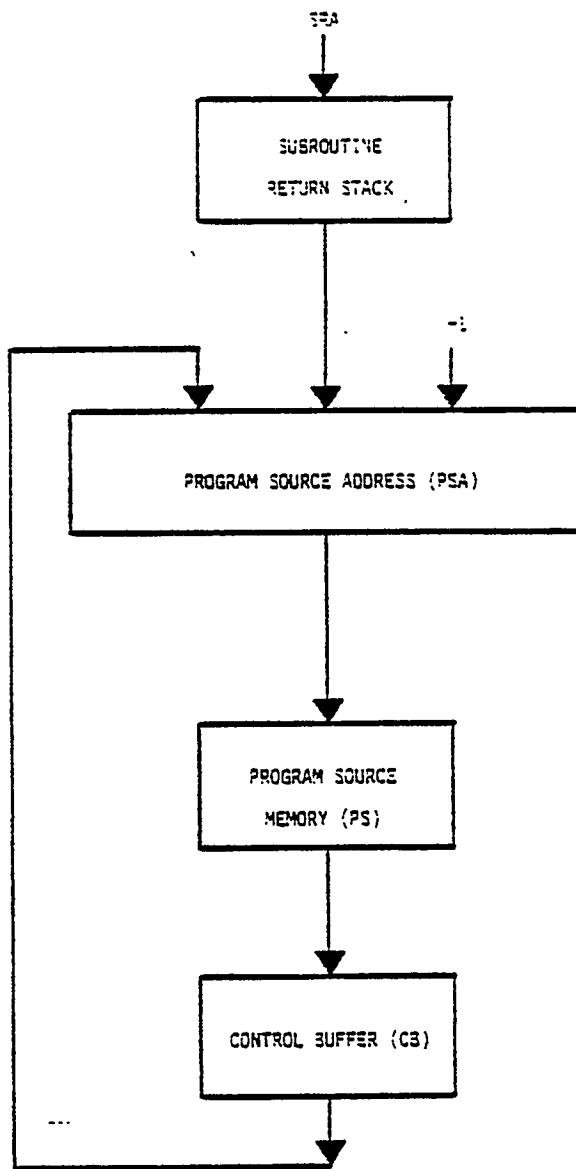
- program source memory (PS)
- program source address (PSA) register
- control buffer (CB) with decoding logic
- subroutine return stack (SRS)

The operation of the AP is controlled by the execution of 64-bit instruction words which reside in program source (PS) memory. The program word for the next instruction to be performed is selected by the address in the program source address (PSA) register. At the initiation of the next machine cycle, this program word is transferred to the control buffer (CB) where it is decoded and executed. The PSA is incremented by one unless a branch in the current instruction causes the PSA to move to another location in program source memory. Access to program source memory and instruction decoding is overlapped so that the AP can operate at a 6-MHz rate (167ns).

Branching is accomplished in two ways. A short-range branch is provided by adding the 5-bit branch displacement field to the current PSA. This gives a branch range of from -20₈ to +17₈. A long-range jump to any location in PS is accomplished by loading the desired target address into PSA.

Subroutine jumps are made by a JSR instruction which saves the current PSA in the subroutine return stack and sets PSA to the subroutine address. Return is via a return, which loads the PSA with the last entered return address on the SRS.

Subroutine return address (SRA) is the subroutine return stack pointer, which is automatically incremented or decremented as subroutines are called and returns are made from the subroutine.



3987

Figure 2-1 Control Unit

2.3 S-PAD UNIT

This unit, illustrated by Figure 2-2, performs the integer address indexing, loop counting, and control functions necessary to direct completion of a given algorithm. In form, it is similar to familiar minicomputers such as the PDP-11 and Nova.

The s-pad contains sixteen 16-bit directly-addressable registers. The contents of these registers pass through a special integer ALU associated with this unit.

The output of the ALU may be directed back to the specified s-pad destination register and also may be directed to any of the following address memory registers: memory address (MA), table memory address (TMA), or data pad address (DPA).

The s-pad integer ALU functions include the following:

<u>function</u>	<u>effect</u>
move	S \rightarrow D S-source register
logical complement	S \rightarrow D D-destination register
clear	0 \rightarrow D
increment	S+1 \rightarrow D
decrement	S-1 \rightarrow D
add	D+S \rightarrow D
subtract	D-S \rightarrow D
logical AND	D AND S \rightarrow D
logical OR	D OR S \rightarrow D
logical equivalence	D EQV S \rightarrow D

The output of the s-pad ALU (called S-PAD FUNCTION or SPFN) may be used unmodified, shifted left once, shifted right once, or shifted right twice.

A hardware bit-reverse function included in the s-pad accomplishes the bit swapping necessary to access data in scrambled order after an FFT.

The s-pad ALU also sets three condition bits in the AP status register depending upon the output of the ALU/shifter:

- N: set if result <0; cleared otherwise
- Z: set if result =0; cleared otherwise
- C: set if a carry occurred; cleared otherwise

These bits may be tested by the next AP instruction, and a branch made, depending upon whether the specified condition is true.

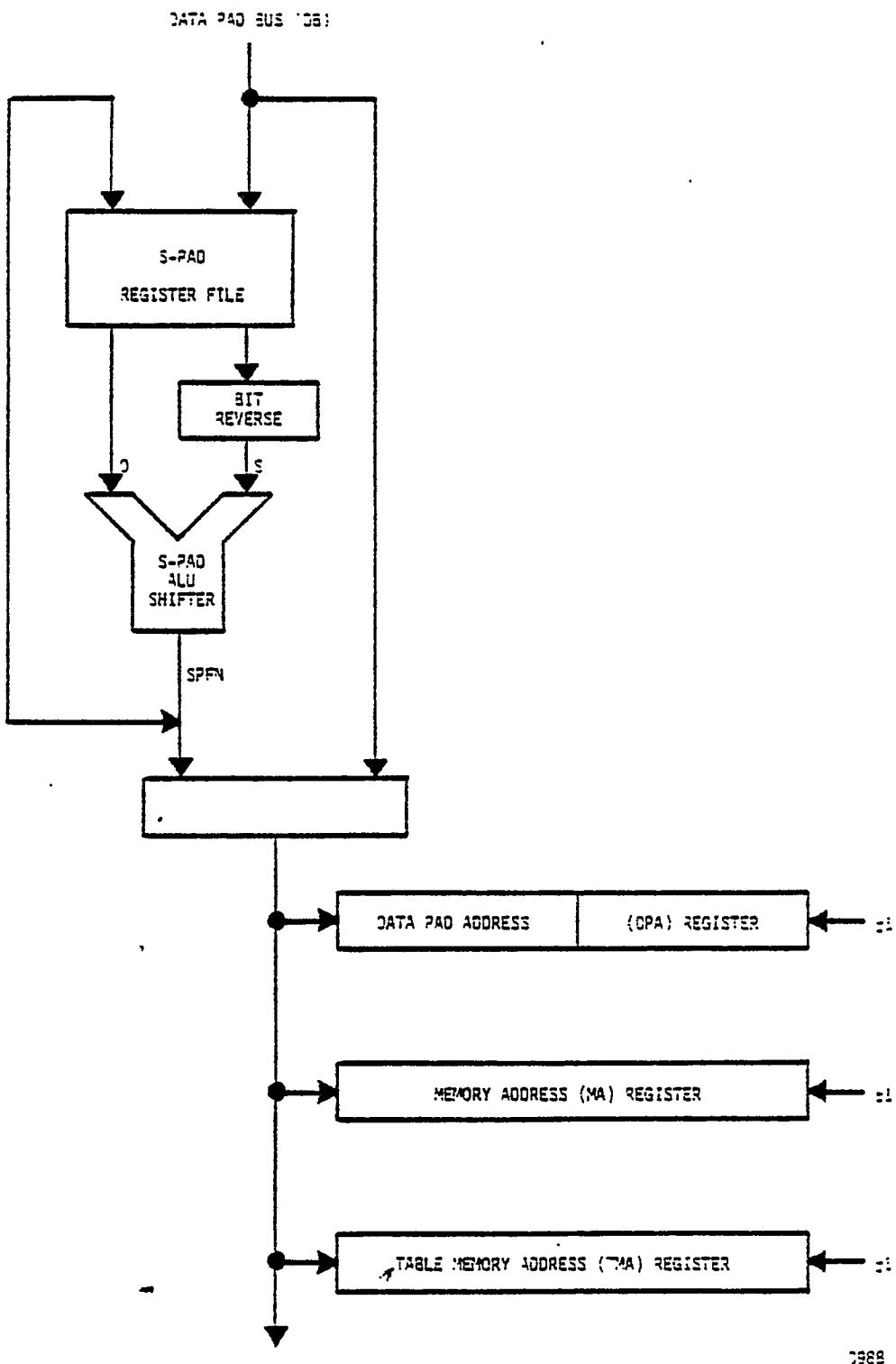


Figure 2-2 S-Pad Unit

2.4 FLOATING-POINT ADDER UNIT

The floating-point adder, shown in Figure 2-3, performs addition or subtraction operations on the contents of the adder input registers (A1 and A2). The operation is performed in two stages, each of which takes one machine cycle.

In the first stage, the exponents of the two numbers are compared and the fractions are aligned by shifting the fraction of the smaller number right. The fractions are then added or subtracted. In the second stage, the resultant fraction is normalized and convergently rounded.

Since the two stages are independent of each other, a new pair of numbers can be entered into A1 and A2 every AF cycle (167ns). The result is available for use two cycles later (333ns).

In effect, the floating adder (FA) is a pipeline where new inputs can be entered into the pipeline stream every cycle. Initiation of an add operation loads the two numbers to be added into the A1 and A2 input registers. The previous adder input is pushed down the pipeline to the adder buffer register. One cycle later, the completed result (called FA) from the buffer is available for storage or use by another unit. Thus, a new add can be started every 167ns, and the result is ready 333ns later.

A1 may be loaded from data pad (DP), from the output of the floating multiplier (FM), or from table memory (TM). A2 may be loaded from data pad (DP), from the output of the floating adder (FA), or from main data memory (MD).

The output of the floating adder (FA) may be directed to the multiplier (M2), to the adder (A2), to data pad (DP), or to memory input (MI).

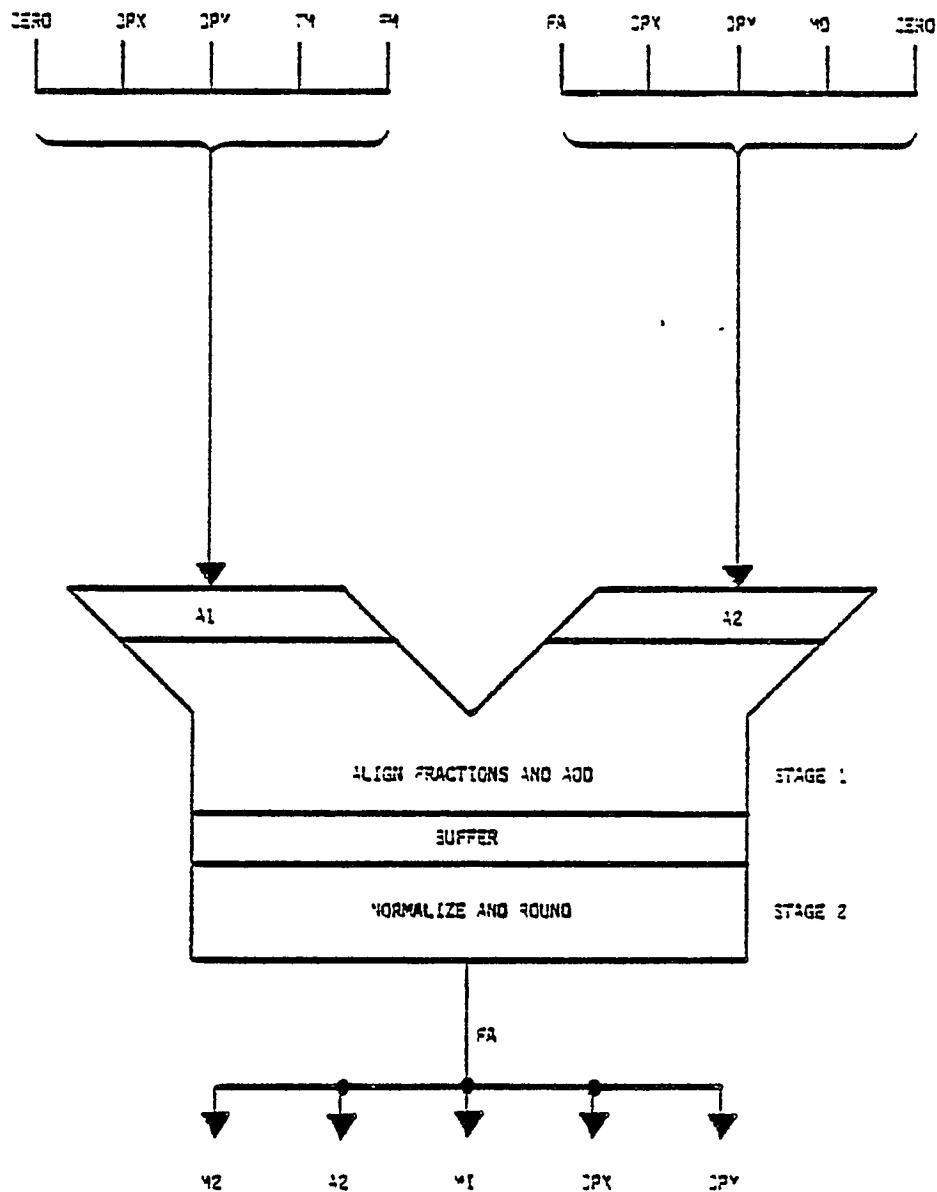
The operations performed by the floating adder are:

- $A_1 + A_2$
- $A_1 - A_2$
- $A_2 - A_1$
- $A_1 \text{ EQV } A_2$
- $A_1 \text{ AND } A_2$
- $A_1 \text{ OR } A_2$
- convert A_2 from signed magnitude to 2's complement format
- convert A_2 from 2's complement to signed magnitude format
- scale A_2
- absolute value of A_2
- fix A_2

Four condition bits in the AP status register are set or cleared by the floating adder depending upon the current result:

FZ	Set to one if result is zero, else cleared to zero.
FN	Set to one if result is negative, else cleared to zero.
FO	Set to one if exponent overflow occurred. The result is forced to the signed maximum value.
FU	Set to one if exponent underflow occurred. The result is forced to zero.

The overflow and underflow bits remain set until cleared by the program. These bits may be tested by the instruction after the floating adder result is completed (i.e., three cycles after the floating adder operation is initiated).



1983

Figure 2-3 Floating-Point Adder Unit

2.5 FLOATING-POINT MULTIPLIER UNIT

The floating multiplier, as illustrated in Figure 2-4, forms the product of the two multiplier input registers (M1 and M2). The product is formed in three stages, each of which takes one machine cycle.

In the first stage, the 56-bit product of the two 28-bit fractions are partially completed. The second stage completes the product of the fractions. In the third and final stage, the exponents are added, and the mantissa product is normalized and convergently rounded.

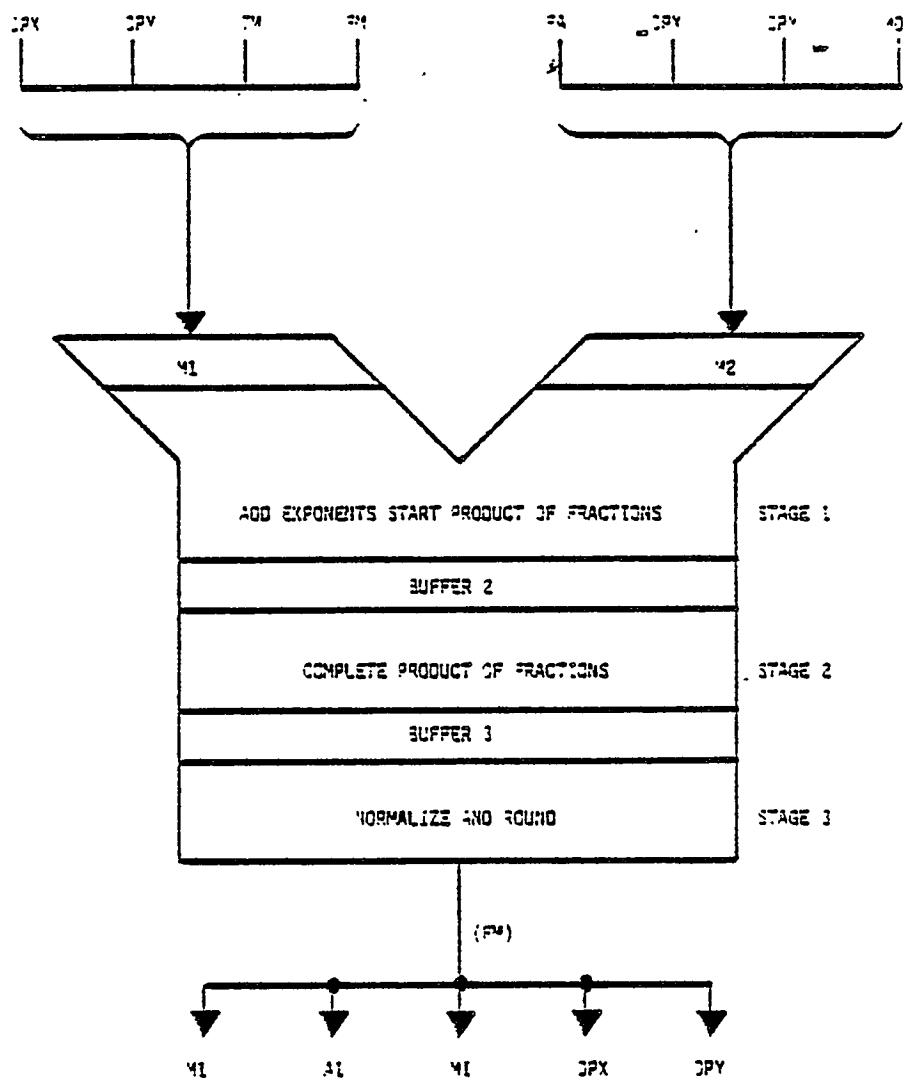
The floating multiplier, like the floating adder, is organized like a pipeline. Initiation of a multiply loads the two numbers to be multiplied into the M1 and M2 input registers. The two previous multiplier inputs are pushed down the pipeline to buffer 2 and buffer 3, respectively. One cycle later, the result from buffer 3 is available for storage or use by another unit.

Thus, a new product can be started every 167ns, and the result is ready 500ns later.

M1 can be loaded from data pad (DPX or DPY), from the output of the floating multiplier (FM), or from table memory (TM). M2 is loaded from data pad (DPX or DPY), from the adder (A1), from the multiplier (M1), or from the main data memory (MD).

Two error bits in the AP status register are affected by the floating multiplier:

FO	Set if exponent overflow occurred. The result is forced to the signed maximum value.
FU	Set if exponent underflow occurred. The result is forced to zero.



2399

Figure 2-4 Floating Multiplier

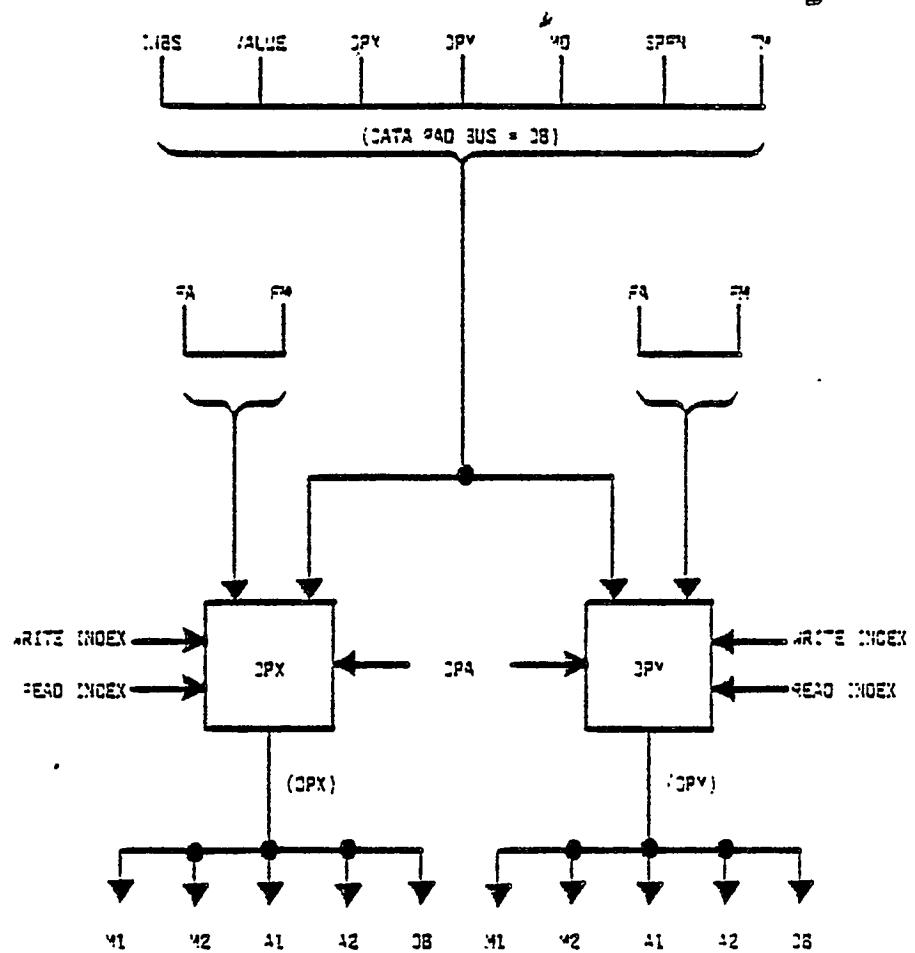
2.6 DATA PAD UNIT

Data pad, illustrated in Figure 2-5, consists of two fast accumulator blocks (each with 32 floating-point locations) called data pad X (DPX) and data pad Y (DPY). In a single-machine cycle, the contents of one location from each data pad can be read out and used. In addition, data can also be stored into one location in each data pad in the same cycle. For example, in a single instruction (167ns), a multiply can be initiated specifying one argument from DPX and another from DPY; an adder result (FA) can be stored into a DPX location, and a data element in main data stored into a DPY location. On the very next instruction, similar multiple data pad accessing could be accomplished again.

The two memories are addressed via a combination of the data pad address (DPA) register and four index field values contained in a given instruction word. DPA can be thought of as a base address register or stack pointer. It can be loaded from the s-pad (SPFN) or its contents can be incremented or decremented by one.

For a given read or write operation (i.e., reading from data pad X) an index value contained in the instruction is added to the current contents of DPA to give the effective address for that particular operation. The four index fields (one each for read DPX, read DPY, write DPX, and write DPY) are each three bits wide and have a range from -4 to +3 relative to DPA.

Data from either data pad can be used by the multiplier (M1, M2), adder (A1, A2), or memory input (MI). Data can be stored into data pad from the adder (FA), multiplier (FM), s-pad function output (SPFN), command buffer value (VALUE), or from data pad (DP).



0391

Figure 2-5 Data Pad

2.7 DATA MEMORY UNIT

The data memory unit, as illustrated in Figure 2-6, is the primary data store for the AP. It is available in 38-bit wide 8K modules which have an interleaved cycle time of 333ns (for the standard memory) and 167ns (for the fast memory).

The memory unit contains a main data memory (MD) buffer and a memory input (MI) buffer. Data read from memory is placed by the controller into MD, while data is written into memory from the MI. The memory address (MA) register points to the desired memory location.

In referencing memory for read or write operations, the selected operation is initiated by making a change to the memory address (MA) register. The MA register can be loaded from the s-pad (SPFN) or its contents incremented or decremented by one.

A write operation is specified by loading MI with the data to be written during the same instruction in which MA is changed. This data is then written into memory from MI during the next two AP cycles. Data can be loaded into MI from the floating adder (FA), floating multiplier (FM), data pad (DP), main data memory (MD), table memory (TM), the input bus (INBS), s-pad function (SPFN), or the command buffer value (VALUE). A memory operation can be initiated every other cycle. The intervening cycle can be used for any other AP function except another memory initiate.

When a memory read is initiated, the requested memory data is placed by the memory controller into the main data memory (MD) register three cycles after the request is made. Two instructions after the read request, another memory operation can be initiated. Again, the intervening cycle can be used for any non-memory function. Data in MD can be used by the floating adder (A2), floating multiplier (M2), or data pad (DP).

To optimize the operation of the AP, it is necessary for the programmer to look ahead and initiate memory reads prior to the actual time that arguments from data memory are used in a calculation.

The system provides a memory lock-out which serves to ensure that erroneous reads and writes of memory do not occur. If a memory initiate occurs while memory is busy, further program execution is halted until the previous memory cycle is completed.

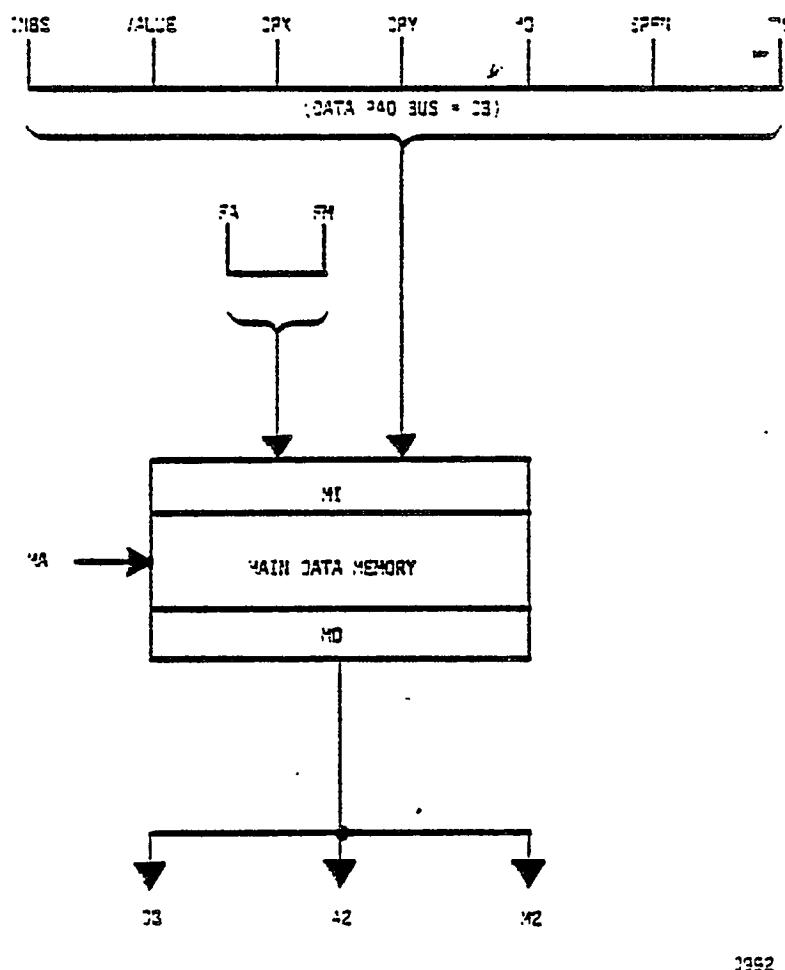


Figure 2-6 Data Memory Unit

2.8 TABLE MEMORY UNIT

The repeated use of standard constants (such as complex roots of unity and transcendental values) in signal processing routines dictates their ready availability to the programmer. A separate table memory, as illustrated in Figure 2-7, eliminates memory accessing conflicts by allowing data values (constants) to be placed in separate memory banks.

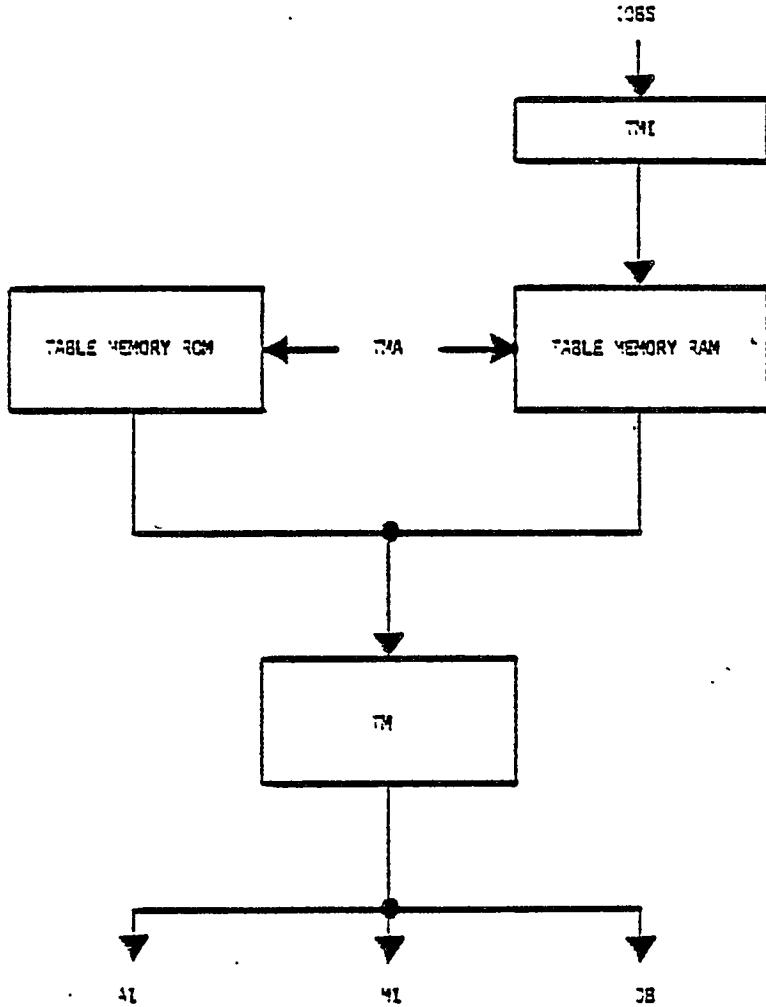
Values read from table memory are placed by the controller into the table memory buffer register. The table memory address (TMA) register serves as a pointer to the desired location.

A table memory read is initiated by changing the contents of TMA either by loading a value from the s-pad (SPFN) or by incrementing or decrementing the contents of TMA.

A new table value may be requested every machine cycle. This value is available for use two cycles later. The value can be used by the floating adder (A1), floating multiplier (M1), or data pad (DP).

In FFT mode (i.e., when FFT is being computed), the address in TMA is interpreted by the hardware to be an angle which points to the appropriate root of unity for a particular step in the algorithm. This allows the full table of roots of unity to be compressed into a single quadrant of cosines.

Refer to Programmer's Reference Manual Part One (FPS 860-7319-000) for information on TMRAM.



3293

Figure 2-7 Table Memory

2.9 INTERNAL FLOATING-POINT FORMAT

Floating-point data internal to the AP is represented as follows:

EXONENT	MANTISSA
0	1.0
52	29 M0
	2994

where:

mantissa 28-bit 2's complement fraction

exponent 10-bit binary exponent, biased by 512

The value of a floating-point number in this format is defined as:

$$\text{mantissa} * 2^{(\text{exponent} - 512)}$$

The dynamic range of this format is from $0.5 * 2^{-512}$ to $(1-2^{-28}) * 2^{511}$, or from 3.7×10^{-155} to 6.7×10^{153} .

The 28-bit fraction, combined with the convergent rounding algorithm used in the floating adder and multiplier, gives a maximum relative error of 7.5×10^{-9} per arithmetic operation. This is a precision of 8.1 decimal digits. As a comparison, unrounded IBM 360 format gives only 6.0 decimal digits of arithmetic accuracy.

The convergent rounding hardware rounds up when the magnitude of the remainder is greater than one-half of the least significant bit of the mantissa. This serves to minimize truncation errors in long series of arithmetic calculations.

Format conversion between host format and AP format occurs in the interface and in the floating adder unit. The dynamic range of the internal format is large enough to accommodate IBM 360 format and other host formats. The extended precision of the AP internal format ensures that accuracy is maintained during critical stages of data analysis.

CHAPTER 3

PROGRAMMING CONSIDERATIONS

3.1 INTRODUCTION

This chapter provides an introduction to programming the AP. The principal operations which control each of the six functional units are described below. A complete listing of the AP instruction word fields can be found in Appendix B.

In the coding examples, a semi-colon (;) is used to separate operations within a complete instruction word. A comma (,) separates operands. A quote mark ("") is used to denote a comment. A less than sign (<) is used to mean " \leftarrow " (replaced by) where the operation involved is a data transfer.

3.2 FLOATING-POINT ADDER

The following sections describe the floating-point adder.

3.2.1 FLOATING ADDER OPERATIONS

Floating adder operations are initiated by the following instructions:

<u>instruction</u>	<u>operands</u>	<u>operations initiated</u>
FADD	A1,A2	A1+A2
FSUB	A1,A2	A1-A2
FSUBR	A1,A2	A2-A1
FAND	A1,A2	A1 AND A2
FOR	A1,A2	A1 OR A2
FEQV	A1,A2	A1 EQV A2
FABS	A2	ABS(A2)
FIX	A2	Convert A2, floating-point number to fixed integer.
FSM2C	A2	Convert A2, signed magnitude to 2's complement.
F2CSM	A2	Convert A2, 2's complement to signed magnitude.
FSCALE	A2	Scale A2.

where A1 and A2 are any of the following data sources:

A1:	FM	floating multiplier result
	DPX	data pad X accumulator
	DPY	data pad Y accumulator
	TM	last data read from table memory
	ZERO	floating-point zero
A2:	FA	floating adder result
	DPX	
	DPY	
	MD	last data read from data memory
	ZERO	

Any data source listed under A1 may be combined with any data source listed under A2. For example, to add a number from data pad X to another from data pad Y:

FADD DPX, DPY "DPX+DPY

To subtract a number read out of data memory from a constant in table memory:

FSUB TM,MD "TM-MD

A reverse subtract changes the order of the subtraction; i.e.,

FSUBR TM,MD "MD-TM

subtracts a constant from table memory from a number in data memory.

To negate a number from DPX:

FSUB ZERO, DPX "0.0 - DPX = -DPX

To take the absolute value of a number from data memory:

FABS MD "ABS (MD)

To fix (convert from floating-point to integer format) a number from DPY:

FIX DPY "FIX (DPY)

3.2.2 ADDER PIPELINE

The floating adder is a two-stage pipeline. A FADD instruction loads the designated operands into the A1 and A2 registers. The previous contents of A1 and A2 are pushed down the pipeline to the buffer register. One AP cycle later, the new contents of the buffer have been normalized and rounded and are then available for use or storage elsewhere.

Example 1 illustrates how the adder pipeline works, where A,B,...,G,H are floating-point numbers to be added.

Example 1

TIME	CYCLE	INSTRUCTION	ADDER PIPELINE		ADDER RESULT (FS)
			A1,A2	SUMMER	
3	1.	FADD A,B	A,B	--	--
137ns	2.	FADD C,D	C,D	A,B	--
333ns	3.	FADD E,F	E,F	C,D	A+B
500ns	4.	FADD G,H	G,H	E,F	C+D
667ns	5.	FADD	--	G,H	E+F
833ns	6.	--	--	G,H	C+D

1295

The FADD without arguments in cycle 5 is used only to push the last computation into the buffer register and hence to the end of the pipeline. Thus, it is a dummy add because the results are unimportant and are never used. In Example 1, the floating-point adds are completed in one microsecond. During cycles 2 through 4, when the pipeline is full, adds are done every 167ns, the maximum rate. The completed results as they come out of the adder pipeline are referred to by the mnemonic FA. FA is dynamic in the sense that it must be used or stored elsewhere before being changed by the next floating adder instruction. The programmer, however, has complete control over the pipeline. Arguments advance only when pushed through the pipeline by floating adder instructions.

3.2.3 AN EXAMPLE

A complete computational sequence to do the vector sum $A_i = A_i + B_i$, $i=0,1,2,3$, is shown in Example 2. A_i is stored in data pad X locations 0-3, and B_i is stored in data pad Y location 0 through 3.

Example 2

- | | |
|----------------------------------|---|
| 1. FADD DPX(0),DPY(0) | "Do $A_0 + B_0$ |
| 2. FADD DPX(1),DPY(1) | "Do $A_1 + B_1$ |
| 3. FADD DPX(2),DPY(2); DPX(0)<FA | "Do $A_2 + B_2$, $A_0 + B_0$ is now done, save it in A_0 |
| 4. FADD DPX(3),DPY(3); DPX(1)<FA | "Do $A_3 + B_3$, $A_1 + B_1$ is now done, save it in A_1 |
| 5. FADD; DPX(2)<FA | "Push Adder; save $A_2 + B_2$ in A_2 |
| 6. DPX(3)<FA | "Save $A_3 + B_3$ in A_3 |

Example 3 is a chart of this computation showing the state of the adder pipeline and data pad after each instruction is executed.

Example 3

CYCLE	ADDER PIPELINE		ADDER RESULT	DATA PAD			
	A1,A2	SUFER		0	1	2	3
1.	A ₀ ,B ₀	--	--	A ₀	A ₁	A ₂	A ₃
2.	A ₁ ,B ₁	A ₀ ,B ₀	--	A ₁	A ₁	A ₂	A ₂
3.	A ₂ ,B ₂	A ₁ ,B ₁	A ₀ +B ₀	A ₂ +B ₀	A ₁	A ₂	A ₂
4.	A ₃ ,B ₃	A ₂ ,B ₂	A ₁ +B ₁	A ₃ +B ₀	A ₁ +B ₁	A ₂	A ₁
5.	--	A ₃ ,B ₃	A ₂ +B ₂	A ₂ +B ₀	A ₁ +B ₁	A ₂ +B ₂	A ₁
6.	--	A ₃ ,B ₃	A ₁ +B ₃	A ₂ +B ₀	A ₁ +B ₁	A ₂ +B ₂	A ₁ +B ₃

3997

3.2.4 FLOATING ADDER TESTS

Table 3-1 lists the conditional branches that test the floating adder result (FA):

Table 3-1 Floating Adder Tests

BR LOOP	"Branch unconditionally to program location "LOOP"
BFEQ LOOP	"Branch if FA=0.0
BFNE LOOP	"Branch if FA \neq 0.0
BFSZ LOOP	"Branch if FA $=$ 0.0
BFGT LOOP	"Branch if FA>0.0

1059

The branches test FA one instruction cycle after it is ready for use. That is, an adder result may be tested one cycle after it comes out of the adder pipeline. This is shown in Example 4.

Example 4

1. FSUB DPX,DPY "Do a computation"
2. FADD "Push the result out"
3. DPX<FA "Save the result"
4. BEQ LOOP "Test the result here (branch to
" location "LOOP" if result was
" zero)"

0998

Compound tests may also be made. Test MD to see if it is between a lower limit contained in DPX (1) and an upper limit in DPX (2) (i.e., see if $DPX(1) \leq MD \leq DPX(2)$). This is shown in Example 5.

Example 5

1. FSUBR DPX(2),MD "Do MD-DPX(2)"
2. FSUB DPX(1),MD "Do DPX(1)-MD"
3. FADD "Push first test result out"
4. BGT BIG "Was too big"
5. BGT SMALL "Was too small"
6. . . . "OK"

0999

The branches are made relative to the current program source address (PSA) with a 5-bit displacement value. This means that the conditional branch target address must be within -20₈ to +17₈ locations of the current instruction.

3.2.5 FLOATING-POINT LOGICAL OPERATIONS

Instructions FAND, FOR, and FEQV perform logical operations on floating-point numbers. Exponent alignment occurs as for a normal floating-point add. The two mantissas are then combined using the specified logical operation. The result is then normalized and rounded.

3.3 FLOATING-POINT MULTIPLIER

The following sections describe the floating-point multiplier.

3.3.1 MULTIPLY INSTRUCTION

Floating-point multiplies are initiated by the following instruction:

FMUL M1,M2

which initiates a multiply between M1 and M2, where M1 and M2 are any of the following data sources:

M1	FM	floating multiplier result
	DPX	data pad X accumulator
	DPY	data pad Y accumulator
	TM	last data read from table memory
M2	FA	floating adder result
	DPX	
	DPY	
	MD	last data read from data memory

Thus, any of the data sources listed under M1 can be multiplied by any of the data sources in M2. For example, to multiply a number read from data memory by a constant from table memory:

FMUL TM,MD "TM * MD

or, to multiply a number in data pad X by another number in data pad Y:

FMUL DPX,DPY "DPX * DPY

3.3.2 MULTIPLIER PIPELINE

The floating multiplier is a three-stage pipeline. An FMUL instruction loads the specified operands into the M1 and M2 registers. The two previous partially-completed products are pushed down the pipeline to buffer 2 and buffer 3, respectively. One AP cycle later, the new contents of buffer 3 have been normalized and rounded and are then available for use or storage elsewhere.

The instruction sequence shown in Example 6 illustrates how the multiplier pipeline works where A,B,...,G,H are floating-point numbers to be multiplied together.

Example 6

TIME	CYCLE	INSTRUCTION	MULTIPLIER PIPELINE			MULTIPLIER RESULT (FM)
			M1,M2	S BUFFER 2	BUFFER 3	
0	1.	FMUL A,B	A,B	--	--	--
167ns	2.	FMUL C,D	C,D	A,B	--	--
333ns	3.	FMUL E,F	E,F	C,D	A,B	--
500ns	4.	FMUL G,H	G,H	E,F	C,D	A*B
667ns	5.	FMUL	--	G,H	E,F	C*D
833ns	6.	FMUL	--	--	G,H	E**F
1.0us	7.	--	--	--	G,H	E**H

1300

The FMUL in cycles 5 and 6 are dummy multiplies used to push the last two computations to the end of the pipeline. In Example 6, four floating-point multiplies in 1.0us are completed. During cycles 3 and 4, while the pipeline is full, products are done every 167ns, the maximum rate.

The completed products as they come out of the multiplier pipeline are referred to by the mnemonic FM. FM is dynamic in that it must be used or stored before being changed by the next FMUL instruction.

3.3.3 AN EXAMPLE

A computation example to square the elements in a vector is shown in Example 7.

Example 7

$A_i = A_i \cdot A_i$, $i=0,1,2,3$. A_i is stored in Data Pad X.

1. FMUL CPX(0),CPX(0) 'Do A_0^2
2. FMUL CPX(1),CPX(1) 'Do A_1^2
3. FMUL CPX(2),CPX(2) 'Do A_2^2
4. FMUL CPX(3),CPX(3); CPX(3)<FM 'Do A_3^2 , save A_3^2
5. FMUL; CPX(1)<FM 'Save A_1^2
6. FMUL; CPX(2)<FM 'Save A_2^2
7. CPX(3)<FM 'Save A_3^2

1301

Example 8 illustrates this computation showing the state of the multiplier pipeline and data pad X after each instruction is executed.

Example 8

CYCLE	MULTIPLIER PIPELINE			MULTIPLIER RESULT (FM)	DATA PAD X			
	M1,M2	SUFFER 2	SUFFER 3		0	1	2	3
1.	A ₀ ,A ₀	--	--	--	A ₀	A ₁	A ₂	A ₃
2.	A ₁ ,A ₁	A ₀ ,A ₀	--	--	A ₀	A ₁	A ₂	A ₃
3.	A ₂ ,A ₂	A ₁ ,A ₁	A ₀ ,A ₀	--	A ₀	A ₁	A ₂	A ₃
4.	A ₃ ,A ₃	A ₂ ,A ₂	A ₁ ,A ₁	A ₀ ²	A ₀ ²	A ₁ ²	A ₂ ²	A ₃ ²
5.	--	A ₂ ,A ₃	A ₂ ,A ₂	A ₁ ²	A ₂ ²	A ₂ ²	A ₂ ²	A ₂ ²
6.	--	--	A ₃ ,A ₃	A ₂ ²	A ₃ ²	A ₃ ²	A ₃ ²	A ₃ ²
7.	--	--	A ₃ ,A ₃	A ₃ ²				

1002

3.3.4 MULTIPLY-ADDS

The full floating-point computational power of the AP is utilized when a process involving both multiplies and adds is considered. The dot product of two eight-element vectors $A_i \cdot B_i = \sum A_i B_i$, $i = -4, -3, \dots, 1, 2, 3$, where A_i is in Data Pad X and B_i is in Data Pad Y, is formed in Example 9.

<u>Example 9</u>		
Fill the Multiplier Pipeline	1. FMUL JPY(-4),JPY(-4) 2. FMUL JPY(-3),JPY(-3) 3. FMUL JPY(-2),JPY(-2) 4. FMUL JPY(-1),JPY(-1); FA00 FM,ZERO	'Co A- ₁ B- ₁ . 'Co A- ₂ B- ₂ . 'Co A- ₃ B- ₃ . 'Co A- ₄ B- ₄ . A- ₁ B- ₁ is now done, save it in the adder. 'Co A- ₂ B- ₂ . A- ₁ B- ₁ is now done, save it in the adder.
Fill the Adder Pipeline	5. FMUL JPY(0),JPY(0); FA00 FM,ZERO	'Co A- ₅ B- ₅ . A- ₁ B- ₁ is now coming out of the multiplier, and A- ₂ B- ₂ from the adder, add them together.
Both pipelines full	6. FMUL JPY(1),JPY(1); FA00 FM,FA	'Co A- ₆ B- ₆ . A- ₁ B- ₁ is now coming out of the multiplier, and A- ₂ B- ₂ from the adder, add them together.
Empty the Multiplier Pipeline	7. FMUL JPY(2),JPY(2); FA00 FM,FA	'Co A- ₇ B- ₇ . A- ₃ B- ₃ is now coming out of the multiplier, and (A- ₁ B- ₁ + A- ₂ B- ₂) from the adder, add them together.
	8. FMUL; FA00 FM,FA	'Co A- ₈ B- ₈ . A- ₃ B- ₃ is coming out of the multiplier, and (A- ₁ B- ₁ + A- ₂ B- ₂) from the adder, add them together.
Empty the Adder Pipeline	9. FMUL; FA00 FM,FA	'Co A- ₉ B- ₉ is coming out of the multiplier, and (A- ₁ B- ₁ + A- ₂ B- ₂ + A- ₃ B- ₃) from the adder, add them together.
	10. FA00 FM,FA	'Co A- ₁₀ B- ₁₀ is coming out of the multiplier, and (A- ₁ B- ₁ + A- ₂ B- ₂ + A- ₃ B- ₃ + A- ₄ B- ₄) from the adder, add them together.
Empty the Adder Pipeline	11. FA00 FM,FA	'Co A- ₁₁ B- ₁₁ is coming out of the multiplier, and (A- ₁ B- ₁ + A- ₂ B- ₂ + A- ₃ B- ₃ + A- ₄ B- ₄ + A- ₅ B- ₅) from the adder, add them together.
	12. FA00; JPY(3)<#4	'Co A- ₁₂ B- ₁₂ is coming out of the adder, save it in JPY(3).
Empty the Adder Pipeline	13. FA00 JPY(3),FA	'Co A- ₁₃ B- ₁₃ is coming out of the adder, add it to (A- ₁ B- ₁ + A- ₂ B- ₂ + A- ₃ B- ₃ + A- ₄ B- ₄ + A- ₅ B- ₅) which was saved in JPY(3).
	14. FA00	'Push result out of Adder
	15. JPY(3)<#4	The result: (A- ₁ B- ₁ + A- ₂ B- ₂ + A- ₃ B- ₃ + A- ₄ B- ₄ + A- ₅ B- ₅) + A- ₁₃ B- ₁₃ , saved in JPY(3).

In accumulating the sum-of-products, the even term sum is kept in one-half of the adder pipeline and the odd term sum in the other half. During cycles 5 through 7 when both pipelines are full, floating-point multiply adds are computed every 167ns. This is 12 million floating-point computations per second. A longer sum of products calculation involving more terms would maintain this maximum computation rate, because nearly all of the time was spent filling and emptying pipelines. Even so, the seven adds and eight multiplies take 15 cycles (2.5us) to complete (an overall rate of 333ns per floating-point multiply add).

Example 10 summarizes the computation as a further aid in understanding the multiply add interaction in the sum-of-products computation of Example 9.

Example 10

CYCLE	MULTIPLIER		COPPER:		DATA 240: 3
	A1, 42	B1	A1, 42	B2	
1.	A_{-4}, B_{-4}	---	---	---	---
2.	A_{-3}, B_{-3}	---	---	---	---
3.	A_{-2}, B_{-2}	---	---	---	---
4.	A_{-1}, B_{-1}	$A_{-2} \cdot B_{-2}$	$A_{-2} B_{-2}, B_{-2}$	---	---
5.	A_2, B_2	$A_{-1} \cdot B_{-1}$	$A_{-1} B_{-1}, B_{-1}$	---	---
6.	A_1, B_1	$A_{-2} \cdot B_{-2}$	$A_{-2} B_{-2}, A_{-1} B_{-1}$	$A_{-1} B_{-1}$	---
7.	A_2, B_2	$A_{-1} \cdot A_{-1}$	$A_{-1} B_{-1}, A_{-1} B_{-1}$	$A_{-1} B_{-1}$	---
8.	A_3, B_3	$A_2 \cdot A_2$	$A_2 B_2, ES_2$	ES_2	---
9.	---	$A_1 \cdot A_1$	$A_1 B_1, OS_2$	OS_2	---
10.	---	$A_2 \cdot A_2$	$A_2 B_2, ES_3$	ES_3	---
11.	---	$A_3 \cdot A_3$	$A_3 B_3, OS_3$	OS_3	---
12.	---	---	---	ES_4	ES_4
13.	---	---	OS_4, ES_4	OS_4	ES_4
14.	---	---	---	---	ES_4
15.	---	---	---	$OS_4 + ES_4$	$OS_4 - ES_4$

NOTE

ES is n terms of the even term sum: $A_i B_j, i = -4, -2, 0, 2$
 OS is n terms of the odd term sum: $A_i B_j, i = -1, -1, 1, 3$

3.4 DATA PAD

The following sections describe the data pad.

3.4.1 DATA PAD ADDRESSING

Data pad is a block of 64 high-speed accumulators used to store intermediate results during a computation. In any given AP instruction, the programmer has 16 of the data pad accumulators to work with; eight in data pad X and eight in data pad Y. They are addressed relative to the current value of the data pad address register which functions as a base register for data pad. For example, if DPA has a value of 24_8 , locations 20_8 through 27_8 would be available for use. This is illustrated in Figure 3-1.

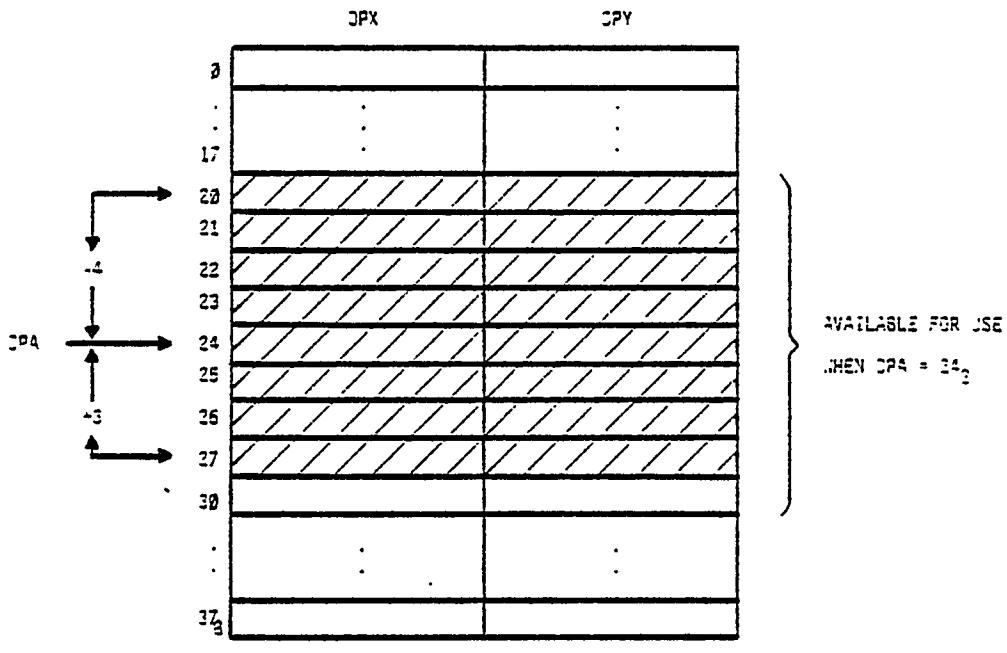


Figure 3-1 Data Pad Address

A displacement value from -4 to +3 may be specified when using DPX and DPY (i.e., if $DPA=24_8$):

DPX(3)	means DPX location $24+3=27$
DPY(-4)	means DPY location $24-4=20$
DPX(0)	means DPX location $24+0=24$
DPY	means DPY location $24+0=24$

Four separate displacements are provided, one each for reading and writing DPX and DPY. Thus, four separate locations in data pad may be used in a given instruction. With DPA=24g, the following instruction occurs in one cycle:

FADD DPX(3),MD; FMUL TM,DPY(-2); DPX(-3)<FA; DPY(1)<FM
(read DPX) (read DPY) (write DPX) (write DPY)

This would add DPX location 27 to the last data read from data memory, multiply the last data read from table memory by the contents of DPY location 22, store the results of a previous add into DPX location 21, and store the results of a previous multiply into DPY location 25.

All 64 locations of data pad are accessed by changing the DPA pointer:

INCDPA	"Increments DPA by 1
DEC DPA	"Decrements DPA by 1
SETDPA	"Loads DPA with the current S-PAD "function (SPFN, refer to section 3.7)

Changes in DPA take effect the next instruction after they occur (i.e., if DPA=24):

FADD DPX(0),DPY(0);	INCDPA	"DPA is still 24 so "DPX ₂₄ is added to "DPY ₂₄
FADD DPX(0),DPY(0);	INCDPA	"Now DPA=25, so "DPX ₂₅ is added to DPY ₂₅
FADD DPX(0),DPY(0)		"Now DPA=26, so "DPX ₂₆ is added to "DPY ₂₆

Thus, by successively incrementing DPA, the data pad can be used as a queue; or by properly incrementing and decrementing DPA, the data pad can be used as a stack. Data pad address is circular. That is, with successive increments of DPA the next location after 37g is 0; with successive decrements of DPA the next location after 0 is 37g.

3.4.2 WRITING INTO DATA PAD

Data may be stored into DPX and DPY from FA, FM, or DB (the data pad bus).

and

DPX<FA	"Store adder result into DPX
DPX<FM	"Store multiplier result into DPX
DPX<DB	"Store data pad bus into DPX

DPY<FA	"Store into DPY
DPY<FM	
DPY<DB	

The following may be selected onto the data pad bus (DB):

DB=ZERO	"Floating-point zero
DB=INBS	"Input Bus
DB=VALUE	"A 16-bit immediate value
DB=DPX	"DPX
DB=DPY	"DPY
DB=MD	"Last data read from data memory
DB=SPFN	"S-pad function (16-bit integer)
DB=TM	"Last data read from table memory

Thus, if DPA=248, the following instruction is possible:

DPX(3)<FA; DPY(-2)<DB; DB=MD

This stores the current adder result into DPX location 27 and stores the last data read from the main data memory into DPY location 22 via the data pad bus.

3.4.3 DATA PAD BUS

Data to be stored into DPX and DPY can be moved through three pathways: FM, FA, and DB. While FM and FA are fixed in meaning (output from the floating multiplier and adder, respectively), the data pad bus (DB) pathway can be connected to any one of eight possibilities depending upon the programmer's choice.

Examples:

- MD is put into both DPX and DPY:

DPX<DB; DPY<DB; DB=MD

MD is put onto the data pad bus, and the data pad bus is stored into DPX and DPY.

- MD is put into DPX and TM into DPY:

DPX<DB; DB=MD; DPY<DB; DB=TM

This is an error. Only one choice at a time can be made for the data pad bus. This double transfer would take two separate instructions to accomplish.

- FA is stored into DPX and MD into DPY:

DPX<FA; DPY<DB; DB=MD

MD is put onto the data pad bus in order to get it into DPY. FA goes directly into DPX.

To simplify notation, data transfers involving data pad bus can be written in a shorthand manner.

shorthand

DPX<MD; DPY<MD

DPX<MD; DPY<TM

(still an error no matter how it is written)

DPX<FA; DPY<MD

longhand

DPX<DB; DPY<DB; DB=MD

DPX, DB; DB=MD; DPY, DB; DB=TM

DPX<FA; DPY<DB; DB=MD

In the shorthand notation, choices for the data pad bus are not explicitly indicated. Transfers are written as if there were a direct connection between the source and destination while in fact it is the data pad bus which does the connecting. Remember, however, that the programmer is still making a data pad bus choice and only one choice is allowed per instruction. Errors like the one shown above (where two data pad bus choices are attempted) are detected and flagged by the assembler.

3.5 DATA MEMORY

The following sections describe data memory.

3.5.1 MEMORY ADDRESSING

Main data memory cycles are initiated by changing the memory address register which points the memory location to be read from or written into:

INCMA	"Increment MA by 1
DECMA	"Decrement MA by 1
SETMA	"function (SPFN, refer to section 3.7)

All of the above initiate a memory cycle at the address pointed at by the new contents of MA. If a memory input (MI) field is also included in the instruction, then the memory cycle is a write cycle. Otherwise, a read cycle is initiated. When sequential memory locations are accessed, a new memory cycle may be initiated by every other AP instruction.

3.5.2 DATA MEMORY READS

Data read from memory is available for use three instruction cycles after the read is initiated. The instruction sequence shown in Example 11 illustrates how memory data is accessed: A, B, and C are floating-point numbers in memory locations 101, 102, and 103, respectively. It is assumed that MA is set to 100 before starting.

Example 11

TIME	AP CYCLE	INSTRUCTION	MEMORY ADDRESS (MA)	MEMORY DATA RESULT (MD)
0	1.	INCMA	101	---
157ns	2.	---	101	---
333ns	3.	INCMA	102	---
500ns	4.	---	102	A
667ns	5.	INCMA	103	A
833ns	6.	---	103	B
1.0us	7.	---	103	B
1.17us	8.	---	103	C

1006

Three AP cycles after a given memory location is read, data from that location is ready in the memory data register and available for use. MD may be used by the adder or the multiplier as follows:

FADD DPX(3),MD; FMUL DPY(-2),MD, "Do MD+DPX and MD * DPY

It can also be placed on the data pad bus and stored in data pad or back into memory as follows:

DPX(2)<MD "store MD into DPX.

3.5.3 AN EXAMPLE

Example 12 loads a vector A_i , $i=0,1,2$ stored in memory locations 101, 102, and 103 into DPX locations 10, 11, and 12. It is assumed that MA is set to 100 and DPA is set to 10 before starting.

Example 12

1. INCMA
"Fetch A₀ from memory
 2. --
 3. INCMA
"Fetch A₁ from memory
 4. CPX<MD; INCOPA
"Store A₀ into CPX location 10
" and bump CPA counter to 11
 5. INCMA;
"Fetch A₂ from memory
 6. CPX<MD; INCOPA
"Store A₁ into CPX location 11
" and bump CPA counter to 12
 7. --
 8. CPX<MD
"Store A₂ into CPX location 12

Example 13 illustrates the transfer of Example 12 showing the state of each component after each instruction.

Example 13

CYCLE	MEMORY		DATA PAD			
	MA	MD	DPA	DPX ₁₀	DPX ₁₁	DPX ₁₂
1.	101	---	10	---	---	---
2.	101	---	10	---	---	---
3.	102	---	10	---	---	---
4.	102	A ₉	10	A ₉	---	---
5.	102	A ₉	11	A ₉	---	---
6.	103	A ₁	11	A ₉	A ₁	---
7.	103	A ₂	12	A ₉	A ₁	---
8.	103	A ₂	12	A ₉	A ₁	A ₂

DCS

3.5.4 DATA MEMORY WRITES

Data memory write cycles are indicated by the following:

MI<FA "write the adder result into memory
MI<FM "write the multiplier result into memory
MI<DB "write data pad bus into memory

These instructions load data into the memory input buffer register from where it is written into memory. Data may be written into sequential memory locations by every other AP instruction.

3.5.5 AN EXAMPLE

Example 14 squares the elements of a vector A_i , $i=0,1,2$, in DPX locations 10, 11, and 12 and stores the results into data memory locations 101, 102, and 103. It is assumed that MA is set to 100 and DPA is set to 10 before starting.

<u>Example 14</u>	1. FMUL CPX,DPX: INCOPA	'Square A_0 , bump DPA counter " to 11
	2. FMUL	"Push down the multiplier " pipeline
	3. FMUL CPX,DPX: INCOPA	'Square A_1 , bump DPA counter " to 12
	4. FMUL: M1<FM: INCMIA	"Write A_1^2 into memory location " 101
	5. FMUL CPX,DPX	'Square A_2
	6. FMUL: M1<FM: INCMIA	"Write A_2^2 into memory location 102
	7. FMUL	"Gummy FMUL to empty pipeline
	8. M1<FM: INCMIA	"Write A_2^2 into memory location 103

1009

Example 15 illustrates the sequential data memory write computation.

Example 15

CYCLE	DPA	MULTIPLIER		MEMORY	
		M1,M2	FMI	MA	MI
1.	10	A_0, A_1	--	--	--
2.	11	---	--	--	--
3.	11	A_1, A_2	--	--	--
4.	12	---	$\frac{1}{2}$	101	$\frac{1}{2}$
5.	12	A_2, A_2	--	101	$\frac{1}{2}$
6.	12	---	$\frac{1}{2}$	102	$\frac{1}{2}$
7.	12	---	--	102	$\frac{1}{2}$
8.	12	---	$\frac{1}{2}$	103	$\frac{1}{2}$

1010

3.5.6 MEMORY INTERLEAVE

Data memory is divided into 16 banks of 4K words each using MA00-MA02 and MA15 as a memory bank select. (These are the three highest-order bits and the least-significant bit of MA.) Memory references to different banks may be made every two AP cycles, while references to the same bank may be made every three AP cycles. For some possible memory addressing sequences refer to Table 3-2.

Table 3-2 Memory Interleave Sequence

MEMORY ADDRESS SEQUENCE (OCTAL)	MEMORY BANK SEQUENCE	MEMORY REFERENCE TIMING
101, 102, 103, 104, ...	1, 2, 1, 2, ...	every 2 AP cycles
156, 155, 154, 153, ...	2, 1, 2, 1, ...	every 2 AP cycles
100, 102, 104, 106, ...	2, 2, 2, 2, ...	every 3 AP cycles
233, 10374, 234, 10376, ...	1, 2, 3, 2, ...	every 2 AP cycles

1011

Thus, references to successive sequential memory locations can be made every other AP cycle, but references to successive-odd or successive-even locations must be three cycles apart.

3.5.7 MEMORY LOCKOUT

If memory references are made too rapidly for memory to handle, the CPU suspends program execution and spins until the memory is no longer busy. Thus, suppose the following were coded:

1. INCMA "referencing memory every cycle
 2. INCMA
 3. INCMA

The following execution is the result:

0ns	1. INCMA
167ns	2. INCMA
333ns	"SPIN"
500ns	3. INCMA
667ns	"SPIN"

The processor waits an extra cycle after instructions 2 and 3 because memory is still busy from the previous memory references. This arrangement is fine if there is no useful computing to do during the spin cycles. Otherwise, it is better to space out the INCMAs and to do something useful during the cycle between memory references.

3.6 TABLE MEMORY

The following sections describe table memory.

3.6.1 TABLE MEMORY ADDRESSING

Constants stored in table memory are read by setting the table memory address (TMA) register to the address of the desired table memory location. This is done with the following instructions:

INCTMA	"increments TMA by 1
DECTMA	"decrements TMA by 1
SETTMA	"set TMA to the current s-pad "function (SPFN)

Each of the above initiates a fetch from the table memory location pointed at by the new contents of TMA. Two AP cycles later, the contents of the desired locations are available for use. A new location can be fetched every AP cycle. The sequence in Example 16 illustrates how table memory is accessed. K0, K1, and K2 are constants stored in table memory location 235, 236, and 237. It is assumed that TMA is set to 234 before starting.

Example 16

TIME	AP CYCLE	INSTRUCTION	TABLE MEMORY ADDRESS (TMA)	TABLE MEMORY RESULT (TMR)
3	1.	INCTMA	235	---
167ns	2.	INCTMA	236	---
333ns	3.	INCTMA	237	K0
500ns	4.	---	237	K1
667ns	5.	---	237	K2

1012

Two cycles after a given table memory location is fetched, the data is ready in the table memory data register and is available for use. TM can be used by the adder or the multiplier:

FADD TM,DPX(2);FMUL TM,DPY(-3) , "do TM+DPX and TM*DPY

or put on the data pad bus and stored into data pad:

DPX(-1)<TM "store TM into DPX

3.6.2 AN EXAMPLE

Example 17 forms the vector sum $A_i = B_i + K_i$, $i=0,1,2$, where A_i is in DPX locations 10-12, B_i is in DPY 10-12, and K_i is a series of constants stored in table memory location 235-237. A_i is stored back into DPX. It is assumed that DPA is set to 10 and TMA is set to 234 before starting.

Example 17

- | | | |
|----|-----------------------------|--|
| 1. | FNCTMA | "Seton ζ_0 |
| 2. | FNCTMA | "Seton ζ_1 |
| 3. | FNCTMA; FADD TM,DPY; INCOPA | "Do $\zeta_0 + \beta_3$, bump CPA to 11 |
| 4. | FADD TM,DPY; INCOPA | "Do $\zeta_1 + \beta_1$, bump CPA to 12 |
| 5. | FADD TM,DPX(0); DPX(-2)<FA | "Do $\zeta_2 + \beta_2$, store A in DPX ₁₂ |
| 6. | FADD; DPS(-1)<FA | "Store A ₁ in DPX ₁₁ |
| 7. | DPX(0)<FA | "Store A ₂ in DPX ₁₂ |

13

Example 18 illustrates the computations of Example 17.

Example 18

CYCLE	TABLE MEMORY		ADDER		DATA PATH			
	TMA	TM	A1,A2	FA1	CPA	I0	I1	I2
1.	235	--	--	--	10	--	--	--
2.	236	--	--	--	10	--	--	--
3.	237	K ₀	K ₀ , B ₀	--	10	--	--	--
4.	237	K ₁	K ₁ , B ₁	--	11	--	--	--
5.	237	K ₂	K ₂ , B ₂	K ₀ +B ₀	12	A ₀	--	--
6.	237	K ₂	--	K ₁ -B ₁	12	A ₀	A ₁	--
7.	237	K ₂	--	K ₂ -B ₂	12	A ₀	A ₁	A ₂

三

3.6.3 A COMPLEX MULTIPLY

An example using both memories, a complex multiply from the FFT (fast fourier transform) algorithm, is shown in Example 19. The multiply is between a complex signal point held in data memory and a complex exponential value (a root of unity, $e^{j\theta}$), fetched from table memory. The computation is:

$$X_R = C_R * W_R - C_I * W_I$$

$$X_I = C_R * W_I + C_I W_R$$

Where C is the data point and W is the complex exponential, R and I denote real and imaginary parts, respectively. C is in main data memory, and W is in table memory.

Example 19

Fetch the	1. INCHIA	'Fetch C_R from data memory
four arguments	2. INCTMIA	'Fetch X_R from table memory
	3. INCHIA: INCTMIA	'Fetch C_I , fetch i
	4. FMUL FM,NO	'Do $C_R * X_R$
Do the	5. FMUL FM,NO: DECXTMA	'Do $C_R * X_I$, fetch i
multiplicies	6. FMUL FM,NO	'Do $C_I * X_R$
	7. FMUL FM,NO: CPX(0)<FM	'Do $C_I * X_I$, save $C_R X_R$, in CPX
	8. FMUL: CPX(1)<FM	'Save $C_R X_I$ in CPX
Do the two	9. FMUL: FEUBR FM,CPX(0)	'Do $C_R * C_R X_R - C_I X_I$
adds	10. FADD FM,CPX(1)	'Do $C_I = C_R X_I - C_I X_R$
	11. CPX(0)<FA; FADD	C_R is ready, save in CPX
	12. CPX(1)<FA	X_I is ready, save in CPX

1016

The total elapsed time is 12 cycles or 2us. In practice, however, all but cycles four through seven with the preceding and following computations can overlap. The complex multiply then takes only 667ns when mixed in with other computations.

Example 20 summarizes the complex multiply.

Example 20

CYCLE	MEMORIES		MULTIPLIER		ADDER		DATA P40	
	TM	40	M1,M2	FM	A1,A2	FA	I	O
1.	--	--	--	--	--	--	--	--
2.	--	--	--	--	--	--	--	--
3.	--	--	--	--	--	--	--	--
4.	w_R	c_R	w_R, c_R	--	--	--	--	--
5.	w_I	c_R	w_I, c_R	--	--	--	--	--
6.	w_I	c_I	w_I, c_I	--	--	--	--	--
7.	w_R	c_I	w_R, c_I	$w_R c_R$	--	--	$w_R c_I$	--
8.	--	--	--	$w_I c_R$	--	--	$w_R c_R$	$w_I c_R$
9.	--	--	--	$w_I c_I$	$w_I c_I, w_R c_R$	--	$w_R c_R$	$w_I c_R$
10.	--	--	--	$w_R c_I$	$w_R c_I, w_I c_R$	x_R	$w_R c_R$	$w_I c_R$
11.	--	--	--	--	--	x_I	x_R	$w_I c_R$
12.	--	--	--	--	--	--	x_R	x_I

1016

3.7 S-PAD

The s-pad is a 16-bit wide integer unit used primarily to compute memory address pointers and to test loop counters. It is similar in capability to a minicomputer and is programmed like the register-to-register instructions of the Nova and PDP-11 computers. There are 16 registers in the s-pad unit.

3.7.1 SINGLE OPERAND INSTRUCTIONS

Table 3-3 lists the single operand instructions. One item can be chosen from each column.

Table 3-3 Single Operand Instructions

OPERATION	SHIFT	'10 LOAD	DESTINATION REGISTER
INC	---	---	DST
DEC	2	-	
COM	4	-	
CLR	RR	-	

1017

The operation is performed upon the contents of the destination register (DST), and that result is shifted. The shifted result is stored in the destination register unless a no load (#) is specified. The shifted result is the s-pad function (SPFN), which may be stored into an address register (MA, TMA, or DPA) or placed onto the data pad bus (DB=SPFN). Some examples where SP_n refers to the contents of s-pad register "n" are illustrated in Example 21.

Example 21

INC 6	"SP ₁ -1)→SP ₁
DEC R 3	"SP ₁ -1), 2→SP ₁
COM 3; CPY<SPF1	SP ₁ -SP ₁ →CPY
CLR# 2; SETOPA	'0-OPA; because of ≠ (no load) SP ₁ remains unchanged

1019

3.7.2 DOUBLE OPERAND INSTRUCTIONS

Table 3-4 lists the double operand instructions. One item can be chosen from each column.

Table 3-4 Double Operand Instructions

OPERATION	SHIFT	NO LOAD	DECIMATE	SOURCE REGISTER	DESTINATION REGISTER
MOV	---	---	---	src,	dst,
ADD	R	*	3		
SUB	L				
AND	RR				
OR					
EQV	-				

1013

The operation is performed between the source (SRC) and destination (DST) registers. If bit reverse (X) is specified, the contents of source are bit-reversed before being used. The shift is performed on the result which is then stored into the destination register unless no load (#) is specified. The shifted result is the s-pad function (SPFN), which may be stored into TMA, MA, or DPA or placed onto the data pad bus.

Example 22

MOV 3.15	SP ₁ -SP ₂
ACCL 6.10; SETMA	((SP ₁) - (SP ₂)) * 2; SP ₁ -MA
SUB 7.13	(SP ₁ -SP ₂) SP ₁
AND 6.11; SETDPA	(SP ₁) AND SP ₂ -DPA
CR= 16.7; SETTMA	(SP ₁ OR SP ₂ (bit-reversed))-TMA
MOVRR 2.2	(SP ₁) 4-SP ₂

1020

For purposes of program clarity, the assembler allows names to be given to the s-pad registers. If register PTR is a pointer to an array in data memory, and register STEP contains the increment value used to step through the array, then the following instruction word advances the array pointer by the proper increment and fetches the next array element from memory:

ADD STEP,PTR; SETMA

3.7.3 S-PAD TEST

The following conditional branches test the s-pad function:

BR LOOP	"branch unconditionally to program location "LOOP"
BEQ LOOP	"branch if SPFN=0
BNE LOOP	"branch if SPFN \neq 0
BGE LOOP	"branch if SPFN \geq 0
BGT LOOP	"branch if SPFN>0

The above branches test the s-pad result from the immediately preceding AP instruction. Thus, an s-pad operation must be done one instruction cycle before it is desired to test the result.

An example of loop counting is shown in Example 23.

Example 23

DEC 2	"decrement SP ₂
BNE LOOP	"branch to "LOOP if SP ₂ has not yet reached zero

Example 24 tests the contents of SP₃ to see if it is between a lower limit contained in SP₂ and an upper limit in SP₄ (i.e., if SP₂<SP₃<SP₄.

Example 24

SUB # 3,2	
SUB 4,3; BGT SMALL	"Too small, SP ₃ <SP ₂
BGT BIG	"Too big, SP ₃ >SP ₄

The branches are made relative to the current program source address with a 5-bit displacement value. This means that the branch target address must be within -20_g to +17 locations of the current instruction.

3.7.4 AN EXAMPLE

Example 25 loads data pad X with an array A, with N elements starting at main data memory location 3721_g. CTR is in s-pad register which is used as a counter.

Example 25

1. CLRA CTR: SETCPA Set CPA to 0
2. LDMA: DB=3721 Fetch the "first element"
3. LDSP1 CTR: DB=N Initialize "CTR" to N
4. LOOP: INCIA; DEC CTR Fetch next element, A_{i+1}
5. CPX<MD: INCCPA: BNE LOOP Store A_i into CPX_i, advance CPA and test counter

1021

Example 26 shows the loop in Example 25 for the N=3 elements.

Example 26

INSTRUCTION NUMBER	MEMORY		DATA #40				S-#40 "CTR" TEST
	MA	M1	DPA	0	:	2	
1.	--	--	0	--	--	--	--
2.	3721	--	0	--	--	--	--
3.	--	--	0	--	--	--	3
4.	3722	--	0	--	--	--	3
5.	--	A ₀	0	A ₀	--	--	2 true
6.	3723	--	1	A ₁	--	--	2
7.	--	A ₁	1	A ₀	A ₁	--	1 true
8.	--	--	2	A ₀	A ₁	--	1
9.	--	A ₂	2	A ₀	A ₁	A ₂	0 false

1022

A generalization on the previous example to fetch array A from every Kth memory location is shown in Example 27. The increment is stored in s-reg register STEP, and the array pointer is stored in PTR.

Example 27

1. LDSP1 STEP: DB=N "Initialize 'STEP' to K
2. CLR# CTR; SET CPA "Set CPA to 0
3. LDMIA; DB=BASE "Fetch the first element, A_0
4. LDSP1 CTR: DB=N "Initialize 'CTR' to 1
5. LOOP: A0D STEP,PTR: SETMA
SEQ CONE \ "Advance memory pointer. Fetch
" next element, A_{i+1} . Test
" counter and jump out if
" done.
6. CPX<40; INC CPA
DEC CTR: BR LOOP "Store A_i into CPX_i, advance CPA
" Decrement 'CTR' and jump
" back to LOOP.
7. CONE: --

0023

CHAPTER 4

INTERFACE

4.1 INTRODUCTION

This chapter describes the interface between the host computer and the AP. The interface is composed of two basic parts: a simulated front panel and direct memory access control. The front panel allows the host computer to examine or modify the internal AP registers, as well as provides for block transfer of data from the host computer to the AP, and vice versa.

4.2 FRONT PANEL

The AP panel is used for bootstrap operations (loading and starting programs) and for debugging user software (inserting hardware breakpoints and examining and modifying AP registers and memory). The panel consists of three 16-bit registers which are under the control of the host via the host interface. The functioning of these registers closely parallels that of the switches and lights on the console of a stand-alone computer. The host can examine and/or set these registers at any time, regardless of the state of the AP. The front panel and host interface is shown in Figure 4-1.

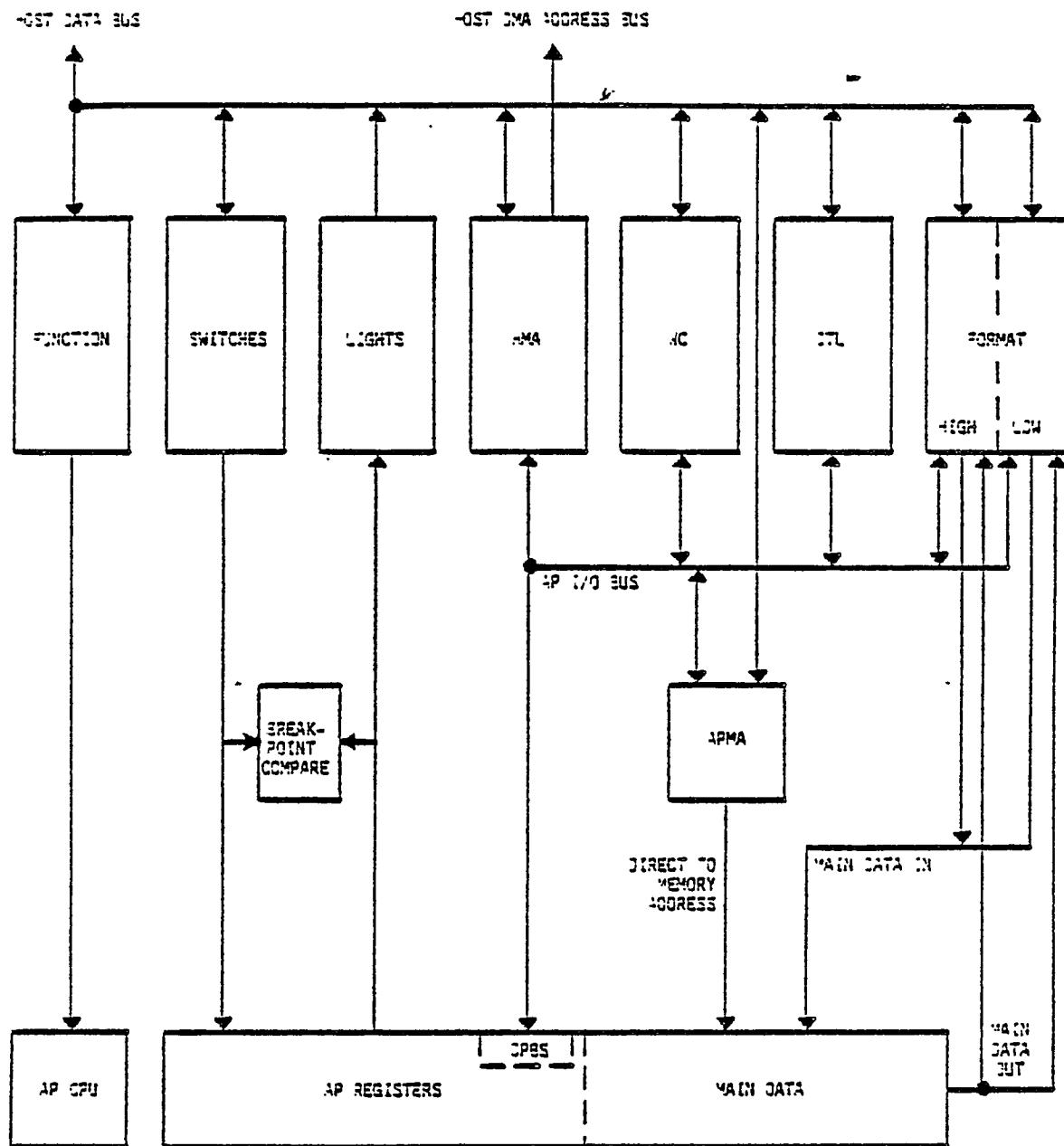


Figure 4-1 AP Panel and Host Interface

4.2.1 SWITCH REGISTER

The switch register (SWR) is used to enter data and addresses into the AP. The SWR can be read and written by the host computer. An executing AP program can also read the switches.

4.2.2 LIGHTS REGISTER

The lights register (LITES) simulates front-panel lights and is used to display the contents of internal AP registers. This register can only be read by the host. The executing AP program can set the lights register.

4.2.3 FUNCTION REGISTER

The function register (FN) provides front-panel control operations (start, stop, continue, etc.). It can be read or written by the host. The format of the function register is shown in Figure 4-2.

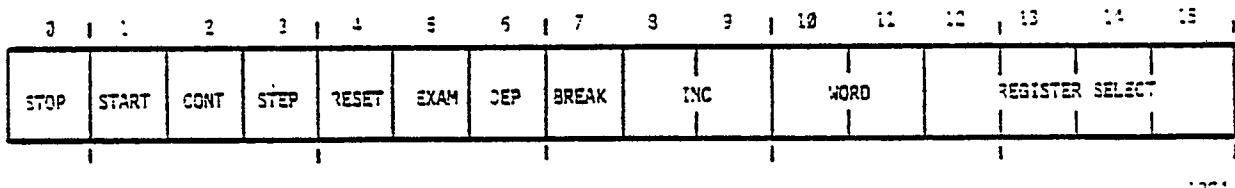


Figure 4-2 Panel Function Register Format

When the AP is running, only the STOP and RESET panel functions are valid. The other panel functions can only be exercised after the AP has halted. The panel functions are described in Table 4-1.

Table 4-1 Function Register Bits

BIT	MONIKONIC	EFFECT
0	STOP/HALTED	Stop AP program execution upon completion of the current instruction. When the host reads the F4 register, this bit reflects the current state of the processor. This bit is set if the AP is halted. (See note.)
1	START	Start program execution at the address specified in SXR.
2	CONT	Continue program execution at the instruction pointed at by PSA (program source address).
3	STEP	Execute the instruction pointed at by PSA and then halt. Advance PSA to point to the next instruction.
4	RESET	Stop the AP immediately. Clear s-ead register 0. Set SPEN to SP _{SPP} . Clear the AP status register. Stop the host DMA (C16 bit 13 set to 0) and clear main data memory timing.
5	EXAM	Examine the register or memory selected by the register select ffield. Display the portion selected by the WORD ffield in the panel display register.
6	DEP	Deposit the contents of the switch register into the register or memory selected by the register select ffield. Deposit into the portion selected by the WORD ffield.
7	BREAK	Enables hardware breakpoints if PSA, MA, or TA is specified in the register select ffield. The breakpoint causes the AP to halt one instruction after any instruction where the contents of the selected register was equal to the contents of the switch register. Thus, if a breakpoint is specified with PSA selected the AP halts after executing the instruction at the program location set in the switch register. PSA points to the next micro-instruction in sequence. If a breakpoint is called for on MA or TA, the AP halts after executing the instruction following the one that referenced the trapped memory location. PSA points to the second sequential instruction after the one that caused the breakpoint. Memory breakpoints aid in debugging those elusive errors that modify memory unexpectedly.
8 - 9	INC	Increment MA, TA, or CPA following completion of the other specified panel functions. This allows sequential memory locations to be examined or deposited into. (Refer to Table 4-2.)
10 & 11	WORD	Specifies which portion of a register is being examined or deposited into. (Refer to Table 4-3.)
12 - 15	REG.SELECT	Specifies which AP internal register or memory location to examine or deposit into. (Refer to Table 4-4.)

NOTE

If the current instruction performs a SPIN while waiting for I/O or memory, the STOP does not take effect until the spin condition is satisfied and the instruction completed.

Table 4-2 Bits 8-9

VALUE IN BITS 8 & 9	ADDRESS REGISTER TO BE INCREMENTED
0	None
1	N/A (Memory Address)
2	DPA (Data Pad Address)
3	TMA (Table Memory Address)

1058

Table 4-3 Bits 10-11

VALUE SET IN BITS 10 & 11	<16-BIT REGISTER	38-BIT REGISTER	64-BIT REGISTER
0	ALL	N/A	Bits 0-15
1	N/A	Exponent Bits 30-39; right-justified in 16-bit field.	Bits 16-31
2	N/A	High mantissa Bits 00-11; right-justified	Bits 32-47
3	N/A	Low mantissa	Bits 48-63

1027

Table 4-4 Octal Values

OCTAL VALUE SET IN BITS 12-15	REGISTER OR MEMORY SELECTED	DESCRIPTION
0	PSA	Program Source Address
1	SPD	S-Pad Destination Address
2	MA	Main Data Address
3	TMA	Table Memory Address
4	DPA	Data Pad Address
5	SPFN	S-Pad Function (EXAM)
	SP _{SPN}	S-Pad address by SPD (080S17)
6	AP STATUS	AP Internal Status Reg.
7	DA	Device Address Register
10	PS _{-MA}	Program Source Memory addressed by TMA
11	I08S	Examine I/O service output register addressed by DA
12	CB	Control Buffer, Bits 48-63 (EXAM only)
13	DPX _{-SP4-4}	Data Pad X addressed by (SP4-4)
14	DPY _{-CPA-4}	Data Pad Y addressed by (CPA-4)
15	MD _{MA}	Main Data Memory addressed by MA
16	SPFN	S-Pad Function (EXAM) only
17	T _{-TMA}	Table Memory Addressed by TMA (EXAM only)

1023

4.3 NOTES ON THE USE OF THE FRONT PANEL AND BREAKPOINT

4.3.1 WHERE DOES THE AP STOP ON A BREAKPOINT?

- With the breakpoint set on PSA, the AP stops with PSA pointing to the next instruction to be executed.

Thus, breaking on a branch instruction and then examining PSA shows whether the branch condition is true or false.

- With the breakpoint set on TMA, the AP stops with PSA pointing to the second instruction following the one that set TMA to the break address.
- With the breakpoint set on MA, the AP stops on either the next instruction or the second instruction after the one that set MA to the break address, depending on the state of the memory lockout hardware (next instruction if memory lockout, second instruction if no memory lockout).

Thus, the stopping point following an MA breakpoint has a one-instruction uncertainty.

4.3.2 DOES THE INSTRUCTION ON WHICH THE AP STOPS EXECUTE?

Since SPFN is current, it is set to the operation specified in the instruction that PSA is pointing to. Otherwise, the instruction that PSA is pointing to remains unexecuted. It executes correctly when the user steps or proceeds from the breakpoint.

4.3.3 WHAT ABOUT MD TIMING AND LOCKOUT ON A BREAKPOINT IN THE MIDDLE OF AN MD MEMORY CYCLE?

- The hardware is designed so that the AP can be stopped in the middle of a memory cycle. The hardware remembers where the memory timing is when the AP stops so that the processor can continue correctly from a breakpoint that occurs during a memory cycle.
- However, the user must not examine MD nor should there be any DMA transfers going to or from MD while the AP is stopped if the user wishes to proceed from the breakpoint.

Thus, for example, it is possible to break in the tight-to-memory portions of the FFT and examine data pad or the address registers (PSA, SPA, etc.) and then proceed. It is not possible to proceed if the user or the host interface disturbs the memory timing by reading or writing MD or DM.

4.3.4 SUMMARY OF THE RULE FOR PROCEEDING FROM BREAKPOINT

If the breakpoint causes the AP to stop in the middle of the memory cycle (PSA pointing to first or second instruction following SETMA, INCMA, DECMA, or LDMA), the user should not try to examine or modify MD.

4.3.5 WHAT ABOUT STEPPING THE AP?

The same rules for proceeding from a breakpoint apply to stepping the AP through a program. The user can examine and modify any register or memory within the constraints mentioned in section 4.3.4.

4.3.6 WHAT OTHER PITFALLS ARE THERE IN THE USE OF THE VIRTUAL FRONT PANEL?

- Note that the panel always examines SPFN, not SP_{SPD}. Thus, the user must force SPFN = SP_{SPD} to see SP_{SPD}. This can most easily be done via the panel reset function which has the side effect of also clearing SP(0).
- To examine TM, the user should first set TMA and then do a dummy panel operation (deposit TMA again, for example) in order to enter the output of table memory into the table memory buffer register. The user can then proceed to examine the addressed location using the appropriate panel functions.
- MD: setting MA from the panel initiates an MD memory read cycle. Depositing into MD from the panel initiates an MD memory write cycle.

Thus, to write MD and then examine what was just written, the user must perform a deposit into MA operation (with the same address) to initiate a read cycle before examining MD.

- Using the increment field in the FN register: DPA and TMA always increment after the EXAM or DEP operation is complete (remember that TMA is used to address program source memory for panel operations).
MA post-increments and initiates a new memory read cycle on an EXAM operation.
MA pre-increments on a DEP operation.

- The recommended procedure for starting the AP is as follows:

1. Set the SWR to the starting address and do a deposit into PSA.
2. Set the SWR to the desired breakpoint and do a continue to start the AP.

This procedure has the significant advantage of placing the necessary breakpoint code into the user's program should the AP program need debugging.

The panel START function can be used, but the user should observe the following restrictions on the first instructions executed by the AP. The first instruction should not branch, jump, or modify PSA in any way other than to advance to the next instruction. The first instruction should not use the SPEC and I/O fields. In fact, the preferred first instruction is a NOP (all zeros).

4.4 DIRECT MEMORY ACCESS

In addition to the panel function, the AP contains four 16-bit registers that are used for direct memory access (DMA) to both host and AP data memory, plus a 38-bit format conversion register that acts as a buffer between the two memories. These registers may be read and/or loaded from either the host computer or the AP.

4.4.1 HOST MEMORY ADDRESS REGISTER

The host memory register (HMA) points to consecutive locations in the memory of the host computer. It operates in either auto-increment or auto-decrement mode during DMA transfers to and from host memory. HMA is device address 1 for AP internal I/O transfers.

4.4.2 WORD COUNT REGISTER

The word count register (WC) counts the number of host memory words transferred in a DMA operation. It is preset to the desired number of words to be transferred and counts down as the transfer proceeds, stopping the DMA transfer when it reaches zero. Hardware logic prevents this register from being counted past zero. WC has AP device address 0.

4.4.3 AP DIRECT MEMORY ADDRESS REGISTER

The AP direct memory address register (APDMA) points to consecutive locations in AP main data memory during DMA transfers to and from MD. This register can operate in either auto-increment or auto-decrement mode. APDMA has AP device address 3.

4.4.4 CONTROL REGISTER

The control register (CTL) acts as a control over the DMA and interrupt functions of the host interface. This register controls the direction and mode of transfer (DMA or program control) and the type of data format and provides certain bits of status information pertaining to the transfer. CTL has AP device address 2. The format of the control register is shown in Figure 4-3. The bit descriptions are contained in Table 4-2.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-C=3	INTR -I2	IAP -I2	-ALT?	-I2	-I2	-I2 S118	SERR	SLATE	CC	-IP DMA	-RT -OST	-DEC DMA	-DEC DMA	-RT	-DMA START

1023

Figure 4-3 DMA Control Register Format

Table 4-5 DMA Control Register Description

BIT	MNEMONIC	EFFECT
0	WC = 0	Indicates that the word count register is zero. Note that WC is decremented only during DMA transfers to/from host memory (read only bit). Should not be used to monitor DMA activity.
1	INTR AP	Sets the INTRQ (interrupt request) flag in the AP.
2	IAPWC	Sets INTRQ (interrupt request) flag in the AP when the DMA transfer is done.
3	IHALT	Enables a host interrupt when the AP halts.
4	IHWC	Enables a host interrupt when the DMA transfer is done.
5	IHENB	Interrupt Host Enable. interrupt host if AP attempts to set this bit. This bit can actually be written only by the host. (This is not supported on all host systems.)
6	FERR	Format error. Indicates that exponent underflow or overflow occurred in conversion from AP format to host floating-point format.
7	DULATE	Data late. Indicates that the AP did not empty the format buffer before the host attempted to reload it. On some hosts this bit also indicates an attempt to access non-existent host memory. In either case the DMA transfer is terminated.
8	CC	Consecutive cycle. Block DMA transfers to/from host memory occur without interruption. On typical hosts, the host CPU is locked out but other higher priority DMA devices still have access to host memory.
9	APDMA	Allows the interface to perform DMA transfers to/from AP memory. Depending on the direction of transfer, a main data memory cycle is initiated every time the host finishes reading or loading the format register, whether via DMA or program control. On the AP side, the format register is loaded from the main data bus instead of the data pad bus.
10	WRTHHOST	Write to host. This bit controls the direction of transfer. If set, data is read from the AP, passed through the format register, and written to the host. If clear, the direction of transfer is reversed.
11	DECAPMA	Decrement APMA. If set, APMA is decremented during DMA transfers to/from AP Main Data memory. If clear, APMA is incremented. (This capability is not present on all host systems.)
12	DECCHMA	Decrement HMA. If set, HMA is decremented during DMA transfers to/from host memory. If clear, HMA is incremented.
13 & 14	FMT	Format Register Control. (See note.)
15	HDMA start/busy	Host DMA start. Initiate DMA transfers to/from host memory. When read, the state of this bit reflects the status of the host DMA activity ('1' if active, '0' if inactive). Transfers continue until WC = 0.

NOTE

The format register mode of operation is controlled entirely by bits 9, 10, 13 and 14 of the control register. Thus, even the host and the AP can load and read the format register via program control I/O transfers at any time. The programmer must be sure that the type of transfer he performs is consistent with these bits of CTL for the transfer to be meaningful. (Refer to Table 4-6.)

Table 4-6 Bits 13-14

VALUE IN BITS 13 & 14	FORMAT TYPE
0	32-bit Integer. No format conversion. Used to transfer integers or program half-words.
1	16-bit Integer. 16-bit integers from host are converted to unnormalized 32-bit AP FPNs. Low 16-bits of AP FPN are sent to host.
2	Conversion of "signed-magnitude mantissa with binary exponent" format to/from AP floating point format. Includes logic to handle "phantom bit" formats.
3	Conversion of IBM 32-bit format to/from AP format. IBM format can be specified to have either sign-magnitude or two's complement mantissa.

NOTE

For format types 2 and 3, the format register has the necessary logic to detect overflow and underflow on conversion from AP format and to force a signed maximum quantity on overflow or floating point zero on underflow.

.331

4.5 FORMAT CONVERSION REGISTER

This 38-bit double-buffered register is used for all transfers of floating-point numbers (FPNs) between the host and the AP. It also provides the most efficient path for transfer of microcode half-words (32 bits). It performs bi-directional format conversions under the direction of bits 9, 10, 13, and 14 of the CTL register. The programmer must be aware of the fact that the format conversion is a slave to these CTL bits. Nonsense results if transfers to and from the formatter are not consistent with these CTL bits. The host and AP can read the output of the formatter at any time without restriction; however, the input to the formatter is controlled by CTL bits 9 and 10.

Table 4-7 CTL Register Bits 9-10

CTL29	CTL10	INPUT PATH TO FORMATTERS
0.1	0	Host Data Bus
0	1	AP I/O Bus
1	1	AP Main Data Output

1032

The formatter has a ready indicator that can be sampled by the AP. This indicator tells the AP when to load new data into the formatter (CTL10=1) and when to read data from it (CTL10=0) after the host has finished reading or loading the last 16-bit word of a FPN.

Note that in 16-bit host computers, the interface expects to receive words in different order depending on CTL bit 12 (DECHMA). If bit 12 is clear (i.e., the host DMA interface is going through memory in forward order from low to high addresses), then the interface expects to receive the high word of an FPN followed by the low word. If bit 12 is set, the interface expects to receive the low word followed by the high word. This is done so that arrays of FPNs are always stored in forward order in host memory.

If the format CTL bits (bits 13 and 14) specify a 16-bit transfer (FMT=1) then the integer is loaded and read from the low word of the formatter. That word is considered to be the last word transferred.

There is no corresponding indicator to the host since the AP can transfer data to and from the formatter faster than most host processors. The DLATE bit in the CTL register (CTL bit 7) does indicate when an error of this type occurs (i.e., when the host transfers data faster than the AP).

4.6 AP INTERNAL INTERFACE TO HOST INTERFACE

The registers in the host interface are accessible to the AP via its input/output (I/O) instructions (FADD=7).

Table 4-8 AP Device Address for Host Interface Registers

I/O DEVICE	DEVICE ADDRESS
HOST INTERFACE	
DMA REGISTERS:	
WORD COUNT REGISTER (WC)	1
HOST MEMORY ADDRESS REGISTER (HMA)	1
CONTROL REGISTER (CTL)	2
AP MEMORY ADDRESS REGISTER (APMA)	3
FORMATTER (FMT)	4
WRITABLE TABLE MEMORY (WTRAM)	5
PAGE SELECT SELECT OPTION	
MEMORY ADDRESS EXTENSION (MAE)	30
APMA EXTENSION (APMAE)	31
MASK (including MODE and I/O)	32
ADDITIONAL DEVICE ADDRESSES:	
First IOP16	10-14
Second IOP15	20-24
Parity Counter	33-37
First IOP	100, 101, 110-117

1033

An IN, OUT, or SNSA instruction at DA=4 (FORMAT) generates an IODRDY response if the format register is ready to accept data from the AP (CTL bit 10=1) or if it has formatted data ready for the AP (CTL bit 10=0). If CTL bit 9 is 1, the AP cannot load the formatter via I/O instructions since the input multiplexer to the format register is set to select main data instead of the AP I/O bus. Note that the AP cannot change the state of CTL bit 5. An interrupt of the host is generated if it attempts to set this bit when the bit has already been set by the host. The AP can read the CTL at any time without interfering with the host interface. If both the host and the AP try to write CTL or access HMA, WC, or APMA at the same time, the host selection and data has priority over that of the AP.

Access to the format conversion register is controlled by CTL bits 9, 10, 13, and 14. Refer to section 4.4 for a description of the function of these bits.

4.7 AN EXAMPLE OF LOADING PROGRAMS INTO THE AP

Loading and running a program in the AP from a cold start is a five-step process which illustrates use of the front panel.

1. Using the AP front panel from the host computer, finger switch in a three-instruction bootstrap program into program memory.
2. Start the bootstrap running.
3. Set the address in the AP where the loaded program is to go.
4. Start a DMA transfer of program words from host computer memory to the AP. The bootstrap program running in the AP stores these words into program memory.
5. When the DMA transfer is done, stop the bootstrap program in the AP and then restart the AP executing the newly-loaded program.

These five steps are detailed in the remainder of Chapter 4. DMA control and front panel interrogation is done from the host computer by setting various interface registers. The actual host computer I/O instructions to accomplish this, of course, depend upon the particular host computer. For the purposes of this explanation, the indicated numbers are loaded into a designated interface register in order to accomplish the desired goals.

Step 1:

For the purpose of this example, the bootstrap program is put into program source memory locations 0, 1, and 2.

1. Set TMA to 0 (TMA is the pointer used by the panel functions for examining or depositing into program memory):

0 → SWR Put 0 into the switches.
1003 → FN Put 1003 into the function register
 (causing a deposit into TMA).

2. Put bits 0-63 of bootstrap program program word no. 1 into program memory location 0 using four deposits of SWR → P_{TMA}.

(bits 0-15) → SWR Put bits 0-15 into the switches.
1010 → FN Put 1010 into the function register
 (causes a deposit into bits
 0-15 of P_{TMA}).
..

(bits 16-31) → SWR Put bits 16-31 into bits
1030 → FN 16-31 of P_{TMA}.
..

(bits 32-47) → SWR Put bits 32-47 into 32-47
1050 → FN of P_{TMA}.
..

(bits 48-63) → SWR Put bits 48-63 into bits
1370 → FN 48-63 of P_{TMA} and
 increments TMA to point to
 location 1.

3. Repeat the second and third bootstrap program words in no. 2 above.

It is necessary to perform these steps only once.

Step 2:

Set the address in the AP program memory where the program is to be loaded by the bootstrap into TMA. For this example, this address is 200:

200 → SWR Put 200 in the switches.
1003 → FN Put 1003 into the function register
(causes a deposit into TMA).

Step 3:

Start the bootstrap program running in the AP.

Set the switches to 0 and do a start.

0 → SWR
40000 → FN Start the AP at location 0.

The bootstrap program (as demonstrated in step 4) spins while waiting for words to come across the DMA from the host computer.

Step 4:

Start the DMA transfer from host memory into the AP. For this example, it is assumed that the program is in host memory at location 20000. The program to be loaded is 200 AP program words (or 800 16-bit host words) long. The actual host memory location and length could be any particular value.

20000 → HMA Set host DMA address to 20000.
800 → WC Set word count to 800 host words
(assuming a 16-bit host word width).
201 → CTL Start the DMA.

Note in particular the CTL bits. Bit 15 initiates the DMA and bit 8 requests consecutive memory cycles from the host. By not setting bits 10 or 11, the transfer is set to go to the AP, but not into main data memory. Instead, the data goes only as far as the formatter which the bootstrap reads. If bit 4 is set, the host computer is interrupted when the DMA is done.

Step 5:

Finally, the three-word bootstrap program is ready to run in the AP.

1. LDDA; DB=4 "set DEVICE ADDRESS to 4

This instruction sets the device address register so that future I/O instructions refer to device no. 4, which is the DMA formatter (where the data from the host computer ends up).

2. LOOP:SPININ; "wait for some data
DB=INBS; "get the data
LPSLT "put it into the left half of P.S.

The SPININ causes the processor to hang until the current I/O device address (in this case, the DMA formatter) has some new data. Then, to read that data, the DB=INBS puts the input data onto the data pad bus. The LPSLT puts what is on the data pad bus into the left half (bits 0 through 31) of the program memory location pointed at by the TMA register.

Two points should be considered:

- The formatter is 32 bits wide on the AP end; every time the interface receives 32 bits of data from the host computer, the SPIN stops waiting, and another 32 bits of data are processed. Since the program words loaded are 64 bits wide, they are halved (left, right, left, right, etc.) and stored accordingly into program memory.
- TMA is used as a pointer indicating where the bootstrap should place the program it is loading; thus, the LPSLT puts the program words into the proper place.

3. SPININ; "wait for data
DB=INBS; "get the data
LPSRT; "put it into the right half
INCTMA; "increment pointer
BR LOOP. "go back for more

This does basically the same as no. 2 above except that this processes the right half (bits 32-63) of a 64-bit program word. The INCTMA increments the storing pointer so instruction no. 2 stores its data into the next word. The branch uses loop waiting for more program half-words.

Step 6:

Back in host, waiting for the DMA transfer is accomplished by:

- reading the CTL register
- tasting for bit 15 (the LSB) equal to 1
- if so, going back to step 1

Enabling a host interrupt on DMA completion is also possible.

When DONE, the bootstrap program is stopped (which otherwise would run forever) with a panel RESET function, and the newly-loaded program is started (example starts at location 200):

```
4000 → FN      "reset the AP
200  → SWR     "new program address
1000 → FN      "set 200 into PSA
20000 → FN     "continue (from 200) (i.e., start
                  at AP location 200)
```

To set a program breakpoint, the user can set the breakpoint address into the SWR and use 20400 (continue + break on PSA) for the final panel function.

NOTE

The simplest way for the running AP program to indicate to the host computer that it is done with its task is to HALT. When this happens, bit 0 in the panel function register is set (which the host can test for) or a host interrupt can be enabled (CTL bit 3).

APPENDIX A

AP REGISTERS/DATA PATH NAMES

Table A-1 Registers and Data Paths

<u>mnemonic</u>	<u>width</u>	<u>name</u>
SP	16 bits	scratch pad registers (16)
SPD	4	s-pad destination address register
SPFN	16	scratch pad ALU/shifter function output
PNBLS	16	panel bus
SWR	16	panel switch register
LITES	16	panel display register
APSTATUS	16	AP status register
PS	64	program source memory
CB	64	command buffer
PSA	16	program source address register
SRS	16	subroutine return stack
SRA	16	subroutine return stack pointer
DPX	38	data pad X registers (32)
DPY	38	data pad Y registers (32)
DB	38	data pad bus
DPA	16	data pad address register
TM	38	table memory output register
TMA	16	table memory address register
MD	38	data memory output register
MI	38	data memory input register
MA	16	memory address register
A1	38	floating adder input register no. 1
A2	38	floating adder input register no. 2
FA	38	floating adder output register
M1	38	floating multiplier input register no. 1
M2	38	floating multiplier input register no. 2
FM	38	floating multiplier output register
IODEVICE		I/O device
DA	16	I/O device address
INBS	38	I/O input bus
IODRDY	1	I/O data ready flag
A	1	I/O device condition A flag
B	1	I/O device condition B flag

Subscripts indicate addressing within memory element (i.e., PS_{PSA} means the location in program source memory pointed to by the program source address register).

Superscripts indicate portions of word (i.e., A_{2E} means the exponent portion of the A₂ register).

Parentheses around a symbol indicates the contents of a register (i.e., (A₁) means the contents of the A₁ register).

Table A-2 AP Internal Status Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OVF	UNF	SINVZ	FZ	FN	±	¶	C	PERR	PENG	SRAO	IFPT	FF			BIT REVERSE

1.034

bits mnemonic meaning

0 OVF Set when the current adder or multiplier (FA or FM) has overflowed. Overflow occurs when an exponent value is increased above 511. The offending result is set to the signed maximum of value of $(1-2^{-27}) * 2^{511}$, which is roughly $6.7 * 10^{153}$. This bit remains set until cleared by the microprogram or host computer.

1 UNF Set when the current adder or multiplier result (FA or FM) has underflowed. Underflow occurs when an exponent value is decreased below -512. The minimum legal magnitude which numbers can take without underflowing is roughly $3.7 * 10^{-155}$. The offending value is set to zero. This bit remains set until cleared by the microprogram or host computer.

Table A-2 Internal Status Register (cont.)

<u>bits</u>	<u>mnemonic</u>	<u>meaning</u>
2	DIVZ	A divide by zero has occurred. The result was set to the value of the dividend. This bit remains set until cleared by the microprogram or host computer.
3	FZ	Set when the current adder result (FA) is zero.
4	FN	Set when the current adder result (FA) is negative.
5	Z	Set when the current s-pad function (SPFN) is zero.
6	N	Set when the current s-pad function (SPFN) is negative.
7	C	S-pad carry bit. If no s-pad shift is specified, carry is the carry bit from the s-pad ALU. If a shift is specified, carry is the last bit shifted off the end of the s-pad result by the shift.
8	PERR	(Optional). Set when a main data memory parity error has occurred. Three parity bits are used, one each to check the exponent, high mantissa, and low mantissa portions of the memory word. If PENB is set, the processor halts on this error. (See Page Select/Parity Option Manual (FPS 860-7365-000) for more information.)
9	PENB	(Optional). Enables halt on memory parity error. If set, the processor halts when a memory parity error is detected.
10	SRAO	Subroutine return stack overflow. Set if more than 16 levels of nested subroutine calls occur.

Table A-2 Internal Status Register (cont.)

<u>bits</u>	<u>mnemonic</u>	<u>meaning</u>
11	IFFT	Inverse FFT flag. When set in conjunction with the FFT flag, bit 12, roots of unity table references are interpreted as positive angles.
12	FFT	FFT flag. When set, table memory addresses are interpreted as negative angles referencing the roots of unity table contained in table memory.
13-15	bit reverse	15- $\log_2 N$ where N is the length of a complex data array to which the s-pad address bit reverse operator is being applied.

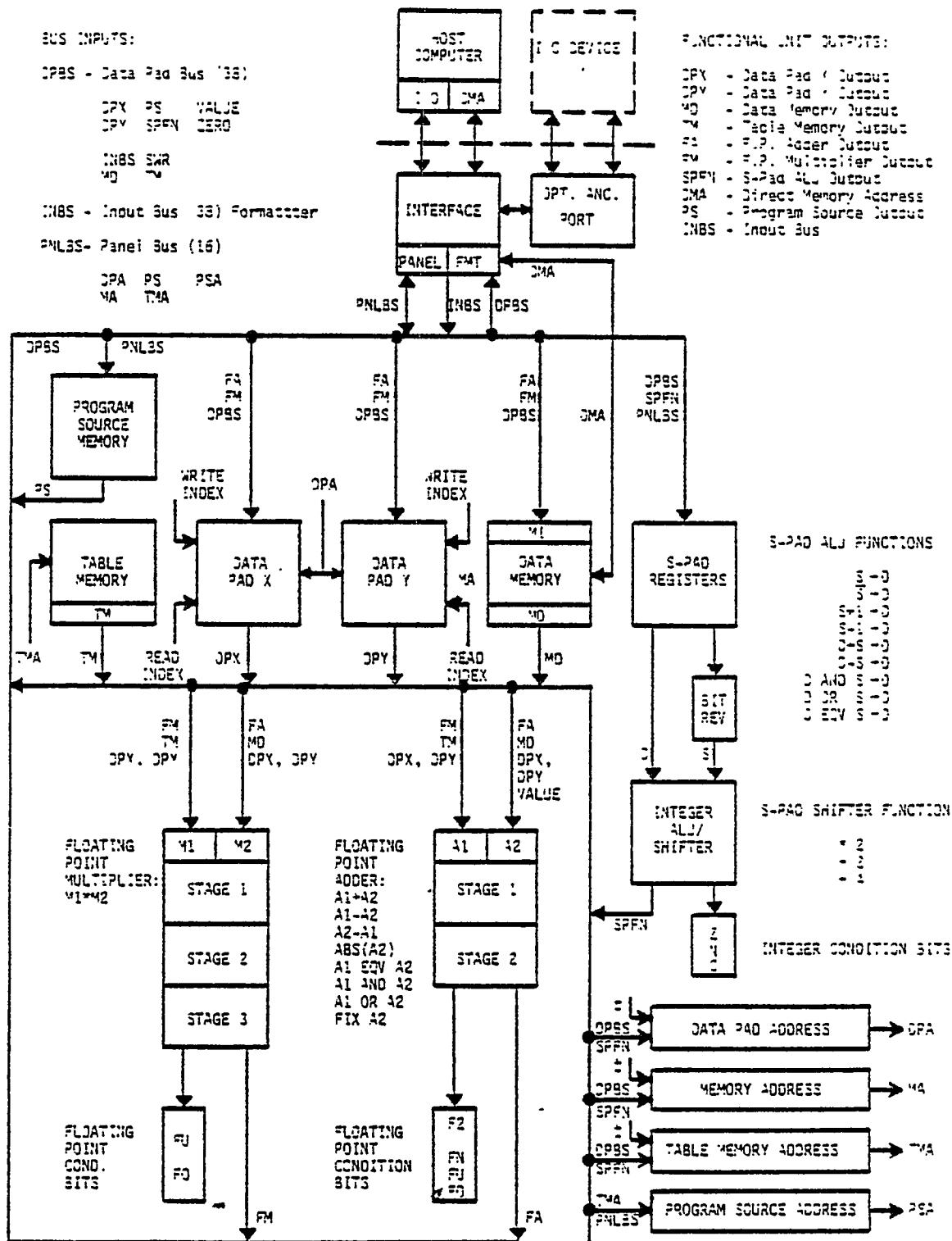


Figure A-1 AP Functional Units

Table A-3 AP Instruction Summary

UNCONDITIONAL FIELDS Each of the following fields may be used in any given instruction word.

OCTAL CODE	FIELD NAME											OCTAL CODE
	3	SOP	SOP1	S4	SPS	SPO	F400	F4001	A1	A2		
0	NOP	SOP1	NOP	NOP	(S-PAD)	(S-PAD)	F4001	NOP	IC	IC		0
1	SPEC	ARTEXP	L	Source	Cest.	FSUBR	FIX	F4	PA			1
2	ADD	WRTMN	RR	Reg.)	Reg.)	FSUB	FIXT	SPX	CPX			2
3	SUB	WRTMN	R			F400	FSCLT	SPY	CPY			3
4	MOV	NOP		(0-17)	(0-17)	FEQV	FSM2C	TM	XO			4
5	AND	NOP				FANO	FECSM	ZERO	ZERO			5
6	OR	NOP				FOR	FSCALE	ZERO	XOPX			6
7	ESQ	NOP				IO	FAGS	ZERO	SCPX			7
10	CLR											10
11	INC											11
12	DEC											12
13	COM											13
14	LOSPML											14
15	LOSPSE											15
16	LOSPI											16
17	LOSPST											17

OCTAL CODE	FIELD NAME											OCTAL CODE
	COND	CISP	CPX	CPY	CPBS	XR	Y2	W	Y4	FM	S4	
0	NOP	(Branch	NOP	NOP	ZERO	(CPX	(CPY	(CPX	(CPY	NOP		0
1	#	Displace-	BB	BB	INBS	Read	Read	Write	Write	FM		1
2	SR	ment)	FA	FA	VALUE*	(Index)	(Index)	(Index)	(Index)			2
3	3INTRO	(0-37)	FM	FM	CPX							3
4	B1CN				CPY	(0-7)	(0-7)	(0-7)	(0-7)			4
5	B1OZ				40							5
6	B1PE				SPFM							6
7	RETURN				TM							7
10	B2EQ											10
11	B2PE											11
12	B2GE											12
13	B2GT											13
14	SEQ											14
15	NE											15
16	GE											16
17	GT											17

OCTAL CODE	FIELD NAME						OCTAL CODE
	M1	M2	M4	MA	CPA	TMA	
0	FM	FA	NOP	NOP	NOP	NOP	3
1	CPX	OPX	FA	INCMA	INCJPA	INCTMA	1
2	CPY	JFY	FM	DECMA	DECOPA	DEC TMA	2
3	TM	40	CB	SETMA	SETOPA	SET TMA	3

* This instruction uses a 16-bit immediate VALUE as a constant or address (in bits 48-63 of this instruction). The Y4, FM, M1, M2, MA and CPA fields are then disabled for this instruction word.

Table A-4 SPEC Fields

SPEC FIELDS One of the SPEC Fields may be used per instruction word. The 3-bit Fields (S, S0P, S0P1, SH, EPS, and SPO) are then disabled for this instruction.

OCTAL CODE	FIELD NAME								OCTAL CODE
	SPEC	STEST	HOSTPNL	SETPSA	PSEVEN	PSODD	PS	SETEXIT	
0	STEST	SFLT	PMLIT	JMPA*	RPS0A*	RPS1A*	RPSLA*	NOP	0
1	HOSTPNL	SLT	DBELIT	JSRA*	RPS2A*	RPS3A*	RPSFA*	SETEXA*	1
2	SPMDA	ENC	DBHLIT	JMP*	RPS0*	RPS1*	RPSLT*	NOP	2
3	NOP	SZC	DBLLIT	JSR*	RPS2*	RPS3*	RPSF*	SETEXT*	3
4	NOP	DBBN	NOP	JMPT	RPS0T	RPS1T	RPSLT	NOP	4
5	NOP	DBBZ	NOP	JSRT	RPS2T	RPS3T	RPSFT	SETEXT	5
6	NOP	BIFN	NOP	JMPP	NOP	NOP	RPSLP	NOP	6
7	'NOP	BIFE	NOP	JSRP	NOP	NOP	RPSFP	SETEXP	7
10	SETPSA	NOP	SW08	NOP	WPS0A*	WPS1A*	LPSLA*	NOP	10
11	PSEVEN	NOP	SW0E	NOP	WPS2A*	WPS3A*	LPSRA*	NOP	11
12	PSODD	NOP	SW08H	NOP	WPS0*	WPS1*	LPSL*	NOP	12
13	PS	NOP	SW0BL	NOP	WPS2*	WPS3*	LPSR*	NOP	13
14	SETEXIT	SFL0	NOP	NOP	WPS0T	WPS1T	LPSLT	NOP	14
15	NOP	SFL1	NOP	NOP	WPS2T	WPS3T	LPSRT	NOP	15
16	NOP	SFL2	NOP	NOP	NOP	NOP	LPSL2	NOP	16
17	NOP	SFL3	NOP	NOP	NOP	NOP	LPSRP	NOP	17

* This instruction uses a 16-bit integer VALUE (in bits 48-63 of the instruction word). The YW, FM, M1, M3, MI, MA, TMA, and PDA Fields are then disabled for this instruction word.

Table A-5 I/O Fields

I/O FIELDS One of the I/O fields may be used per instruction word. The floating adder fields (FA0D, FA0D1, A1, and A2) are then disabled for this instruction word.

OCTAL CODE	FIELD NAME							OCTAL CODE
	IO	LDREG	R0REG	INOUT	SENSE	FLAG	CONTROL	
0	LDREG	NOP	RPSA	OUT	SNSA	SFL0	HALT	0
1	R0REG	LDSPD	RSPD	SPNOUT	SPININ	SFL1	IRST	1
2	SPMDAV	LDMA	RTMA	OUTDA	SNSADA	SFL2	INTEN	2
3	'NOP	LDTMA	RTMA	SPOTDA	SPNADA	SFL3	INTA	3
4	INOUT	LDOPA	R0PA	IN	SNSB	CFL0	REFR	4
5	SENSE	LDSP	RSPN	SPININ	SPINB	CFL1	ARTEX	5
6	FLAG	- LDAPS	RAPS	OUTDA	SNSBDA	CFL2	ARTMAN	6
7	CONTROL	LODA	RDA	SPINDA	SPNODA	CFL3	NOP	7

1036

APPENDIX B

INSTRUCTION SUMMARY

Table B-1 AP Instruction Field Layout

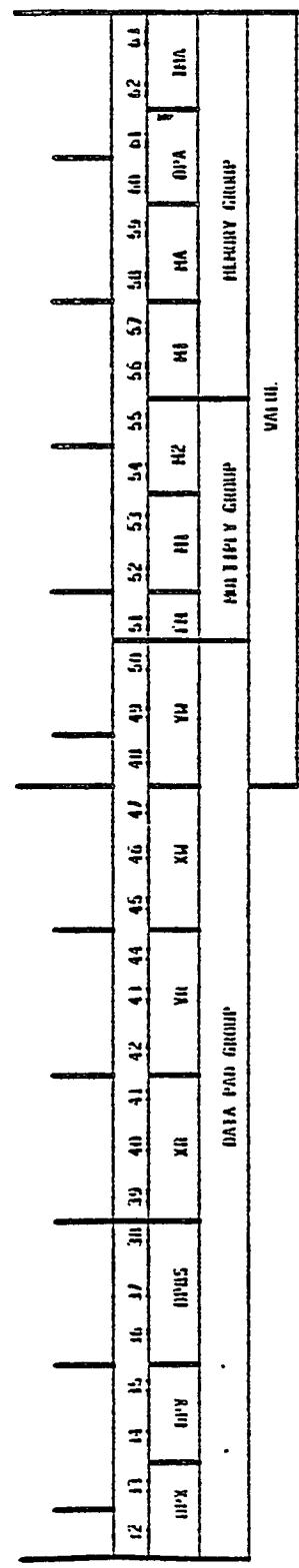
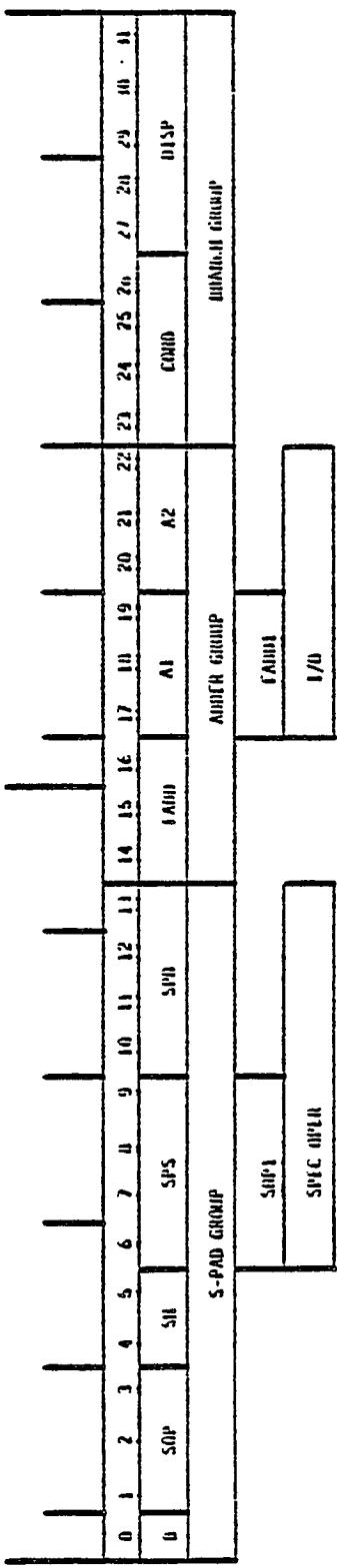


Table B-2 S-pad Group

2	:	3	4	5	6		7	10		11
5	SOP	SH		SPS			SPO			
SOPI										

FIELD	OCTAL CODE	MNEMONIC	EFFECT
SOP	0	-	No-op
	1	4	Use SP _{SPS} (bit-reversed)
SPS	0	-	See SOPI field
	1	-	See Special Operations Group
	2	ADD	(SP _{SPO}) + (SP _{SPS}) → SPFN
	3	SUB	(SP _{SPO}) - (SP _{SPS}) → SPFN
	4	MOV	(SP _{SPS}) → SPFN
	5	AND	(SP _{SPO}) AND (SP _{SPS}) → SPFN
	6	OR	(SP _{SPO}) OR (SP _{SPS}) → SPFN
	7	EQV	(SP _{SPO}) XOR (SP _{SPS}) → SPFN
SH (see NOTE)	0	-	No-op
	1	L	SPFN*2 → SPFN (left shift)
	2	RR	SPFN+4 → SPFN (double right shift)
	3	R	SPFN+2 → SPFN (right shift)
SPS	0-17 ₃	0-17 ₃	S-pad Source Operand Address
SPO	0-17 ₃	0-17 ₃	S-pad Destination Address, SPFN → SP _{SPO} unless inhibited by No Load (CONO = 1)

NOTE

These are logical shifts:

Right shift 0

0-15

15

Left shift 0

0-15

0

Table B-2 S-pad Group (cont.)

FIELD	BITAL CODE	MEMONIC	EFFECT (see NOTE) →
SOP1	0	-	'lo-oo'
	1	ARTEXP	Restricts SPX, SPY & MI fields to Write Exponent Only
	2	ARTHIN	Restricts SPX, SPY & MI fields to Write High Mantissa Only (Bits 30-11)
	3	ARTLHN	Restricts SPX, SPY & MI fields to Write Low Mantissa Only (Bits 12-27)
	4	-	-
	5	-	-
	6	-	-
	7	-	-
	10	CLR	0 → SPFN
	11	INC	(SP _{SPO}) + 1 → SPFN
	12	DEC	(SP _{SPO}) - 1 → SPFN
	13	COM	(SP _{SPO}) → SPFN Logical complement
	14	LDSPL	SP _{SPO} → SPFN, PNLS3 → SP _{SPO}
	15	LDSPE	SP _{SPO} → SPFN, 38 ^{ES} - E12 → SP _{SPO}
	16	LDSPI	SP _{SPO} → SPFN, 38 ^{ML} → SP _{SPO}
	17	LDSPT	SP _{SPO} → SPFN, 38 ^{MT} → SP _{SPO}

NOTE

MH = Mantissa High = Mantissa bits 30-11

ML = Mantissa Low = Mantissa bits 12-27

MT = Mantissa bits for single lookups = Mantissa bits 32-38

E = Exponent

Table B-3 Special Operations Group

3	2	1	SPEC	10	11
				STEST	
				-HOSTPNL	
				SETPSA	
				PSEVEN	
				PS000	
				PS	
				SETEXIT	

FIELD	OCTAL CODE	MNEMONIC	EFFECT
SPEC	0	-	See STEST Field (B-6)
	1	-	See HOSTPNL Field (B-7)
	2	SPMDA	Scan until MD available
	3	-	-
	4	-	-
	5	-	-
	6	-	-
	7	-	-
	10	-	See SETPSA Field, inhibit TEST except No Load (B-8)
	11	-	See PSEVEN Field (B-9)
	12	-	See PS000 Field (B-10)
	13	-	See PS Field (B-11)
	14	-	See SETEXIT Field (B-12)
	15	-	-
	16	-	-
	17	-	-

1039

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE)»
TEST	0	BFLT	Branch if $F_4 < 0.0$
	1	BLT	Branch if $SPP_4 < 0$
	2	BNC	Branch if S-Pad carry bit = 1
	3	BZC	Branch if S-Pad carry bit = 0
	4	BDN	Branch if $CB < 0.0$
	5	BDBZ	Branch if CB positive and unnormalized
	6	BIFN	Branch if Inverse FP flag = 1
	7	BIFZ	Branch if Inverse FP flag = 0
	10	-	-
	11	-	-
	12	-	-
	13	-	-
	14	BFL0	Branch if Flag 0 = 1
	15	BFL1	Branch if Flag 1 = 1
	16	BFL2	Branch if Flag 2 = 1
	17	BFL3	Branch if Flag 3 = 1

NOTE

If the above specified condition is true OR
the condition specified in the CONO field is
true, a branch occurs to (PSA) - DISP-20.

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 1)
-05TPNL	0	PNLSET	PNLBS → LITES
	1	OBELIT	OB ^E → PNLBS → LITES
	2	OBHLIT	OB ^{MH} → PNLBS → LITES
	3	OBLLIT	OB ^{ML} → PNLBS → LITES
	4	-	-
	5	-	-
	6	-	-
	7	-	-
	10	SWDB	(SWR) → PNLSS → DS
	11	SWDBE	(SWR) → PNLBS → OB ^E and WRTEXP (see NOTE 2)
	12	SWDBH	(SWR) → PNLBS → OB ^{MH} and WRTHMAN (see NOTE 2)
	13	SWDBL	(SWR) → PNLBS → OB ^{ML} and WRTLWAN (see NOTE 2)
	14	-	-
	15	-	-
	16	-	-
	17	-	-

NOTE

1) MH = Mantissa High = Mantissa bits 20-11

ML = Mantissa Low = Mantissa bits 12-27

E = Exponent

2) Restrict DPS, DPY and MI to:

WRTEXP: Write Exponent only

WRTHMAN: Write High Mantissa only (bits 20-11)

WRTLWAN: Write Low Mantissa only (bits 12-27)

Table 3-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT see NOTE
SETPSA	0	JMPA	VALUE \rightarrow PSA
	1	JSRA	(SRA) + 1 \rightarrow SRA, (PSA) + 1 \rightarrow SRS _{SRA} , VALUE \rightarrow PSA
	2	JMP	VALUE + (PSA) \rightarrow PSA
	3	JSR	(SRA) - 1 \rightarrow SRA, (PSA) - 1 \rightarrow SRS _{SRA} , VALUE - (PSA) \rightarrow PSA
	4	JMPT	(TMA) \rightarrow PSA
	5	JSRT	(SRA) + 1 \rightarrow SRA, (PSA) + 1 \rightarrow SRS _{SRA} , (TMA) \rightarrow PSA
	6	JMPO	(SWR) \rightarrow ONLESS \rightarrow PSA
	7	JSRP	(SRA) - 1 \rightarrow SRA, (PSA) - 1 \rightarrow SRS _{SRA} , (SWR) \rightarrow PNLESS \rightarrow PSA

NOTE

VALUE = Bits 48-63 of this instruction (C348-C363)

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 2)
PSEVEN	0	RPS0A	(PS ^{Q0} /VALUE) → PNLS → LITES
	1	RPS2A	(PS ^{Q2} /VALUE) → PNLS → LITES
	2	RPS0	(PS ^{Q0} /VALUE-PSA) → PNLS → LITES
	3	RPS2	(PS ^{Q2} /VALUE-PSA) → PNLS → LITES
	4	RPS0T	(PS ^{Q0} _{74A}) → PNLS → LITES
	5	RPS2T	(PS ^{Q2} _{74A}) → PNLS → LITES
	6	-	-
	7	-	-
	10	WPS0A	(SWR) → PNLS → PS ^{Q0} /VALUE
	11	WPS2A	(SWR) → PNLS → PS ^{Q2} /VALUE
	12	WPS0	(SWR) → PNLS → PS ^{Q0} /VALUE-PSA
	13	WPS2	(SWR) → PNLS → PS ^{Q2} /VALUE-PSA
	14	WPS0T	(SWR) → PNLS → PS ^{Q0} _{74A}
	15	WPS2T	(SWR) → PNLS → PS ^{Q2} _{74A}
	16	-	-
	17	-	-

NOTE

- 1) This field requires 2 cycles to execute.
- 2) VALUE = Bits 48-63 of this instruction (C348-C363)
 - Q0 = Quarter zero of Program Source Word (PS00-PS15)
 - Q2 = Quarter two of Program Source Word (PS31-PS47)

Table B-3 Special Operations Group (cont.)

FIELD (see NOTE 1)	OPCODE CODE	Mnemonic	EFFECT (see NOTE 2)
PS000 (see NOTE 1)	0	RPSLA	(PS ³¹ _{16-VALUE}) → PHLES → LITES
	1	RPSAA	(PS ³³ _{16-VALUE}) → PHLES → LITES
	2	RPSI	(PS ³¹ _{16-VALUE-PSA}) → PHLES → LITES
	3	RPS3	(PS ³³ _{16-VALUE-PSA}) → PHLES → LITES
	4	RPSIT	(PS ³¹ _{16-A}) → PHLES → LITES
	5	RPSAT	(PS ³³ _{16-A}) → PHLES → LITES
	6	-	-
	7	-	-
	8	APSIA	(SWR) → PHLES → PS ³¹ _{16-VALUE}
	9	APSAA	(SWR) → PHLES → PS ³³ _{16-VALUE}
	10	APS1	(SWR) → PHLES → PS ³¹ _{16-VALUE-PSA}
	11	APS3	(SWR) → PHLES → PS ³³ _{16-VALUE-PSA}
	12	APSIT	(SWR) → PHLES → PS ³¹ _{16-A}
	13	APSAT	(SWR) → PHLES → PS ³³ _{16-A}
	14	-	-
	15	-	-
	16	-	-
	17	-	-

NOTE

- 1) This field requires 2 cycles to execute.
- 2) VALUE = Bits 48-63 of this instruction (C348-C363)
 - 31 = Quarter one of Program Source Word (PS16-PS31)
 - 33 = Quarter three of Program Source Word (PS48-PS63)

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 2)
PS (see NOTE 1)	0	RPSLA	'PS ^{LH} /VALUE' → DB
	1	RPSFA	(PS ^{FP} /VALUE) → DB
	2	RPSL	(PS ^{LH} /VALUE-PSA) → DB
	3	RPSF	(PS ^{FP} /VALUE-PSA) → DB
	4	RPSLT	(PS ^{LH} /TMA) → DB
	5	RPSFT	(PS ^{FP} /TMA) → DB
	6	RPSLP	(PS ^{LH} /PNLBS) → DB
	7	RPSFP	(PS ^{FP} /PNLBS) → DB
	10	LPSLA	DB → PS ^{RH} /VALUE
	11	LPSRA	DB → PS ^{RH} /VALUE
	12	LPSL	DB → PS ^{RH} /VALUE-PSA
	13	LPSR	DB → PS ^{RH} /VALUE+PSA
	14	LPSLT	DB → PS ^{RH} /TMA
	15	LPSRT	DB → PS ^{RH} /TMA
	16	LPSLP	DB → PS ^{RH} /PNLBS
	17	LPSRP	DB → PS ^{RH} /PNLBS

NOTE

- 1) This field requires 2 cycles to execute.
- 2) VALUE = Bits 48-63 of this instruction (C848-C363)
 - LH = Left half of Program Source Word (Bits 30-31)
 - RH = Right half of Program Source Word (Bits 32-63)
 - FP = Program Source bits 26-63, used for floating-point literals

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE)
SETEXIT	0	-	-
	1	SETEX4	VALUE \longrightarrow SRS _{SRA}
	2	-	-
	3	SETEX	VALUE + (PSA) \longrightarrow SRS _{SRA}
	4	-	-
	5	SETEXT	PSA \longrightarrow SRS _{SRA}
	6	-	-
	7	SETEXP	PSA - 1 \longrightarrow SRS _{SRA}

NOTE

Sets the current subroutine return address as indicated above.
 SRA does not change.
 VALUE = Bits 48-63 of this instruction.

Table B-4 Floating Adder Group

14	15	17	19	20	22
F100		A1		A2	
		FADD1			

FIELD	OCTAL CODE	MNEMONIC	EFFECT
FADD	0	-	See FADD1 field
	1	FSUBR	Subtract: (A2) - (A1)
	2	FSUB	Subtract: (A1) - (A2)
	3	FACD	Add: (A1) + (A2)
	4	FEQV	Logical Equivalence: (A1) XOR (A2)
	5	FAND	Logical and: (A1) AND (A2)
	6	FOR	Logical or: (A1) OR (A2)
	7	-	See I/O Group
A1	0	NC	(A1) → A1
	1	FM	FM → A1
	2	DPX(IDX)	(DPX _{DPA+IDX}) → A1 where R = IDX+4
	3	DPY(IDX)	(DPY _{DPA+IDX}) → A1 where YR = IDX-4
	4	TM	(TM) → A1
	5	ZERO	0.0 → A1
	6	-	-
	7	-	-

NOTE

- All floating adder op-codes:
- 1) Align exponents
- 2) Perform the specified arithmetic, logical, or shift operation
- 3) Normalize
- 4) Convergently round

Table B-4 Floating Adder Group (cont.)

FIELD	JCTAL CODE	MNEMONIC	EFFECT
A2	0	NC	(A2) \rightarrow A2
	1	FA	FA \rightarrow A2
	2	DPX(10X)	(DPX _{SPY+10X}) \rightarrow A2, where X2 = 10X+4
	3	DPY(10X)	(DPY _{SPY+10X}) \rightarrow A2, where Y2 = 10X+4
	4	WD	(WD) \rightarrow A2
	5	ZERO	3.2 \rightarrow A2
	6	4DPA(10X)	SPFY+612 \rightarrow A2 ² , (DPY ⁴ _{SPY+10X}) \rightarrow A2 ⁴
	7	EDPX(10X)	(DPX _{SPY+10X}) \rightarrow A2 ² , SPFY \rightarrow A2 ⁴ (32-31), A \rightarrow A2 ⁴ (32-27)
FAD01	0	-	Ye=00
	1	F0X	Convert (A2) to an integer
	2	FEXT	Convert (A2) to an integer (result truncated)
	3	FEGLT	Shift (A2) right and increment A2 ² until A2 ² = (SPFY+611) (result truncated).
	4	FEM2C	Convert (A2), from signed magnitude to 2's complement.
	5	F2CSM	Convert (A2) from 2's complement to signed magnitude.
	6	FSCALE	Shift (A2) right and increment A2 ² until A2 ² = SPFY+611.
	7	FABS	Take the absolute value of (A2).

1348

Table B-5 I/O Group

14	15	16	17	18	19	20	21	22
:	:	:	:	0				LOREG
								SPREG
								INOUT
								SENSE
								FLAG
								CONTROL

FIELD	OCTAL CODE	MNEMONIC	EFFECT
I/O	0	-	See LOREG field
	1	-	See RDREG field
	2	SPMDAV	Spin until MD available
	3	REXIT	SRS(SPA) → SPNLS
	4	-	See INOUT field
	5	-	See SENSE field
	6	-	See FLAG field
	7	-	See CONTROL field
LOREG	0	-	No-op
	1	LDSPD	OPBS → SPD
	2	LDMA	OPBS → MA
	3	LDMA	OPBS → TMA
	4	LODPA	OPBS → DPA
	5	LDSP	SP _{SPD} → SPFN, OPBS → SP _{SPD}
	6	LOAPS	OPBS → APSTATUS
	7	LDIA	OPBS → DA

1049

Table B-5 I/O Group (cont.)

FIELD	TOTAL CODE	MEMONIC	EFFECT
PREG	0	RPSA	(PSA) → PHLES
	1	RSPD	(SPD) → PHLES
	2	RMA	(MA) → PHLES
	3	RTMA	(TMA) → PHLES
	4	RCPA	(CPA) → PHLES
	5	RSPFN	SPFN → PHLES
	6	RAPS	(APSTATUS) → PHLES
	7	ROA	CA → PHLES
INOUT	0	SUT	SPBS → (CODEVICE) _{DA}
	1	SPNOUT	SPIN if CODROY _{DA} = 0 SPBS → (CODEVICE) _{DA}
	2	SUTDA	SPBS → (CODEVICE) _{DA} , SPFN → CA
	3	SPOTCA	SPIN if CODROY = 0, SPFN → CA SPBS → (CODEVICE) _{DA}
	4	ON	(CODEVICE) _{DA} → ONBS
	5	SPININ	SPIN if CODROY _{DA} = 0 (CODEVICE) _{DA} → ONBS
	6	ONCA	(CODEVICE) _{DA} → ONBS, SPFN → CA
	7	SPINDA	SPIN if CODROY _{DA} = 0, SPFN → CA (CODEVICE) _{DA} → ONBS

1360

Table B-5 I/O Group (cont.)

FIELD	DIGITAL CODE	MNEMONIC	EFFECT (see NOTE)
SENSE	0	SNSA	$A_{DA} \rightarrow IODRDY$ flag
	1	SPINA	$A_{DA} \rightarrow IODRDY$, SPIN if $IODRDY = 0$
	2	SNSADA	$A_{DA} \rightarrow IODRDY$, SPIN $\rightarrow DA$
	3	SPNADA	$A_{DA} \rightarrow IODRDY$, SPIN if $IODRDY = 0$, SPEN $\rightarrow DA$
	4	SNSB	$B_{DA} \rightarrow IODRDY$ flag
	5	SPINB	$B_{DA} \rightarrow IODRDY$, SPIN if $IODRDY = 0$
	6	SNSBDA	$B_{DA} \rightarrow IODRDY$, SPEN $\rightarrow DA$
	7	SPNBDA	$B_{DA} \rightarrow IODRDY$, SPIN if $IODRDY = 0$, SPIN $\rightarrow DA$
FLAG	0	SFL0	$1 \rightarrow FLAG_0$
	1	SFL1	$1 \rightarrow FLAG_1$
	2	SFL2	$1 \rightarrow FLAG_2$
	3	SFL3	$1 \rightarrow FLAG_3$
	4	CFL0	$0 \rightarrow FLAG_0$
	5	CFL1	$0 \rightarrow FLAG_1$
	6	CFL2	$0 \rightarrow FLAG_2$
	7	CFL3	$0 \rightarrow FLAG_3$

NOTE

A and B are I/O device dependent conditions, either 1 or 0.

1051

Table B-5 I/O Group (cont.)

FIELD	DECAL CODE	MNEMONIC	EFFECT
CONTROL	0	ALT	Alt
	1	CRST	I/O reset
	2	INTEN	If CTL05 is set see Programmer's Reference Manual Part III, page 89.
	3	INTA	Interrupt acknowledge. Device Address of interrupting device put onto JPSS.
	4	REFR	Memory refresh sync
	5	RTEN	Restricts JPX, JPY & YI to write exponent only
	6	RTMAN	Restricts JPX, JPY & YI to write Mantissa only (bits 3-27)

1362

Table B-6 Branch Group

	23	26	27	31
	COND	DISP		
FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 1)	
COND	0	-	No-op	
	1	#	Inhibit load of SPFN \rightarrow SP _{SFD}	
	2	BR	Branch always	
	3	BINTRO	Branch if INTREQ (Interrupt Request flag = 1)	
	4	BION	Branch if IODRDY _{CA} flag = 1	
	5	BIOZ	Branch if IODRDY _{CA} flag = 0	
	6	BFPE	Branch on floating-point arithmetic error (overflow, underflow, or divide by zero).	
	7	RETURN (see NOTE 1)	(ERS _{SRA}) \rightarrow PSA, (SRA) - 1 \rightarrow SRA (subroutine return jump).	
	10	BFEQ	Branch if FA = 0.0	
	11	BFNE	Branch if FA \neq 0.0	
	12	BFGE	Branch if FA \geq 0.0	
	13	BFGT	Branch if FA $>$ 0.0	
	14	BEQ	Branch if SPFN = 0	
	15	BNE	Branch if SPFN \neq 0	
	16	BGE	Branch if SPFN \geq 0	
	17	BGT	Branch if SPFN $>$ 0	
DISP (see NOTE 3)	0 to 37 -	1	If branch condition is true, (PSA) + DISP - 20 \rightarrow PSA	

NOTE

- 1) "RETURNS" may not be made in two successive instructions.
- 2) FA and SPFN are tested as to their state for the previous instruction.
- 3) Thus the effective Branch Range is -20 to +17 relative to the current instruction.

Table 3-7 Data Pad Group

12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30		
OPX	OPY	OPSS																		
#CODE																EFFECT				
OPX (see NOTE 1)	0	-																		
	1	OPX(1CX)<08																		
	2	OPX(1CX)<FA																		
	3	OPX(1CX)<FM																		
OPY (see NOTE 1)	0	-																		
	1	OPY(1CX)<08																		
	2	OPY(1CX)<FA																		
	3	OPY(1CX)<FM																		
OPSS	0	CB=ZERO																		
	1	CB=MBS																		
	2	CB=VALUE																		
	3	CB=OPX(1CX)																		
	4	CB=OPY(1CX)																		
	5	CB=MO																		
	6	CB=SPFN																		
	7	CB=TM																		

NOTE

- 1) All bits written unless ARTEXP, ARTMANT or ARTLMAN set.
See S0P1 and -0STPML field.
- 2) OPSS forced to 0 if -0STPML field = 10 to 13
ML = Mantissa Low (Mantissa Bits 12-27)
MH = Mantissa sign (Mantissa Bits 30-31)
E = Exponent
VALUE is a 16-bit 2's complement number, contained in
bits 48-63 of the instruction word.

Table B-7 Data Pad Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT
XR	3 to 7		CPX Read EFA is (CPA) + XR - 1
YR	3 to 7		CPY Read EFA is (CPA) + YR - 1
XW	3 to 7		CPX Write EFA is (CPA) + XW - 1
YW	3 to 7		CPY Write EFA is (CPA) + YW - 1, YW=XW if VALUE is used in another field.

1066

Table 3-8 Floating Multiplier Group

	51	52	53	54	55
	51	41	4	42	
FIELD	OCTAL CODE	MNEMONIC		EFFECT	
41	0	-		No-op	
	1	MUL		Multiply: $(M1) \times (M2)$	
41	0	FM		$FA \rightarrow M1$	
	1	CPX(1DX)		$(CPX_{CPA+1DX}) \rightarrow M1$, Where $X=1DX+4$	
	2	CPY(1DX)		$(CPY_{CPA+1DX}) \rightarrow M1$, Where $R=1DX+4$	
	3	TM		$(TM) \rightarrow M1$	
42	0	FA		$FA \rightarrow M2$	
	1	CPX(1DX)		$(CPX_{CPA+1DX}) \rightarrow M2$, Where $X=1DX+4$	
	2	CPY(1DX)		$(CPY_{CPA+1DX}) \rightarrow M2$, Where $R=1DX+4$	
	3	40		$(40) \rightarrow M2$	

NOTE

These fields are not in effect if VALUE is used in another field.
 Arguments that are unnormalized by more than one position will
 produce incorrect results.

1086

Table B-9 Memory Group

	56	57	58	59	60	61	62	63
	MI	MA		DPA		TMA		
FIELD (see NOTE 1)	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 3)					
MI	0	-	No-op					
	1	MI<FA	FA → MI, Write MI into Data Memory (see NOTE 2)					
	2	MI<FM	FM → MI, Write MI into Data Memory (see NOTE 2)					
	3	MI<DB	DB → MI, Write MI into Data Memory (see NOTE 2)					
MA	0	-	No-op					
	1	INCMA	(MA)+1 → MA, initiate a Data Memory cycle					
	2	DECMA	(MA)-1 → MA, initiate a Data Memory cycle					
	3	SETMA	SPFN → MA, initiate a Data Memory cycle					
DPA	0	-	No-op					
	1	INCDPA	(DPA)+1 → DPA					
	2	DECDDPA	(DPA)-1 → DPA					
	3	SETDPA	SPFN → DPA					
TMA	0	-	No-op					
	1	INCTMA	(TMA)+1 → TMA, initiate a read from Table Memory					
	2	DECTMA	(TMA)+1 → TMA, initiate a read from Table Memory					
	3	SETTMA	SPFN → TMA, initiate a read from Table Memory					

NOTE

- 1) These fields are not in effect if a value is used by another field. Changes made in MA, TMA, or DPA do not affect the values of these registers used by other fields during the current instruction.
- 2) All bits written unless WRTEXP, WRTHMAN or WRTLTMAN is set. See SOP1 and HOSTPNL fields.
- 3) DB is used in place of SPFN if LDRREG field is used.

APPENDIX C

DATA GENERAL ECLIPSE INTERFACE

C.1 ECLIPSE INTERFACE I/O INSTRUCTIONS

<u>I/O Instruction Assignments</u>	<u>Interface Register</u>
DOA/DIA AP0	Panel Switches (SWR)
DOB/DIB AP0	Panel Function (FN)
DIC AP0	Panel Lights (LITES)
DOA/DIA AP1	Word Count (WC)
DOB/DIB AP1	Host Memory Address (HMA)
DOC/DIC AP1	DMA Control (CTL)
DOA/DIA AP2	AP Memory Address (APMA)
DOB/DIB AP2	Formatter High Word (FMT 00-15)
DOC/DIC AP2	Formatter Low Word (FMT 16-31)
DOA/DIA AP3	Page Select Register
<u>Other I/O Instructions</u>	<u>Effect</u>
NIOS AP0	Sets AP0 Busy, Clear Done
NIOC AP0	Clears AP0 Busy and Done
NIOS AP1	Sets AP1 Busy, Clears Done
NIOC AP1	Clears AP1 Busy and Done
NIOS AP2	Sets AP2 Busy, Clears Done
NIOC AP2	Clears AP2 Busy and Done
NIOP AP0	Resets: DMA Logic Interrupt Busy/Done Logic AP (same as panel reset)

<u>Other I/O Instructions</u>	<u>Effect</u>
IIRST	Same effect as NIOP APO, also presets system and user accessible MSKO flip-flops.
SKPBN AP0	Skip if AP Running
SKPBN AP1	Skip if AP DMA in Progress } see
SKPBN AP2	Skip if Waiting for } NOTE CTL05 Interrupt
SKPDN AP0	Skip if AP Done Interrupt
SKPDN AP1	Skip if AP DMA Done Interrupt
SKPDN AP2	Skip if AP CTL05 Interrupt

NOTE

Effective only if appropriate NIOS
instruction issued when enabling
the respective operation.

<u>Other I/O Instructions</u>	<u>Effect</u>
DOC APO BIT5=0	Sets user accessible MSKO flip-flop, enables interrupt
DOC APO BIT5=1	Clears user accessible MSKO flip-flop, disables interrupt
MSKO BIT5=0	Sets system MSKO flip-flop, enables interrupt
MSKO BIT5=1	Clears system MSKO flip-flop, disables interrupt

C.2 COMMENTS

NIOS AP1 must be issued before attempting to start a DMA transfer (DOC AP1 with bit 15 set to 1). Completion of the DMA transfer results in an interrupt unless disabled by a DOC APO (Bit5=1) or MSKO (Bit5=1) instruction.

HOST DATA

2	3	4	7	S	11	12	15
MODE	I/O		MASK		APMAE		MAE

HOST DATA BUS TO PAGE SELECT REGISTERS:

<u>AP DEVICE ADDRESS</u>	<u>REGISTER</u>
30	MAE
31	APMAE
32	MODE, I/O, MASK

1065

Figure C-1 Bit Map for Page Select Registers

Notice to the Reader

- Help us improve the quality and usefulness of this manual.
 - Your comments and answers to the following READERS COMMENT form would be appreciated.
-
-

To mail: fold the form in three parts so that Floating Point Systems' mailing address is visible; seal.

Thank you

READERS COMMENT FORM

Document Title _____

Your comments and answers will help us improve the quality and usefulness of our publications. If your answers require qualification or additional explanation, please comment in the space provided below.

How did you use this manual?

- () AS AN INTRODUCTION TO THE SUBJECT
- () AS AN AID FOR ADVANCED TRAINING
- () TO LEARN OF OPERATING PROCEDURES
- () TO INSTRUCT A CLASS
- () AS A STUDENT IN A CLASS
- () AS A REFERENCE MANUAL
- () OTHER _____

Did you find this material . . .

- | | YES | NO |
|-----------------------|-----|-----|
| • USEFUL? | () | () |
| • COMPLETE? | () | () |
| • ACCURATE? | () | () |
| • WELL ORGANIZED? | () | () |
| • WELL ILLUSTRATED? | () | () |
| • WELL INDEXED? | () | () |
| • EASY TO READ? | () | () |
| • EASY TO UNDERSTAND? | () | () |

Please indicate below whether your comment pertains to an addition, deletion, change or error; and, where applicable, please refer to specific page numbers.

Page	Description of error or deficiency

From:

Name _____
Firm _____
Address _____
Telephone _____

Title _____
Department _____
City, State _____
Date _____

First Class
Permit No. A-757
Portland,
Oregon

BUSINESS REPLY

No postage stamp necessary if mailed in the United States

Postage will be paid by:

FLOATING POINT SYSTEMS, INC.

P.O. Box 23489
Portland, Oregon 97223

Attention: Technical Publications

IRP

FAST RECONSTRUCT PROCESSOR

AP120B

By FPS TECHNICAL PUBLICATIONS STAFF

GENERAL ELECTRIC
IMAGE RECONSTRUCTION
PROCESSOR (IRP)
MANUAL

Publication No. 7379
First Edition, April 1978

NOTICE

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

Floating Point Systems, Inc. assumes no responsibility for any errors which may appear in this publication.

PROPRIETARY INFORMATION

This document contains proprietary information and is supplied for identification, maintenance, engineering evaluation or inspection purposes only and shall not be duplicated or disclosed without written permission of FLOTTING POINT SYSTEMS, INC.

By accepting this document the recipient agrees to make every effort to prevent unauthorized use of this information.

Copyright © 1978 by Floating Point Systems, Inc.
Beaverton, Oregon 97005

All rights reserved. No part of this publication may be reproduced in any form or by any means without permission in writing from the publisher.

Printed in USA

CONTENTS

	Page
CHAPTER 1 IRP THEORY OF OPERATION	
1.1 INTRODUCTION	1-1
1.2 IRP ARCHITECTURE	1-1
1.3 BOARD LAYOUT	1-5
1.4 OPERATION	1-8
1.5 ADDITIONAL INFORMATION	1-9
CHAPTER 2 IRP DIAGNOSTICS	
2.1 INTRODUCTION	2-1
2.2 IMAGE RECONSTRUCTION PROCESSOR DIAGNOSTIC	2-1
2.3 USING IRPTST	2-2
2.4 EXAMPLES	2-4
2.5 TEST DESCRIPTIONS	2-6
2.5.1 Test 1 — Control Register Test	2-6
2.5.2 Test 2 — K Register File Test	2-7
2.5.3 Test 3 — LAMBDA Address Test	2-7
2.5.4 Test 4 — PES Test	2-8
2.5.5 Test 5 — PAS Test	2-8
2.5.6 Test 6 — LPE Test	2-8
2.5.7 Test 7 — LPA Test	2-8
2.5.8 Test 8 — WGT Test	2-9
2.5.9 Test 9 — DETNO Test	2-10
2.5.10 DMA Test	2-11
APPENDIX	
A.1 LOADING AND RUNNING IRP DIAGNOSTIC	A-1

ILLUSTRATIONS

Figure No.	Title	Page
1-1	IRP to AP Data Paths	1-2
1-2	IRP Internal Architecture	1-3
1-3	ICTL Format	1-4
1-4	Block Diagram of IRP	1-6
1-5	Stage One Timing Relationship	1-10
1-6	ROM Address Field Relationships	1-11

TABLES

Table No.	Title	Page
1-1	Related Documents	1-1
1-2	IRP Device Address Assignments	1-4
1-3	PERP Round Truth Table	1-11
1-4	WGT Packed to Floating-Point Exponent Conversion	1-12
2-1	Operator Command Set	2-3
A-1	Function Filename	A-1

CHAPTER 1

THEORY OF OPERATION

1.1 INTRODUCTION

This manual describes the operation for the Image Reconstruction Processor (IRP) option from a hardware point of view. It is intended to be used as an aid to understanding the schematic diagrams of IRP logic.

Table 1-1 Related Documents

RELATED SCHEMATICS	
IRP1 Board 275 Schematic	512-3275-000
IRP2 Board 276 Schematic	512-3276-000

0255

1.2 IRP ARCHITECTURE

The image reconstruction processor was designed to enhance the execution of a specific proprietary algorithm in the AP. The IRP is used to calculate an address in TMRAM and provide a weighting factor for the data found in TMRAM.

Figure 1-1 shows the logical placement of the IRP within the AP. The weight (WGT) is applied to the M2 input of the floating multiplier (FM). Once activated, the IRP modifies the "FMUL TM, MD" instruction so that MD will be replaced by the W output of the IRP.

Figure 1-2 illustrates the internal architecture of the IRP. Four stages of pipelining enable the IRP to generate data at the 6 MHZ Rate of the AP. Some overhead is involved in initializing the pipeline as a series of "FMUL" instructions are required to push the data from one stage to the next until the pipeline is filled. Once filled, each successive "FMUL" pushes new data out every AP cycle.

The first stage of the pipeline contains five sets of register files. Each register file is organized as a rotary file of variable length (1, 2, 4, 8, or 16 registers deep). The length is determined by the number of sets of view data present in TMRAM. Each "FMUL" instruction causes the files to advance to the next set of view data. After one pass has been made through all of the view sets the rotary file counter resets to the first set. The contents of files PERP and PARL are replaced by the algebraic sum of that file and its respective increment file (PERP+PERPI, PARL+PARLI) at the end of each cycle. This allows the IRP to automatically step through the view sets in TMRAM, freeing the AP from the overhead of indexing the data. The fifth file--K--holds base addresses of the view data in TMRAM.

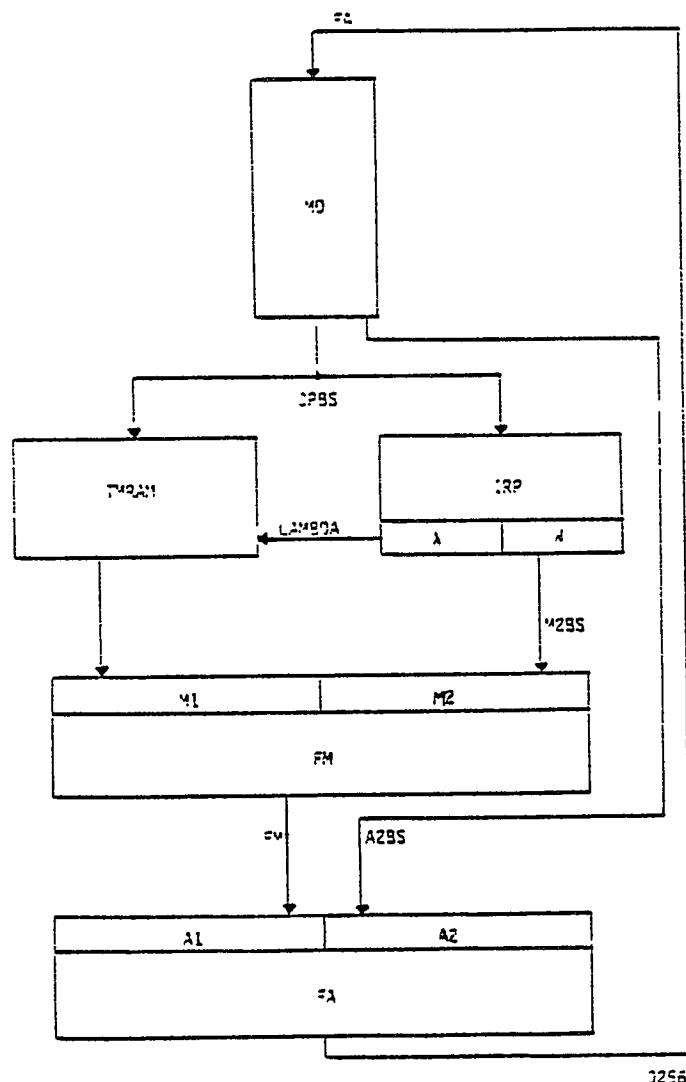


Figure 1-1 IRP to AP Data Paths

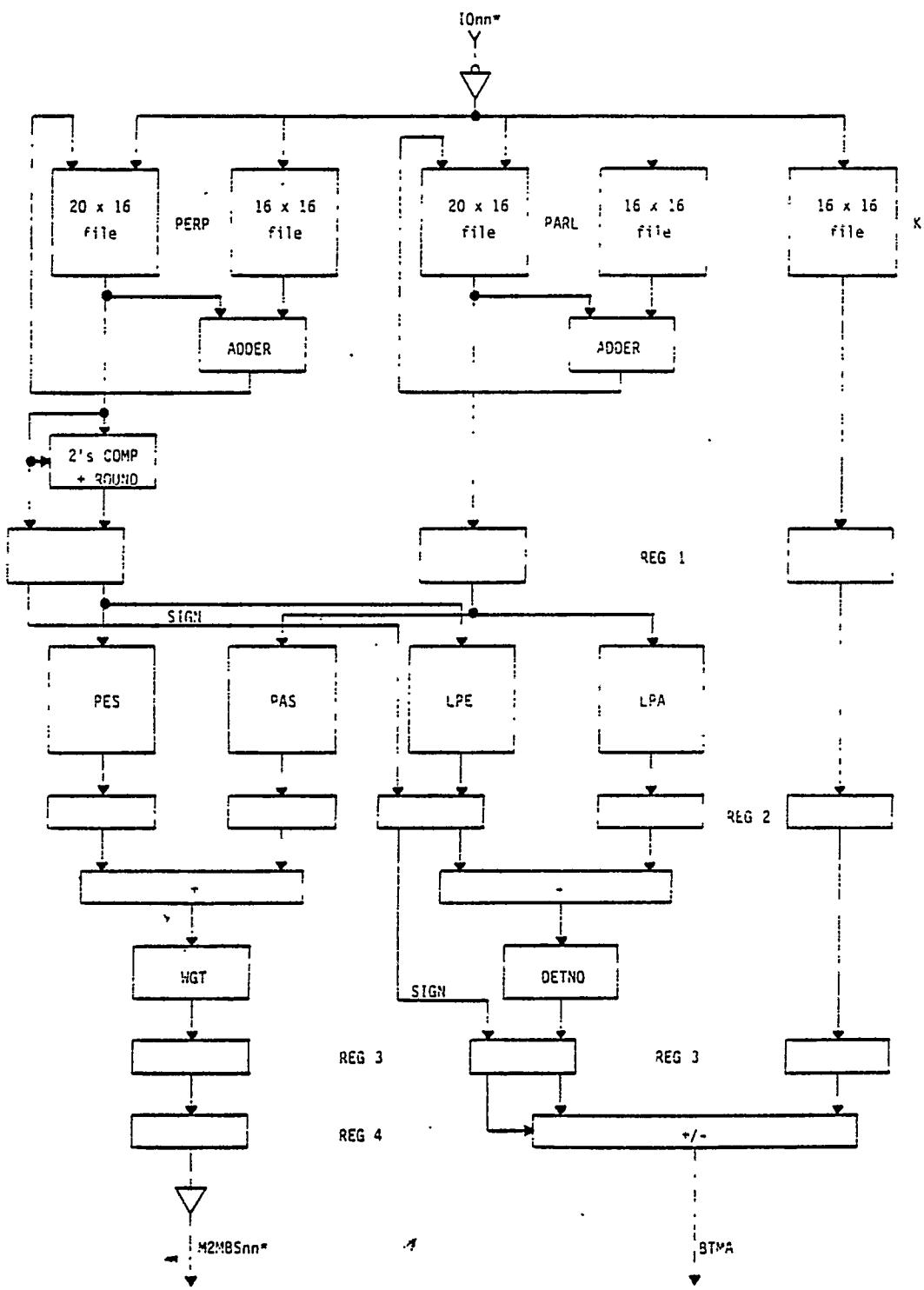


Figure 1-2 IRP Internal Architecture

A separate control register is provided in the IRP that contains the IRP enable bit, a reset bit, a field that specifies the number of view data sets, and a diagnostic bit that disables the output of the DETNO ROM. The bit assignment in the ICTL register (IRP Control) is shown in Figure 1-3.

I/O BUS BITS	33	34	35	36	37	38	39
DETNO ZERO			VIEW		RESET	ENABLE	

0258

Figure 1-3 ICTL Format

ICTRL is a read/write AP I/O device at device address 105. The five register files, PERP, PARL, PERPI, PARLI, and K are write only from the I/O bus. Several diagnostic paths are provided in the IRP to enable the output of the PES, PAS, LPE, LPA and DETNO ROMS to be read onto the I/O bus. The WGT ROM output is available through multiplying by 1.0 from TMROM. Table 1-2 lists the IRP device address assignments for both IN and OUT instructions.

Table 1-2 IRP Device Address Assignments

Octal Device Address	IN (read)	OUT (write)
100	PERP	PES
101	PERPI	LPE
102	PARL	PAS
103	PARLI	LPA
104	K	DETNO+K
105	ICTL	ICTL

0262

Before the IRP is run, the register files must be initialized with the setup parameters. Consecutive loads of PERP, PARL, etc., cause the rotary file counter to advance to the next location; thus, for example, it is possible to load the PERP's, for the views in TMRAM via a series of OUT instructions to device address 100. DA can then be changed to 101 and the PERPI's loaded and so on.

1.3 BOARD LAYOUT

A detailed block diagram of the image reconstruction processor is shown in Figure 1-4. The IRP logic is contained on two etch circuit boards installed in the AP chassis, IRP1 (board 275), and IRP2 (board 276). All of stage one of the pipeline and stage two of K is on board 275. The ROMS, their ALU's, and bus drivers are on board 276.

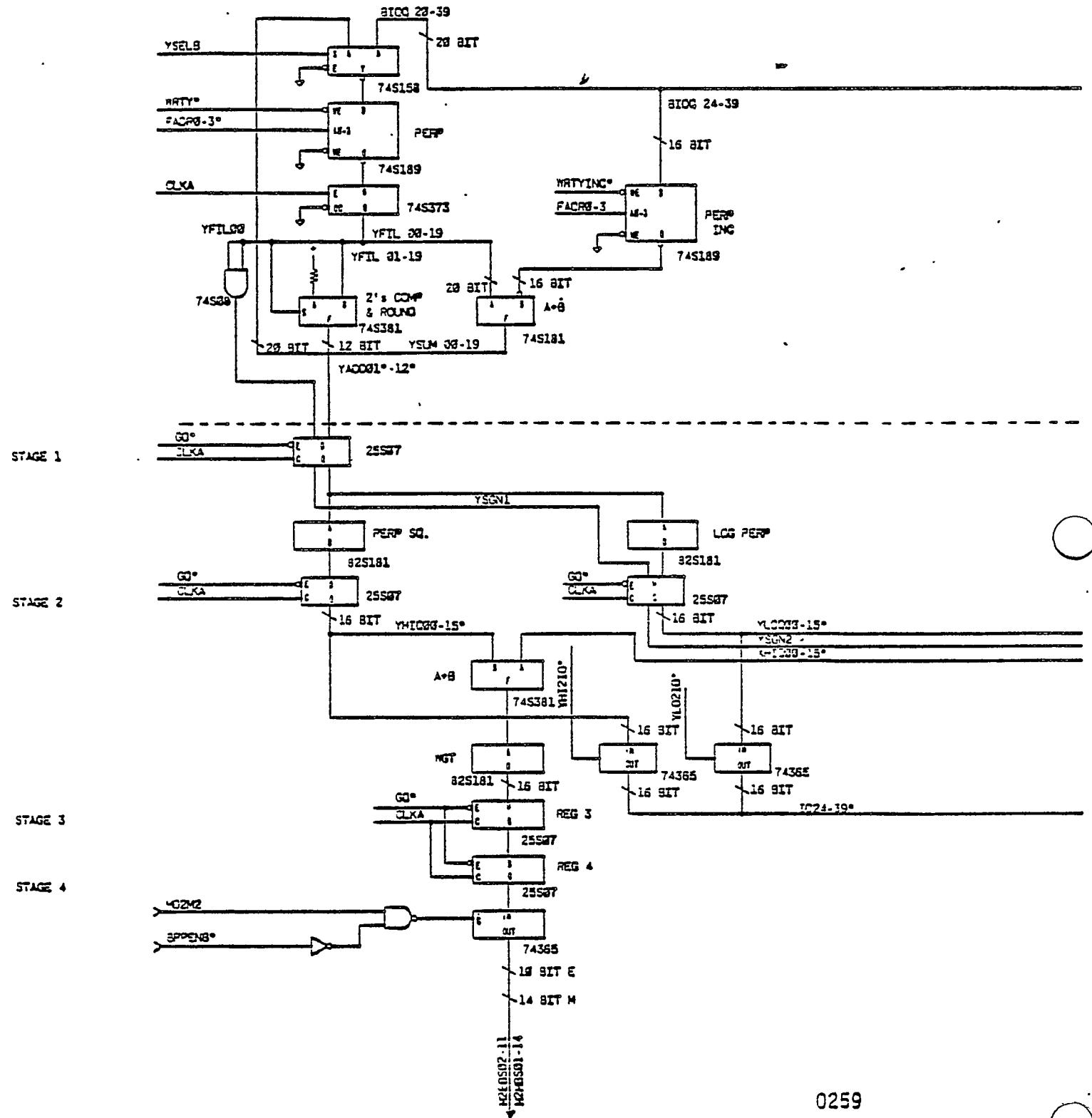


Figure 1-4 Block Diagram of IRP

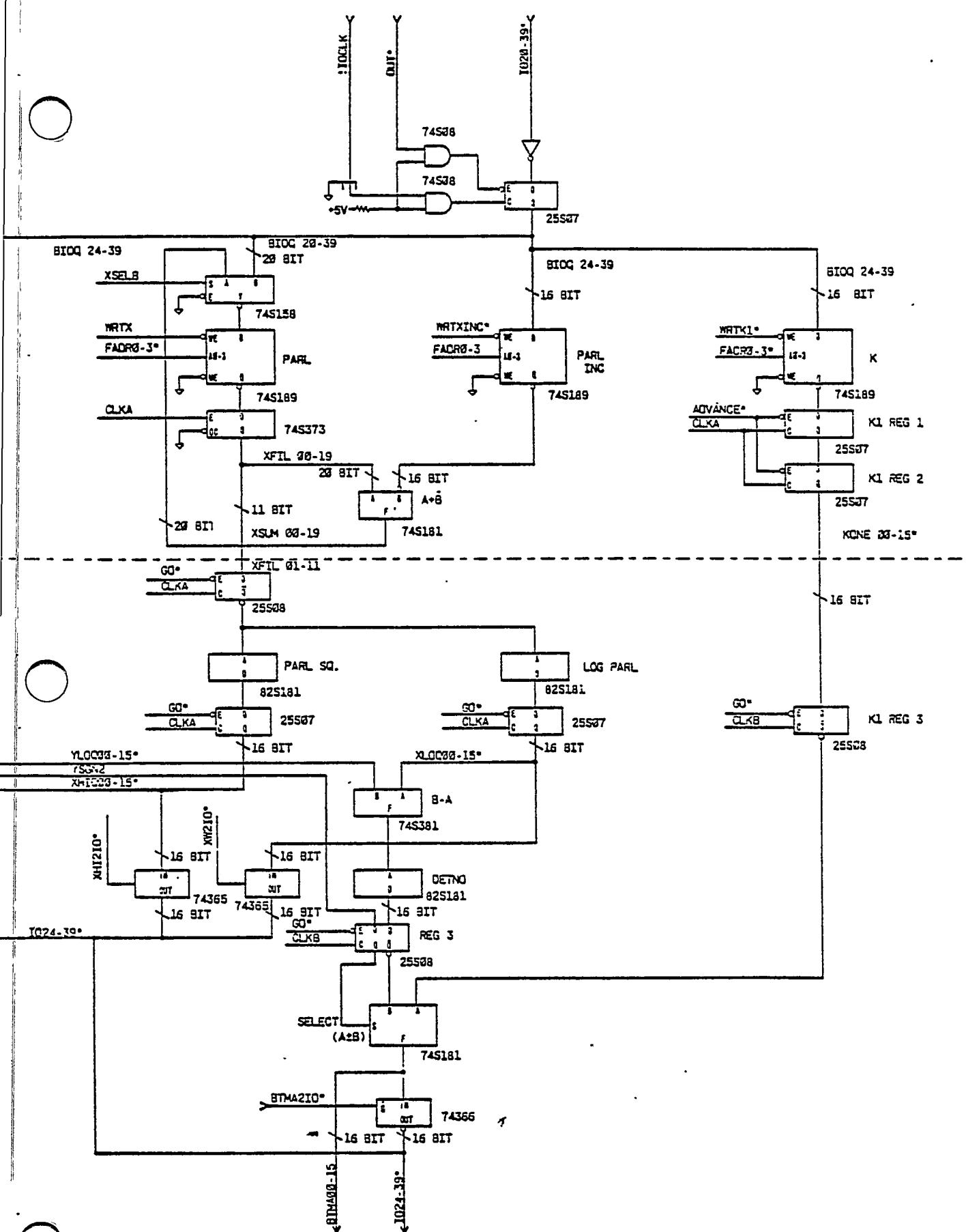


Figure 1-4 Block Diagram of IRP (cont.)

1.4 OPERATION

The first stage of the IRP performs the PERP and PTRL update and enters the results along with K into the first pipeline register. The second stage accesses the PES, PAS, LPE and LPA ROMS and enters the result into the second pipeline register. The third stage forms $PES+PAS$ and $LPE-LPA$ and then accesses the WGT and DETNO ROMS storing the result in the third pipeline register. The fourth stage adds/subtracts DETNO to/from K that is passed down to form the TMRAM address LAMBDA. In this stage WGT is passed to a holding register so that it can be applied to M2BS in the correct time relationship to match the TMRAM output. More detailed descriptions of the function of each stage are given in the next section.

1.5 ADDITIONAL INFORMATION

Note: {275-1} indicates a reference to the schematic for board 275, page 1.

The rotary files are made up of a series of 16 location x 4 bit register file chips (74S189) ganged together to make up the appropriate file widths (20 bits for PERP, PARL, 16 bits for PERPI, PARLI and K) {275-2,3,4,5,6}. All of the files share the same address bits which are generated by a binary counter (74S169) loaded from the NVIEW field of the ICTRL register {275-1}. The counter operates in a decrement mode and is clocked at the end of a cycle following a cycle in which a FMUL instruction is executed. (See Figure 1-5.) When the counter decrements to zero it is reloaded from the NVIEW field of ICTRL on the next FMUL. The counter is also loaded when the reset bit in the ICTRL register is written.

The output of the PERP+PERPI and PARL+PARLI adders {275-4,6} are available at the inputs of the PERP and PARL files along with the latched I/O bus lines. A 2:1 multiplexer selects the adder outputs so long as the files are not being loaded by OUT instructions. The contents of PERPI and PARLI files are considered to be negative if the most significant bit (MSB) is one. These files are sign-extended to 20 bits at the input of the adders to properly handle additions of negative numbers. PERP is also signed; MSB=1 indicates negative quantity. The algorithm implemented in the IRP calls for the DETNO output to be added/subtracted to/from the output of the K file depending on whether the sign of PERP for that case is positive/negative. For this reason the sign of PERP(MSB) is carried through to the fourth stage where it controls the select inputs to the ALU at the output of the DETNO ROM {276-8}. The PARL file is always positive. Figure 1-5 illustrates the timing for stage one.

Updated contents for PERP and PARL are written into the files in the last half of the cycle during WRTY* and WRTX*, respectively. The 74S189's go into a high impedance state during the write cycle, after which the output reflects the new data written until the address change. Transparent latches, whose output follow the input when the clock is high and latch and hold the data present at the input on the negative-going edge of the clock, are used to hold the output from the PERP and PARL files. This ensures that the proper data is maintained at the inputs to stage two through the WRT cycle.

An examination of Figure 1-5 shows that the IRP is activated the cycle after one in which a FMUL is executed, assuming the IRP enable bit is set in ICTL. FMUL*, AND, and the leading edge of IOCLK assert the signals IRPGO, ADVANCE, etc. the next cycle. IRPGO* AND WRT* cause WRTX* and WRTY*. IRPGO and the leading edge of IOCLK cause the addresses to the rotary file to change in anticipation of the next cycle. Similarly, data from an OUT instruction is not written into the files until the second half of the succeeding cycle (which is when the WRTX* and WRTY* pulses occur). For this reason the I/O bus data is latched into a holding reg {275-1} at the end of every OUT cycle so that it will be available at WRT time.

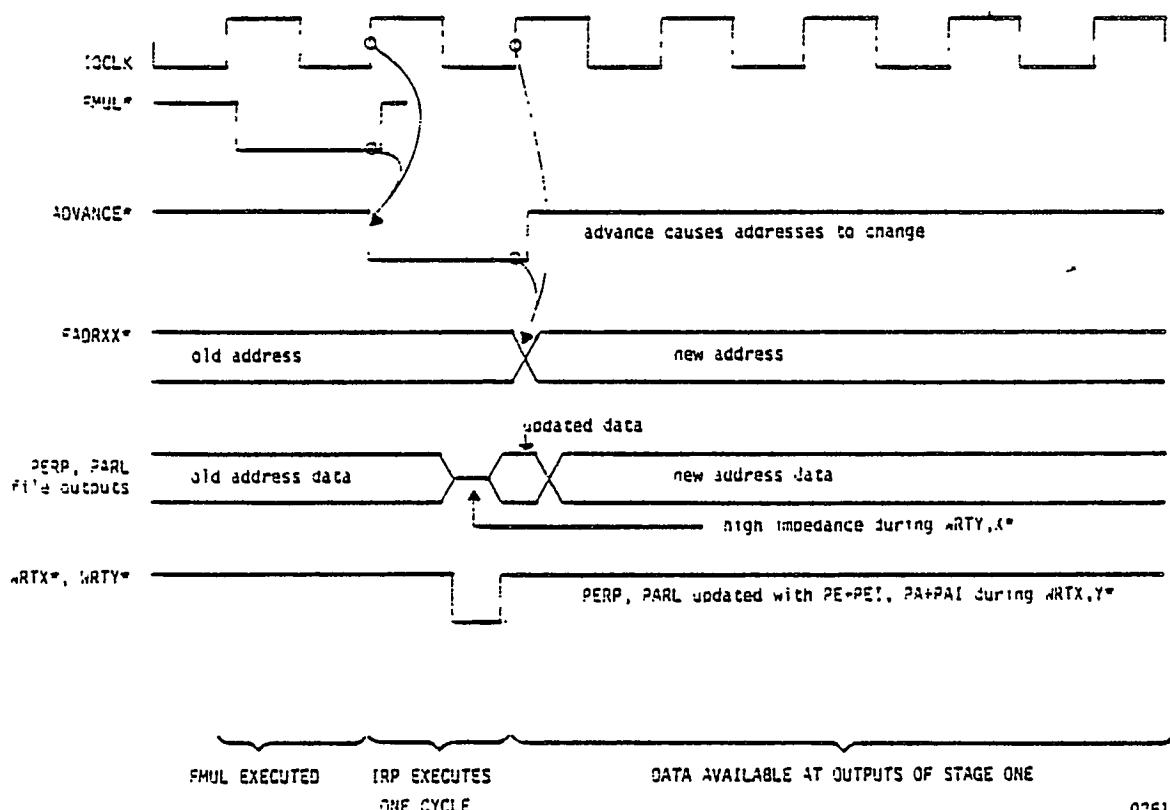


Figure 1-5 Stage One Timing Relationship

Since the PERP file output can be negative, 2's complement and round logic is included in the PERP data path {275-2}. Only 12 bits are necessary for ROM addressing; therefore the decision to round is based on the sign and the thirteenth bit according to Table 1-3.

Table 1-3 PERP Round Truth Table

Sign	Bit 13	Round
0	0	no
0	1	yes
1	0	yes
1	1	no

0263

The round and 2's complement are done in the same step -- the complement by subtracting from 0 and the round by modulating the carry in and LSB of the "A" input to the ALU.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----

PERP, PARM FILE OUTPUT

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

LOGPERP ADDRESS FIELD

YADD

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

PERPSQR ADDRESS FIELD

1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	----	----

LOGPARL ADDRESS FIELD

XFIL

1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	----	----

PARLSQR ADDRESS FIELD

WADR

5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	----	----	----	----	----	----

WEIGHT ADDRESS FIELD

DAOR

3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	----	----	----	----	----	----

DETNO ADDRESS FIELD

0264

Figure 1-6 ROM Address Field Relationships

In stage two the outputs of PERP and PTRL are used to address contents of the PES, PAS, LPE and LPA ROMs. Note that due to the difference in sizes of the LPE and PES ROMS the address fields for each ROM are taken from different significance positions within the PERP output. Refer to Figure 1-6. Note also that dummy holding registers are used to synchronize the K file output with the other data as it passes through the pipeline {275-2}.

LPE-LPA and PES+PAS are derived in stage three in two adders made up of 74S381's {276-5}. The results are used to address the WGT and DETNO ROMs. Note that the outputs of the DETNO ROMs are disabled and go into a high-impedance state by DNOZIP* {276-6}. DNOZIP* true is the result of setting the ZERO bit in ICTL. This is a diagnostic function useful in verifying proper function of the K file data paths. Also in stage three are the drivers that enable the PES, PAS, LPE, and LPA ROM output onto the I/O bus when an IN instruction to the proper device address is executed.

Only part of stage four is on the IRP boards; the rest is made up of TMRAM and TMREG. The output of DETNO is added to or subtracted from K according to the PERP sign {276-8} and the result is passed as an address to TMRAM. At the end of the cycle, the addressed data is latched in TMREG and is available to the floating multiplier. The WGT output is passed to a dummy pipeline register to synchronize with TMRAM output at the bottom of stage four. The 16-bit value held in WGT is unpacked to a 38-bit floating-point number on M2 bus. The two WGT most significant bits become the floating exponent according to Table 1-4. The remaining 14 WGT bits become the floating mantissa most significant bits. The other 13 mantissa bits not driven are zero. The value from WGT is always positive so the floating sign bit (which is not driven) is also always zero.

Table 1-4 WGT Packed to Floating-Point Exponent Conversion

Two MSBts WGT ROM: Binary Value	Octal Floating Point Exponent	True Value
00	1000	0
01	1001	1
10	0776	-2
11	0777	-1

J265

CHAPTER 2

IRP DIAGNOSTICS

2.1 INTRODUCTION

This manual tells how to run and to interpret output from the image reconstruction processor diagnostic. It is assumed the reader understands the Image Reconstruction Processor Theory of Operation.

2.2 IMAGE RECONSTRUCTION PROCESSOR DIAGNOSTIC

The IRP diagnostic (IRPTST) program was written specifically to verify proper operation of the image reconstruction processor (IRP) option in the array processor. The bulk of IRPTST is written in APAL and is resident in the array processor. A stand-alone driver that handles operator I/O and loads AP program source and data memories with the diagnostic program and necessary data tables is written in NOVA assembly language to run in the host Data General machine.

The purpose of IRPTST is to determine whether the IRP is functioning properly and, if not, to give some indication of what might be the trouble. A software simulator written in APAL is included in the test program to provide a standard to which the IRP can be compared. Diagnostic paths have been included in the hardware of the IRP so that the IRP can be divided into several functional units to simplify debugging. Nine individual pieces of the IRP are tested alone before being tested together.

Several subroutines are called on within an individual test. First, the random number generator is called to set up random input parameters for the IRP in buffers in main data memory. A load subroutine is called to set up the IRP hardware with the random inputs. The IRP is then run for 32 AP cycles the results of which are held in data pad temporarily before being moved to "actual" result buffers in main data memory. After the hardware run, the simulator is called on to produce "expected" results from the random input buffers in main data memory. Finally, a checking subroutine compares the "actual" hardware results with the "expected" simulator results for discrepancies and an error message is output to the system teletype if any differences are found. DMA transfers of random numbers are overlapped with IRP operation in order to more closely duplicate an actual operating environment.

The main driving routine that calls each test in turn makes a number of decisions about the flow of the calls to each test based on flags set by user inputs. These flags enable the operator to better understand the nature of the problem before attempting a repair. Note that IRPTST provides only a general idea of where in the hardware the failure is located. Troubleshooting down to the chip level requires the use of an oscilloscope and a working knowledge of TTL logic. Since the test runs random parameters through the IRP there is no specific end to the test. After the sequence of tests is completed the random number generator starting parameters are randomly changed and the sequence repeated. After 8096 passes through the test sequence the program prints a single character of a logging message to indicate successful operation.

2.3 USING IRPTST

IRPTST responds to a number of single-letter commands entered on the system console. These commands are used to modify the flow through the diagnostic and to simplify the isolation of hardware failures. The abbreviation CSI used below stands for Command String Interpreter, the subroutine that interprets teletype input into program commands.

Table 2-1 summarizes the commands of IRPTST. Additional details for their use follow.

Table 2-1 Operator Command Set

COMMAND	FUNCTION
Annnnnn	Set the B command jump. Must be followed by an actual integer.
B	Transfer program control to the address specified by the A command.
C	List the currently enabled user commands.
D	Disable error message typeout. Used to create a loop fast enough in the hardware to refresh an oscilloscope display. Disables error checking.
E	Execute the test program. Used to transfer control from the command string interpreter subroutine to the test subroutines. This is the last command that will be seen by the CSI. Further commands will be ignored.
F	Disable overlapped DMA during I/RP operation.
H	Return to Command String Interpreter (CSI) after executing one test case (HALT). Used to "step" through test cases until a specific one is reached.
L	Unconditionally repeat the last executed test case (LOOP).
Mnnnnnn	Define the AP Program Source size. IRPTST loads with a default size of 2000/base 8 (1024/base 10). Must be followed by an octal integer. Not normally used.
N	Display next failing case within the currently failing test.
R	Reset all flags. Clears D, H, L, W and T commands to disabled state.
Tnn	Force execution of the test number indicated by the following octal integer. The integer should not exceed 11/base 8 or the program will destroy itself.
W	Cause program execution to return to CSI after typing an error message, i.e., after detecting an error (WAIT).
Xnnnnnn,nnnnnn	Set the random number generator starting parameters to the following octal integers. Used to recreate a test case from "X,Y" parameters displayed in an error message.
Z	Enter AP hardware debugger subroutine. See explanation below.
(cr)	"Carriage return" if used to terminate the input of a command string.
(ctrl C)	"Control C" is used to delete an input command string if typed before the carriage return. It is also used to return to the CSI after the "E" command is executed.
(rubout)	Used to selectively clear one of the execution control flags (D, H, L, W and T). Rubout echoes as a slash (""). Type the flag to be cleared followed by Rubout, i.e., "L (rubout)" clears the loop flag.

NOTES

1. The CSI (Command String Interpretation) indicates a ready state by typing "*".
2. If more than six octal numerals are entered as one of the input parameters, only the last six are accepted. It is not necessary to enter leading zeros, i.e., "000123" is the same as "123".

2.4 EXAMPLES

- For normal operation, type "RWE(cr)". This starts the test running so that it returns to command input mode when an error is detected.
- To loop on a failing case: after the program has typed out an error and returned to the command mode, type "LE(cr)" to loop on the failing case to see if it will repeat solidly. Type "LDE(cr)" to enter a fast loop so that the problem can be examined with an oscilloscope.

After the problem has been corrected "D(rubout)E" repeats the case with error checking re-enabled to verify the correction.

"RWE(cr)" returns the test to its normal sequence.

- To leave the loop while scoping, or at any time quit the test sequence and return to command input mode simply type "(cntrl C)".
- For an overnight run type "RE(cr)". The program types out all failures and proceeds. If using a hard copy teletype this leaves a record of any failures. The program halts automatically after 32 failures to prevent excessive use of paper in the event of a catastrophic failure.
- Since a given test runs 32 cases through the IRP it becomes clumsy to display all failing cases within a given pass. For this reason the error output routine shows the first failing case only, even though there may be more failures in the 32 cases. Using the "N" command enables subsequent failures within a single test to be examined. Typing "NLE(cr)" displays any further failures. ("L" is used to guarantee that the current failing test won't be lost.) If there are no additional failures the program returns to the CSI, typing only a "*".

- If during the course of an overnight run, an error is output and the "W" command is not enabled, it is necessary to reset the random number parameters in order to recreate that failing case.
 1. Use the "R" command to reset all program control flags.
 2. Typing "Xnnnnnn,nnnnnn(cr)" resets the parameters. The variables to be entered are output by the error messages.
 3. Use the "T" command to force the failing test.
 4. At this point it is a good idea to set the "H" (unconditional halt) command to force a return to the CSI in the event the failure is intermittent and no error is detected on the first repetition of the test. Typing "HWE(cr)" after typing "Xnnnnnn,nnnnnn(cr)" causes the failing case to be recreated and executed once. Note that the "H" flag is automatically reset to zero once it has been used. The "H" flag must be set before each "E".
 5. Now that the case is recreated it is possible to use the "L" flag. "LWE(cr)" causes the case to repeat until the failure reoccurs (if it didn't occur on "HE(cr)"). Note that "L" must be cleared before using "X" since execution with "L" set causes the last case run to repeat and the "X" command is ignored. After getting the failing case to reappear "L" and "D" may be used as usual to debug the problem.
 6. The above commands can be entered on the same line, i.e., "RXnnnnnn,nnnnnnTmHWE(cr)" causes Test m of pass nnnnnn,nnnnnn to be executed once. Control returns to the operator whether the test fails or not.
- The "F" command allows the user to disable the overlapped DMA. This is useful in narrowing the field of possible causes of a failure. "F" can be set at any time and remains set, disabling DMA, until cleared by the operator. When an error occurs, "LE" determines whether or not it repeats. "FLE" runs the same test without DMA.

2.5 TEST DESCRIPTIONS

As mentioned before, IRPTEST is a collection of nine individual tests, each of which exercises a unique data path through the IRP. Below is a description of each test and the particular path tested. Figure 2-1 is a block diagram of the IRP showing the major signals and data buses.

2.5.1 Test 1 -- Control Register Test

A test pattern of ones and zeros is written into the IRP CTRL register from the AP I/O bus. The contents of the CTRL register is then read again over the I/O bus, and the output pattern and input data are compared. The test pattern consists of fixed series of cases: first all ones, then all zeros, and finally a "sliding bit". A "sliding bit" is a progression of cases that starts in the first case with the least significant bit set and all others cleared. Succeeding cases move the set bit to the left, one position per pass, until the last case where the most significant bit is set and all others cleared.

Error message format:

TEST 1	
EXP eeeeeee	DPA cc
ACT aaaaaaa	X,Y xxxxxxxx yyyyyyy

where

eeeeeee	is the simulated (expected) value
aaaaaaaa	is the hardware (actual) result
cc	is the number for that case (1-40/base 8) (actually the data pad address at which the result was stored.) This is an indication of which case out of the 32/base 10 cases executed per pass is failing.

xxxxxxxx yyyyyyy are the random number generator restart parameters

2.5.2 Test 2 -- K Register File Test

Test 2 writes each of the test bit patterns used in Test 1 into all 16 K registers. The IRP is initialized with the number of views set to 16/base 10 and is run to flush the K register contents through the pipeline at 6MHz. The results are read over the I/O bus.

This test verifies that the input/output control logic in the IRP and the pipeline clocks and enables are properly functioning. DETNO output is disabled during this test by the DETNO ZERO bit in the ICTL register.

Error message format:

```
TEST 2
EXP eeeeeee DPA cc
ACT aaaaaaa X,Y xxxxxxxx yyyyyyy
```

2.5.3 Test 3 -- LAMBDA Address Test

TMRAM is initialized such that each location contains its address. The K registers are filled with random addresses and the IRP is run to drive the addresses onto the BTMA lines. The results are fetched from TMRAM output. The test is successful if the TMRAM output matches the input addresses.

Both the rotary file addressing control logic in the IRP and the BTMA address lines into TMRAM are tested. DETNO output is disabled during this test by the DETNO ZERO bit in the ICTL register.

Error message format:

```
TEST 3
EXP eeeeeee DPA cc
ACT aaaaaaa X,Y xxxxxxxx yyyyyyy
```

2.5.4 Test 4 -- PES Test

2.5.5 Test 5 -- PAS Test

2.5.6 Test 6 -- LPE Test

2.5.7 Test 7 -- LPA Test

Tests 4 through 7 verify proper operation of the rotary register files and stage two ROMS. PERP, PARL, PERPINC, and PARLINC files are filled with random numbers. The IRP is randomly initialized for 1, 2, 4, 8 or 16 views and run for 32 cycles at full speed (6MHz). The results are read from the outputs of the ROMS (PES, PAS, LPE, OR LPA) over the IOBUS and stored in data pad. A simulation is run in the AP and compared to the IRP result.

These tests verify that the PERP, PARL, PERPINC, and PARLINC files, the PERP+/-PERPINC and PARL+/-PARLINC logic, and the addressing for the PES, PAS, LPE, and LPA ROMS and the ROMS themselves are all functioning properly. In other words, the major portions of the IRP pipeline stages one and two are verified.

Error message format:

```
TEST 4 (or 5, 6, or 7)
EXP eeeeeee DPA cc
ACT aaaaaaa X,Y xxxxxx yyyyyyy
```

2.5.8 Test 8 -- WGT Test

Test 8 is an extension of Tests 4 and 5. PERP, PARL, PERPINC, and PARLINC are filled with random numbers. The IRP is run as in Tests 4 through 7, and the resultant WEIGHT values are fetched from the multiplier via a multiply by 1.0. A simulator is run in the AP. Simulated results are compared with the IRP results.

This test checks that the PES+PAS adder, the WGT ROM, and the drivers for M2BUS all work.

Error message format:

```
TEST 10
EXP eeeeeee eeeeeee eeeeeee DPA cc
ACT aaaaaaa aaaaaaa aaaaaaa X,Y xxxxxxxx yyyyyyy
```

Since the results are 38-bit floating-point numbers, the expected and actual data is unpacked into three words. The first word shows the 10-bit exponent; the second the 12-bit high mantissa; and the third word shows the 16-bit low mantissa. TEST 10 is indicated since the output is always in octal format.

2.5.9 Test 9 -- DETNO Test

Test 9 is similar to Test 8. PERP, PARL, PERPINC, PARLINC and K are filled with random numbers. The IRP is run as in tests 4 through 7 with results read from the I/O bus to data pad. A simulator is run in the AP. Simulated results are compared with the hardware.

Since the K file path is verified in test 3, test 9 is intended to find errors in the LPE-LPA logic, DETNO ROM, and K+/-DETNO ALU.

Error message format:

```
TEST 11
EXP eeeeeee DPA cc
ACT aaaaaaa X,Y xxxxxxx yyyyyyy
```

2.5.10 DMA Test

A direct memory access transfer of random numbers is run concurrently with IRP operation in Tests 4 through 9. The results of the transfer are checked in the AP and flags are set to indicate errors.

The DMA is run primarily to verify that the AP SPIN* condition does not induce errors in the IRP. Errors are reported only to aid in debugging any IRP failures.

One of two possible error messages will be output:

DMA REGISTER ERROR

	HMA	WC	CTRL	APMA
EXP	hhhhhh	wwwww	cccccc	aaaaaa
ACT	hhhhhh	wwwww	cccccc	aaaaaa

--OR--

DMA ERROR

EXP	eeeeee
ACT	aaaaaa

"DMA REGISTER ERROR" occurs when the final contents of the four DMA registers in the interface do not match the expected values. The data transferred is not checked.

"DMA ERROR" is output when the four DMA registers are in the proper state but the data transferred contains errors.

APPENDIX A

A.1 LOADING AND RUNNING IRP DIAGNOSTIC

The nine IRP tests for both 289 and 511 detector ROM configurations are contained in five separate NOVA assembly language programs. The tests are separated because of core limitations in the host Eclipse. A single program containing both the AP microcode tables and the ROM data tables for both 511 and 289 detector configurations could exceed 64K of host storage. Table A-1 outlines filename versus function for all IRP test versions.

Table A-1 Function Filename

Function	Detector Configuration	
	289	511
tests 1,2,3	IRPTS3	IRPTS3
tests 4,5,8	E1D289	E1D511
tests 6,7,9	E2D289	E2D511

0267

The typical starting procedure under RDOS to run, for example, tests 6, 7, and 9 on a 511 configuration IRP is:

BOOT E2D511(cr)

(assuming the directory in which the boot occurs contains the save file for E2D511).

The diagnostic responds with:

IRPTEST REV 3.0 3/78 TESTS 6, 7 and 9 (cr)

*

The "*" indicates the diagnostic is ready to accept command input. At this point, the user should enter "RWE(cr)". The diagnostic begins execution and the "Array Processor Running" light on the AP control panel lights. The test runs until it is stopped or it encounters an error. Successful operation of the IRP is indicated by a logging message printed on the system console. A single character of the message is printed for each set of 8096 successful pass through all the tests within a given program.

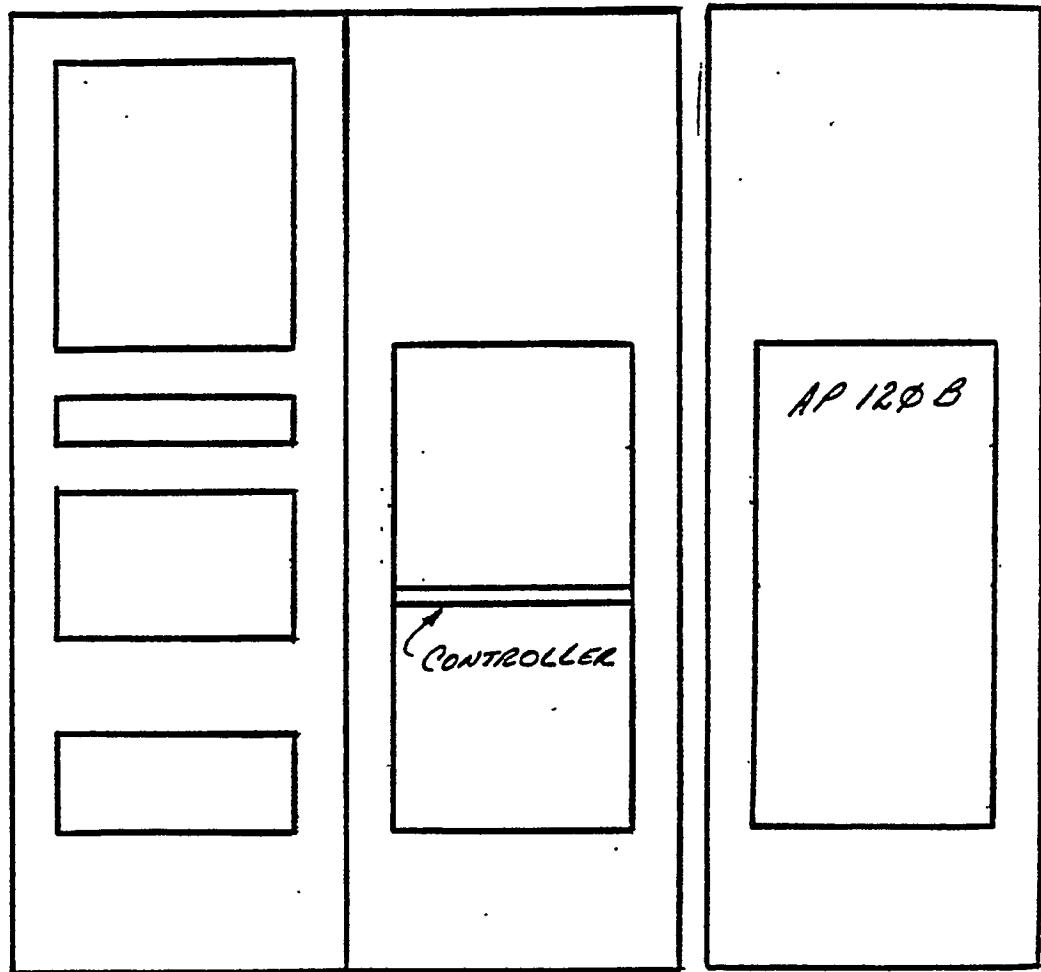
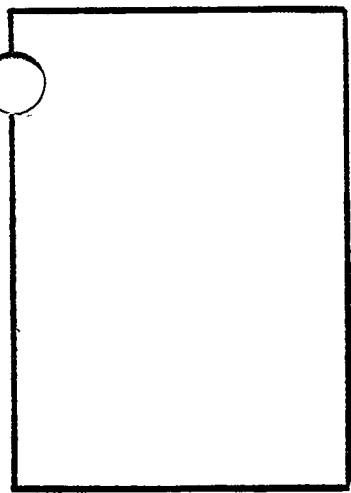
Mantissa
12345 $\times 10^6$ — *Exponent*
Base

**INITIAL
HANDOUTS**

FAST RECONSTRUCT PROCESSOR

AP120B

GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

Where to Start

Guide to Chapter 8

This chapter is divided into three parts. Part I is basically an introduction to the use of the AP diagnostics. This has been separated from the rest of the chapter; because once the user becomes familiar with the AP diagnostics, he need not refer to this part any longer. Part II gives the particulars of each diagnostic routine. This includes how to use each diagnostic and types of error patterns for which to look.

Finally, Part III consists of two sections. The first being block diagrams of AP hardware, and the second section includes a back-plane signal glossary. This glossary gives a cross-reference between signal name and function.

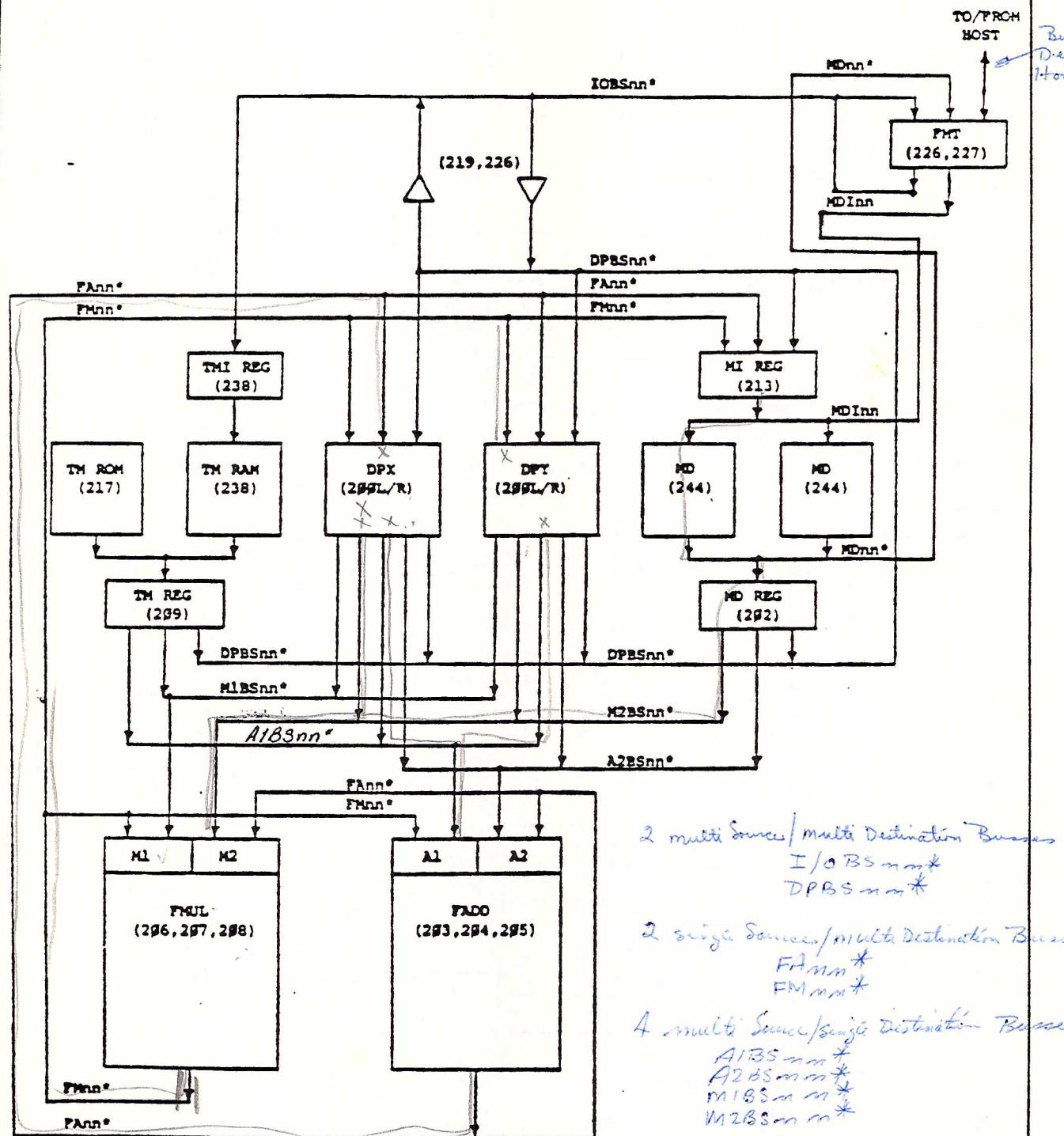
It is suggested the user follow the section of the current diagnostic being run. If an error occurs, find the area in question on the block diagrams. This should give one a better feel of what is being tested. From this point one can go to the signal glossary for hints on what might be causing the failure, or go directly to the schematics. This shall be covered in more detail with the related routine.

Initial functional check/troubleshooting

After the AP has been powered up and voltages have been checked, diagnostics should be run to confirm proper operation of the unit. All diagnostic tests should be run in the order listed below. This also includes the general approach to troubleshooting. If an error is encountered, proceed to the section describing the test in use.

- | | |
|---------------------------|--|
| 1. AP TEST | AP memory test |
| 2. AP PATH | AP data path test |
| 3. AP ARTH | AP arithmetic |
| 4. PS TEST | Program source memory test |
| 5. TM ROM | Table memory ROM test |
| 6. TM RAM | Table memory RAM test |
| 7. IRPTS3 | Image reconstruction processor board test (275) |
| 8. a) E1D289
b) E2D289 | PROM tests for CT/T with standard resolution DAS/Detect configuration. Tests 276 board |

GE MEDICAL SYSTEMS INSTITUTE



2 multi Source / multi Destination Busses
 I/o BSnn*
 DPBSnn*

2 single Source / multi Destination Busses
 FA nn*
 FM nn*

4 multi Source / single Destination Busses
 A1BSnn*
 A2BSnn*
 M1BSnn*
 M2BSnn*

FIGURE SYS 3
 Array Processor 38-Bit Data Paths

Bus size
 Depends on
 Host Bus Size

GE MEDICAL SYSTEMS INSTITUTE

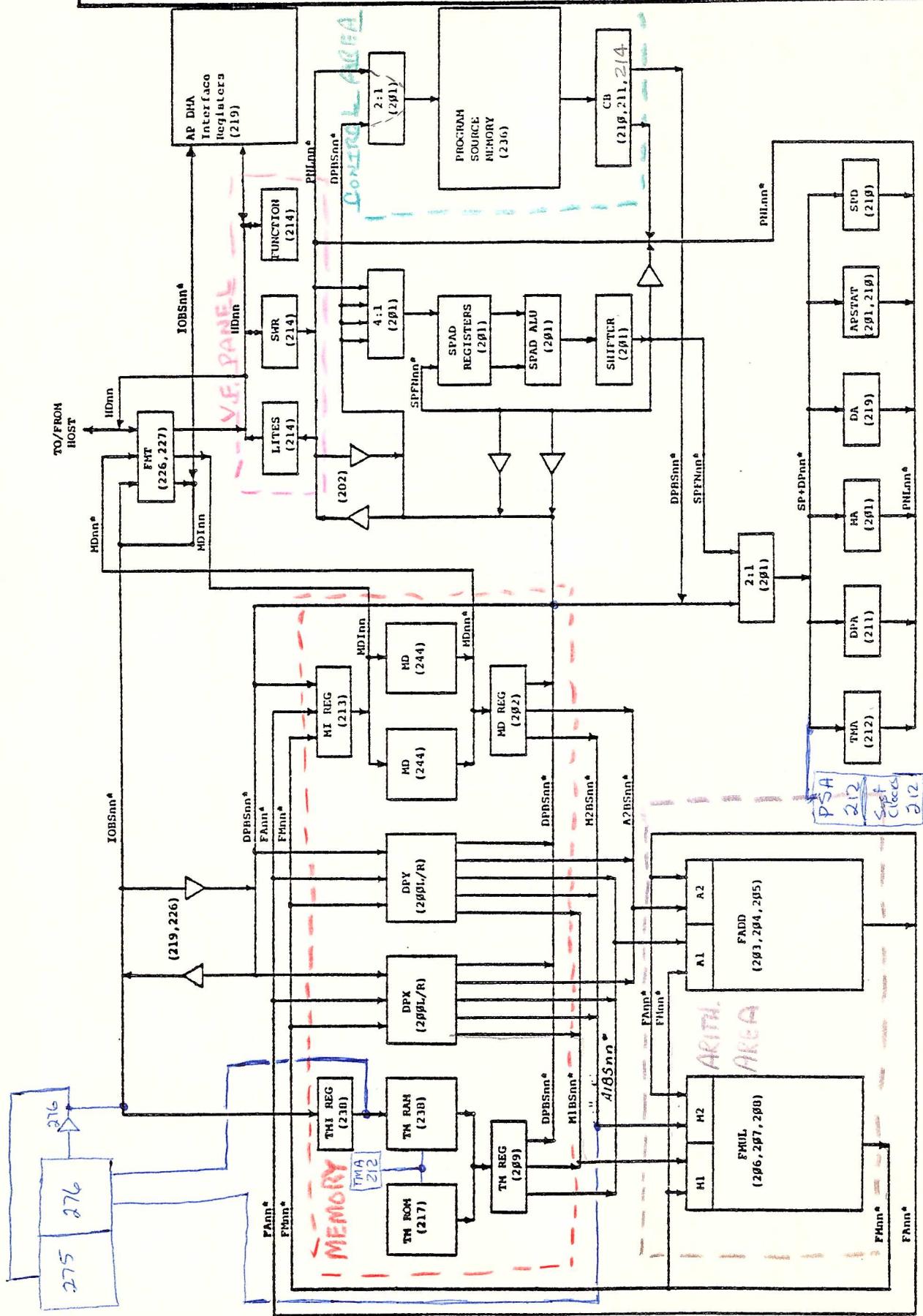


FIGURE SYS4
Array Processor Data Paths

GE MEDICAL SYSTEMS INSTITUTE

ARRAY PROCESSOR ETCH CIRCUIT BOARD LIST

BOARD NUMBER	MNEMONIC	NAME
200	DPL	Data Pad Left
200	DPR	Data Pad Right
201	SPAD	Scratch Pad
202	MDREG	Main Data Register
203	FADD1	Floating Adder 1
204	FADD2	Floating Adder 2
205	FADD3	Floating Adder 3
206	FMULA	Floating Multiplier A
207	FMULB	Floating Multiplier B
208	FMULC	Floating Multiplier C
209	TMREG	Table Memory Register
210	CB1	Control Buffer 1
211	CB2	Control Buffer 2
212	ADDR	Address
213	MIREG	Memory Input Register
214	EXPAN	Expansion of Control Buffer 1
217	TMROM	Table Memory Read Only Memory
219	GPIF	General Purpose Interface
226	FMTM	Format Mantissa
227	FMTE	Format Exponent
236	PSRAM	Program Source Random Access Memory
238	TMRAM	Table Memory Random Access Memory
244	MD	Main Data - Data in From Host & To Host
275	IRP1	Image Reconstruction Processor Number 1
276	IRP2	Image Reconstruction Processor Number 2
280	IOADP	Eclipse I/O Adapter

GE MEDICAL SYSTEMS INSTITUTE

VIRTUAL FRONT PANEL SIGNAL GLOSSARY

ABORT*	Reset the Array Processor
CAP2PNL*	Disable PSA2PNL, MA2PNL or TMA2PNL when the executing program places something on the Panel Bus
!CBCLK1	Control Buffer One (CB1) Clock
!CBCLK2	Control Buffer Two (CB2) Clock
CBCLKE*	Control Buffer Clock Enable (Load)
CCLR	Clear I/O Pulse from Host
DATA00*-DATA15*	Host's 16-Bit I/O Data Bus
DATIA	Data in A I/O Instruction from Host
DATIB	Data in B I/O Instruction from Host
DATIC	Data in C I/O Instruction from Host
DATOA	Data out A I/O Instruction from Host
DATOB	Data out B I/O Instruction from Host
DATOC	Data out C I/O Instruction from Host
DS0*-DS5*	I/O Device Select 0 through 5 Lines from the Host
FN2HD	Function to Host Data Enable
FN2HD*	Function to Host Data Enable (Low True)
HD00*-HD15*	16-Bit Host Data Bus
HRSET*	Host Reset
IORST	I/O Reset
LDFN*	Load Function Register Clock
DLSR*	Load Switch Register Clock
LT2HD	Lights to Host Data Enable
LT2HD*	Lights to Host Data Enable (Low True)

GE MEDICAL SYSTEMS INSTITUTE

9. a) E1D511
b) E2D511

PROM tests for CT/T with high resolution DAS/Detector configuration. Tests 276 board.

10. a) FDFFT
b) FCFFT

Forward/Inverse Fast Fourier Transforms. Intended as exercises to confirm AP operation.

Two (2) passes of each diagnostic should catch any hard failures.

Each test takes approximately 30 seconds to two (2) minutes, except FDFFT and FCFFT. They take in the order of 25 minutes.

IMPORTANT NOTE:

Each diagnostic acknowledges successful completion of a pass by typing a letter (i. e. "A", "T", or "P"), except TM ROM. TM ROM should be run for three (3) minutes. IRPTS3, E1D289, E2D289, E1D511 and E2D511 spell out IMAGE RECONSTRUCTION PROCESSOR as it completes passes. Again, each letter is a successful pass.

Loading AP Diagnostics

1. Press "STOP-RESET" on computer function switches.
2. Mount and thread the DTOS Tape.
3. Place the tape unit on line.
4. Place 100022 in computer data switch (only 9, 11, 14 switches up).
5. Press "PROG LOAD" on the computer function switches.

After following the above steps the DTOS banner should appear, giving the current revision of the tape. If this does not occur, repeat steps 1-5 above. (See banner below.)

GENERAL ELECTRIC MEDICAL SYSTEM
CT/T
COMPUTER SUBSYSTEM DIAGNOSTIC TAPE

GE MEDICAL SYSTEMS INSTITUTE

TOP OF MEMORY = \$77777

DTOS REV NEW = 2

*

In response to the "*" type:

LOAD ----- 'CR'

where:

----- is the mnemonic of the particular diagnostic desired
They are as follows:

AP TEST
AP PATH
AP ARTH
PS TEST
TM ROM
TM RAM
IRPTS3
E1D289
E1D511
E2D289
E2D511
FDFFT

~~E2FFT~~ ~~We do not use this test in our application~~

These mnemonics must be typed exactly as shown above. If desired, a directory listing of DTOS may be output by typing DIR 'CR' in response to the asterisk.

AP Diagnostic User Commands

Given below is a list of user commands followed by some typical uses.

<u>USER</u>	<u>FUNCTION</u>
<u>COMMAND</u>	

Axxxx	Input the starting address of another program in core to which control is to be transferred by the 'B' command.
-------	---

An example in the 'B' command below demonstrates the usefulness of the 'A'.

GE MEDICAL SYSTEMS INSTITUTE

- B Transfer program control to the address specified in the last preceding 'A' command.
1. EX:A77777B'CR' will return program control back to DTOS. This prevents the user from having to reboot the tape.
 2. EX:A600 B'CR' will return program control back to the current diagnostic.
- C Type out the user input flags that have been set.
- D Set the typeout Disable flag. Used for Scoping a hardware fault. Also disables the error detection.
- E Execute the test program used to transfer control from the command input section of the program to the test section.
This must be the last command in the entered command string
- Fnnnn Type out the micro-instruction (Function) at PSA location nnnn. PSA location 4 corresponds to the first randomly selected micro-instruction. The PSA value will be typed on a line followed by the micro-instruction represented as four octal numbers. At the end of the line, the user can type either a line-feed to cause the program to type out next PSA value and micro-instruction or a carriage return to cause the program to return to command input mode.
Used only in AP ARTH. This instruction will be discussed extensively in the section describing AP ARTH.
- H Set the unconditional Halt flag. Following the next 'E' command, the program will execute one test case and return to command Input mode.
- I Set the IO Reset flag. Test program Resets the AP-120B between each test case.

GE MEDICAL SYSTEMS INSTITUTE

<u>USER COMMAND</u>	<u>FUNCTION</u>
L	Set the LOOP flag. Program loops on the last executed test case.
Mnnnn	Define the AP-120B Memory Size.
Nnnnn	Select the Number of micro-instructions (1 - 17 ₈) that are to constitute each test case. Once N is set, the Random Number Generator is inhibited from selecting the size of each Test case. This condition will persist until N is cleared by either the 'R' or Rubout command. The "N" command is used to force the selection of simpler test cases in order to facilitate their interpretation in the case of a solid hardware failure. Again, this command shall be covered in more detail during the discussion of AP ARTH
R	Reset all flags. Clears D, H, I, L, S and W.
S	Set short form format for error typeout. Affects FFT tests only.
W	Set Wait on error flag. If set, the test will return to command input mode after encountering and typing out an error.
Xnnnn, nnnn	Reset the Random Number generator parameters. Used to recreate a test case from the parameters typed out in an error message.
EX:	<u>X054770,066024 'CR'</u> If errors appear to be data related, the "X" command will restart the random number generator to recreate a previously failing case.

GE MEDICAL SYSTEMS INSTITUTE

<u>USER COMMAND</u>	<u>FUNCTION</u>
Rubout	Used to selectively clear one of the flags (D, H, I, L, S, N, S, or W). Type the flag to be cleared followed by a rubout i. e., "Rubout" clears the Loop flag.
'Control' C	Used to break test mode and re-enter to the command input mode. May be typed at any time.
#nnn	Enables the user to examine various parameters utilized during the execution of AP ARTH. Example of its use shall be given in the description of AP ARTH, along with a complete table of arguments (nnn) for the particular parameters of interest.

Typical Command Strings

1. RWE 'CR'

R - Reset all flags

W - Wait on error -

When an error is encountered, print the message and return to the command input mode.

E - Execute this command string

This command (RWE) is the normal command string for all AP diagnostics to be run.

2. LDE "CR"

L - Loop on error

D - Disable printout

E - Execute command string

This command (LDE) is useful for scoping an error.

GE MEDICAL SYSTEMS INSTITUTE

3. ~~RE 'CR'~~

R - Reset all flags

E - Execute command string

This command (RE) is useful for collecting various errors and data. An internal loop inhibits the output and returns to the command input mode after encountering 64 errors.

4. ~~RNIWE'CR'~~

R - Reset all flags

N1 - Run 1 random instruction

W - Wait on error

E - Execute command string

This command (RNIWE) is the normal string to be entered while running AP ARTH after an error has been encountered. Otherwise, RWE should be used.

5. ~~E'CR'~~

E - Execute command string

This command (E) executes the command string. This includes any flags previously set and not cleared, i.e. W, L, N, D.

Important Note:

1. In all cases, an "*" (asterisk) must appear on the CRT before DTOS or any of the AP diagnostics will accept command inputs.
2. Any numeric data needed in the AP diagnostic routines must be entered in octal format and all error codes/data are output in octal format.

GE MEDICAL SYSTEMS INSTITUTE

AP TEST

Brief Description

This stand-alone, Nova assembly-language program exercises the Panel and DMA interface functions of the AP-120B. It tests all of the available memory inside the AP-120B with simple patterns and then with pseudo-random number patterns.

Error Messages

TEST 1 checks the WC, HMA, CTRL, APMA, APSR, and FN registers. Error Message Format:

E nnnn A mmmm cccc

Where nnnn is the expected value,
mmmm is the actual value,
and cccc is a code number between 0 and 5.

TEST 1 Code Number description:

<u>Code Number</u>	<u>Register under Test</u>	<u>Most Likely Failing Board or Boards</u>
0	WC	Interface (219, 280)
1	HMA	Interface (219, 280)
2	CTRL	Interface (219, 280)
3	APMA	Interface (219, 280)
4	APSR	214
5	FN	214

TEST 2 and TEST 2A check the ability of the AP-120B Panel to access all of the internal registers that are available to it.

ERROR output format is the same as TEST 1 except that the code number will be in the range 6 through 31 (except for PS word 2= 100020).

GE MEDICAL SYSTEMS INSTITUTE

TEST 2 and TEST 2A Code Number description:

<u>Code Number</u>	<u>Register Under Test</u>	<u>Most Likely Failing Board or Boards</u>
6	PSA	212, 214, 210
7	SPA	210, 201, 214
10	MA	201, 214
11	TMA	212, 214
12	DPA	211
13	SPAD	201
14	APSTATUS	210, 201 (if low 3 bits)
15	DA	Interface (219, 280)
16	PS word 0	236, 210, 201
17	PS word 1	236, 201
100020	PS word 2	236, 201
20	PS word 3	236, 201
21	DPX EXP	200L, 211
22	DPX HMAN	200R
23	DPX LMAN	200L, 200R
24	DPY EXP	200L, 211
25	DPY HMAN	200R
26	DPY LMAN	200L, 200R
27	MD EXP	244, 202, 213, 210
30	MD HMAN	244, 202, 213
31	MD LMAN	244, 202, 213

GE MEDICAL SYSTEMS INSTITUTE

TEST 3 performs a basic address test on all the memories in the AP-120B. The address of each memory location is written into that location until all locations have been written, then the entire memory is read back and verified.

TEST 3 error message format:

BASIC ADDR E nnnn A mmmmm cccc

Where nnnn is the expected value and is also the address of the failing location,

mmmmmm is the actual result

and cccc is the code number between 32 and 47 that identifies the memory under test.

See TEST 4 for the Code Number descriptions.

TEST 4 performs a random patterns test of each memory by filling it with pseudo-random numbers, and reading them back to check.

TEST 4 error message format:

RND PATTERNS E nnnn A mmmmm CODE = cccc ADDR = aaaa
X, Y = xxxx yyyy

Where nnnn is the expected value,

mmmmmm is the actual value (note that only the bits in this word that lie within the width of the memory portion under test are checked)

cccc is a code number between 32, and 47,

aaaa is the address of the failing location

and xxxx yyyy are the restart parameters for the random number generator.

GE MEDICAL SYSTEMS INSTITUTE

TEST 3 and TEST 4 Code Number descriptions:

<u>Code Number</u>	<u>Memory Under Test</u>	<u>Most Likely Failing Board or Boards</u>
32	SPAD	201, 210
33	DPX EXP	200L, 211
34	DPX HMAN	200R
35	DPX LMAN	200L, 200R
36	DPY EXP	200L, 211
37	DPY HMAN	200R
40	DPY LMAN	200L, 200R
41	PS0	236, 212
42	PS1	236
43	PS2	236
44	PS3	236
45	MD EXP	244, 201, 210
46	MD HMAN	244
47	MD LMAN	244

TEST 5 checks the DMA to DMA interface by transferring Host memory contents into AP-120B Main Data memory in 16-bit integer mode and then checking the contents of MD against Host memory by using the AP-120B Panel.

TEST 5 error message #1:

MEMTST LOAD ERROR

WC wwwwww HMA hhhh APMA aaaa CTRL cccc

GE MEDICAL SYSTEMS INSTITUTE

The correct values for wwww, hhhh, aaaa, and cccc, are \$000000, \$200000, \$200000 and 100102.

Where wwww, hhhh, aaaa, and cccc are the respective contents of the AP-120B WC, HMA, APMA, and CTRL registers after the completion of the DMA transfer.

This error indicates that the transfer did not go to completion correctly. The interface board (Bd. 280) is the most likely failing Board in this case.

TEST 5 error message #2:

MTEST CHECK LOC = 1111 E eeee A aaaa

Where 1111 is the MD location having incorrect contents,

eeee is the expected value (host memory contents)

and aaaa is the actual value.

This error generally indicates a failure in the Interface, or Format Boards, however, the problem could be in Main Data Memory itself, Board 244.

TEST 6 error message #1:

DMA STORE NOT DONE

WC wwww HMA hhhh APMA aaaa CTRL cccc

See description of TEST 5 error message #1.

Correct values here are WC = 0, HMA = APMA = 13777, CTRL = 100142.

TEST 6 error message #2:

MSTORE LOC = 1111 E eeee A aaaa

Where 1111 is Host memory location having incorrect contents.

eeee is the expected (MD) value

and aaaa is the actual value.

GE MEDICAL SYSTEMS INSTITUTE

This error generally indicates an Interface or Format (Bd. 226, 227) board failure, however, the failure could be in MD, Board 244.

TESTS 7 and 8 are not implemented for the NOVA/Eclipse Interface.

TEST 9 tests NOVA format to AP-120B floating point conversion.

Error Message:

TEST 9/10 E eeee eeee eeee

A aaaa aaaa aaaa ANSP pppp

TEST 10 tests AP-120B to NOVA format conversion.

Error Message:

TEST 9/10 E eeee eeee

A aaaa aaaa ANSP pppp

Notes on Tests 9, and 10:

- (1) The only difference in the error messages between Tests 9 and 10 is in the number of octal numbers typed out. For NOVA/Eclipse to AP-120B conversion (Test 9), 3 words each of expected and actual results are typed out. For AP-120B to NOVA/Eclipse conversion (Test 10) 2 words each are typed out.
- (2) Errors in tests 9 and 10 can generally be traced to the interface or format Boards (219, 227, 226).

Use of AP TEST

After booting the DTOS tape, type LOAD AP TEST 'CR'.

DTOS will respond by acknowledging the loading of AP TEST. Upon completion of the loading an "#" will appear. Type RWE 'CR'.

To troubleshoot problems with AP TEST it is necessary to observe patterns in the error messages. Since AP TEST tests the various memories in the AP, some suggestions shall follow to troubleshoot the different memories.

GE MEDICAL SYSTEMS INSTITUTE

- a. DATA PAD - DPX and DPY are located on the 200 L/R boards. If a bit is being lost, try swapping the two 200 L/R boards. The lost bit should move. The 200 boards are the only two boards in the AP that may be swapped.

If none of the data matches, this may indicate an addressing problem. Observe addresses - are they all high or all low? From the block diagram one will note the data pads are divided into high and low groups.

Another trick is: possibly only the exponent portion or the high mantissa or low mantissa may be in error. This could suggest the separate clocks for each section of the word.

EX:	!XECLK	DPX exponent clock
	!XHMCLK	DPX high mantissa clock
	!XLMCLK	DPX low mantissa clock

Refer to the block diagrams, schematics and the signal glossary for other suggestions.

- b. MAIN DATA - MD is located physically on the 244 board. Included with the block diagrams is a drawing showing board layout.

Again, look for patterns in the errors. Is a single bit missing? If it is a single bit, set AP TEST to Loop (LDE 'CR') and monitor that bit.

Like the 200 board, main data can selectively read and write into the exp., high mantissa, and the low mantissa. In this case, the signal names are:

WEEA	}	write enable exponent (A, B)
WEEB	}	
WEHA	}	write enable high mantissa (A, B)
WEHB	}	
WELA	}	write enable low mantissa (A, B)
WELB	}	

Instead of being divided into high and low addresses as DPX, DPY; main data is divided into odd and even addresses. Look for this odd, even pattern in the addresses of the error messages.

GE MEDICAL SYSTEMS INSTITUTE

c. PROGRAM SOURCE memory is rather straight forward. It consists of 1K of PS memory (bits 0 - 63) words 0 - 3. This is apparent when running PS TEST.

In all of the above cases, some consideration must be given to the trigger point to be used. Typically the write enable signals are helpful for triggering.

When pursuing addressing problems, keep in mind these problems are typically a result of open buffers or shorted address lines. Open buffers can be found by comparing input with the output. The signals should be either duplicates or the output should be inverted depending upon the particular buffer chip.

If address lines are shorted, a typical waveform below illustrates this problem.



Note the portion of the waveform where one line is trying to pull high while the other line of the shorted pair is trying to keep the line low.

GE MEDICAL SYSTEMS INSTITUTE

AP PATH

Brief Description

This stand-alone, assembly-language program tests the bulk of the internal data paths within the AP-120B. In contrast to the paths and registers tested by the program APTEST, the data paths and registers tested by the APPATH require the execution of AP-120B micro-instructions. Thus this test also effectively checks most of the AP-120B micro-instruction set.

Detailed Description of Tests

Each path is tested with the following data patterns; zero, all-bits on, and a single true bit which is passed through all bit positions in the path under test starting from the least significant bit and moving left to the most significant bit.

Error output Format:

CODE cccc EXP eeee eeee

ACT aaaa aaaa aaaa PLOC pppp

Where cccc is a code number that indicates the type of test being performed.

eeee etc is a packed 2 to 38-bit value indicating the expected result.

aaaa etc is a packed 2 to 38-bit value indicating the actual result.

and pppp is pointer to the test table entry in the listing corresponding to the current test case. The listing is not provided because one would have to be able to read AP assembly language to decipher the table. Instead, flowcharts have been provided indicating the path under test.

Description of error codes and board level Diagnostic indications.

GE MEDICAL SYSTEMS INSTITUTE

<u>Code</u>	<u>Unique Path or Paths Under Test</u>	<u>Most Likely Failing Board or Boards</u>
100	PS Left half to DPBS	210, 214
101	PS Right half (FP Value) to DPBS	210, 211
102	DPBS to PS Left half	201, 210
103	DPBS to PS Right half	201, 210
104	DPBS (approximation portion of mantissa) to SPAD	201, 210
105	DPBS (Exponent portion) to SPAD	201, 210
106	DPBS to I/O Bus to WC	280, 219, 214
107	DPBS to I/O Bus to HMA	280, 219
110	DPBS to I/O Bus to CTRL	280, 219
111	DPBS to I/O Bus to APMA	280, 219
112	DPBS to I/O Bus to FMT	226, 227
113, 114	Not Implemented	
115	MD to FMT	280, 219, 226, 227
116	DPX to A1 FA to M2 FM to DPY	200L, 200R 203, 204, 205 206, 207, 208
117	DPX to A2 FA to DPY	200L, 200R 203, 205, 204
120	DPY to A1 FA to DPX	200L, 200R
121	DPY to A2 FA to MD	200L, 200R, 213
122	FA to A2	203, 205

GE MEDICAL SYSTEMS INSTITUTE

<u>Code</u>	<u>Unique Path or Paths Under Test</u>	<u>Most Likely Failing Board or Boards</u>
123	DPY to M2 FM to DPX	200L, 200R 206, 207, 208
124	DPY to M1 FM to DPY	200L, 200R 206, 208
125	DPX to M2 FM to MD	200L, 200R, 213
126	DPX to M1 FM to A1	200L, 200R 203, 205
127	MD to M2	202
130	MD to A2	202
131	TM to DPBS	217, 209
132	TM to A1	209
133	TM to M1	209
134	VAL to DPBS (mantissa) SPFN to MA	211 201
135	SPFN to DPBS (mantissa)	201
136	VAL to DPBS (exponent) SPFN to DPBS (exponent)	211 201
137	SPFN to A2 EXP	212, 200L
140	SPFN to A2 mantissa	212, 200L, 200R
141	Not Implemented	
142	VAL to EXIT EXIT to PNLBUS PNLBUS to SPAD	212, 201

GE MEDICAL SYSTEMS INSTITUTE

<u>Code</u>	<u>Unique Path or Paths Under Test</u>	<u>Most Likely Failing Board or Boards</u>
143	VAL + PSA to EXIT (relative JSR)	212
144	SPFN (DPBS) to TMA TMA to EXIT	212
145	VAL to PSA (JMPA)	212
146	VAL + PSA to PSA (JMP)	212
147	EXIT to PSA (Return Jump)	212
150	TMA to PSA (JMPT)	212

NOTES:

Each path typically uses more paths than the unique ones indicated (see flow charts below). The choice of indicated paths is based on the premise that previously tested paths have been verified and are thus no longer under test as the test proceeds.

The mnemonics used in the above Table are described in detail in the AP-120B Processor Handbook.

AP PATH should be used the same as the other diagnostics.

1. Boot DTOS
2. Lead AP PATH
3. Enter "RWE" command

If an error occurs, type E 'CR'. Repeat this (E 'CR') until the error messages fill the CRT screen. With the list of error codes go to the flowcharts (page 11/8) and find a common path. This can isolate the problem as far as a particular input of a board (i. e. the A2 input of the floating adder.)

GE MEDICAL SYSTEMS INSTITUTE

Once this has been done, enter LDE 'CR' to loop on the problem for scoping. Typically a good trigger point is RUN # at A1O on the 214 board.

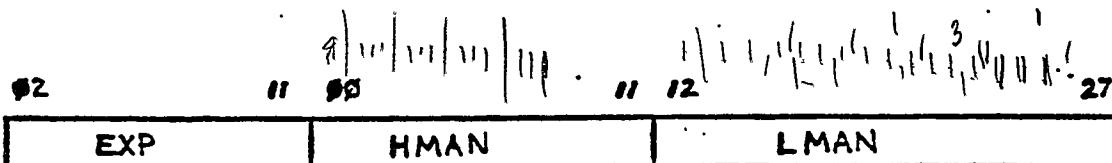
To isolate a bit, compare the expected to the actual data and note which bit is missing.

Example: CODE 121 EXP 000077 173777 177777
ACT 0000 172777 177777 PLOC 3225

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28

The fourth octal digit in the middle word does not compare. In fact, it is the least significant bit in that digit. Now count bits starting from the rightmost bit (remember there are only 16 bits per word). The bit that is different from the expected value is the 26th bit from the right.

Format of the 38 bit data word below:



To find the actual name for the bit being lost, subtract your count above from twenty-eight ($28 - 26 = 02$.) Now look up Mantissa bit 02 on the schematics at the appropriate board, which has been isolated from the various error codes. Now proceed to scope the problem to find the actual point where the bit is lost.

GE MEDICAL SYSTEMS INSTITUTE

AP ARTH

Brief Description

This stand-alone, assembly-language program exercises and verifies the accuracy of the arithmetic hardware in the AP-120B (FA, FM and S-PAD). In addition, due to the heavy use of Data Pad and S-Pad registers, the functioning of these registers is thoroughly tested.

The program utilizes a pseudo-random number generator to produce arguments for Data Pad and S-Pad, to select DPA and the Decimate shift count, and also to select combinations of Floating Adder and S-Pad operations, Data Pad Read and Write Indices, and S-Pad register addresses which are combined into micro-instructions for the AP-120B to execute.

The number (1 to 17 octal) of randomly selected instructions is also selected by the Random Number Generator. A user command ('N') is provided, however, to override this selection and force the generation of simpler test cases in order to help simplify their interpretation.

After generating the data and instructions, the program loads them into AP-120B Data Pad, S-Pad, DPA, APSTATUS and Program Source and starts the AP-120B executing the test. The program then loads the data into a corresponding set of software simulation registers and calls on the simulation package (SPADS) to generate the expected results.

The program then checks all of S-Pad, Data Pad and the APSTATUS register against the predicted results. If any discrepancies are encountered, the expected and actual results are typed out along with the restart parameters for the Random Number Generator.

Error Message Format

The full APARTH error message has the following format:

- 1) APSTATUS E eeee A .aaaa

GE MEDICAL SYSTEMS INSTITUTE

- 2) SPD ssss E eeee A aaaa
- 3) DPX dddd E eeee eeee eeee A aaaa aaaa aaaa
- 4) DPY dddd E eeee eeee eeee A aaaa aaaa aaaa
- 5) N nnnn X, Y xxxx yyyy APSTATUS pppp DPA dddd

Where in general, eeee stands for the expected and aaaa stands for the actual result. Line 1) will appear only if there is a discrepancy in the status register. Line 2) appears if there is a discrepancy in an S-Pad Register. In Line 2), ssss is the address of the S-Pad Register in error. Line 2) could appear up to 16 times if every S-Pad Register were wrong. Line 3) appears if there is an error in a Data Pad X Register. Line 4) appears for an error in a Data Pad Y Register. In lines 3) and 4), dddd stands for the absolute address of the Data Pad Register in error. Lines 3) and 4) could appear up to 8 times each if all Data Pad Registers were in error. Line 5) appears if any of lines 1) to 4) are present indicating an error. Line 5) contains: nnnn, the size of the test case; xxxx yyyy, the restart parameters for the random number generator; pppp the expected value for the APSTATUS Register (same as eeee in Line 1 if Line 1 appears); and dddd, the contents of DPA, the Data Pad base address register.

Error Message Interpretation

The program begins by using the Random Number Generator (GRN) to select the number (ENP) of functions that will constitute the test case. If the user has set the N flag the number of functions is set equal to the user determined parameter "EN". The program then uses the GRN to select input values for S-Pad (16 by 16-bits) and Data Pad (8 by 38-bits for DPX and also for DPY).

It then attacks the problem of generating the Random micro-instructions. These instructions are placed into a block following the location CODEP in the listing. In order to insure that the machine is in a known state prior to the execution of the selected instructions, the block of code begins with a four instruction header that is used to fill the Multiplier and Adder pipelines with zeros.

GE MEDICAL SYSTEMS INSTITUTE

Header Micro-Code:

000000 000001 FADD ZERO, ZERO: FMUL TM, MD

155000

000000

017400

000001 000001 FADD ZERO, ZERO: FMUL TM, MD

155000

000000

017400

000002 000001 FADD ZERO, ZERO: FMUL TM, MD

155000

000000

017400

000003 000003 LDAPS; DB DECIMATE Count

106000

002000

00000n

Where n is set by the program to equal the selected value of the Decimate shift count. Starting at location 4 then the program generates ENP random instructions that consist of randomly selected Decimate, SOP, SH, SPS, SPD, FADD, XR, YR, XW, and YW Fields. The only restrictions being that SPEC, I/O, and NOP operations are not generated in the S-Pad and FADD fields. On the second instruction following the first FADD (i.e., at location 6), the DPY field is set to DPY (YW) < FA. On the next instruction DPX is set to DPX (XW) < FM. The A1, A2, and M1, M2 fields are set to DPY (YR) DPX (XR). Thus the FA and FM operands are the same, DPY > A1, M1 and DPX > A2, M2. This makes the generated operation sequences recursive in that the output of the arithmetic (FA, FM, S-PAD) can become an input argument for a later operation. After the desired number of instructions has been generated, a seven instruction trailer is appended to the code to flush any remaining results out of the FA and FM pipelines and to provide an instruction sequence to set SPFN=SP (SPD) so that all of S-Pad can be examined.

GE MEDICAL SYSTEMS INSTITUTE

Trailer Micro-Code:

n+4	000001 155000 000000 017400	FADD ZERO, ZERO: FMUL TM, MD XW, YW Randomly selected See below for DPX, DPY
n+5	000000 000000 000000 017400	FMUL TM, MD: DPY (YW) < FA XW, YW Randomly selected See below for DPX
n+6	000003 170000 000000 000000	HALT: DPX (XW) < FM XW Randomly selected
n+7	040000 000040 000000 000000	MOV #0, 0
n+8	001403 115000 000000 000000	LDPNL: RSPFN
n+9	000003 170000 000000 000000	HALT
n+10	000000 000000 000000 000000	NOP

NOTES:

- 1) n is the number of randomly selected instructions, and set by the 'N' command, i.e. N4 'CR' or N1 'CR'.
- 2) At instruction n+4 and n+5 DPX will be DPX(XW) < FM if there have been three preceding FMUL's. DPY will be DPY (YW) < FA if there have been two preceding FADD's.

GE MEDICAL SYSTEMS INSTITUTE

The AP-120B is started executing this instruction sequence. It stops when it encounters the HALT at location n+6 (PSA will be pointing at n+7 when it is stopped). The program then reads the APSTATUS Register in order to capture the state of the machine as of the last operation. It then uses the Panel Continue function to cause the AP-120B to execute instructions n+8 and n+9. Instruction n+8 has the effect of setting SPFN equal to SP (SPD) so that the program can then proceed to examine S-PAD and Data Pad to obtain the hardware results.

Error message interpretation. Note that since FM always goes to DPX and FA goes to DPY, errors on DPX are typically caused by the multiplier (FMUL) while errors in DPY are typically caused by the adder (FADD).

Typical error message interpretation proceeds by using the "F" command to type out the micro-instructions, examining the micro-instructions to find the last one that wrote into the register in error, counting back 0, 2 or 3 micro-instructions (0 if S-Pad 2 if FA, 3 if FM), examining that instruction to find the input registers to the function in question, and then using the "#" command to type out the input numbers. At this point, all the information is available. Knowing the location of the failing instruction, the technician can probably go directly to scoping the problem. By setting the NOVA switch Register to one less than the address of the micro-instruction that failed, he will have a convenient pointer for locating it. If the problem is voltage sensitive, this is the preferred course of action. If the problem is solid, however, it may be worthwhile to attempt to carry out the arithmetic by hand, especially if in S-Pad, or FA, in order to be able to localize the problem further.

AP Micro-instructions

Introduction

Before proceeding into Micro-code interpretation, the reader should obtain the Processor Handbook. Constant reference will be made to Appendix B. Also, the reader should obtain the block diagram for the entire AP (see page 48).

The following few pages will explain the AP micro-instruction. It is suggested the reader follow along by relating the verbiage to the block diagram and the Appendix B mentioned above. This will help simplify the task of understanding the instructions.

Another technique used to simplify the instructions is to break down, where possible, the Program Source (instruction) words into four groups:

1. S-PAD (scratch pad)
2. FADD (floating adder)
3. DATA PAD
4. FMUL (floating multiplier)

GE MEDICAL SYSTEMS INSTITUTE

These four groups correspond very closely to the block diagram. By the time the reader finishes these pages, he will have walked through some AP micro-instructions that use the above four groups of hardware.

Brief Summary of Arithmetic Operation

1. S-PAD is a small 16 bit minicomputer. It can add, subtract and perform logical operations determined by the bits in the SOP group. The inputs to the S-PAD come from the S-PAD source and destination accumulators (SPS and SPD respectively). These accumulators may have an address \emptyset -17₈ (see SPS, SPD group).
2. FLOATING ADDER is an ALU that performs operations on 2 (two), 38 bit floating point numbers. This ALU, as in the S-PAD, can perform several operations on 2 (two) input arguments. The particular operation is determined by the bits in the FADD group.

The inputs to the floating adder are A1 and A2. Again, the bits in these fields are responsible for gating the appropriate data to the inputs. Some of the various memories that are available to A1 and A2 are:

DATA PAD X (DPX)

DATA PAD Y (DPY)

TABLE MEMORY (TMRAM, TMROM)

FLOATING MULTIPLIER (FM)

FLOATING ADDER (FA)

Note the octal values (code) necessary in A1 and A2 groups to make the various data paths available.

3. DATA PADS are essentially accumulators for the FLOATING ADDER and FLOATING MULTIPLIER.

As mentioned above, A1 and A2 can get data from the data pads. Each data pad (DPX and DPY) have address locations of \emptyset -37₈.

GE MEDICAL SYSTEMS INSTITUTE

- a) Focus your attention to the data pad group bits. Please note XR and YR. These two fields have an absolute range of $\emptyset\text{-}7_8$. In order to address all 40_8 locations in the data pads, a separate register is added; called the Data Pad Address Register (DPA). To calculate the Effective Address (EFA), the index fields are added to the contents of the DPA, i.e. $^*EFA = DPA(dpa) + XR$, where DPA (dpa) indicates the contents of the DPA register. To keep things interesting, a slight twist was added. Instead of weighting the index fields (XR, YR, XW, YW) with values of $0\text{-}7_8$ as one might think, the index fields are biased by 4 (four). This means, the true EFA is four less than one would have thought; i.e. true $EFA = DPA(dpa) + (XR - 4)$; hence, the "*" in front of the earlier "EFA". In other words, the weights of the index fields are (-4) to $(+3)_8$ and not $\emptyset\text{-}7_8$.

These weights ($XR - 4$) are referred to as IDX in the examples in Section 1.5.3. This will be very important to remember during the use of AP ARTH (AP Arithmetic).

Perhaps at this time, the meanings of the index field might clarify some confusion.

XR - read from data pad x

YR - read from data pad y

XW - write into data pad x

YW - write into data pad y

This means that separate indexes may control reading DPX, DPY and writing DPX, DPY essentially all in the same instruction.

In a nutshell, the index fields (XR, YR, XW, YW) modify the DPA while fields A1, A2, M1, M2 and DPBS control what data pad "accumulator"! (X or Y) is gated onto which bus.

- b) Move your eyes to the DPX and DPY fields.

1. The DPX field controls what data bus is gated to the input to DATA PAD X.

GE MEDICAL SYSTEMS INSTITUTE

2. DPY field controls what data bus is gated to the input to DATA PAD Y.
3. Valid inputs to the data pads are the DPBS, the outputs from the floating adder and multiplier (FA and FM) respectively).

c) IMPORTANT RULES OF THUMB (ROT)

ROT 1

- a) XR has meaning if, and only if, A1, A2, M1, M2 so designates a requested read from DPX.
- b) YR has meaning if, and only if, A1, A2, M1, M2 so designates a requested read from DPY.

ROT 2

- a) XW has meaning if, and only if, the DPX field is non zero.
- b) YW has meaning if, and only if, the DPY field has meaning.

4. FLOATING MULTIPLIER

The floating multiplier fields (FM, M1, M2) behave very similarly to the floating adder fields. The major exception being the FM field has only 1 (one) bit; therefore, the floating multiplier is either multiplying or is inactive.

Decoding Micro-instruction Examples

EX 1:

	<u>INST</u>	<u>ASSEMBLY CODE</u>
PS Word 0	020215	ADD SPS(2), SPD(3); FADD DPX(0), DPY(0)
PS Word 1	123000	
PS Word 2	000440	
PS Word 3	000000	

STEP 1. Write the bit patterns in the layout of the instruction word. (See Ill. FRP 8-1).

GE MEDICAL SYSTEMS INSTITUTE

STEP 2. Isolate bit patterns in their operand group and read as octal numbers (See Ill. FRP 8-2).

STEP 3. Look up the function in the appropriate table in Appendix B of the Processor Handbook.

a) $D = \emptyset$; No decimate (Bit reverse)

NOTE: D and B are used interchangeably in this group.

b) $SOP = 2$; Add S-Pad source and S-Pad destination. The "accumulator" address is determined by SPS and SPD group (see page B-3 in the Processor Handbook).

c) $SH = \emptyset$; No shift

d) $SPS = 2$
 $SPD = 3$; S-Pad locations 2 and 3 are the "accumulators" to be added together (the ADD determined by $SOP = 2$ above)

e) $FADD = 3$; Floating add A1 and A2 (see page B-13 in the Processor Handbook)

f) $A1 = 2$; $DPX > A1$, A1 gets its input from DPX.

DPX (1DX) is the form requested by AP ARTH for use of the "#" command.

$$\begin{aligned} 1DX &= XR-4 \\ &= 4-4 = DPX (\emptyset) \end{aligned}$$

Since DPX (\emptyset) is the input argument of interest, typing #34 'CR' (see 'User Commands') will display the content of DPX (\emptyset).

NOTE: Keep in mind the true data pad location (addresses \emptyset -37(8)) EFA = 1DX + DPA (dpa), where:

DPA (dpa) is the contents of the Data Pad Address register.

(See description of the DATA PAD ADDRESS REGISTER on page 5-7 of the ARRAY PROCESSOR MAINTENANCE MANUAL)

GE MEDICAL SYSTEMS INSTITUTE

PS WORD0 :- 0202158

PS WORD1 - 1230808

0	SOP	SII	SRS	STB	FAND	A1	A2	COND	DISP
S-pool group									
Mult. group									
Branch group									

Page 31

PS WORD2 - 00004408

0	0	0	0	0	0	0	0	0	0
32	33	34	35	36	37	38	39	40	41
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60

Data prod group

Multiply group

Memory group

Value

Instruction Field Layout

III. FRP 8-1

GE MEDICAL SYSTEMS INSTITUTE

0	\emptyset	D	NOP
1-3	2	SOP	ADD SPS, SPD
4, 5	\emptyset	SH	No SHIFT
6-9	\emptyset 2	SPS	SPS(\emptyset 2)
10-13	\emptyset 3	SPD	SPD(\emptyset 3)
14-16	\emptyset 3	FADD	FADD AL, A2
17-19	2	AL	DPX \rightarrow AL
20-22	3	A2	DPY \rightarrow A2
23-26	\emptyset \emptyset	COND	NOP
27-31	\emptyset \emptyset	DISP	COND is NOP so DISP is not applicable
32, 33	\emptyset	DPX	NOP
34, 35	\emptyset	DPY	NOP
36-38	\emptyset	DPBS	ZERO
39-41	4	XR	$IDX = XR - 4 = 4 - 4 = \emptyset$
42-44	4	YR	$IDY = YR - 4 = 4 - 4 = \emptyset$
45-47	\emptyset	XW	DPX is NOP above so XW is NOP
48-50	\emptyset	YW	DPY is NOP above so YW is NOP
51	\emptyset	FM	NOP
52, 53	\emptyset	M1	FM \rightarrow M1 but FM is
54, 55	\emptyset	M2	FA \rightarrow M2 but FM is
56, 57	\emptyset	M1	NOP
58, 59	\emptyset	MA	
60, 61	\emptyset	DPA	
62, 63	\emptyset	TMA	

III. FRP 8 - 2

GE MEDICAL SYSTEMS INSTITUTE

g) A2 = 3; DPY > A2, A2 input (read) from DPY

The calculations for DPY (1DY):

$$1DY = YR - 4$$

$$= 4 - 4 = \emptyset$$

h) COND

DISP; NOP

i) DPX

DPY

DPBS; NOP

j) XR

YR; Since A1 and A2 fields request numbers from Data Pads X and Y, XR and YR are valid. (See DATA Pad description ON PAGE 37)

The general form for use of the AP ARTH '#'
COMMAND TABLES is

DPX (1DX)

DPY (1DY)

where 1DX and 1DY are:

$$1DX = XR - 4 \text{ or } XW - 4$$

$$1DY = YR - 4 \text{ or } YW - 4$$

EXAMPLES: 1. XR field equals \emptyset

$$1DX = \emptyset - 4 = -4$$

#20 'CR' would be used to examine the contents of
DPX (-4).

2. YW field equals 7

$$1DY = 7 - 4 = 3$$

#275 'CR' would be used to examine the
contents of DPY (3).

GE MEDICAL SYSTEMS INSTITUTE

- k) XW
YW; not applicable because DPX and DPY fields are NOP (see g above).
- l) FM
M1
M2; FM is a NOP; therefore, M1, M2 are not applicable.
- m) MI
MA
DPA
TMA; all NOPS

EX 2:

	<u>INST</u>	<u>ASSEMBLY CODE</u>
PS Word 0	000000	
PS Word 1	000000	FMUL DPX(0); DPY(0); DPX(-1) < FA
PS Word 2	100443	
PS Word 3	013000	:

STEP 1 Write bit patterns in the instruction layout (Ill. FRP 8-3)

STEP 2 Isolate bit patterns in their operand group and read as octal numbers (Ill. FRP 8-4)

STEP 3 Look up the function in the tables in Appendix B of the Processor Handbook

- a) Fields D to DISP, all zeroes - translates to all NOPS
- b) DPX = 2; FA > DPX - the output of the floating adder will be stored in Data Pad.
- c) DPY = 0
DPBS = 0; NOP
- d) XR = 4 IDX = XR - 4 = 0, DPX(0)
YR = 4; IDY = YR - 4 = 0, DPY (0)

These index fields are defined because the floating multipliers are requesting inputs from DPX and DPY.

GE MEDICAL SYSTEMS INSTITUTE

PS WORD1 - ~~phos�~~

PS WORD 3 - #13 of 66

Memory drawn										Value									
Memory drawn					Value					Memory drawn					Value				
W1		W2		W3	X1		X2		X3	Y1		Y2		Y3	Z1		Z2		Z3
51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70
22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61

Instruction Field Layout

m1. FRP 8-3

GE MEDICAL SYSTEMS INSTITUTE

<u>Bit(s)</u>	<u>Value</u>	<u>Group</u>	<u>Means</u>
0	\$	D	NOP - no bit reverse
1-3	\$	SOP	SOP _L - under SOP _L , \$ = NOP
4, 5	\$	SH	NOP - no shift
6-9	\$0	SPS	DON'T CARE BECAUSE SOP = NOP
10-13	\$0	SPD	Don't care because SOP = NOP
14-16	\$	FADD	SEE FADD FIELD
17-19	\$	A1	THIS IS FADD FIELD, FADD
20-22	\$	A2	Don't care FLOATING ADDER
23-26	\$0	COND	NOP
27-31	\$0	DISP	Don't care COND is NOP
32, 33	2	DPX	FA > DPX
34, 35	\$	DPY	NOP
36-38	\$	DPBS	DB = ZERO
39-41	4	XR	LDX = XR - 4 = \$
42-44	4	YR	LDX = YR - 4 = \$
45-47	3	XW	LDX = XW - 4 = -1
48-50	\$	YW	NOP
51	1	FM	MULTIPLY M1 * M2
52, 53	1	M1	DPX → M1
54, 55	2	M2	DPY → M2
56, 57	\$	M1	NOP
58, 59	\$	MA	NOP
60, 61	\$	DPA	NOP
62, 63	\$	TMA	NOP

III. FRP 8-4

GE MEDICAL SYSTEMS INSTITUTE

- e) $XW = 3;$ $IDX = XW - 4 = -1,$ $DPX(-1)$
 $DPX(-1)$ is being written into from the output of
 the floating adder (see b above).
- f) $YW = \emptyset;$ This is not applicable because nothing is specified
 to be stored into any DPY location.
- g) $FM = 1;$ Floating Multiply $M1 \neq M2$
- h) $M1 = 1;$ $DPX > M1$, the $M1$ input gets its value from DPX
 (\emptyset). (See d above.)
- i) $M2 = 2;$ $DPY > M2$, the $M2$ input is the value from $DPY(\emptyset)$.
 (See d above.)
- j) $MI = \emptyset$
 $MA = \emptyset$
 $DPA = \emptyset$
 $TMA = \emptyset;$ all NOPS

GE MEDICAL SYSTEMS INSTITUTE

Use of AP ARTH

Running AP ARTH with the intent of verifying arithmetic operation can be done by typing RWE 'CR' and allowing two (2) passes to occur. At the end of a pass, an "A" will be output by the diagnostic.

After encountering an error, the command string should be RN1WE 'CR'.

The following is a sample debugging session:

1. Load and boot the DTOS tape.
2. The DTOS heading on the CRT means the program is ready for commands. Be sure the asterisk is present!
3. Type LOAD AP ARTH 'CR'
A prompt will appear on the screen indicating the load is in progress.
4. a) With the asterisk on the screen, again type RN1WE 'CR'

Sample error:

DPY 000023 E 000732 003065 125070 A 000733 003432 152424

N 000001 X,Y 054770 066024 APSTATUS 002005 DPA 000024

- b) Because the error occurred in DPY, it is probably a FADD error.
- c) To look at the instructions being executed, use the "Fnnnn" command.

Type F4'CR' to display the first random instruction (in this case, there is only one random instruction - NOTE N 000001 in the error message).

In response to the "Fnnnn" command, the instruction requested will be output with the form:

PSA PS 0 PS 1 PS 2 PS 3
nnnn aaaaaa bbbbbbb ccccccc dddddd

Sample printout:

000004 167015 132000 000575 004400

GE MEDICAL SYSTEMS INSTITUTE

- d) At this point AP ARTH is looking for a carriage return or a line feed.
1. The carriage return will put the program back into the command string mode.
 2. The line feed will cause output of subsequent instructions.

Sample:

*F4 'CR'

000004	107015	132000	000575	004400	'LF'
000005	000001	155000	000007	017400	'LF'
000006	000000	000000	020006	077400	'LF'
000007	000003	170000	140007	060000	<u>'CR'</u>
*					

- c) Now decode the instruction(s)

The above instructions decode to:

FADD DPY(3), DPX(1)

FADD ZERO, ZERO

DPY(-1)< FA

HALT

Since the FADDER was suspect, only the FADD group was decoded.

- f) Now the task of finding the input arguments shall be tackled.

1. The input arguments are DPY(3) and DPX(1). This is where the "#" command is used.
2. In the "a) Input Arguments" Section of the APARTH '#: COMMAND Table (Ill. FRP 8-5 page)

#75 for DPY (3)

#37 for DPX (1)

GE MEDICAL SYSTEMS INSTITUTE

3. The arguments shall be displayed in the form:

EXPONENT (EX)

HIGH MANTISSA (HMAN)

LOW MANTISSA (LMAN)

4. To display the exponent of an argument -

Type #75 'CR'

5. Two line feeds cause the HMAN and LMAN to be output.

Sample *#75 'CR'

000701 'LF'

001516 'LF'

021750 'CR'

*#37 'CR'

000733 'LF'

001432 'LF'

152433 'CR'

*

APARTH '#' COMMAND Argument Number to Register Correspondence Table

a) Input Arguments

Octal	
'0 to #17	SP(0 To 17)
#20	DPX (-4)
#23	DPX (-3)
#26	DPX (-2)
#31	DPX (-1)
#34	DPX (0)
#37	DPX (1)
#42	DPI (2)
#45	DPX (3)
#50	DPY (-4)
#53	DPY (-3)
#56	DPY (-2)
#61	DPY (-1)
#64	DPY (0)
#67	DPY (1)
#72	DPY (2)
#75	DPY (3)

III. FRP 8-5

GE MEDICAL SYSTEMS INSTITUTE

b) Results of Software simulation

#100 To #117	SP(0 To 17)
#120	DPX (-4)
#123	DPX (-3)
#126	DPX (-2)
#131	DPX (-1)
#134	DPX (0)
#137	DPX (1)
#142	DPX (2)
#145	DPX (3)
#150	DPY (-4)
#153	DPY (-3)
#156	DPY (-2)
#161	DPY (-1)
#164	DPY (0)
#167	DPY (1)
#172	DPY (2)
#175	DPY (3)

c) Result of AP-120B Hardware

#200 To #217	SP (0 To 17)
#220	DPX (-4)
#223	DPX (-3)
#226	DPX (-2)
#231	DPX (-1)
#234	DPX (0)
#237	DPX (1)
#242	DPX (2)
#245	DPX (3)
#250	DPY (-4)
#253	DPY (-3)
#256	DPY (-2)
#261	DPY (-1)
#264	DPY (0)
#267	DPY (1)
#272	DPY (2)
#275	DPY (3)

II. FRP 8-5 (cont.)

GE MEDICAL SYSTEMS INSTITUTE

PS TEST

This stand-alone, assembly-language program tests the functioning of the APEX Bootstrap micro-code. It uses the Bootstrap to load pseudo-random values into Program-Source memory. It tests Program-Source memory (BD. 236), the Formatter (Bd's 226 and 227) and the Host Interface (BD's 280 and 219) and AP-120B Internal I/O (BD 214). However, all but the program source memory has been checked extensively in the previous tests.

This test prints an "S" on the CRT upon completion of a pass.

The error message format is:

PS LOC = aaaaaa EXP zzzzzz oooooo tttttt rrrrrr
X, Y xxxxxx yyyy yy ACT zzzzzz oooooo tttttt rrrrrr

where

aaaaaa is the program source address
zzzzzz is program source word Zero
oooooo is program source word One
tttttt is program source word Two
rrrrrr is program source word Three
and xxxxxx yyyy yy are the random generator restart parameters.

TM ROM

Brief description

This stand-alone, assembly-language program tests Table-Memory ROM in both the normal and FFT addressing modes. It utilizes a micro-program to transfer pseudo-random locations from table-memory ROM into sequential locations in Main - Data memory. The micro-code is written so that this process tests the minimum access and cycle times for TMROM.

TM ROM does not acknowledge successful completion of a pass.

The results are checked via a simulator in the host that generates the same sequence of random addresses and looks up the expected result in a table in host memory.

GE MEDICAL SYSTEMS INSTITUTE

Error message format

The error message format is of the form:

MA mmmmmmm E eeeeeee eeeeeee A aaaaaaa aaaaaaa aaaaaaa
TMLOC llllll X,Y xxxxxx yyyy APSTA ssssss

where

mmmmmm - is the memory address
eeeeee - is the expected packed 38 bit value
aaaaaa - is the actual packed 38 bit value
llllll - is the table memory location
xxxxxx yyyy - restart parameters for random number generator
sssss - AP status word

Any errors encountered during TM ROM generally indicates a bad 217 Board (TM ROM)

TM RAM

Brief description

This stand-alone, assembly-language program tests table memory RAM. It places a table of pseudo-random 38-bit values in Main-Data memory and then runs a micro-program that moves this table at full speed from MD to TM RAM and then back to a separate buffer in MD.

Error Message Format

The error message format is the same as TM ROM, except the data is displayed as exponent, high mantissa and low mantissa instead of packed. If MA is under 10,000g this generally indicates a bad 238 Full Board.

MA greater or equal to 10,000g generally indicates the 238 Half Board is faulty.

GE MEDICAL SYSTEMS INSTITUTE

FDFFT
FCFFT

Brief Description

This stand-alone, assembly-language program is intended primarily for use as a verification test of the AP-120B. It does not provide board level diagnostic indicators as do the diagnostic programs, APTEST and APPATH. Its use as a diagnostic requires the skill of a technician who is well versed in the theory of operation of the AP-120B.

The test is based on the fact that a forward Fourier Transform of a data set followed by an inverse Fourier Transform of the result of the forward transform should result in the original data set within a predictable error limit.

The program is divided into two sections. The first section uses simple data sets that consist of only one non-zero point with all the remaining points set to zero. The second section utilizes a pseudo-random number generator to produce data sets. After the forward/inverse FFT process the pseudo-random number generator is restarted to recreate the input data set for comparison against the result of the process.

The version supplied for the acceptance test is called "FCFFT". This version utilizes the AP-120B to generate the data set and check the results for the pseudo-random tests. The Host

GE MEDICAL SYSTEMS INSTITUTE

CPU still does this for the simple data sets. Use of the AP-120B for the pseudo-random sets significantly increases the thru-put and provides a nearly 100% duty factor for the test.

This program also performs a test of the transcendental micro-code (SIN/COS, LOG, DIV, ATAN, EXP, SQRT) and related TMROM tables. The transcendental test is table driven.

a) FFT Tests

The impulse and the random FIFFT tests operate by supplying the AP-120B with 28-bit integers through the 32-bit integer, program control to DMA path of the AP-120B interface. An AP-120B micro-program converts these integers to floating point, rearranges the array in bit-reversed order, performs the forward Real FFT, again rearranges the array in bit-reversed order, performs the inverse FFT, and finally converts the array back to 28-bit integers. The test program then uses the 32-bit integer, DMA to program control, interface path to read the results back for comparison.

The first (impulse) test begins with the smallest FFT (8 points) and works its way up sequentially to the largest FFT size that is compatible with the memory configuration (8K Complex FFT in 16K of MD). For each size FFT it slides an impulse (\$04000, \$00000) across the data set. For the small FFT's (< 256 points), the impulse is placed in sequence on every real and complex point

GE MEDICAL SYSTEMS INSTITUTE

of the input data set and the data set is tested. For the larger FFT's, the impulse is moved through the data set in such a manner as to skip larger and larger numbers of points as the size of the data set is increased while still testing real and complex points in an alternating fashion. This is done in order to allow the impulse test to complete in a finite length of time (typically less than 25 minutes for a maximum FFT size of 8K). Upon completion of the first test, the program types:

"END IMPULSE TEST"

The second portion of the test utilizes a very long-sequence, pseudo-random number generator to select the desired data set size and then to generate the set of 28-bit integers. These numbers are acted upon by the AP-120B in exactly the same fashion as in the first test. The micro-program then restarts the pseudo-random number generator and uses it to recreate the input data set for comparison against the FFT output. For every 64 successful random Forward/Inverse FFT's, the program types:

"P"

For a maximum Complex FFT size of 8K (16K MD; 2K TM Roots of unity), the program sets an error limit of 4 in the least significant bits of the 28-bit mantissa for the impulse test. For the random FFT test, the error limit is $m+5$ where m is the power of 2 such that $2^m = N$ (N is the number of complex data points).

GE MEDICAL SYSTEMS INSTITUTE

If the deviation between the input and output is greater than the allowable error limit, then an error message is output in the following format.

LOC, SIZE, llll ssss EXP eeee eeee

ACT aaaa aaaa X, Y xxxx yyyy

Where llll is the location in error,

ssss is the size of the FFT under test

eeee etc. is the expected result,

aaaa etc. is the actual result,

and xxxx yyyy are the restart parameters for the
pseudo-random number generator.

b. Transcendental Tests

This test is run once for every 64 FFT Forward/Inverse tests (just before the "F" is typed). It uses a table of input arguments and expected answers to test the transcendental micro-code and related TMROM entries.

The error message has the following format.

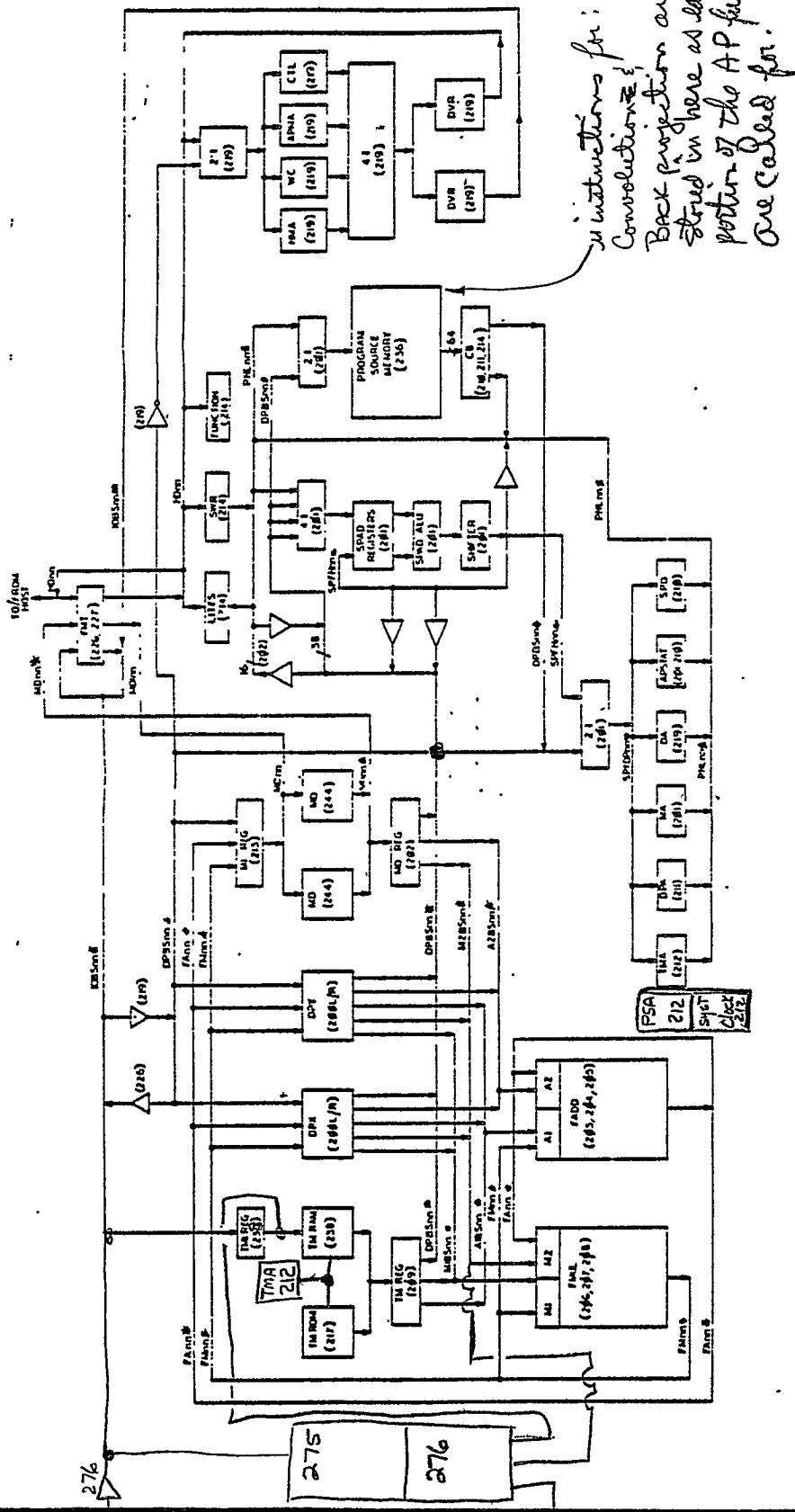
MNEM....EXP eeee eeee eeee ACT aaaa aaaa aaaa.

Where the mnemonic (MNEM) is:

DIVIDE, ATN2, SQRT, SIN X, COS X,

EXPONENT X, LOG X, ATAN X, or LN X to denote the appropriate transcendental function, where eeee etc. denotes the expected result and where aaaa etc. denotes the actual result (in AP-120B Floating Point format).

GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

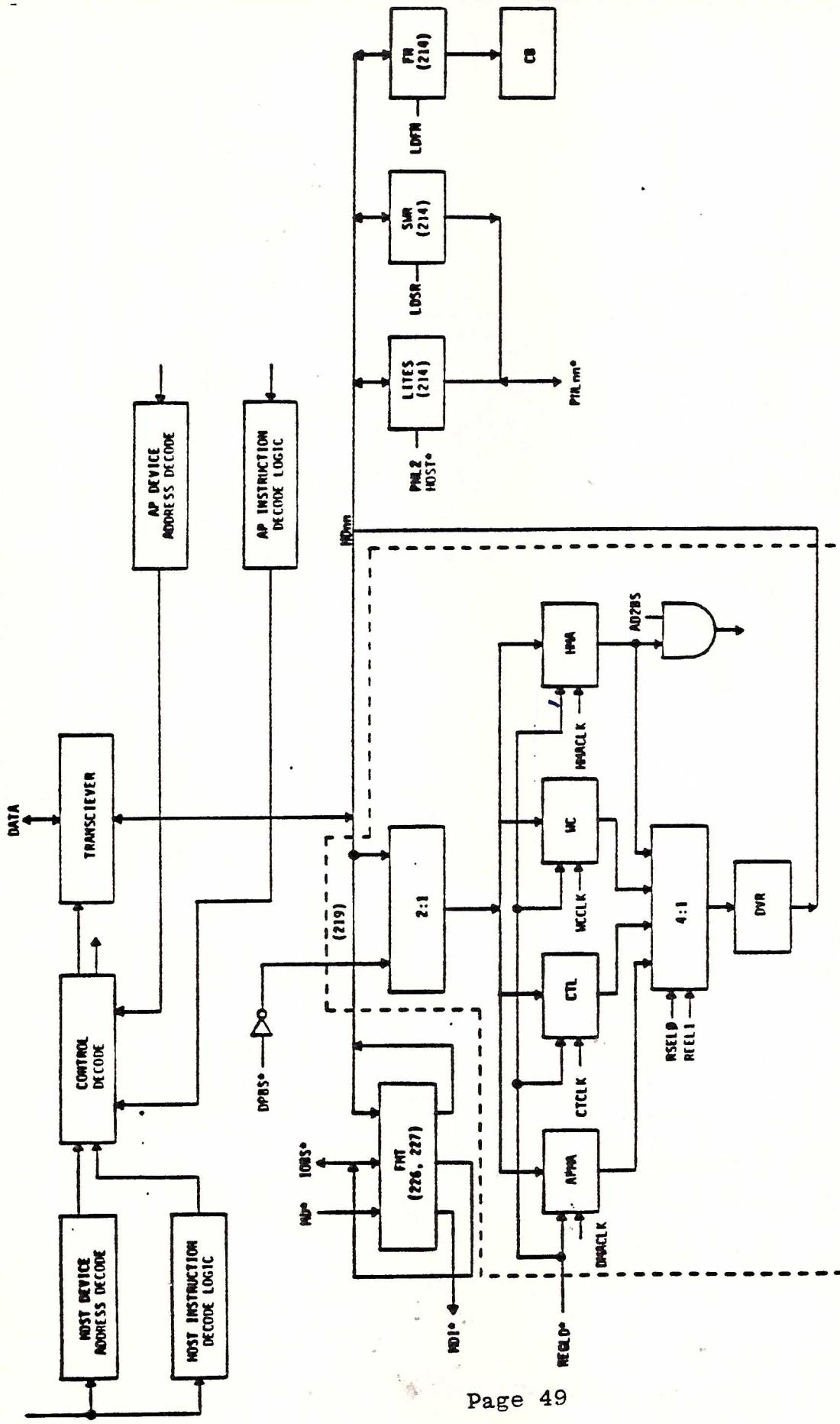


Figure IF1
Interface Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

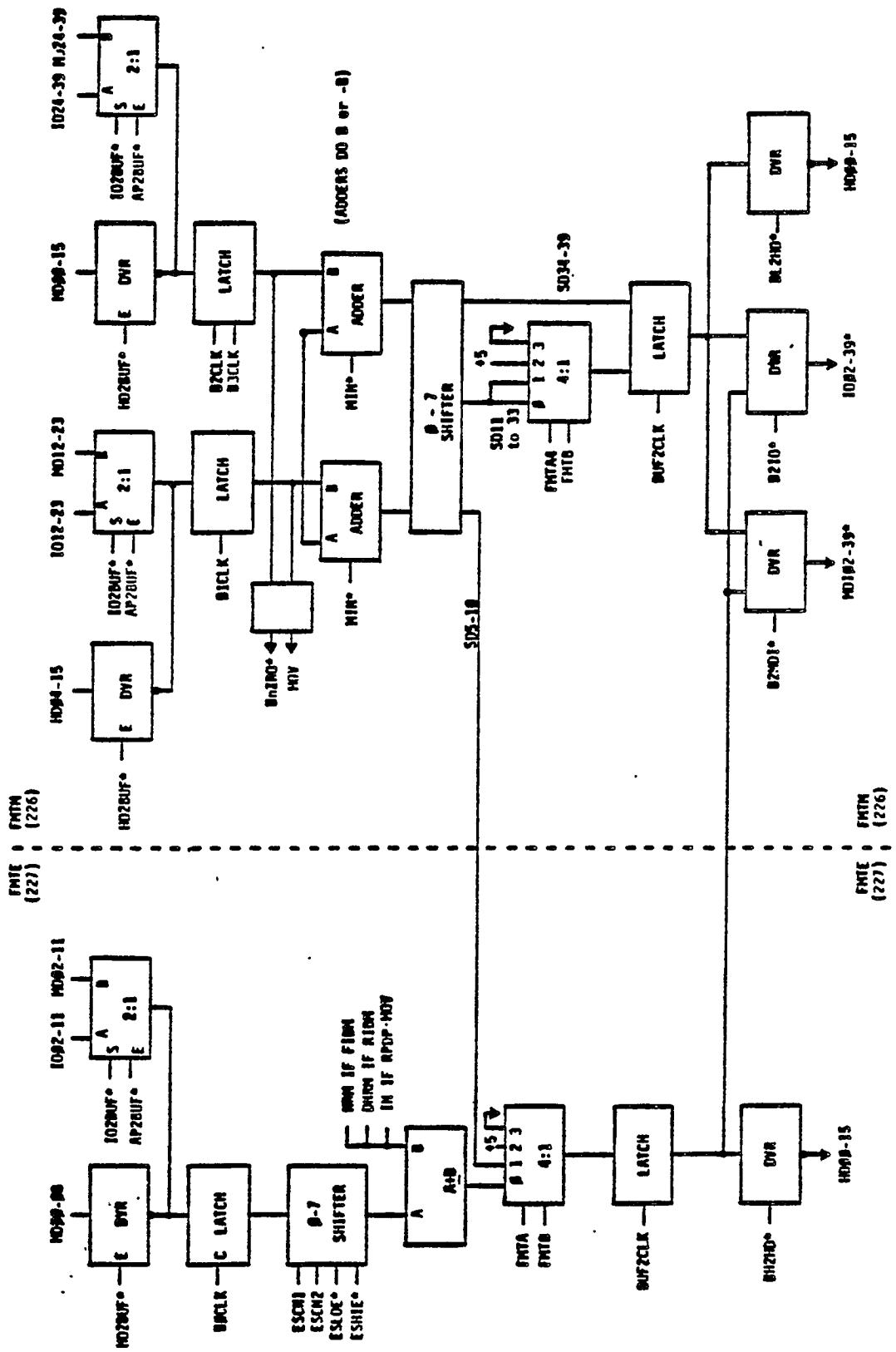


Figure F11 Formatter Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

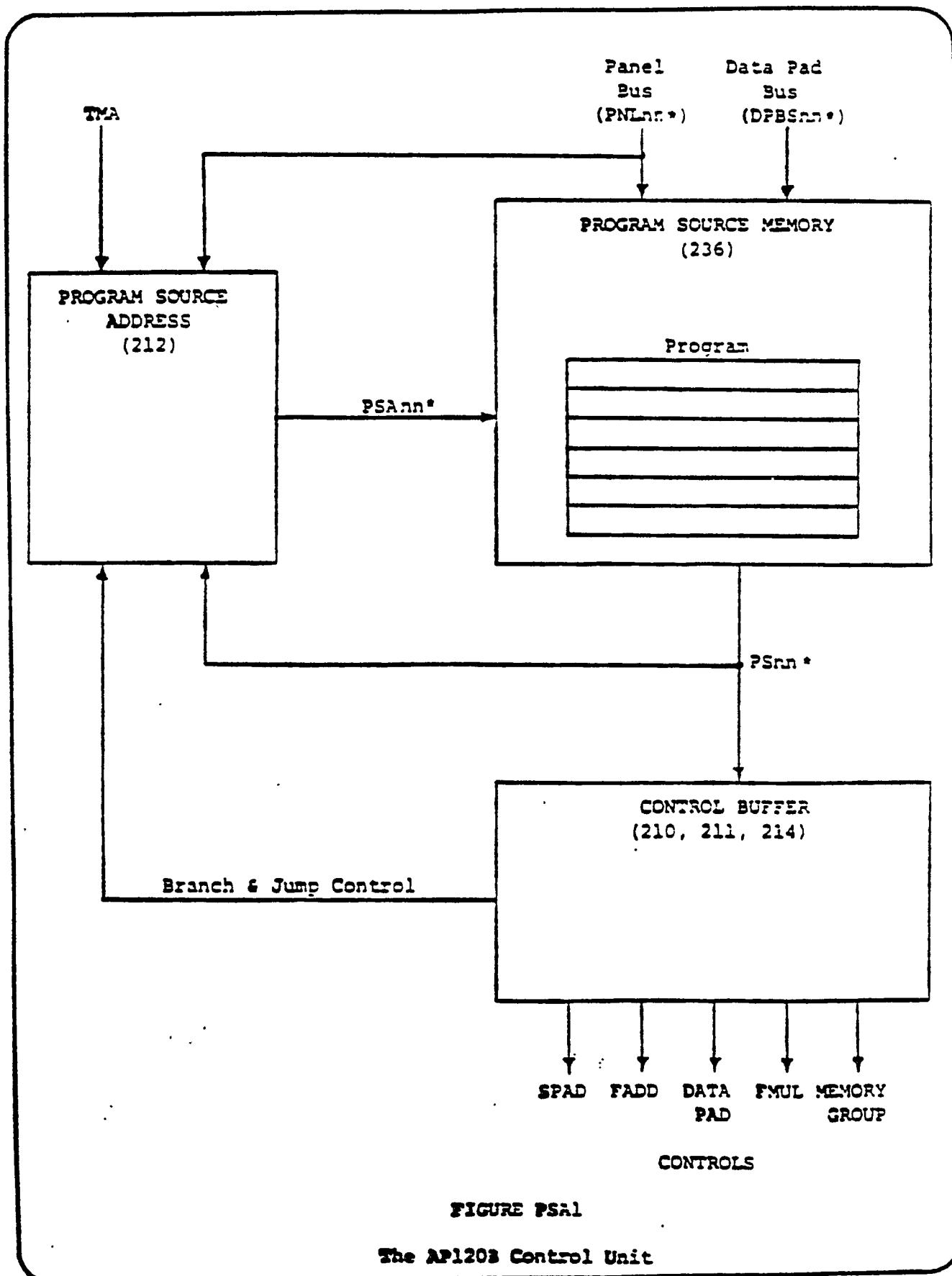


FIGURE PS1

The API203 Control Unit

GE MEDICAL SYSTEMS INSTITUTE

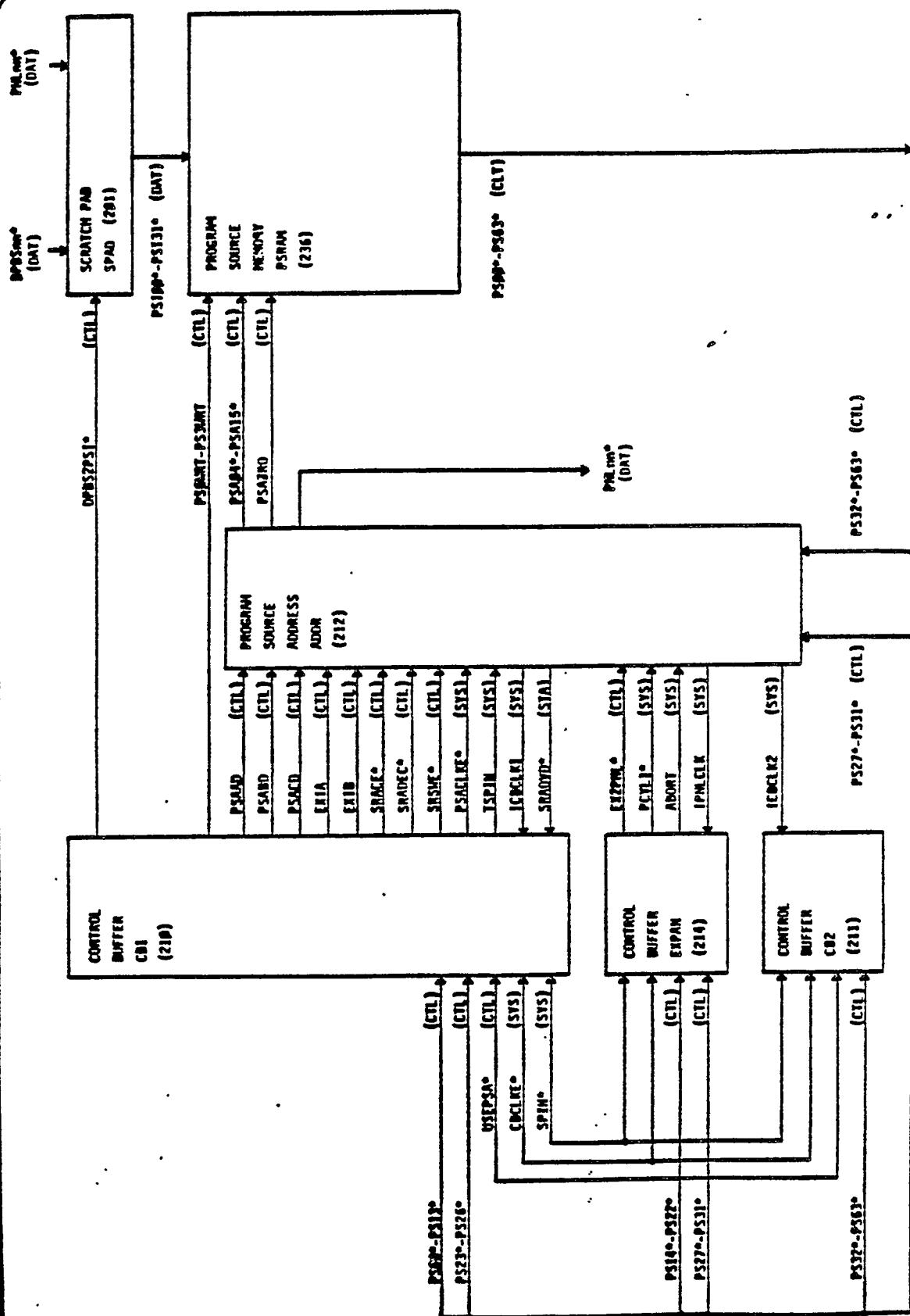
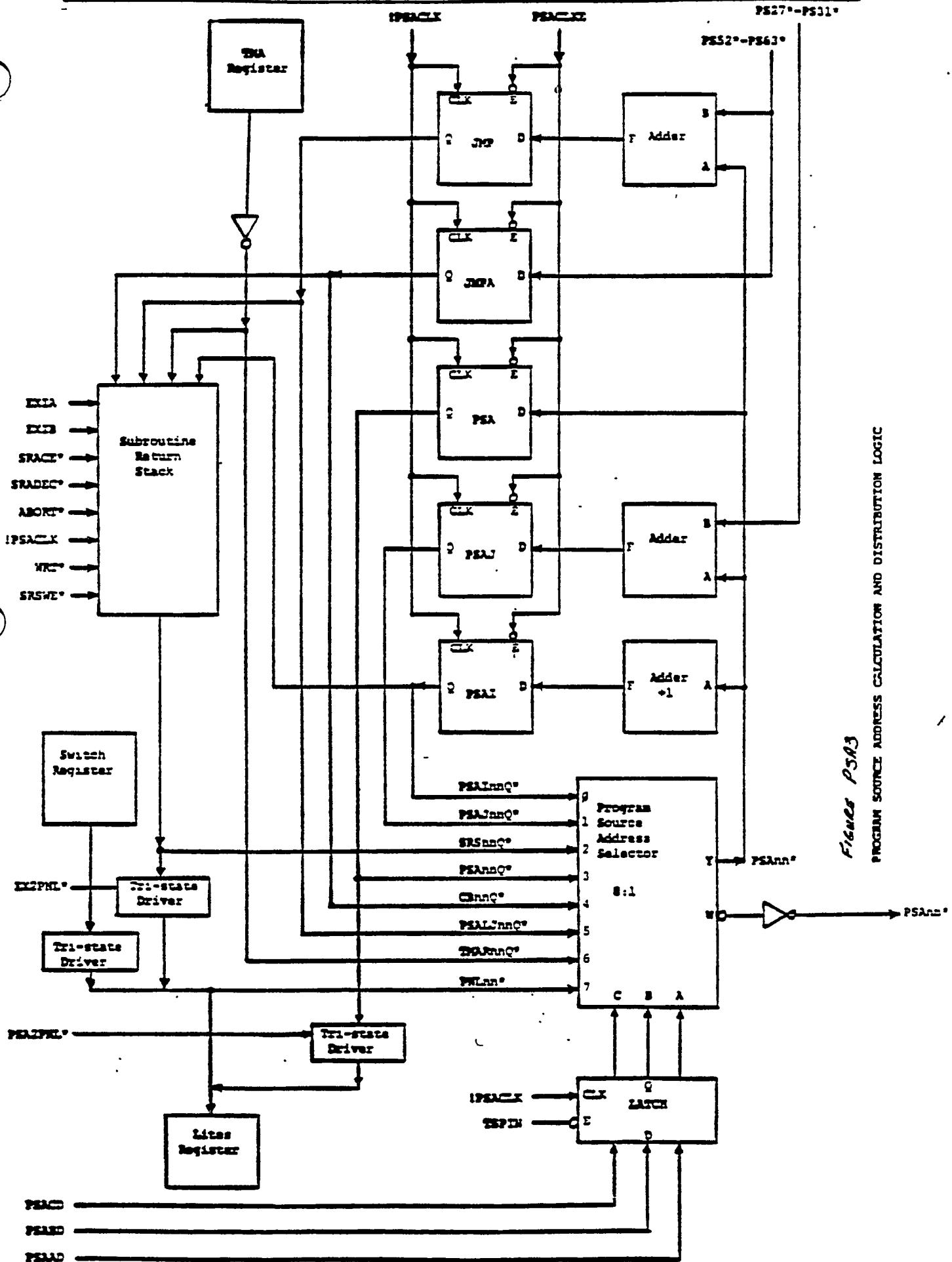


Figure PSA2
Program Source System Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE



PROGRAM SOURCE ADDRESS CALCULATION AND DISTRIBUTION LOGIC

GE MEDICAL SYSTEMS INSTITUTE

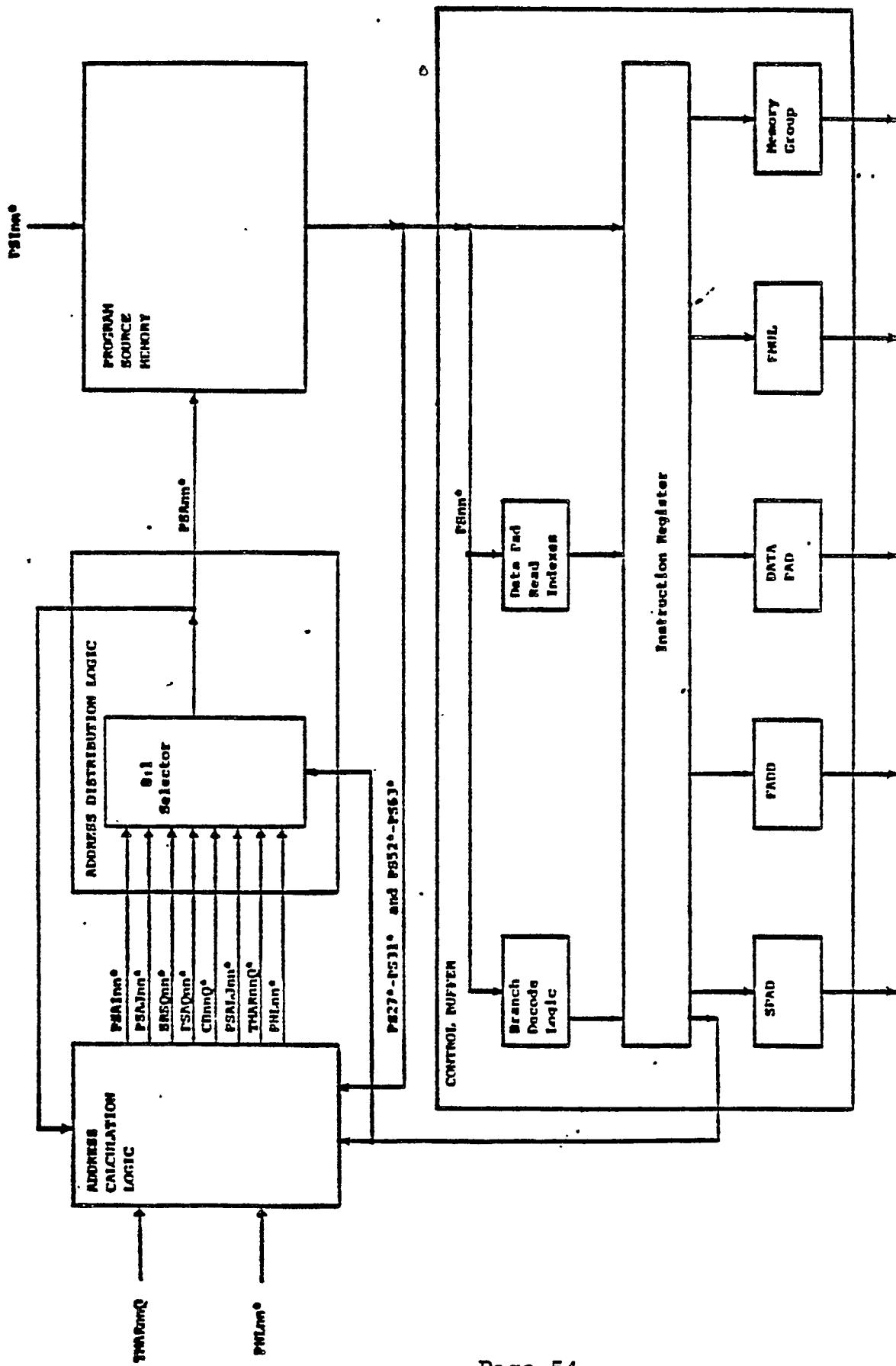


Figure PSA4
Program Source Address Logic

GE MEDICAL SYSTEMS INSTITUTE

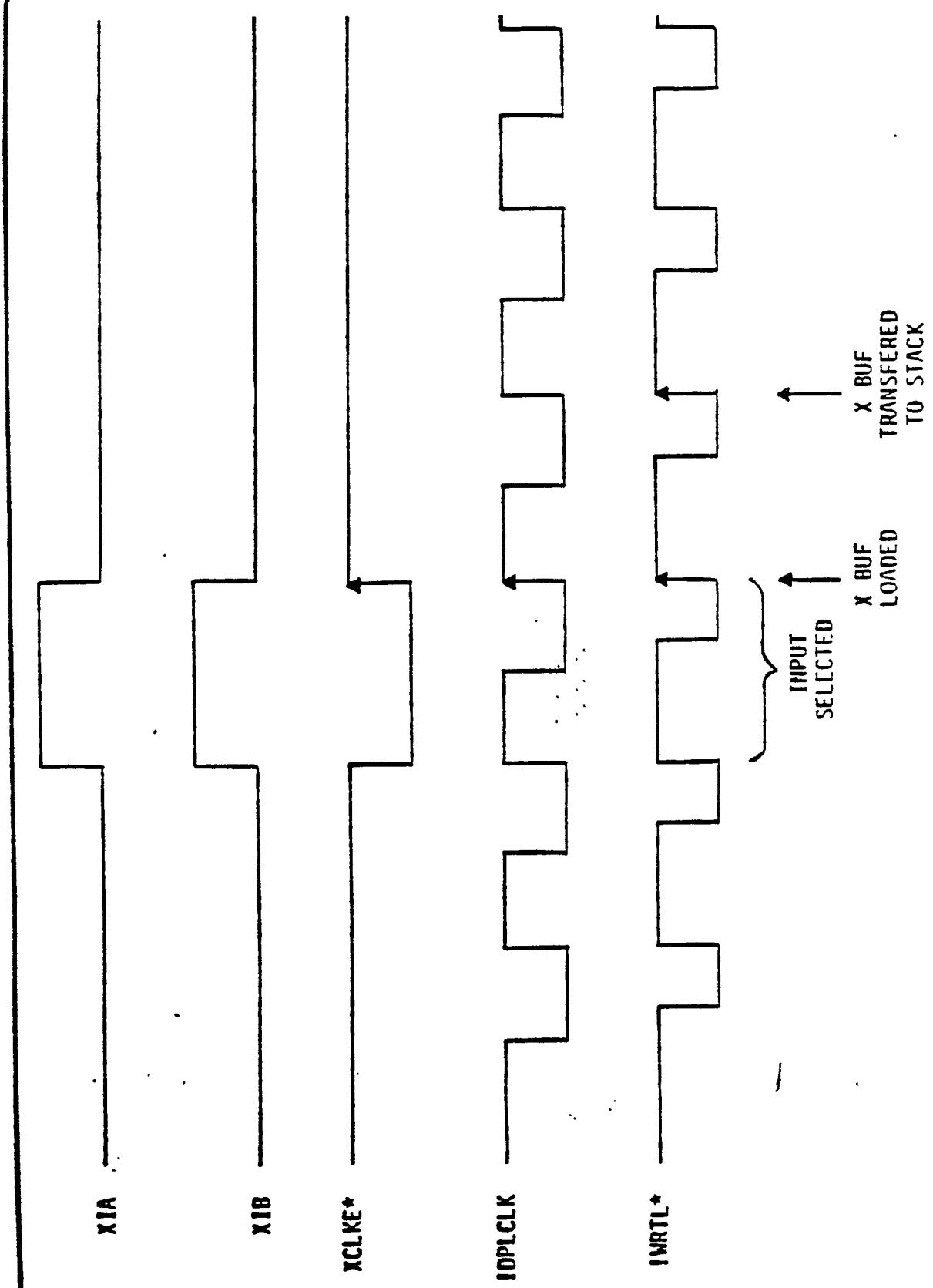


Figure DP&D1

Timing Diagram of Writing DPX

GE MEDICAL SYSTEMS INSTITUTE

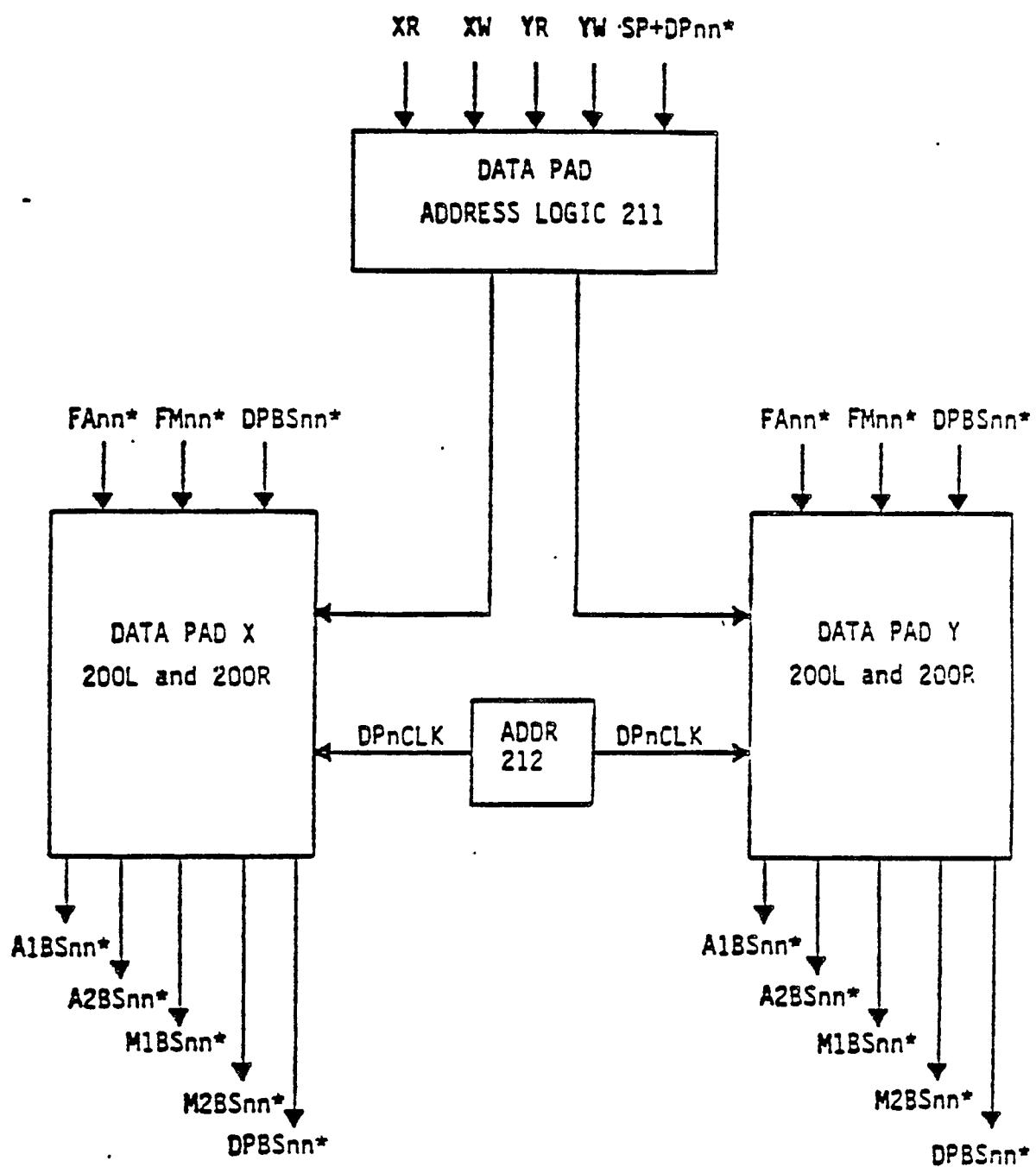


Figure DPAD2
Data Pad Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

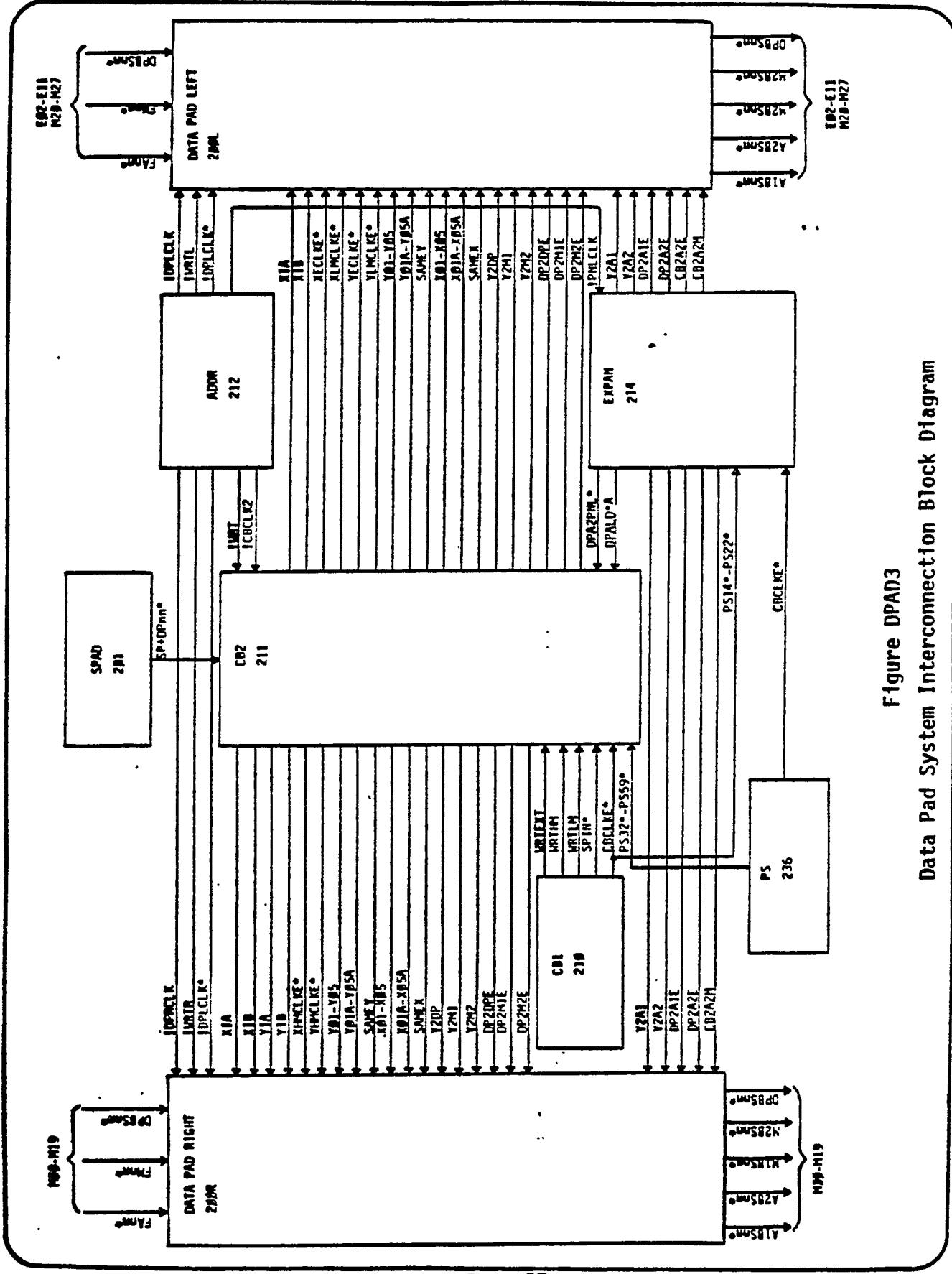


Figure DPAD3 Data Pad System Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

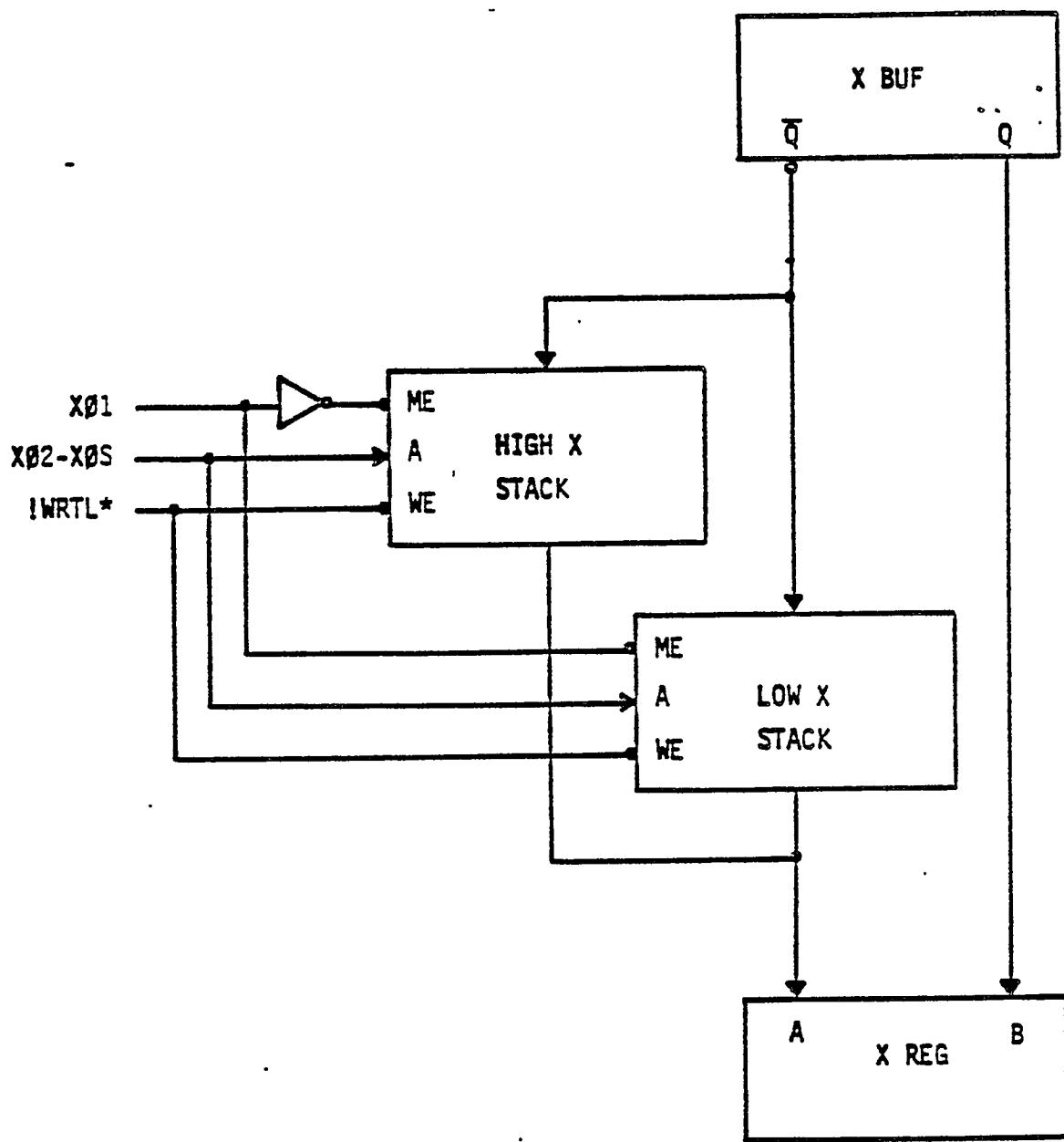


Figure DPAD4
Data Pad Memory Element Detail Showing Stack Address

GE MEDICAL SYSTEMS INSTITUTE

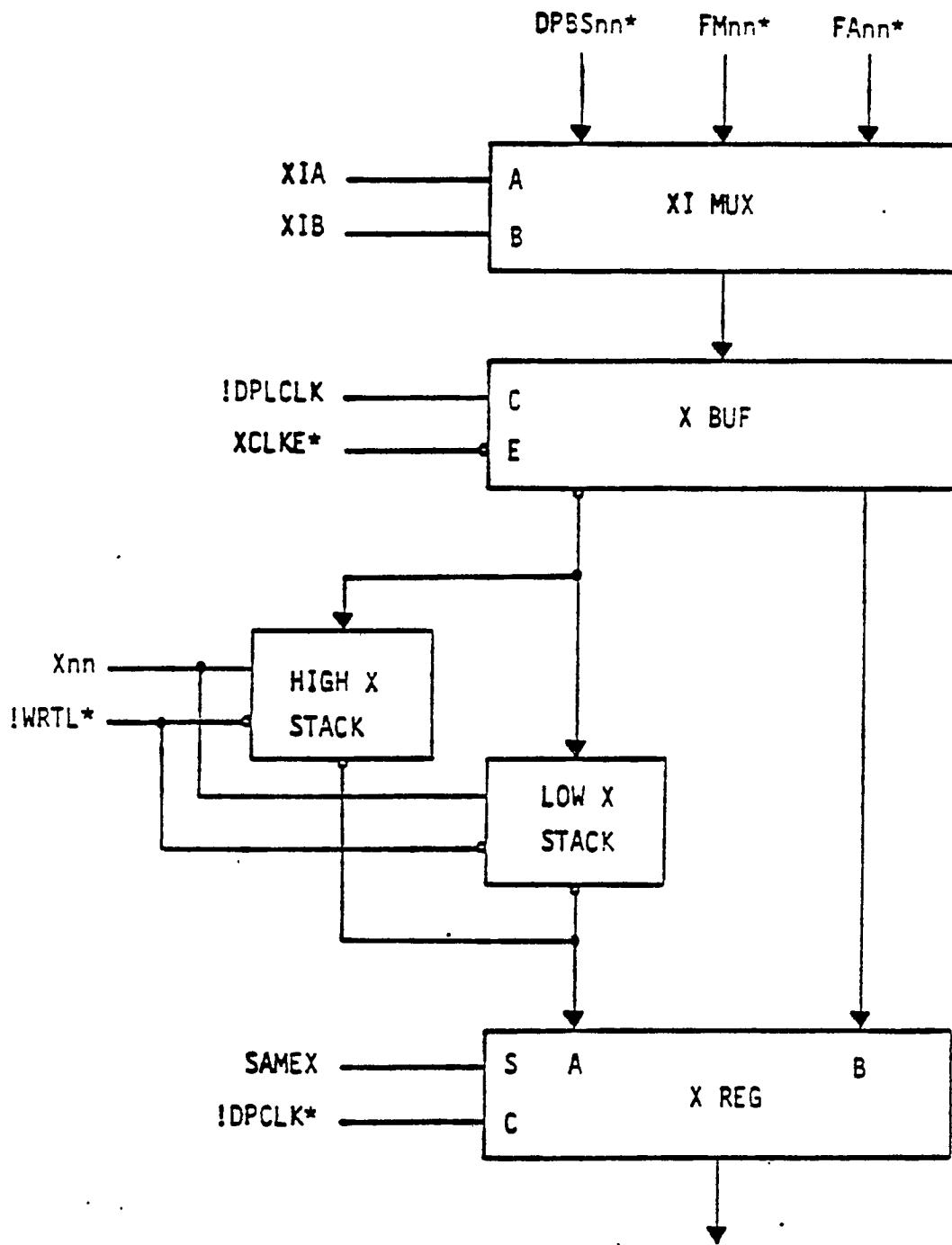


Figure DPAD5
Block Diagram of DPX

GE MEDICAL SYSTEMS INSTITUTE

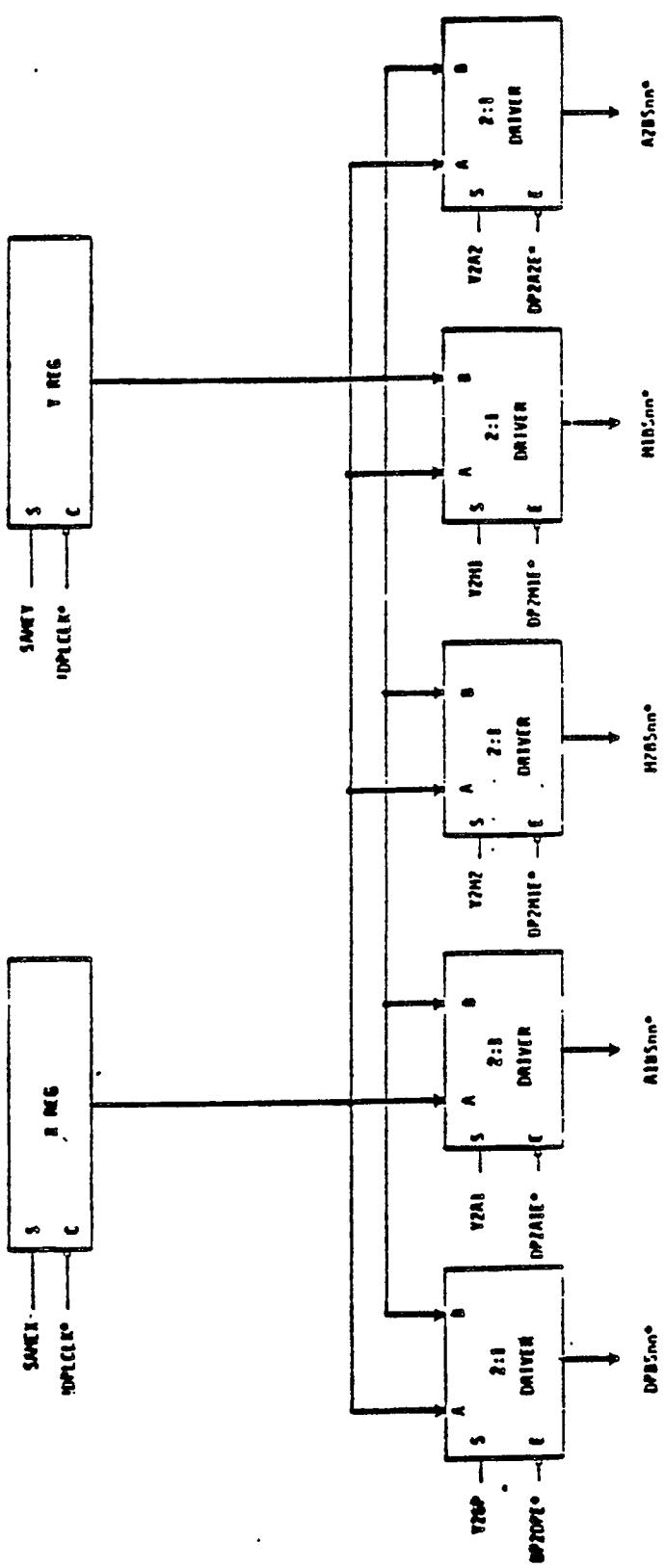


Figure DPADE
Data Pad Output Logic

GE MEDICAL SYSTEMS INSTITUTE

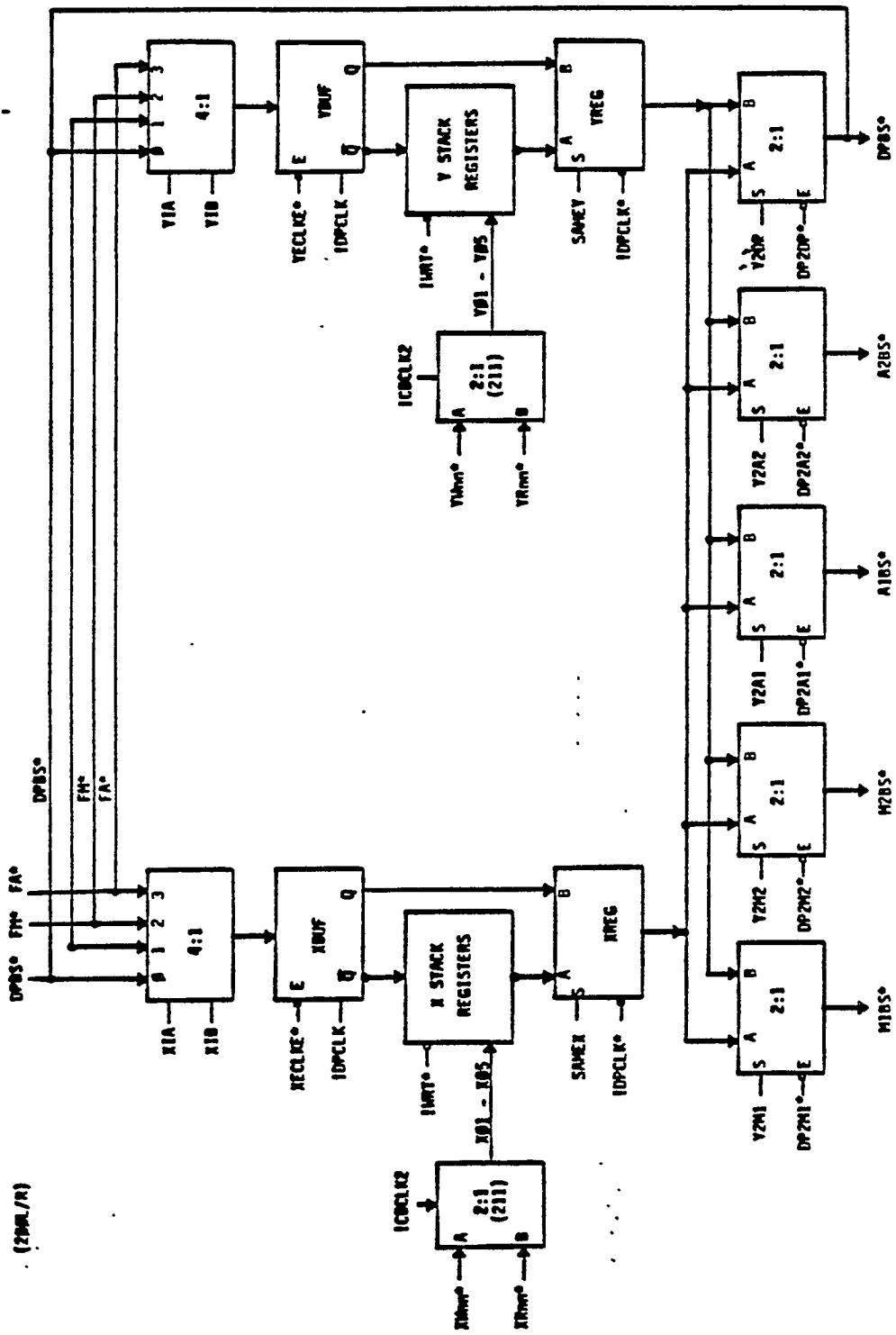
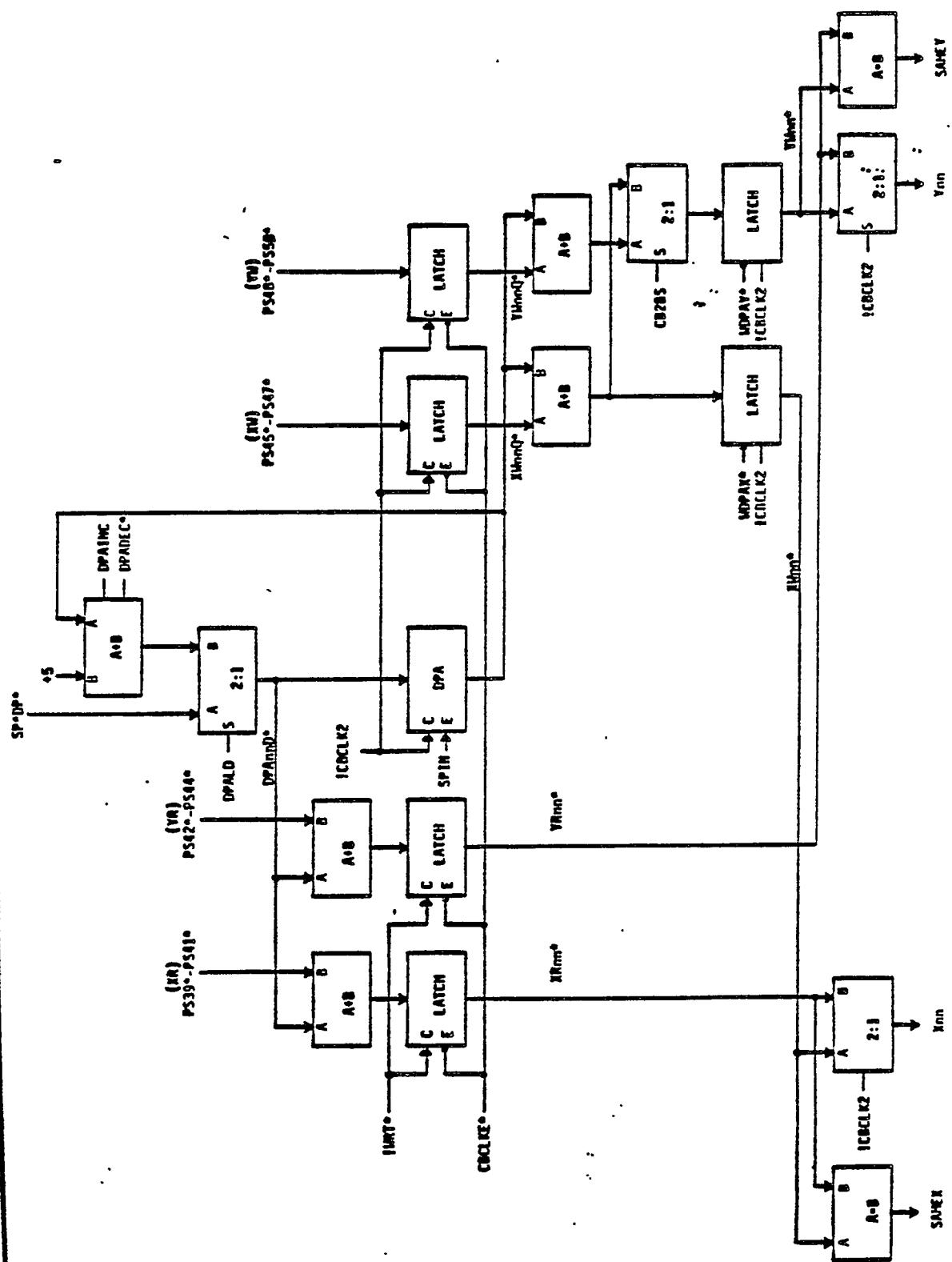


Figure DPAD7
Data Pad

GE MEDICAL SYSTEMS INSTITUTE



Data Pad Address Logic Figure DPA08

GE MEDICAL SYSTEMS INSTITUTE

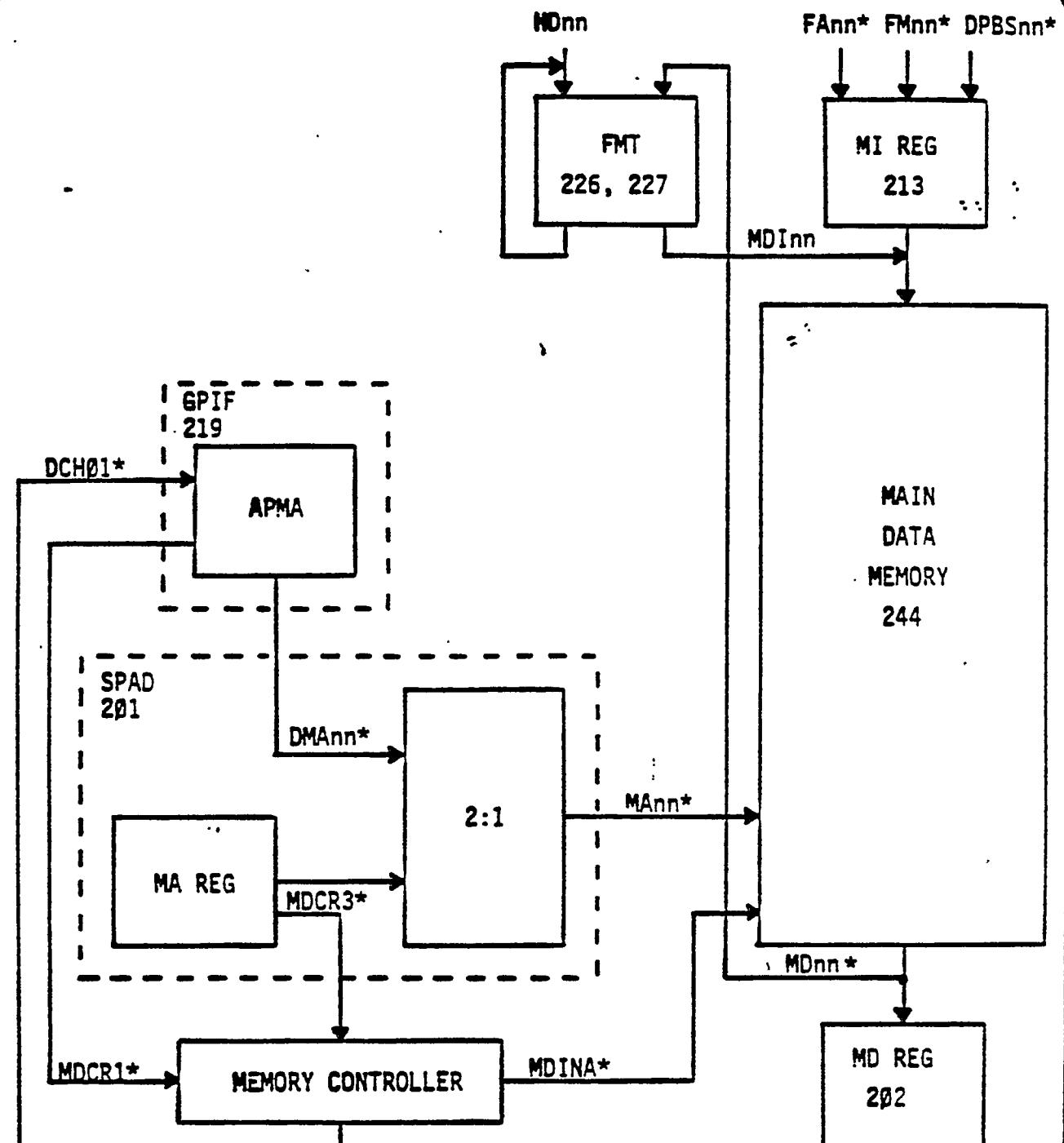


Figure MD1
Main Data System Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

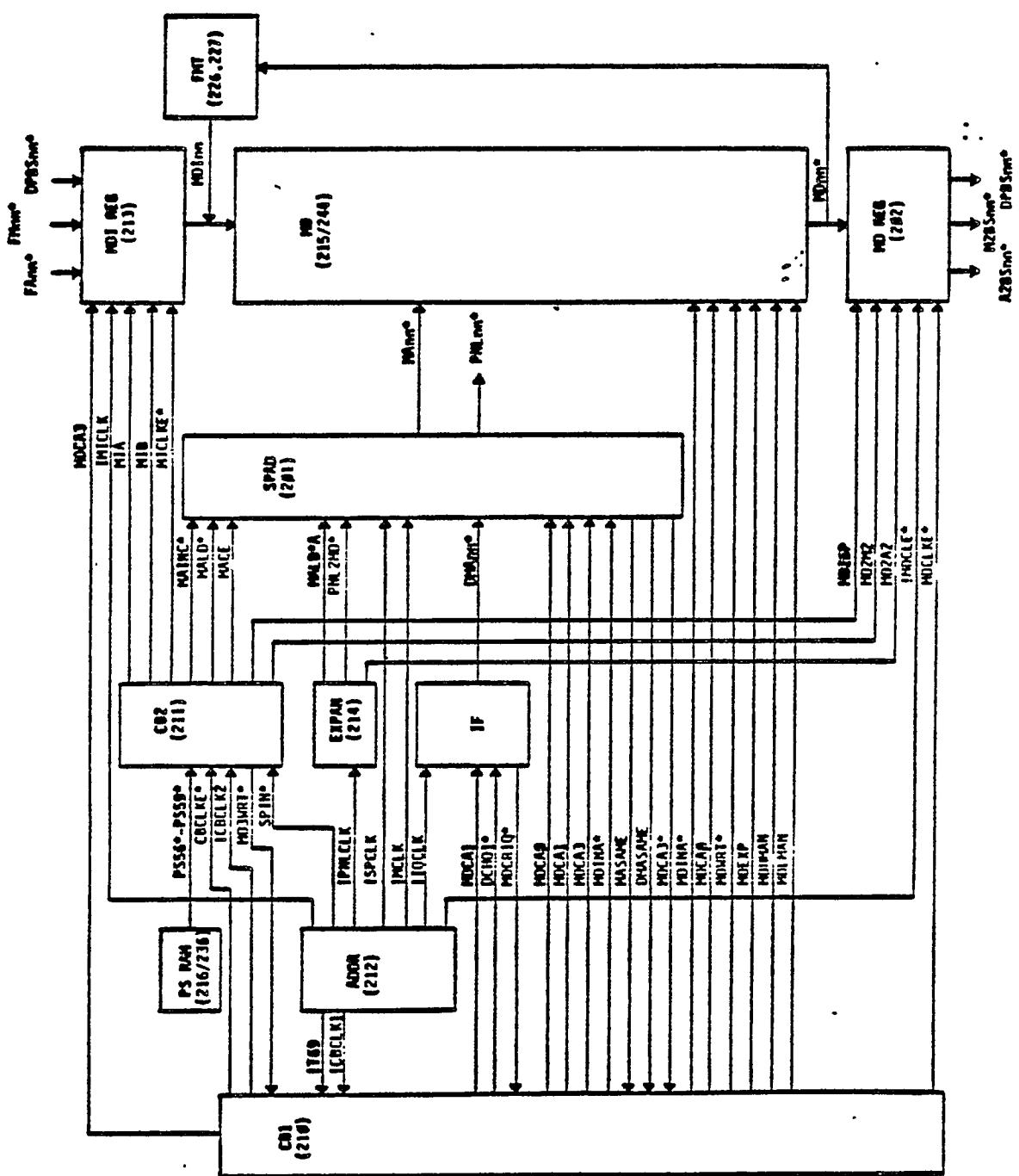
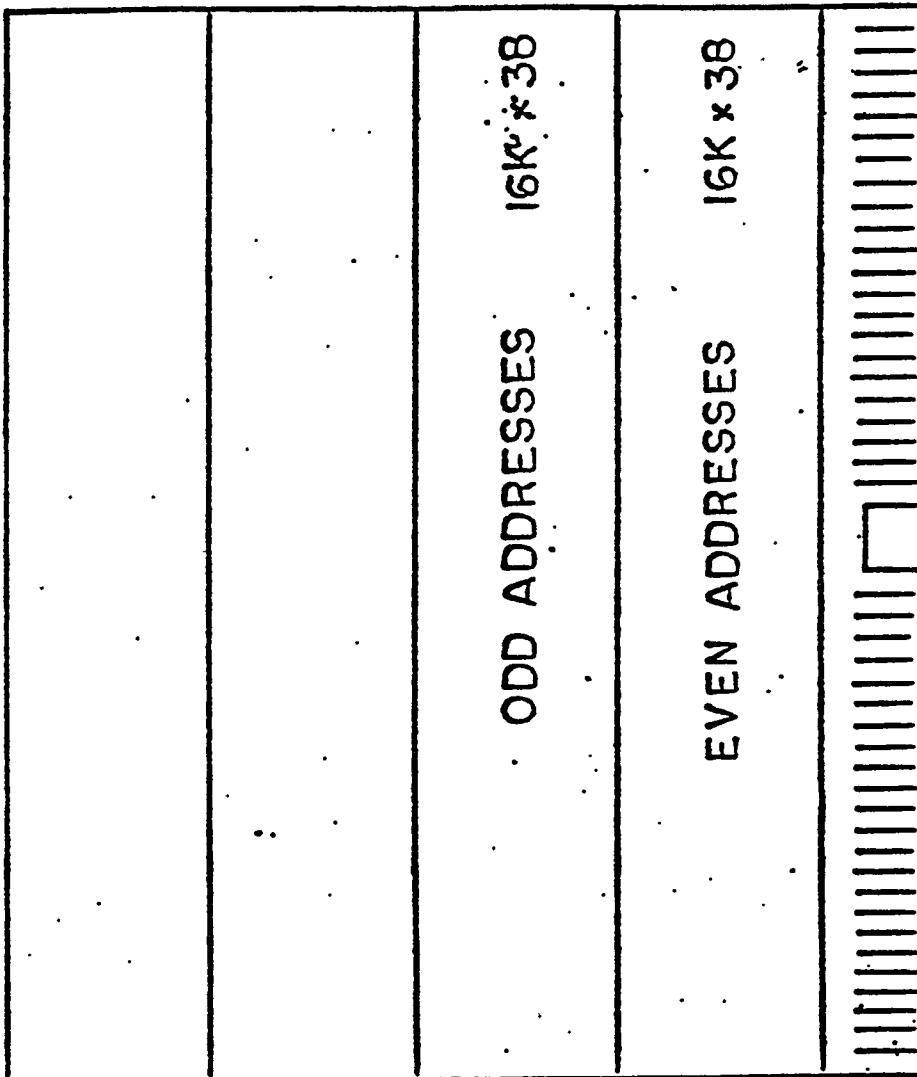


Figure MD2
Main Data System Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

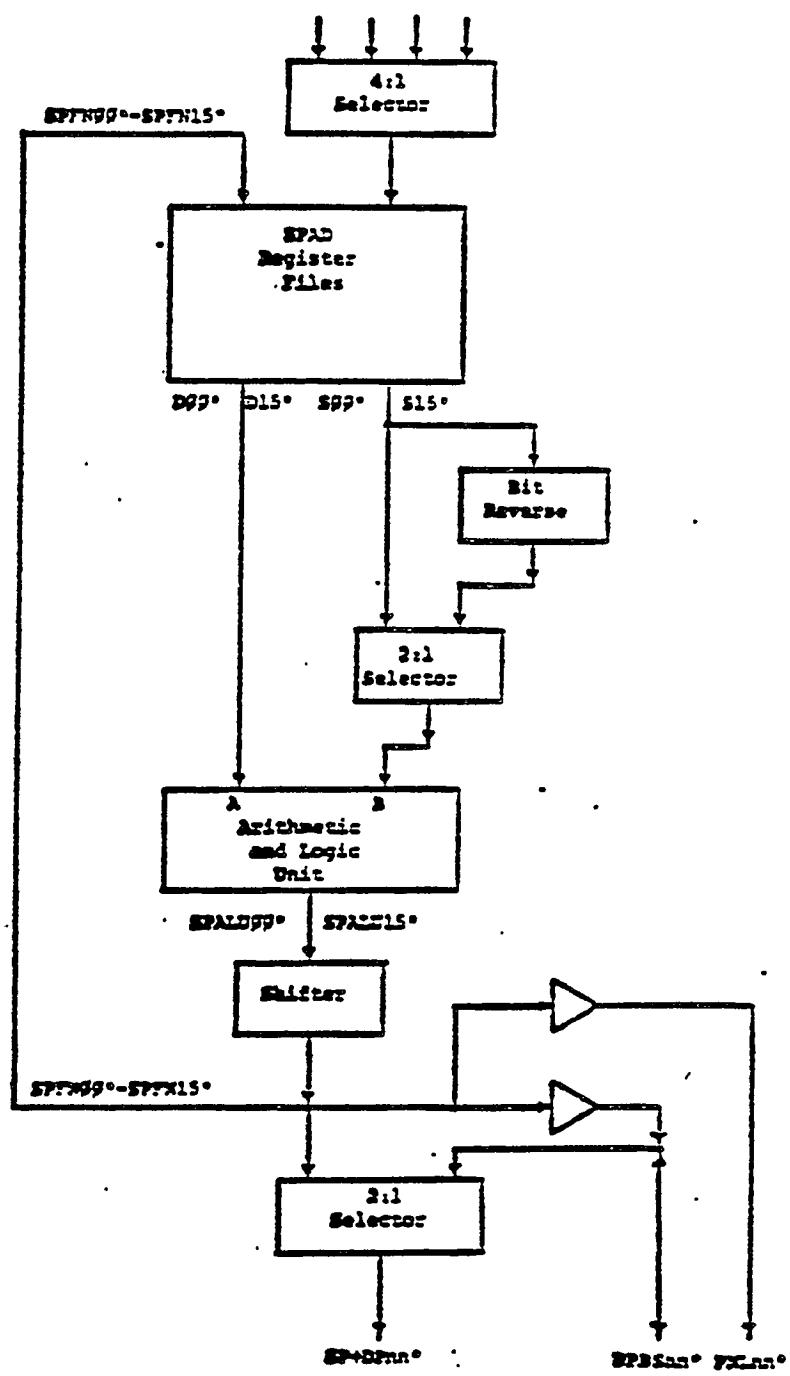
244 BOARD

COMPONENT SIDE UP



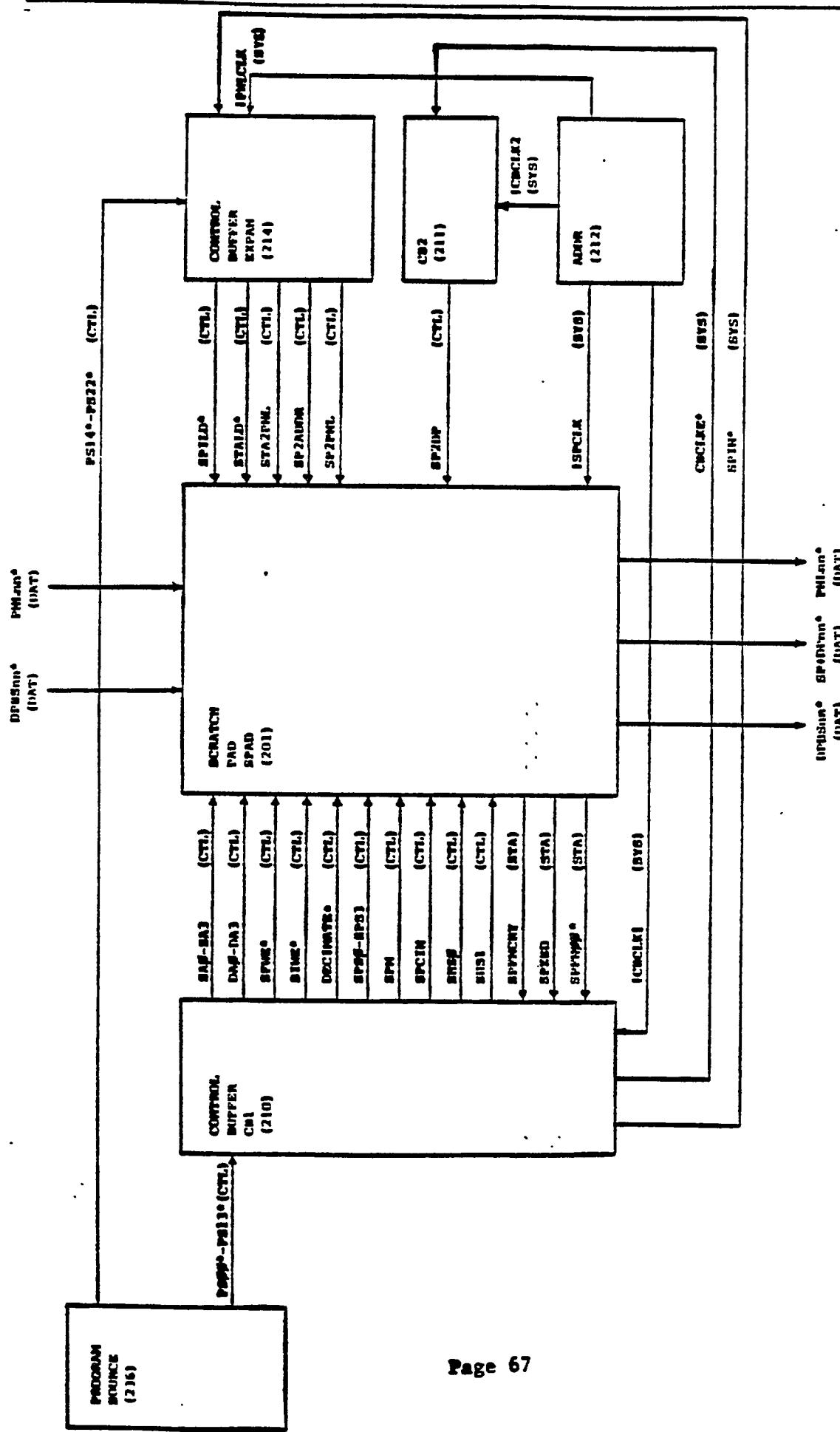
MAIN DATA

GE MEDICAL SYSTEMS INSTITUTE



SPAD Block Diagram

GE MEDICAL SYSTEMS INSTITUTE



SPAD2
SPAD Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

PCGG*-PS13*

9	1	2	3	4	5	6	7	8	9	10	11	12	13
D	SPAD Operation	SPAD Shift		SPAD Source			SPAD Destination				Program Source		

ADDR 9,5

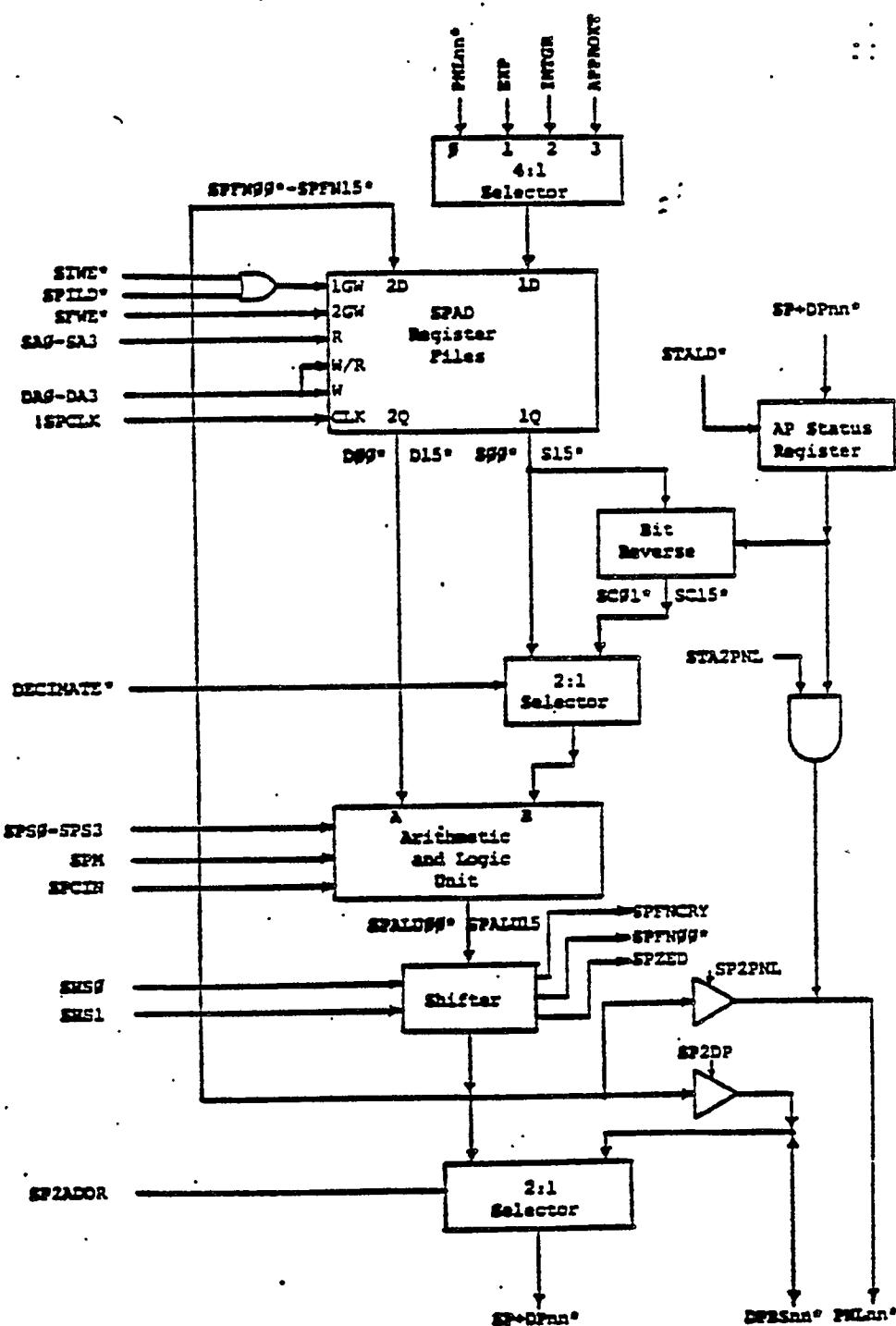


FIGURE SPAD3

Control Detail of SPAD Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

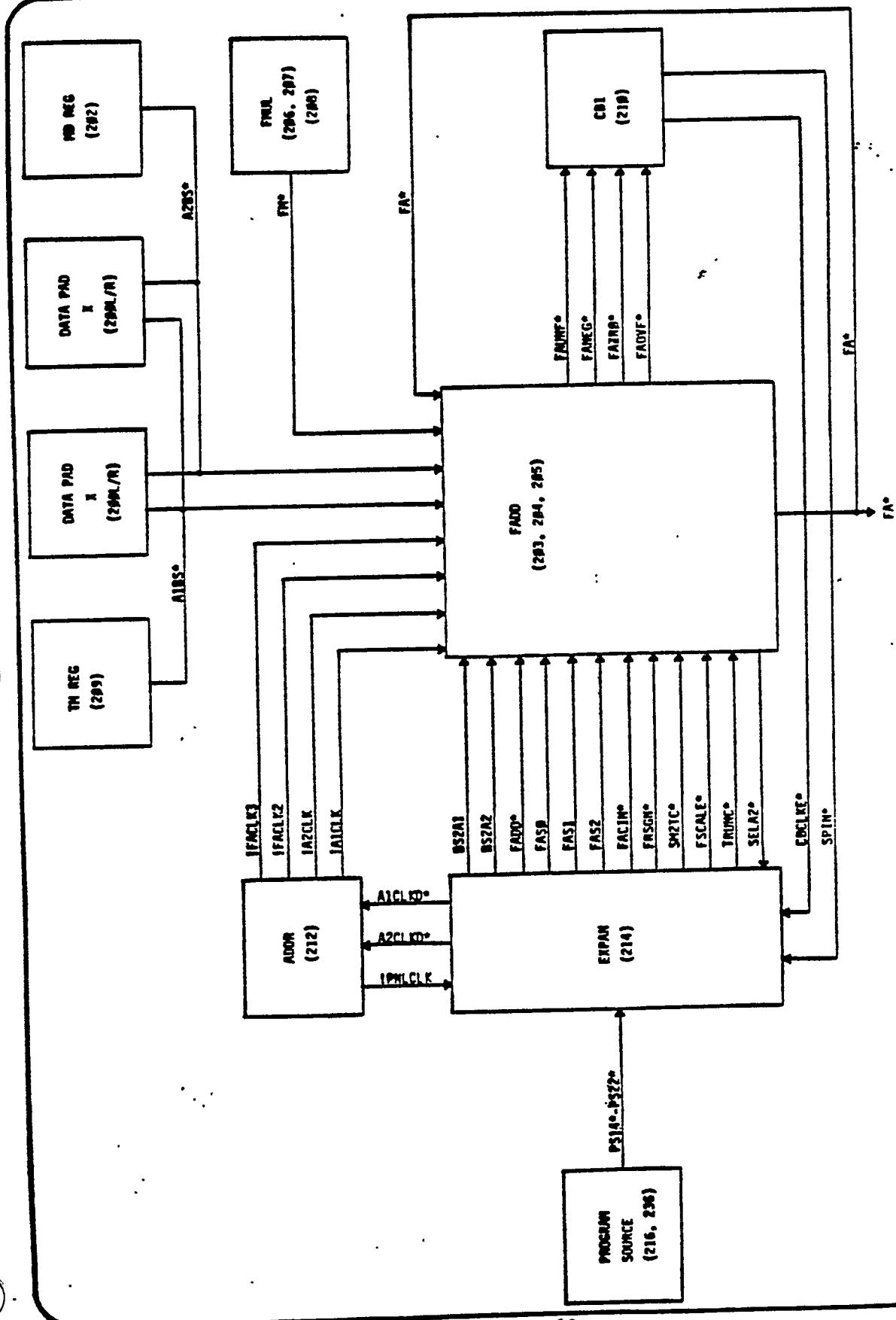
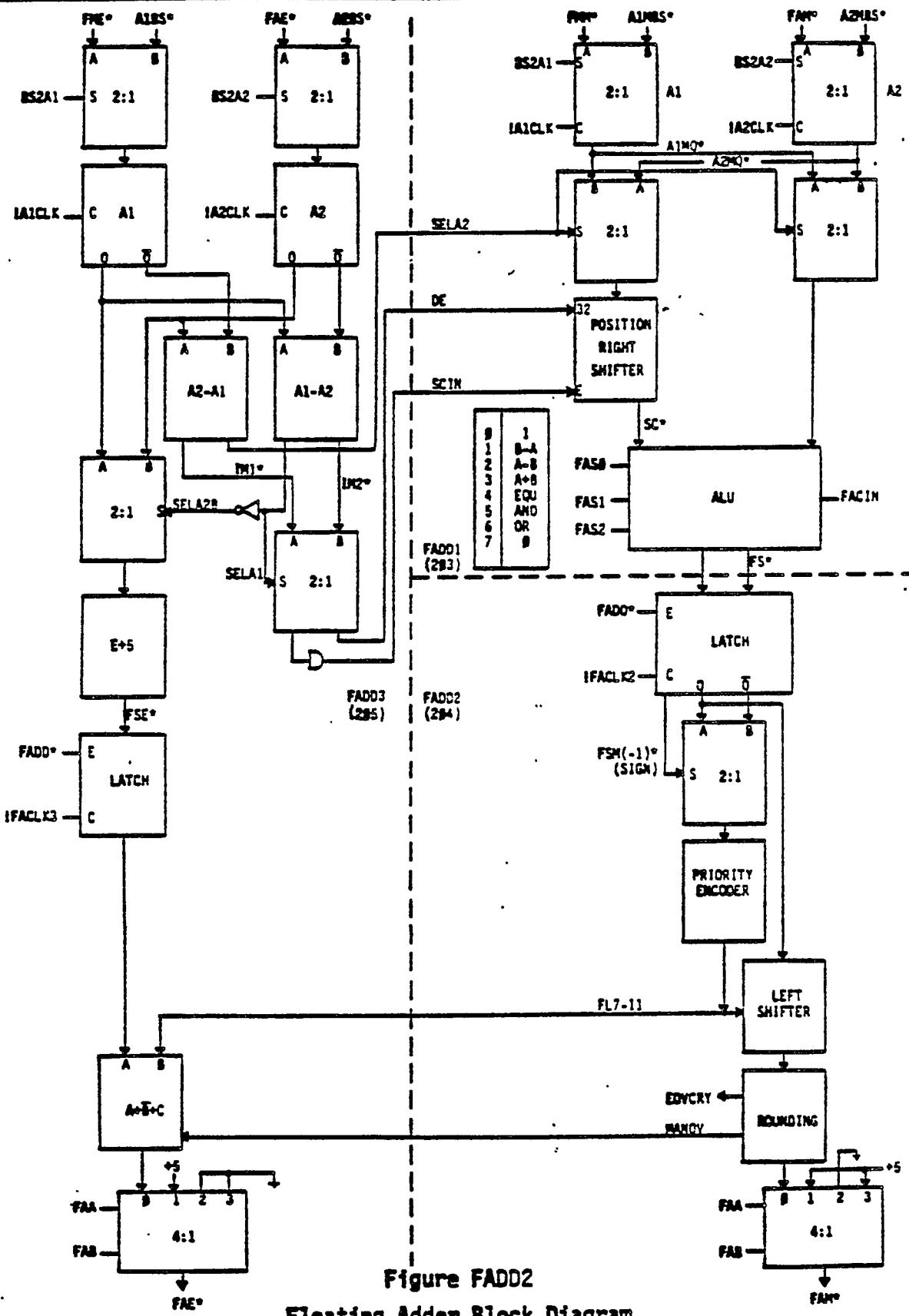


Figure FADD1
Floating Adder System

GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

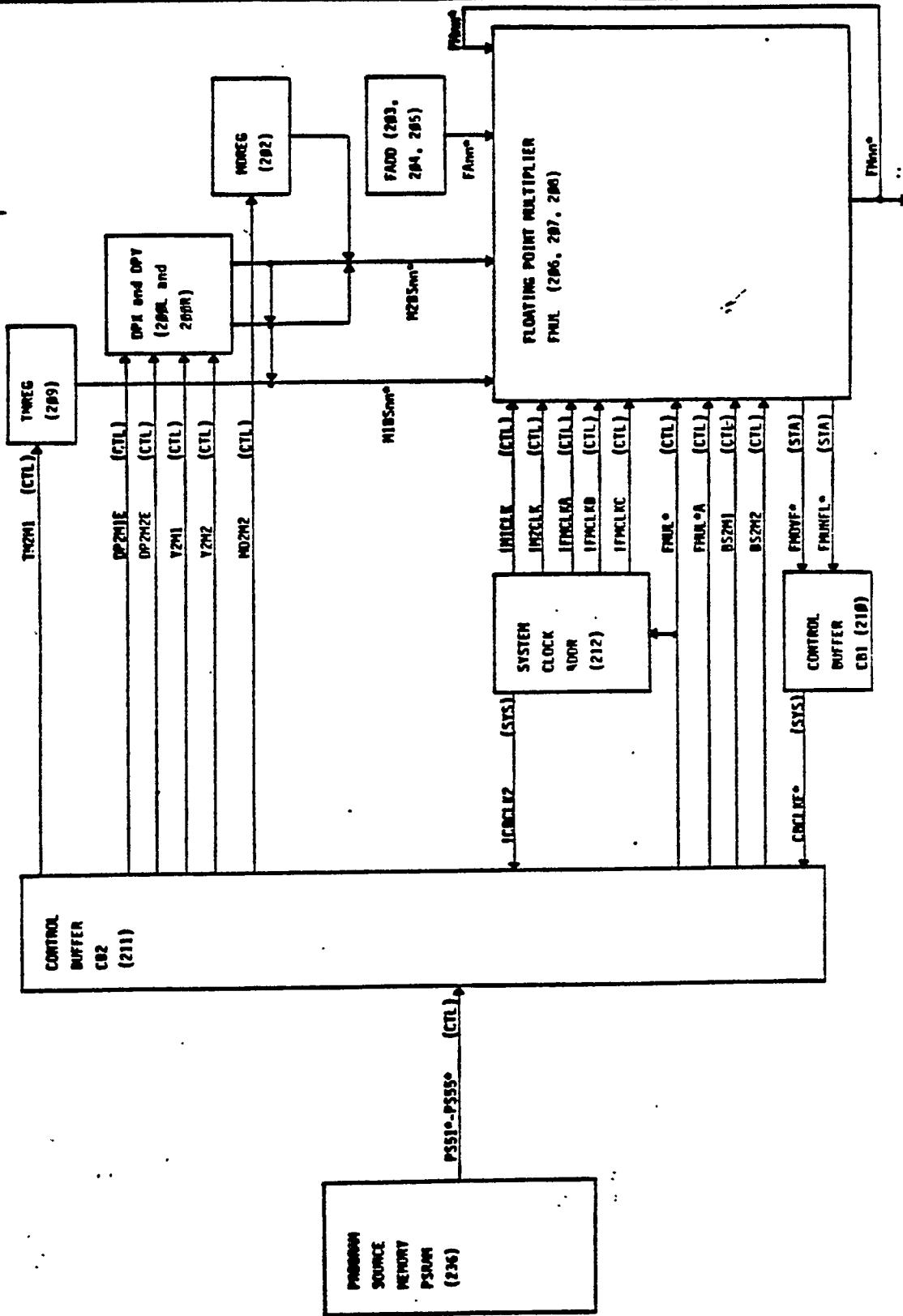


Figure FMUL1
Floating Multiplier System Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

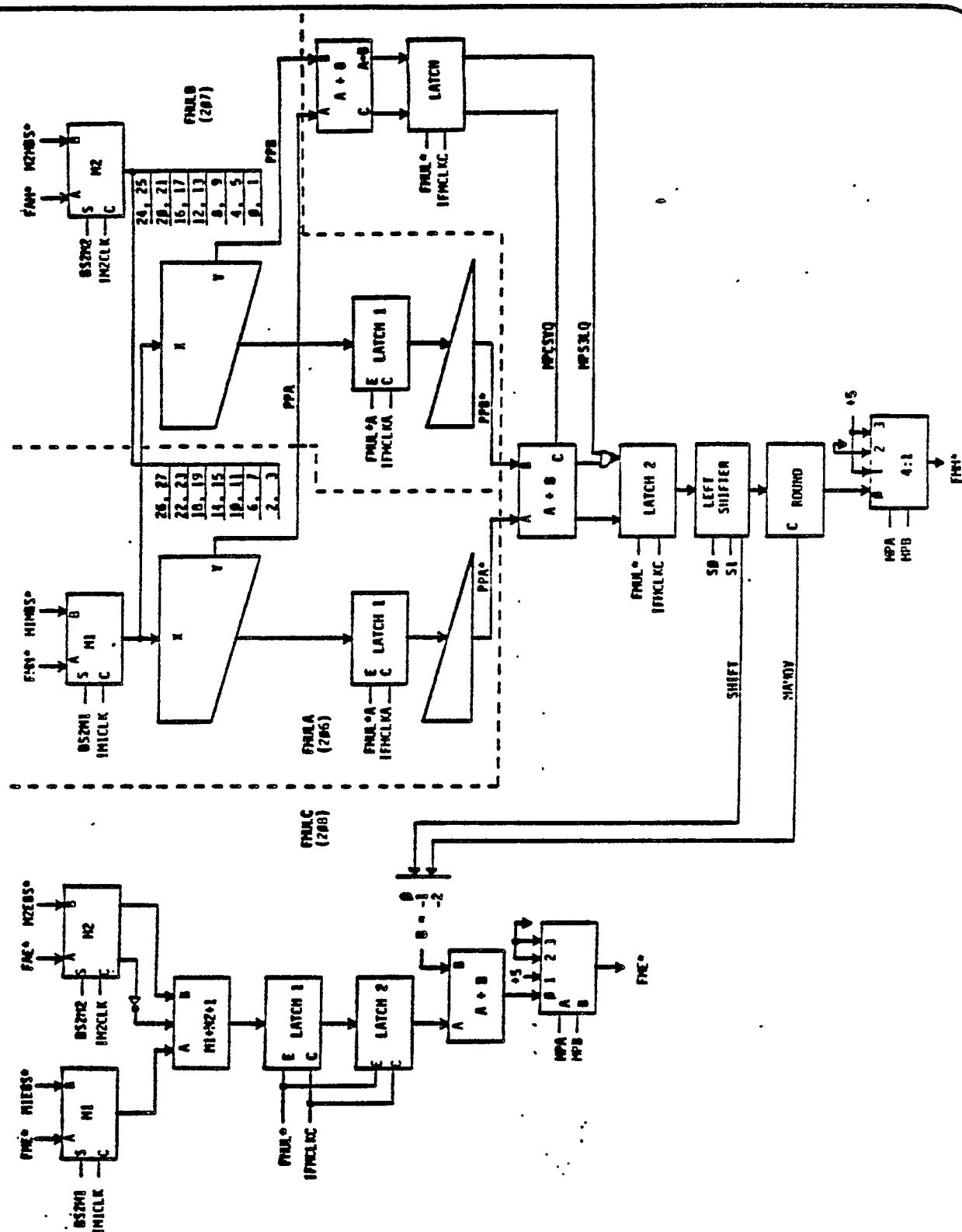


Figure FMUL2
Floating Multiplier Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

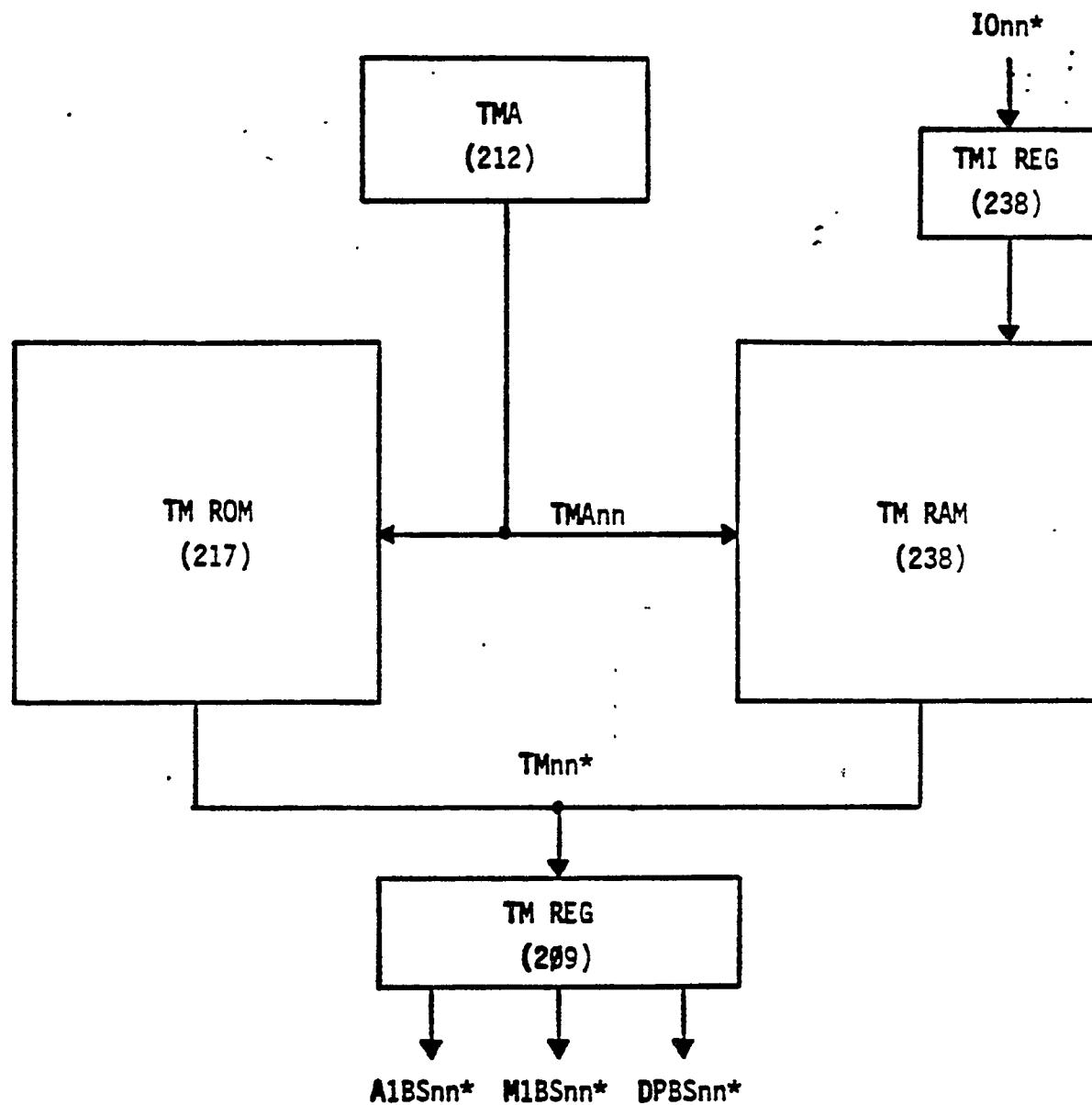


Figure TMA1
Table Memory System Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

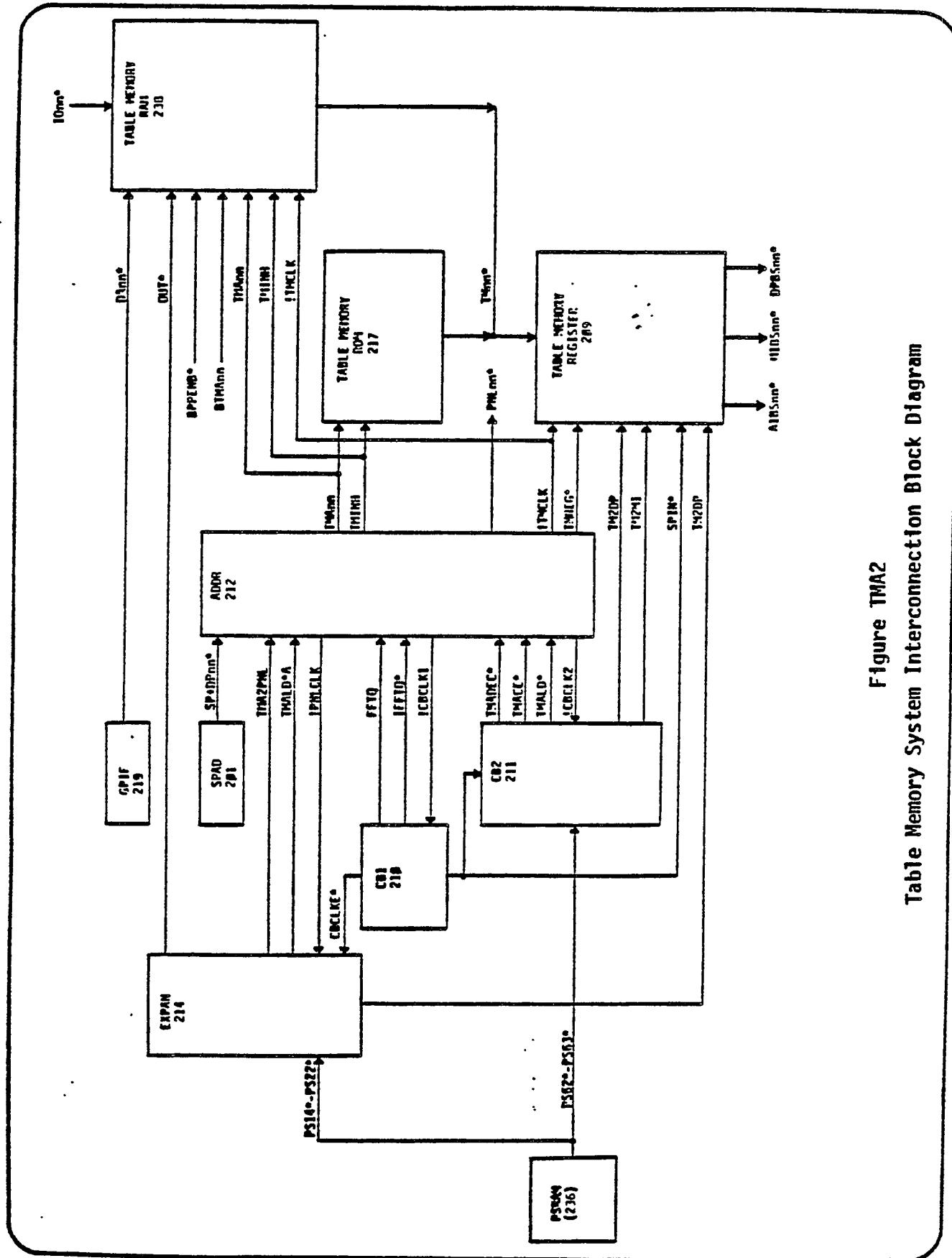


Figure TM22
Table Memory System Interconnection Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

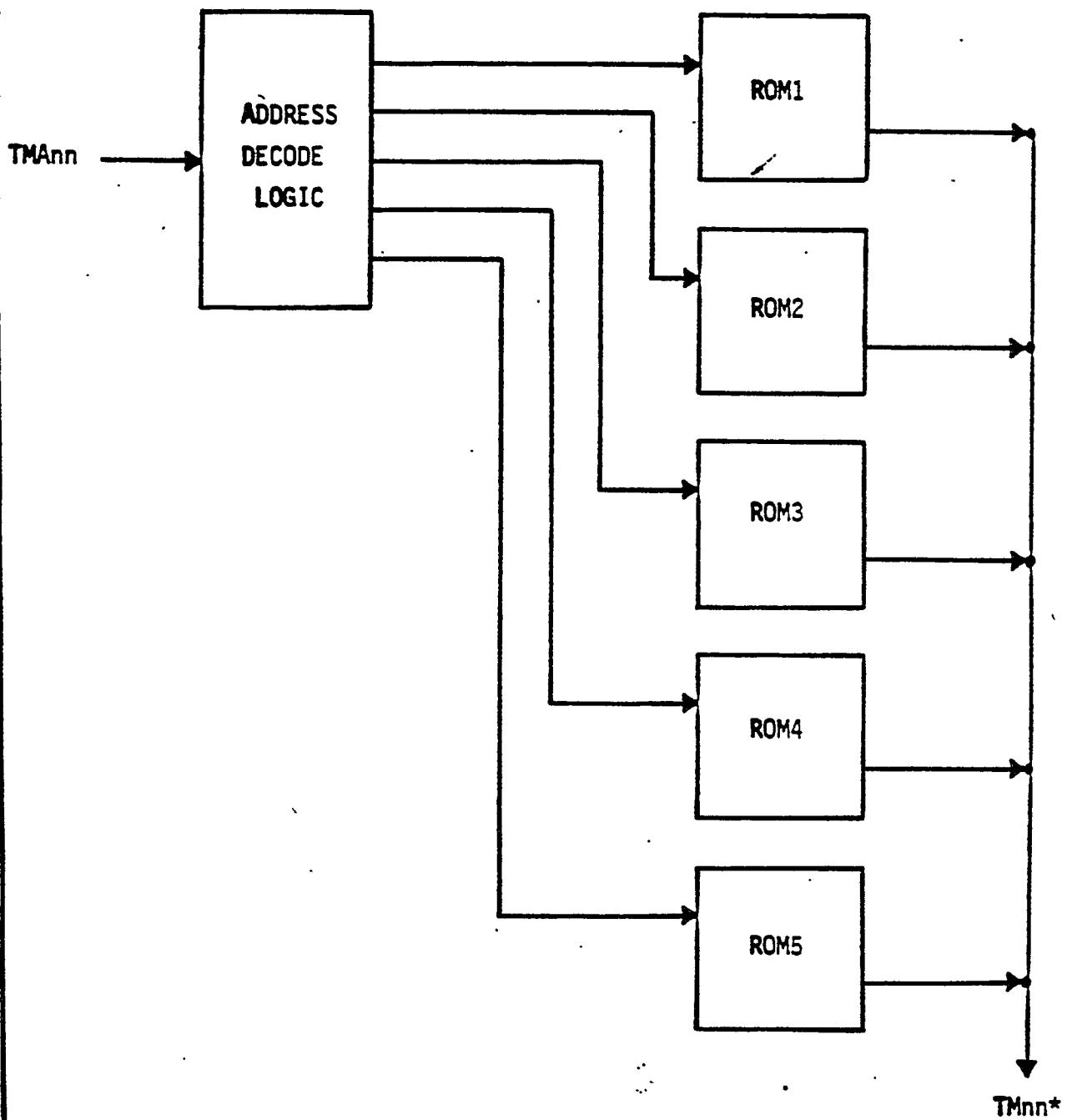


Figure TMA3
TM ROM Memory Elements

GE MEDICAL SYSTEMS INSTITUTE

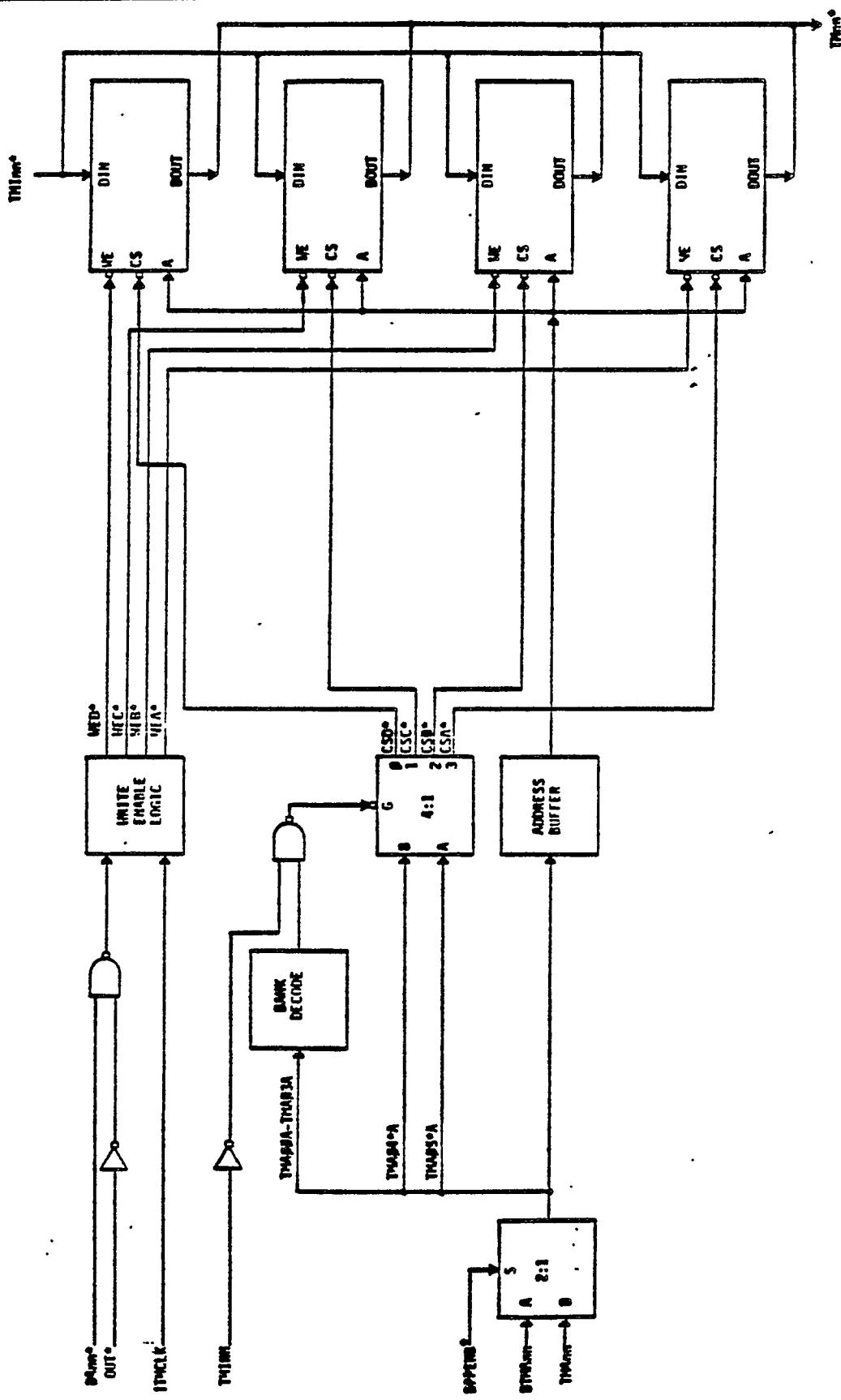


Figure TMA4
TMRAM Memory Elements

GE MEDICAL SYSTEMS INSTITUTE

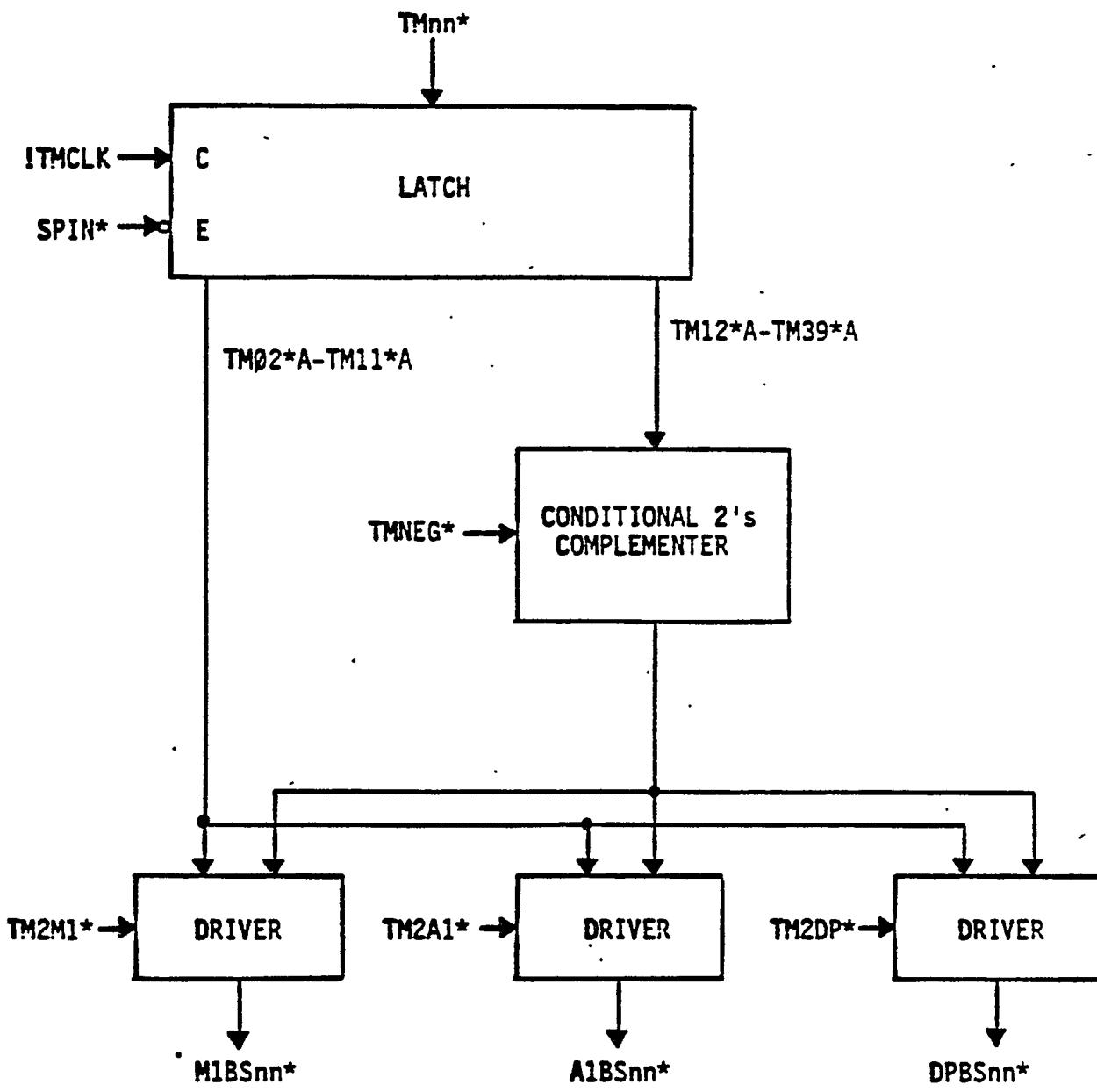


Figure TMA5
Table Memory Register Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

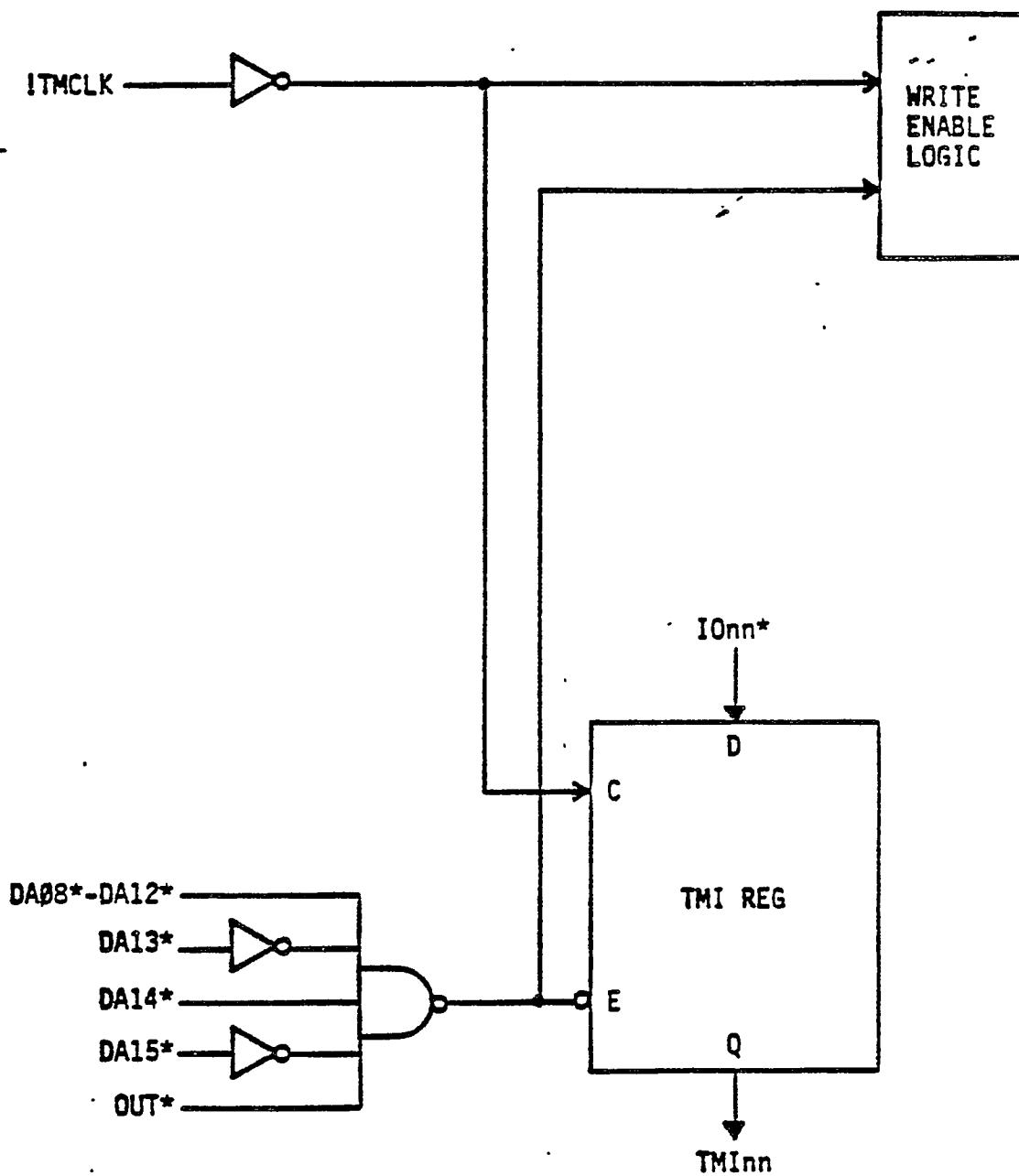


Figure TMA6
Table Memory Input Register Block Diagram

GE MEDICAL SYSTEMS INSTITUTE

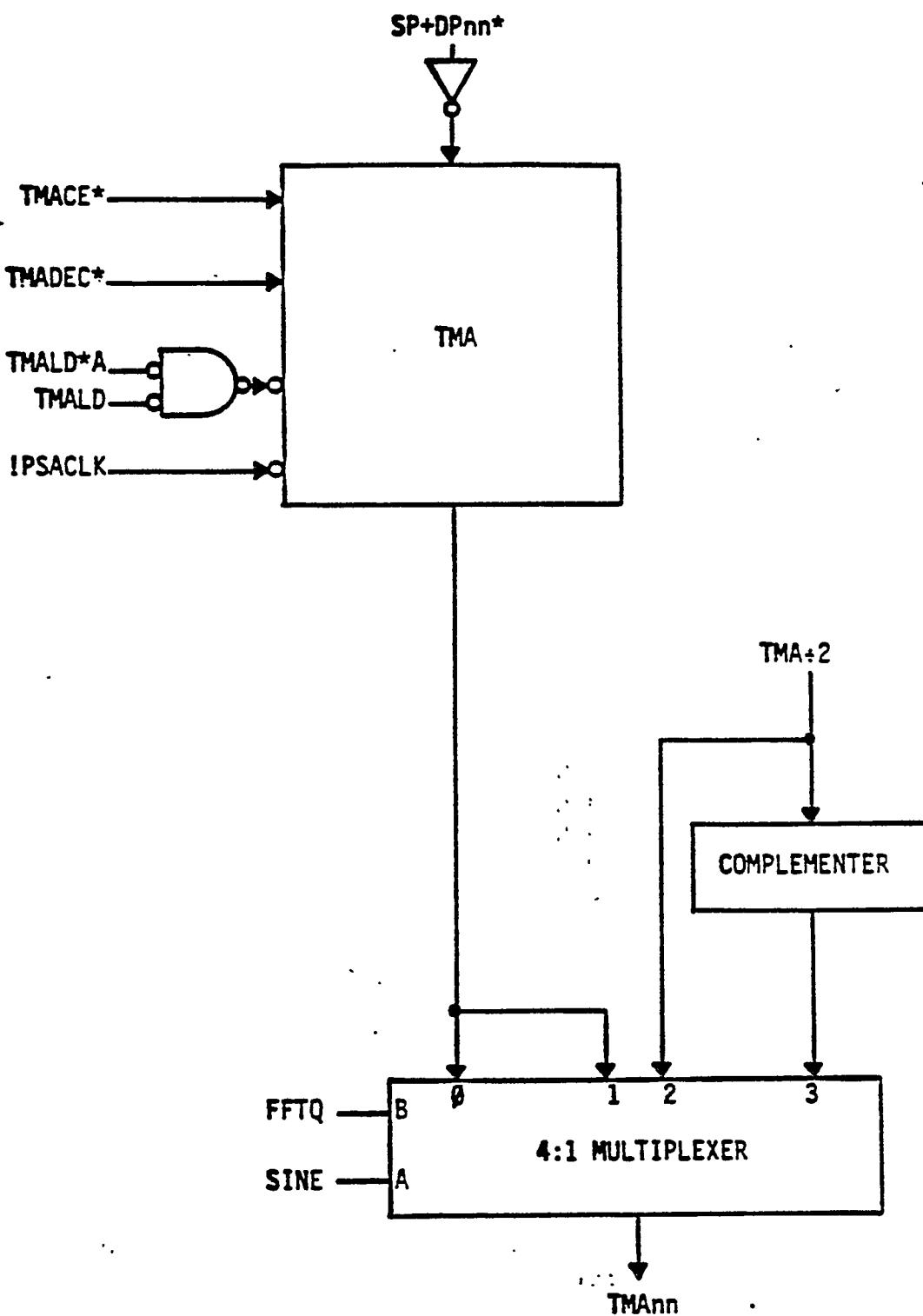


Figure TMA7
Table Memory Address Logic

GE MEDICAL SYSTEMS INSTITUTE

COMPONENT SIDE UP

0-511 (0-777₈)

512-1023 (1000-1777₈)

1024-1535 (2000-2777₈)

1536-2047 (3000-3777₈)

2048-2559 (4000-4777₈)



**TM ROM
217 BOARD**

GE MEDICAL SYSTEMS INSTITUTE

238 BOARD

FULL
BD

COMPONENT SIDE UP

EVEN BITS TM₀₂-TM₃₈

ADDRESS
(6000-7777)

ODD BITS TM₀₃-TM₃₉

EVEN BITS TM₀₂-TM₃₈

ADDRESS
(4000-5777)

ODD BITS TM₀₃-TM₃₉

ADDRESS
(2000-3777)

EVEN BITS TM₀₂-TM₃₈

ADDRESS
(0-1777₈)

ODD BITS TM₀₃-TM₃₉



HIGH EXPONENT	MANTISSA	LOW MANTISSA
---------------	----------	--------------

02	11	12	23	24	39
----	----	----	----	----	----

TABLE MEMORY
WORD FORMAT

0 0 1 7 7 7 | 0 0 7 7 7 7 | 1 7 7 7 7 7

ERROR MESSAGE
DATA FORMAT

GE MEDICAL SYSTEMS INSTITUTE

238 BOARD HALF BD

COMPONENT SIDE UP

EVEN BITS TM02-TM38	ADDRESS (12000-13777)
ODD BITS TM03-TM39	
EVEN BITS TM02-TM38	ADDRESS (10000-11777)
ODD BITS TM03-TM39	

HIGH EXONENT	MANTISSA	LOW MANTISSA
02	11 12	23 24

0 0 1 7 7 7 | 0 0 7 7 7 7 | 1 7 7 7 7 7

TABLE MEMORY
WORD FORMAT

ERROR MESSAGE
DATA FORMAT

**POWER SUPPLY
SETTINGS**

FAST RECONSTRUCT PROCESSOR

AP120B

GE MEDICAL SYSTEMS INSTITUTE

AP - 120B Check

Line Power

Refer to the FPS Array Processor Maintenance Manual Section 3.6.1 for line power checks. If checking the 50Hz incoming power, do not verify peak to peak voltages on the scope. It should be sufficient to check the RMS value.

D.C. Power Supplies

Again, refer to the FPS manual mentioned above. All measurements should be made with the Fluke 8030A or equivalent.

a. +5V 150 A Supplies

PS1 is on the left (from the rear of the AP cabinet), and PS2 is on the right. The wiring for them is identical and they are directly interchangeable.

1. Power off the AP.
2. Check all nut and bolt power connections for tightness.
3. Place a D.V.M. between the center of the ground bracket and the center of the +5 volt bracket on the bottom of the AP backplane.
4. Preset the voltage pots on both power supplies to maximum CCW (low voltage). Note: These are one-turn pots placed very close to the power output terminals. Use a non-conductive tweaker for all adjustments.
5. Preset the overload pots, used to set current limit, maximum CW (no limit).
6. Pull the fuse on PS1.
7. Power up the AP and note that backplane voltage is \approx +4.25.
8. Advance the voltage adjust pot on PS2 CW until a reading of 5.75 volts is attained--then quickly back down the overload pot CCW until AP voltage falls off to 2.80 VDC

GE MEDICAL SYSTEMS INSTITUTE

Leave the voltage across the AP at 5.75 volts for as short a time possible.

9. Power off the AP.
10. Replace the fuse in PS1 and pull the fuse on PS2.
11. Power up the AP, and note voltage $\approx +4.25$.
12. Turn overload pot on PS1 CCW until voltage reads 2.80 ($2.80 \rightarrow 3.15$).
13. Power off AP.
14. Replace the fuse on PS2.
15. Power up AP and note voltage $\approx +4.25$.
16. Adjust PS1 voltage pot CW until voltage at AP backplane equals 5.15 VDC.
17. To verify that adjustments were made properly, make the following measurement. Leave the meter common on the AP backplane. Remove the "+" meter lead and attach it to PS1 (-). On the 200 mv scale, read roughly $-80 \rightarrow -90$ mv. Move the "+" meter lead to PS2 (-), and read roughly $-130 \rightarrow -140$ mv. This indicates that PS2 is outputting ≈ 80 ADC and PS1 is outputting the rest. (What you are measuring is the voltage drop across the return wires from the AP common to the supplies and, hence, is dependent on lead length and the quality of the connections.)

--DO NOT attempt to balance the current flow of the two supplies--this leads to oscillation and possible destruction of the supplies.

--If voltage needs to be "touched-up", use the voltage pot on PS1. DO NOT touch any other pots.

--The AP may be run with one supply by removing the defective supply, taping up the unused power leads (117 and +5), set current limit to maximum CW, and backplane voltage to 5.15.

GE MEDICAL SYSTEMS INSTITUTE

- NOTES:
1. All adjustments are interactive and will require a couple iterations of the steps above.
 2. If the supplies cannot be adjusted because of low values, a supply may be out. Remove power. Wait at least 5 minutes. Disconnect the power cables at the supply end and recheck the supplies individually.
 3. Check both fuses on the supply in question (one internal, one external).

Observe all cautions on the supply. After waiting 5 minutes check the supply with a DVM to insure all remaining charge on the filter capacitors has led off.

b. -5V Supply

The -5V supply should be -5.00 ± 0.05 V. Refer to Array Processor Maintenance Manual for the location of the -5V pad. Since the DVM can go negative, leave the minus lead on common.

c. +12V Supply

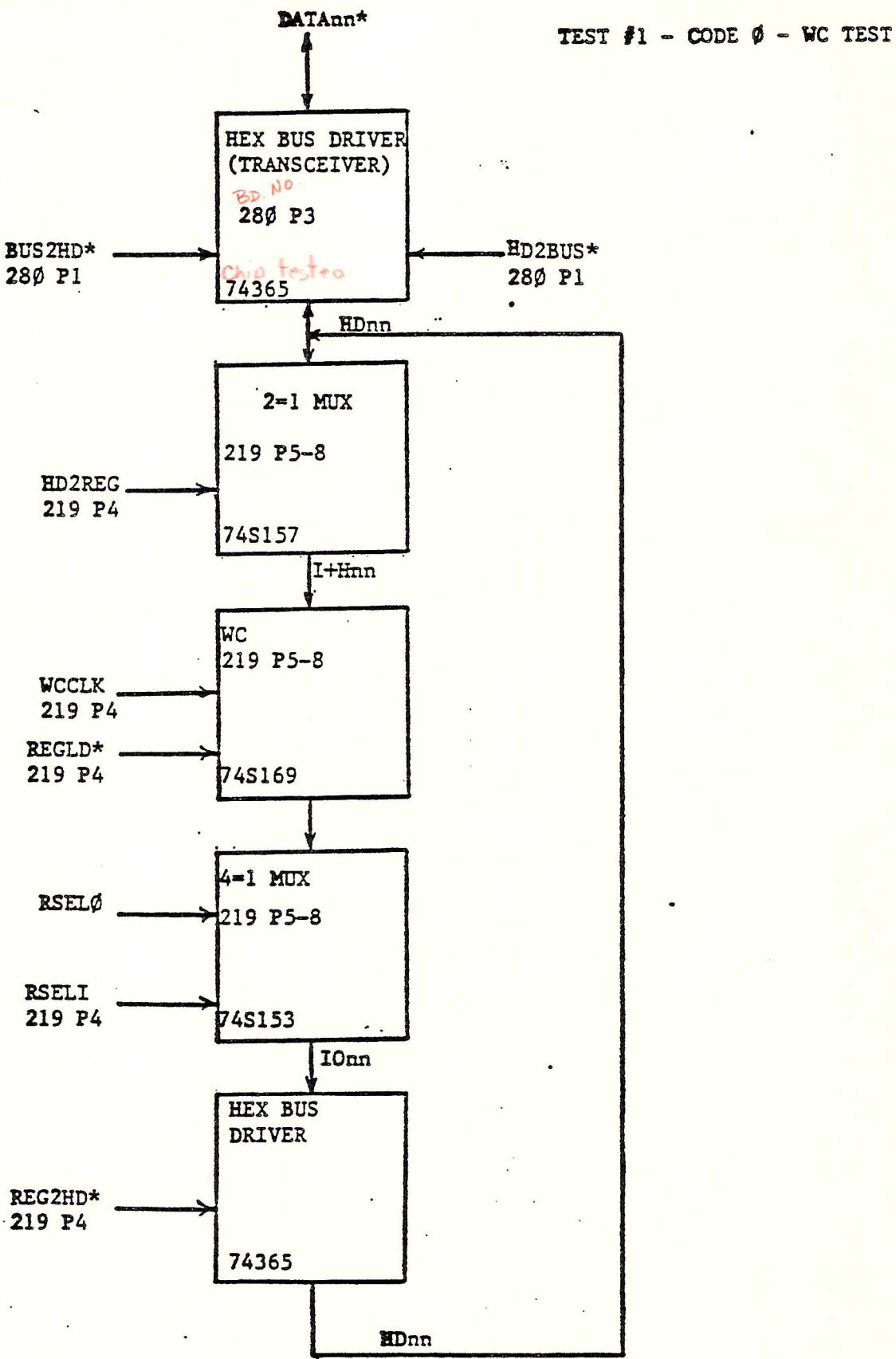
Check the +12V supply as described in the Array Processor Maintenance Manual.

FIXED FLOWS

FAST RECONSTRUCT PROCESSOR

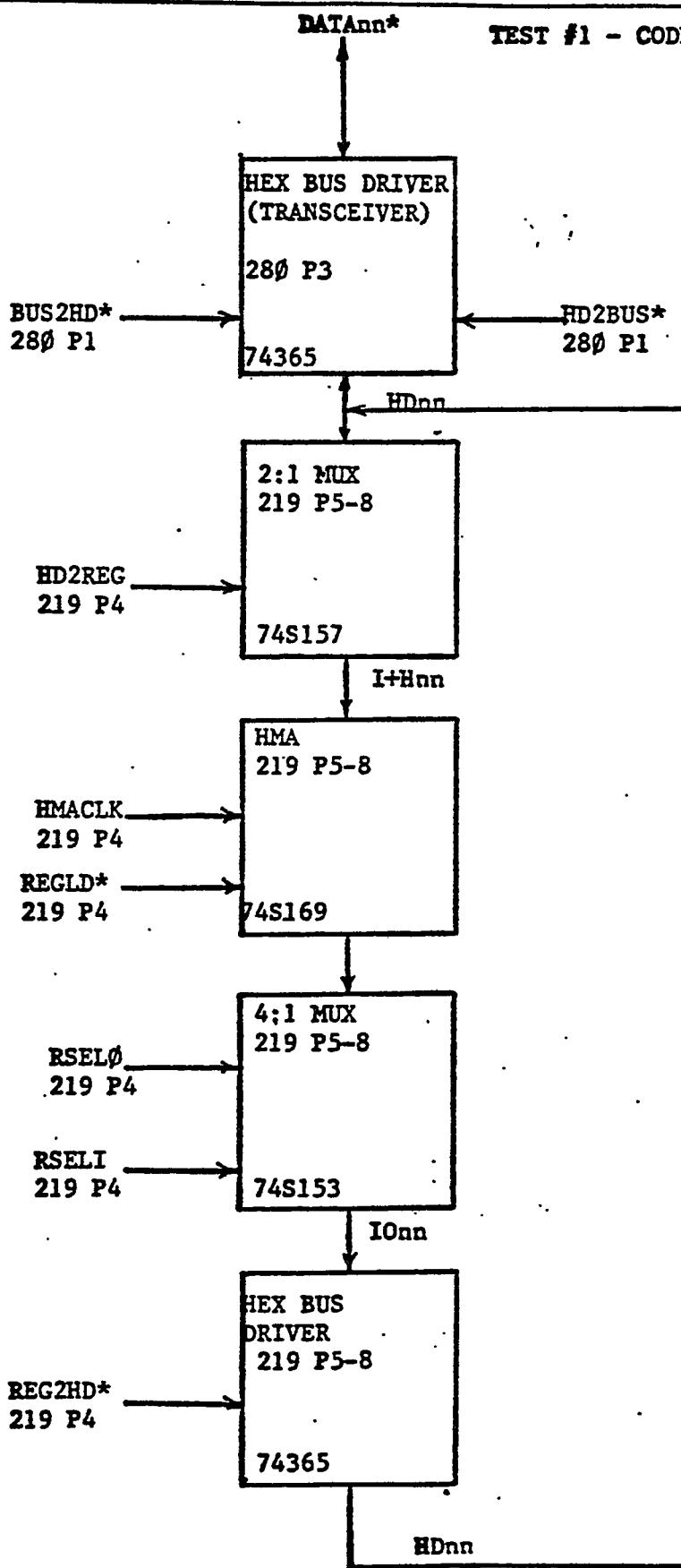
AP120B

GE MEDICAL SYSTEMS INSTITUTE

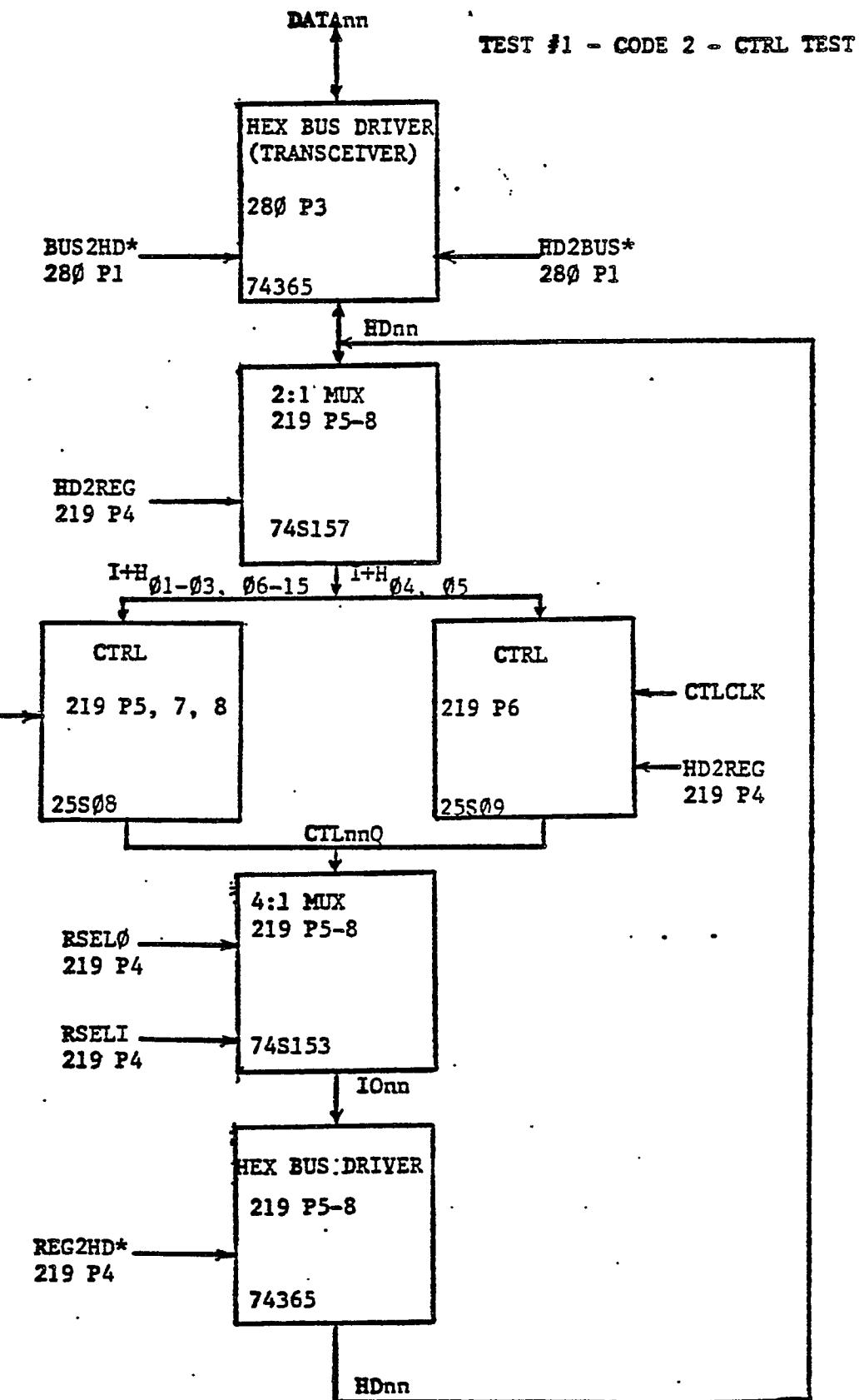


GE MEDICAL SYSTEMS INSTITUTE

TEST #1 - CODE 1 - HMA TEST

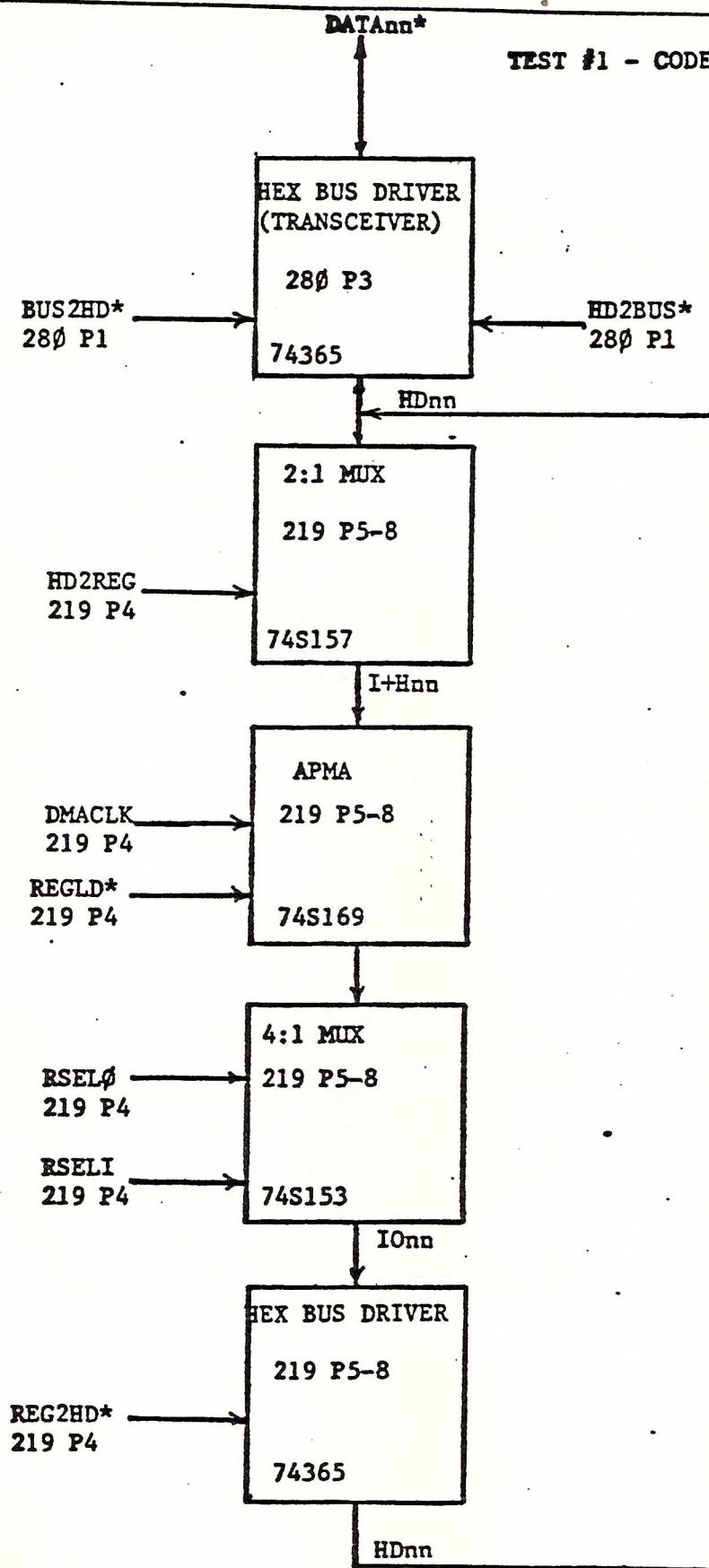


GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

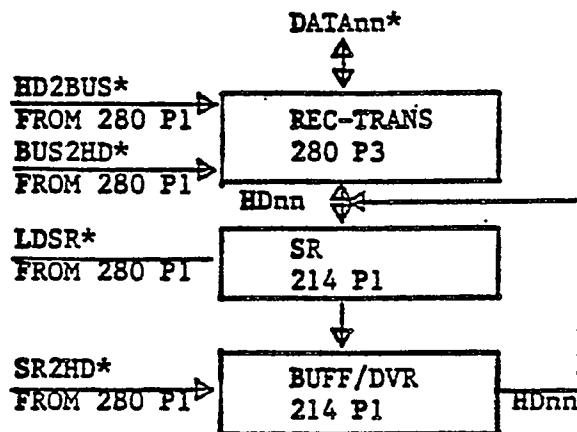
TEST #1 - CODE 3 - APMA TEST



GE MEDICAL SYSTEMS INSTITUTE

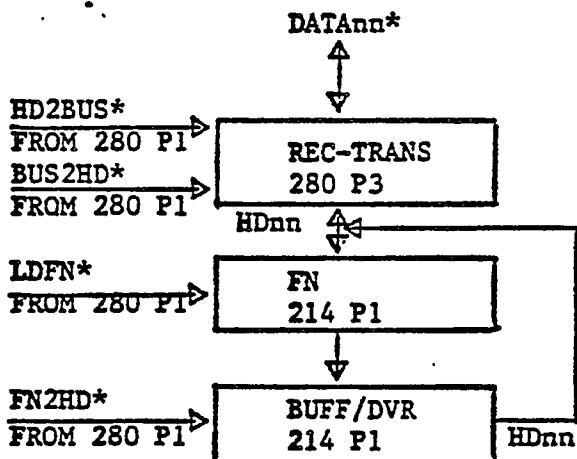
TEST 1

CODE 4 DATA → HD → (LDSR*) SR
 SR → (SR2HD*) HD → DATA



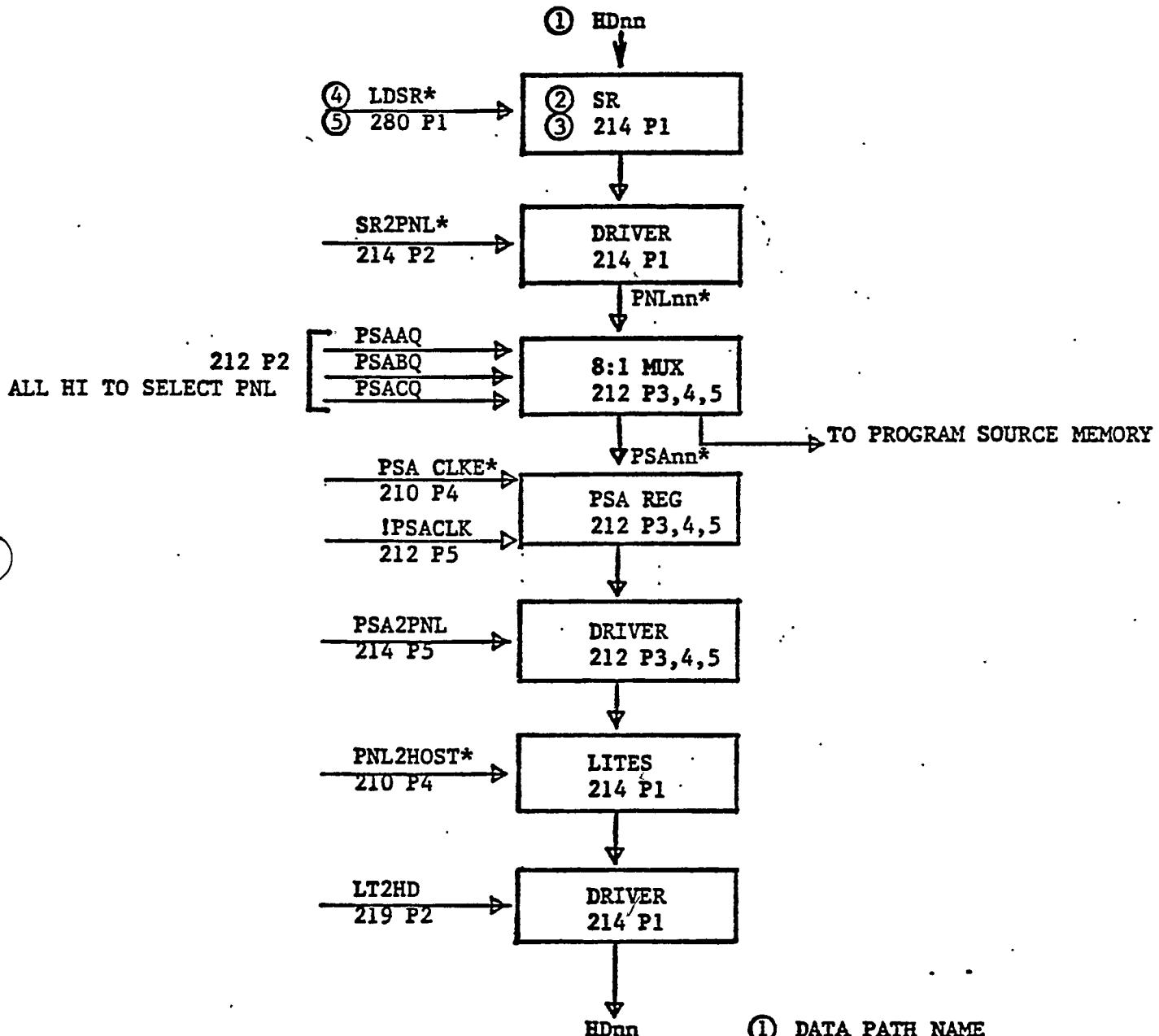
TEST 1

CODE 5 DATA → HD → (LDFN*) FN
 FN → (FN2HD*) HD → DATA



GE MEDICAL SYSTEMS INSTITUTE

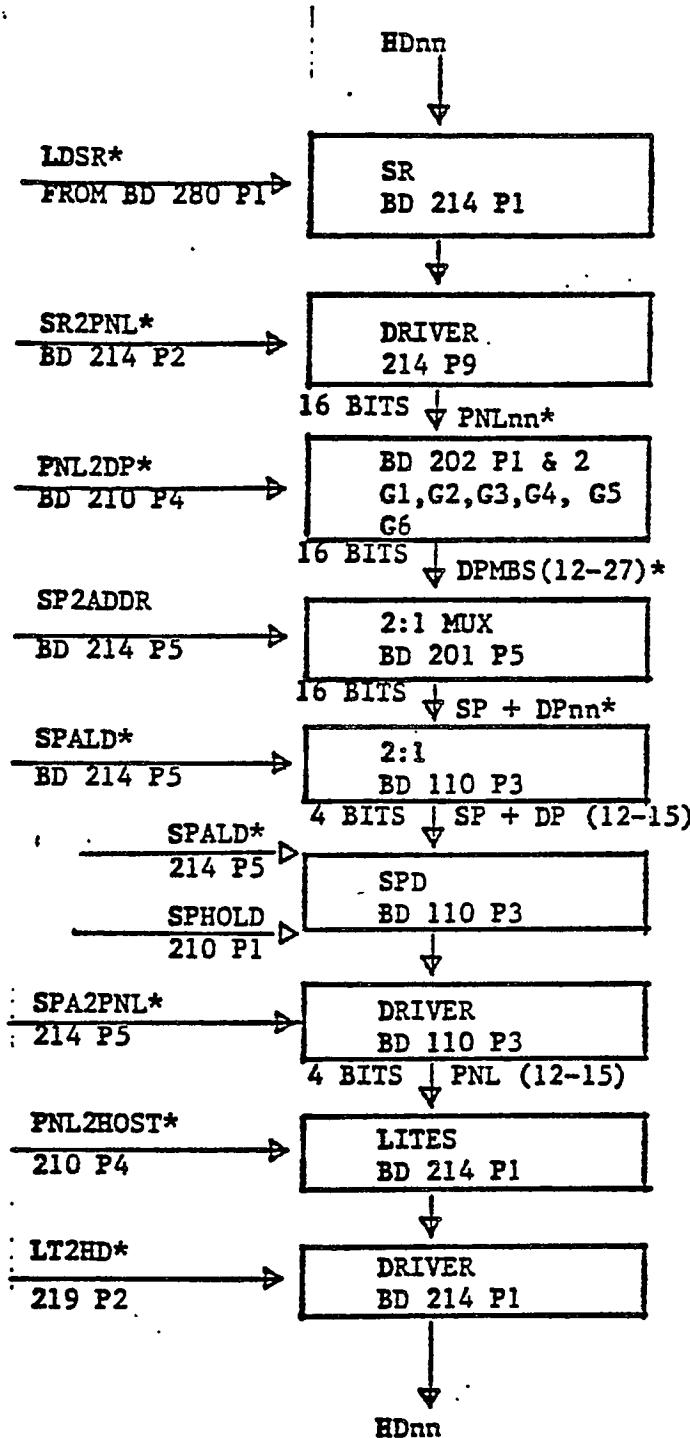
TEST CODE 6



- ① DATA PATH NAME
- ② BLOCK NAME
- ③ BOARD # & PAGE #
- ④ COMMAND SIGNAL NAME
- ⑤ BOARD # & PAGE #

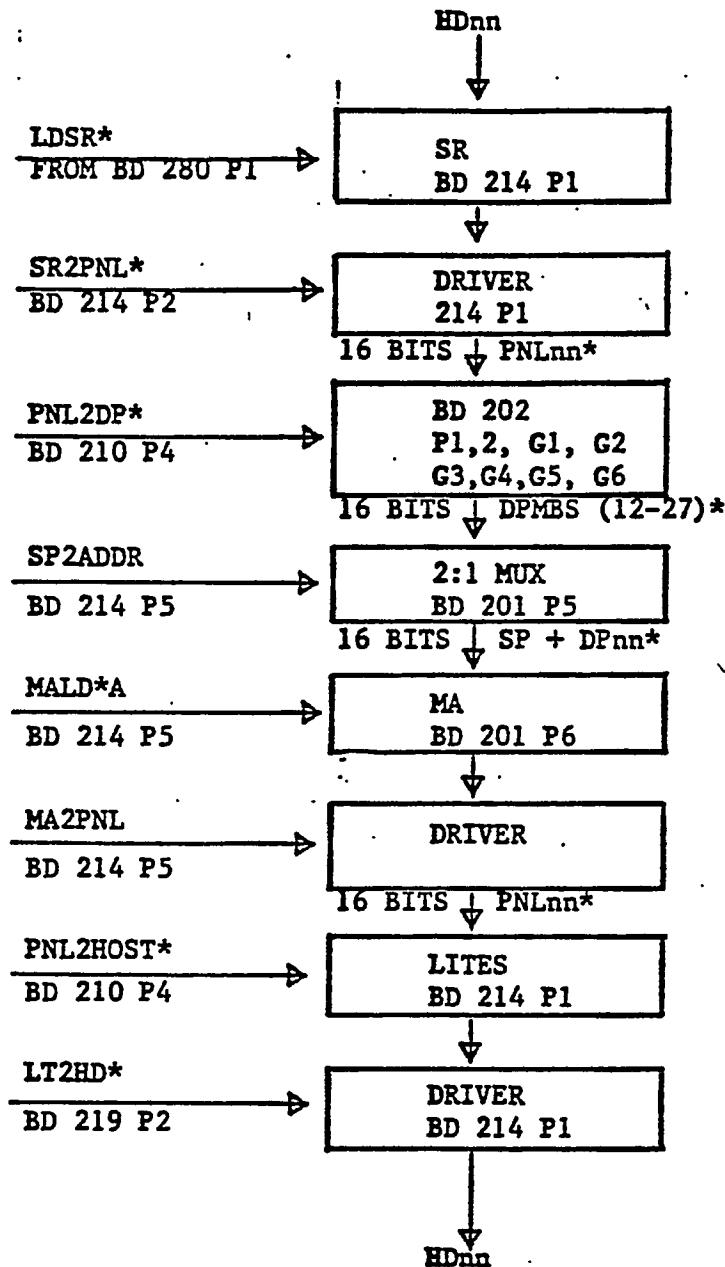
GE MEDICAL SYSTEMS INSTITUTE

CODE 7 TESTING SPA Reg.



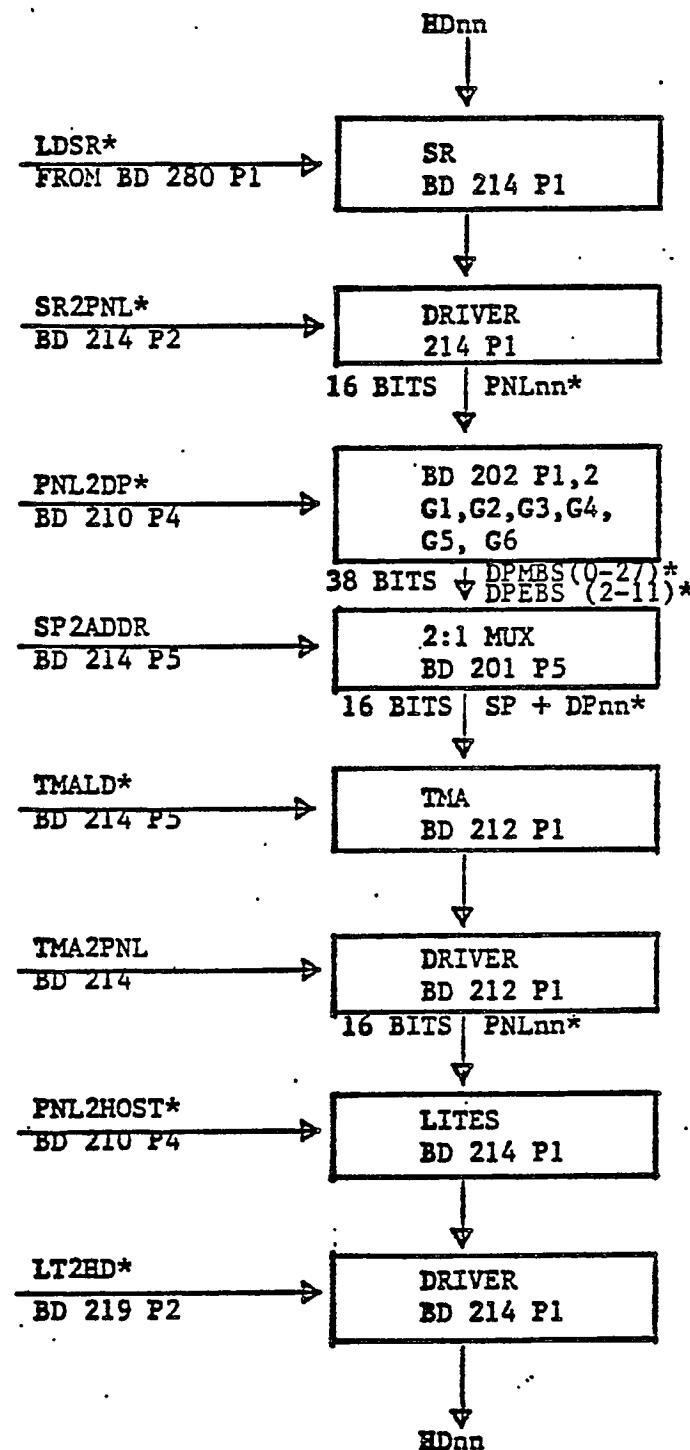
GE MEDICAL SYSTEMS INSTITUTE

CODE 10 TESTING-MA Reg.



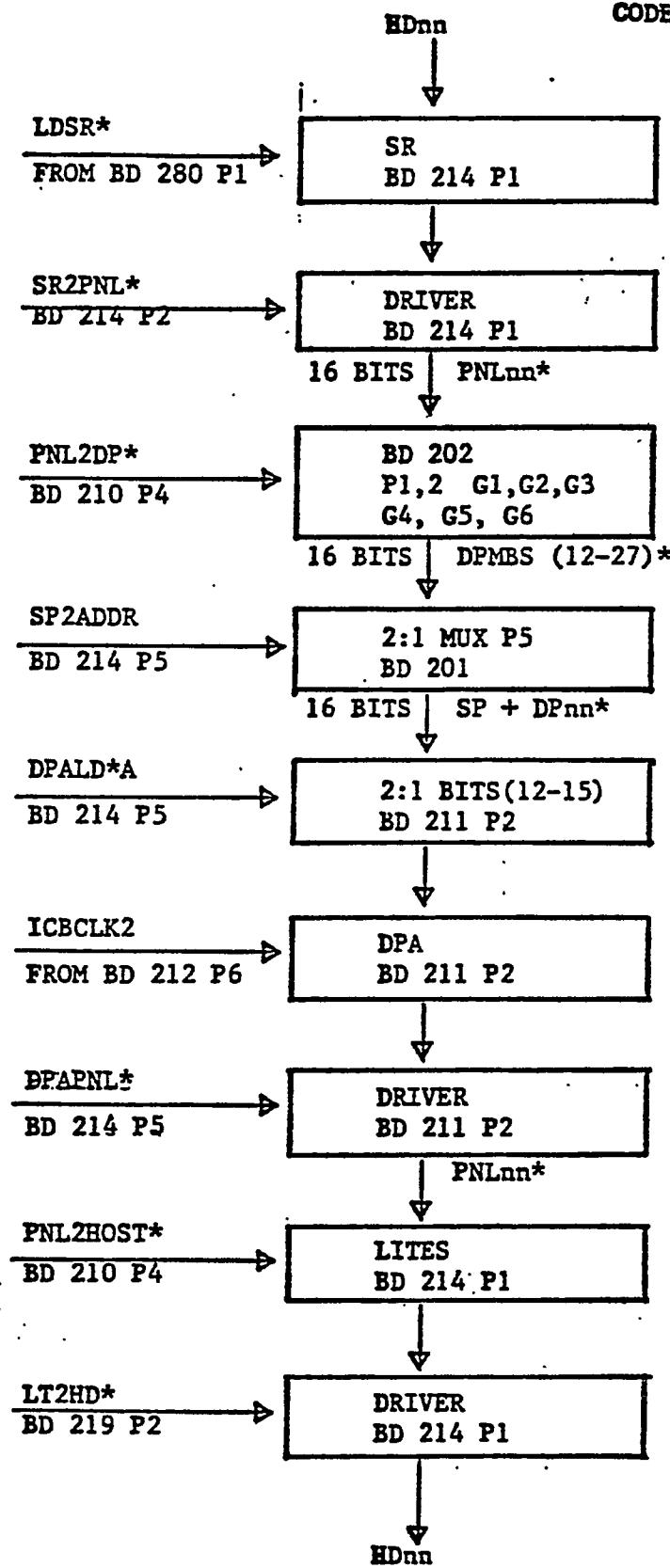
GE MEDICAL SYSTEMS INSTITUTE

CODE 11 TESTING TMA Reg.



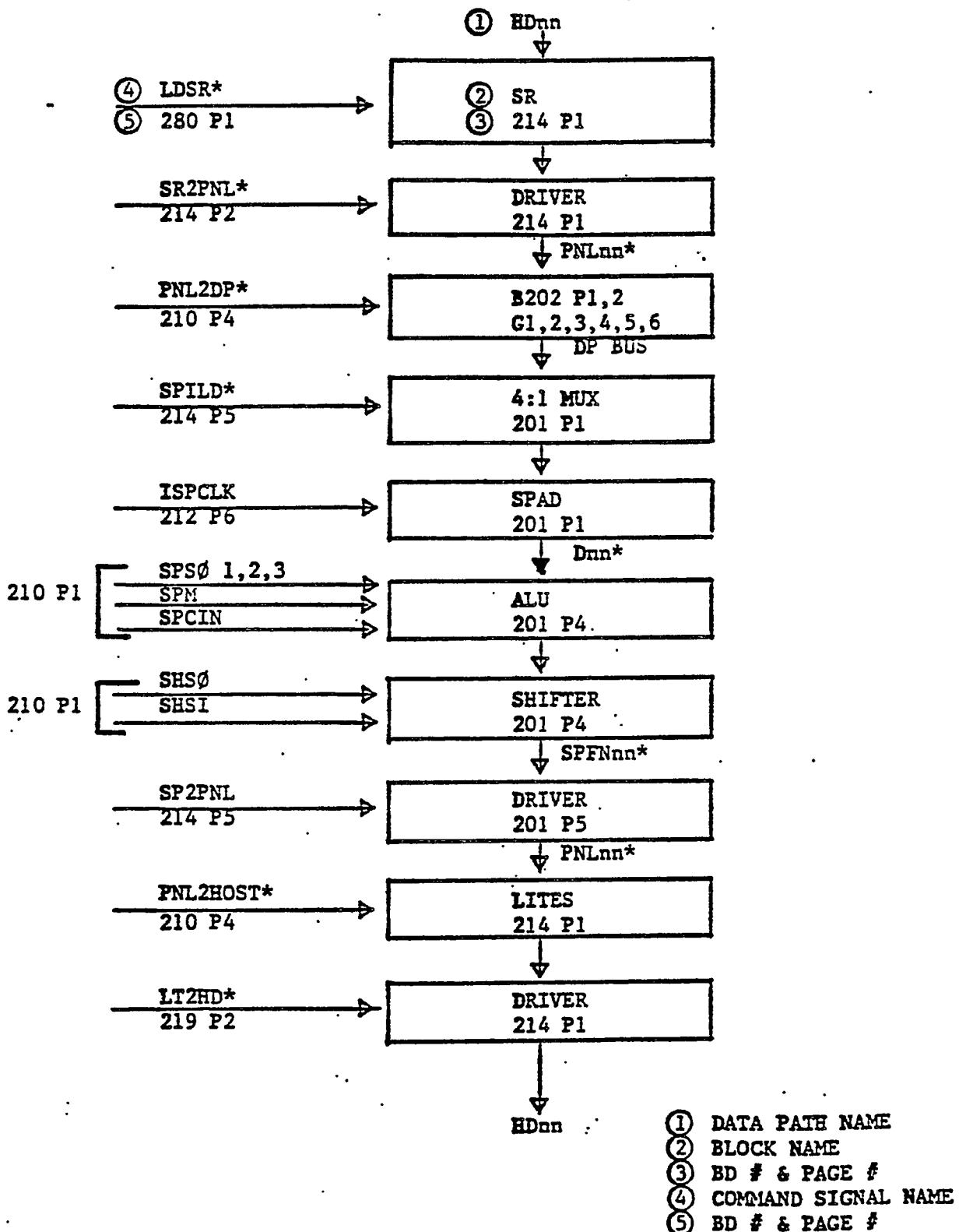
GE MEDICAL SYSTEMS INSTITUTE

CODE 12 TESTING DPA Reg.



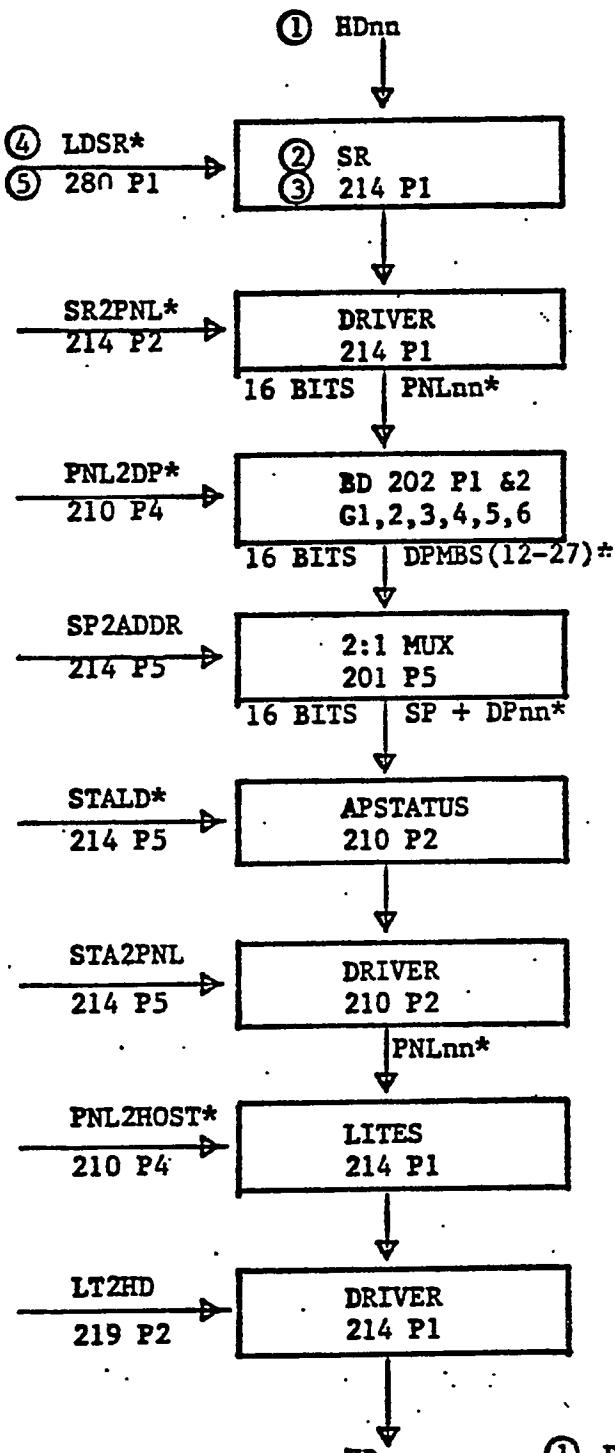
GE MEDICAL SYSTEMS INSTITUTE

TEST CODE 13



GE MEDICAL SYSTEMS INSTITUTE

TEST CODE 14



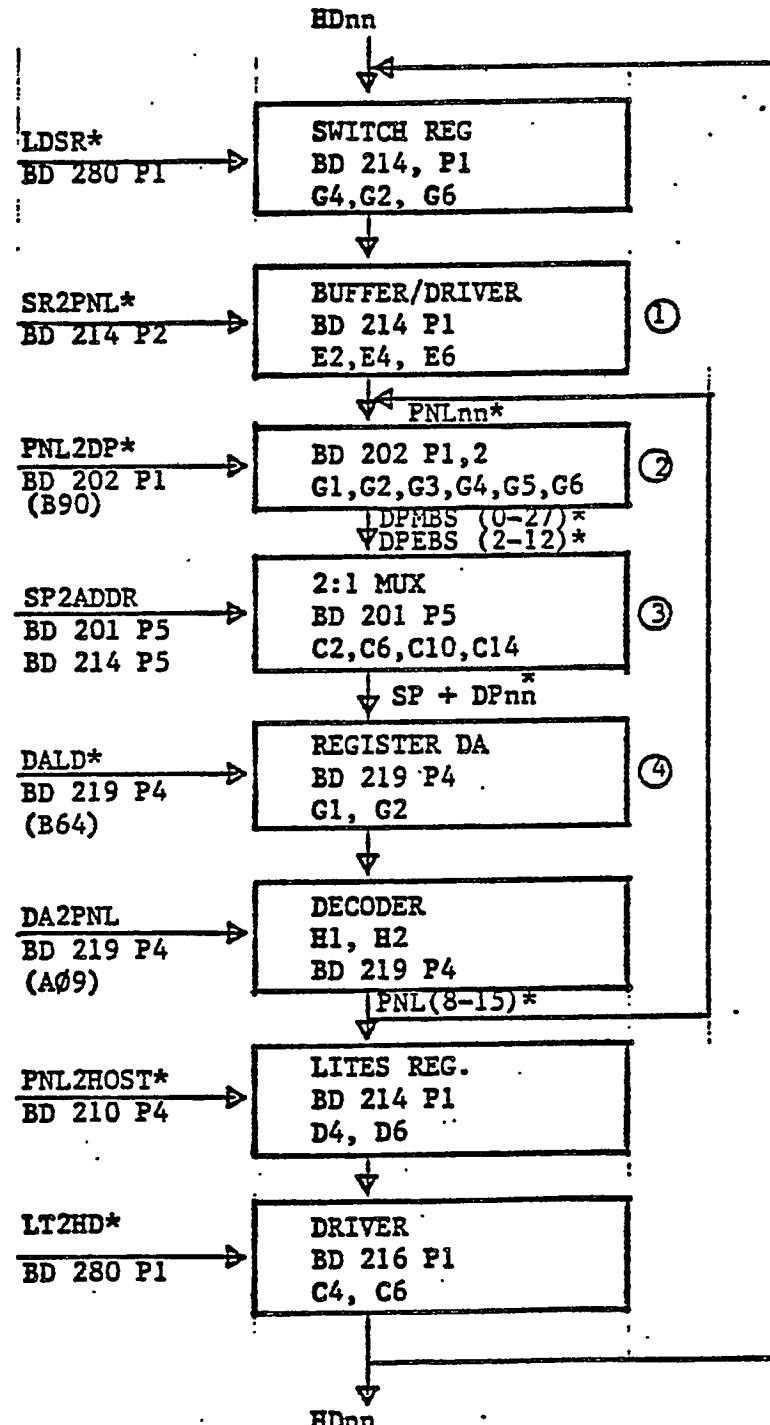
- (1) DATA PATH NAME
- (2) BLOCK NAME
- (3) BD # & PAGE #
- (4) COMMAND SIGNAL NAME
- (5) BD # & PAGE #

GE MEDICAL SYSTEMS INSTITUTE

D= HDnn $\xrightarrow{\text{LDSR}*}$ SR $\xrightarrow{\text{SR2PNL}}$ PNLnn* $\xrightarrow{\text{PNL2DP}*}$ DPBUS $\xrightarrow{\text{SP2ADDR}}$ SP+DPnn* $\xrightarrow{\text{DALD}}$ DA

E= DA $\xrightarrow{\text{DA2PNL}}$ PNLnn* $\xrightarrow{\text{PNL2HOST}*}$ LITES $\xrightarrow{\text{LT2HD}*}$ HDnn

CODE 15

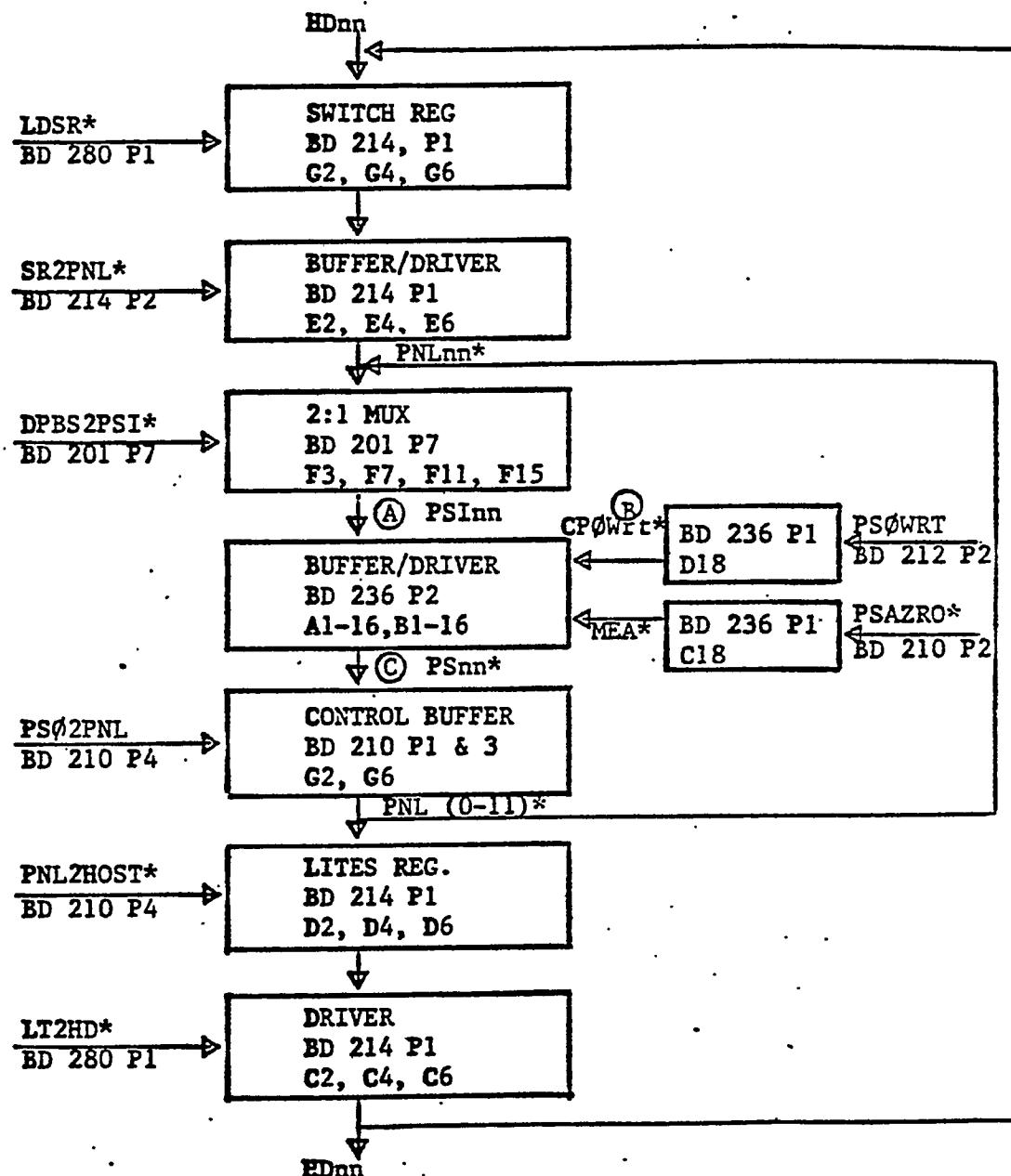


- (1) 16 BITS BUS
- (2) 38 BITS BUS
- (3) 16 BITS BUS
- (4) ONLY BITS 8-15 GO TO THE DA DECODE LOGIC

GE MEDICAL SYSTEMS INSTITUTE

D= SR SR2PNL → PNLnn → DPBS2PSI* → PSI_{nnn} → PSØWRT, PSAZRO* → PSI_{nnn}
 E= PSnn → PSØ2PNL → PNLnn → PNL2HOST* → LITES

CODE 16

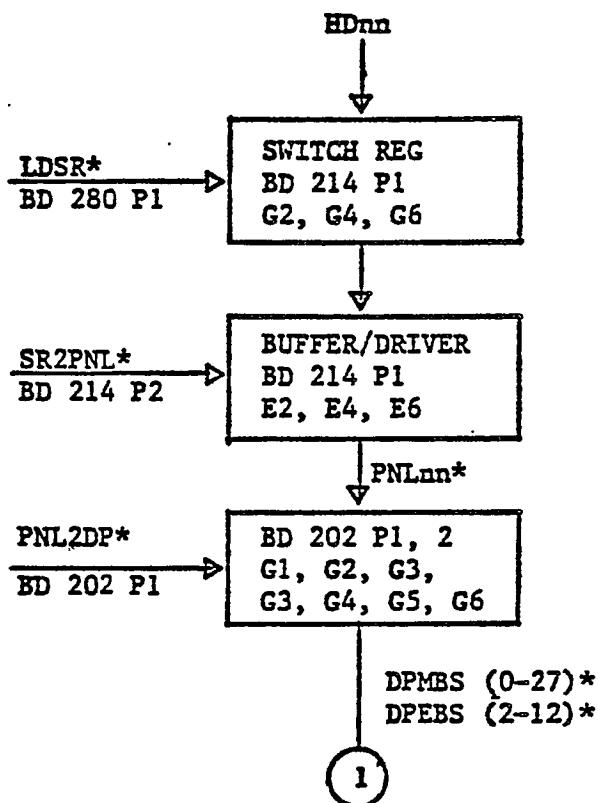


CODE	A	B	C
16	Bits PSI ₀₋₁₅	PSØWRT*/CPØWRT*	PS ₀₋₁₅
17	" PSI ₁₆₋₃₁	PSIWRT*/CP1WRT*	PS ₁₆₋₃₁
100020	" PSI ₀₋₁₅	PS2WRT*/CP2WRT*	PS ₃₂₋₄₇
20	" PSI ₁₆₋₃₁	PS3WRT*/CP3WRT*	PS ₄₈₋₆₃

GE MEDICAL SYSTEMS INSTITUTE

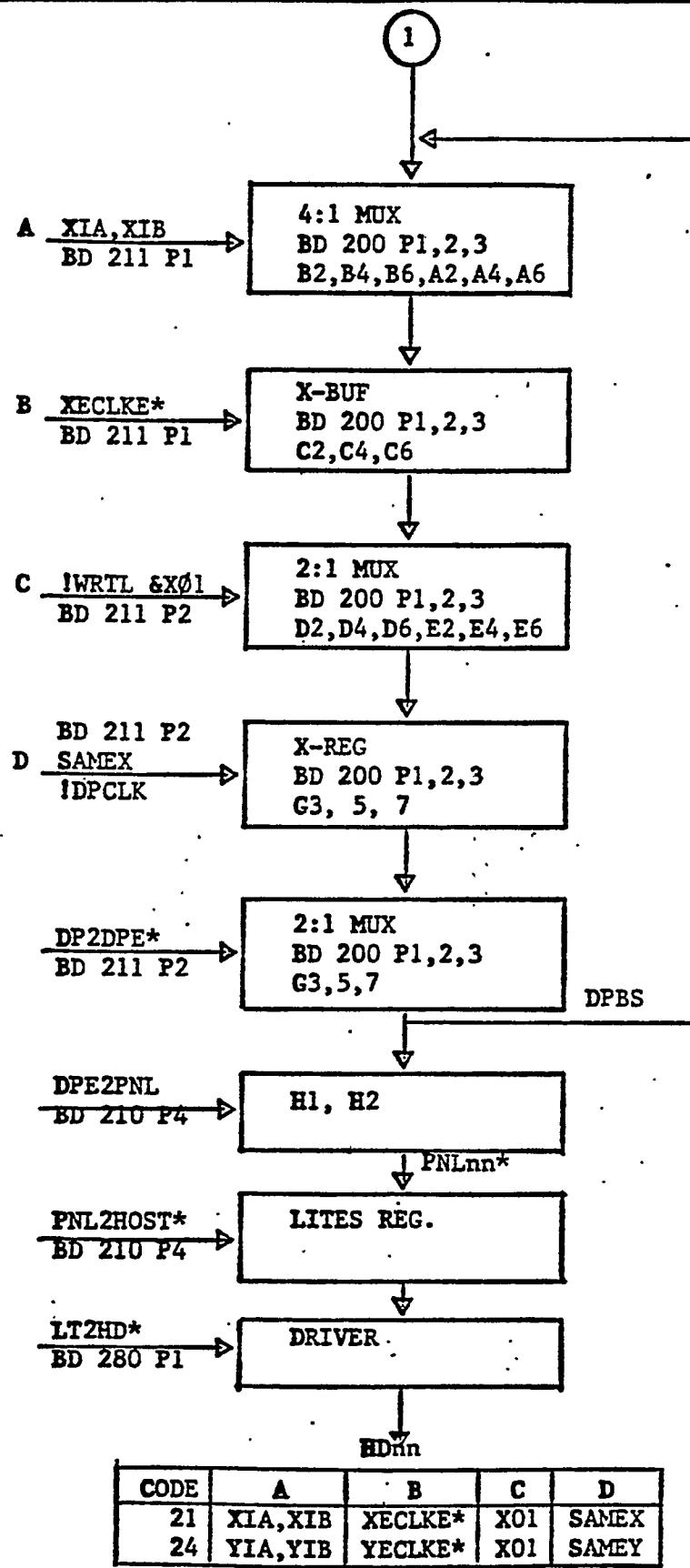
D= SR SR2PNL → PNLnn* → PNL2DP* → DPBS → XIA,XIB, XECLKE* → DPXE
 E= DPXE → DP2DPE* → DPEBS → DPE2PNL → PNLnn* → PNL2HOST* → LITES

CODE 21



CODE	A	B	C	D
21	XIA,XIB	XECLKE*	XO1	SAMEX
24	YIA,YIB	YECLKE*	YO1	SAMEY

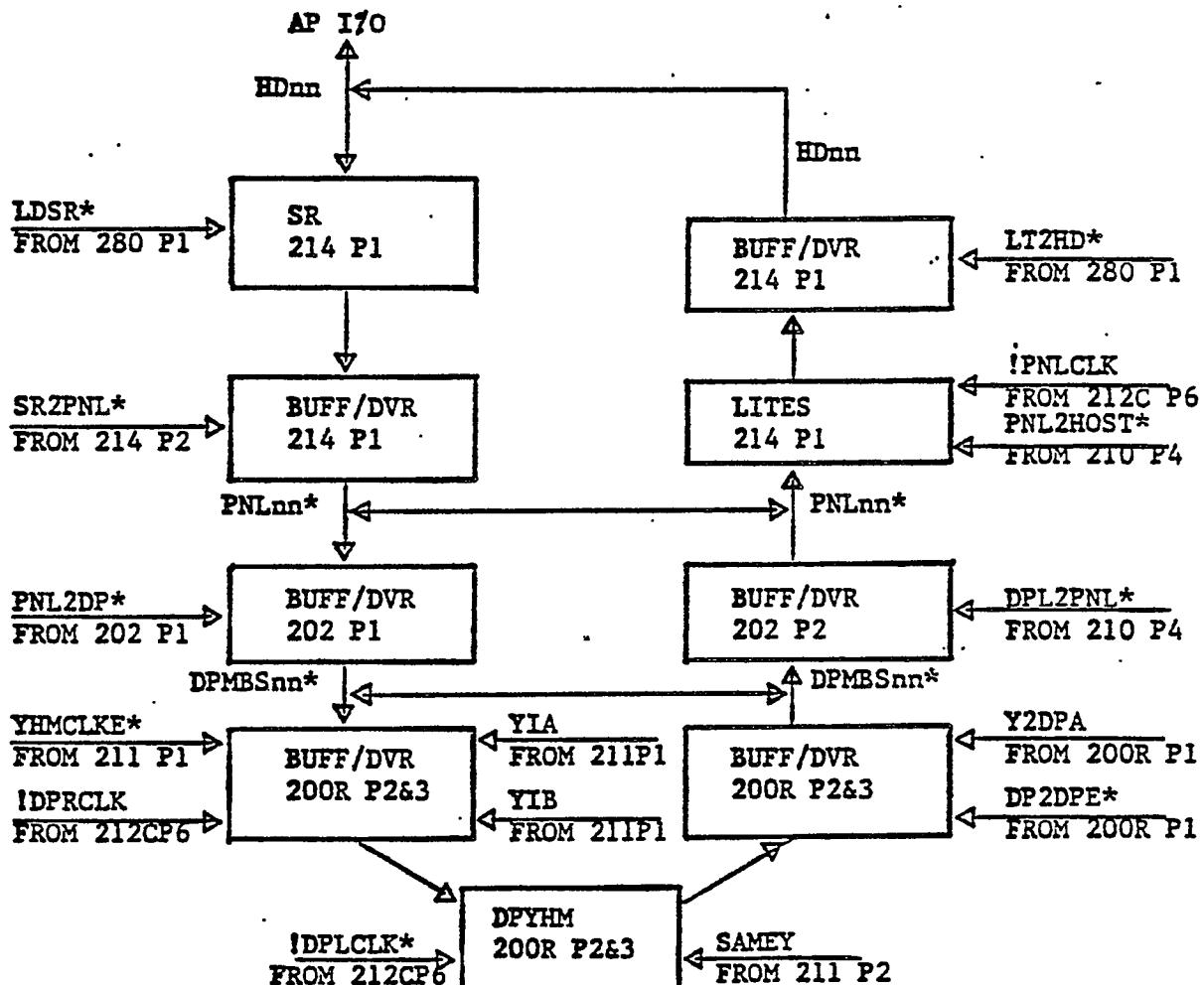
GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

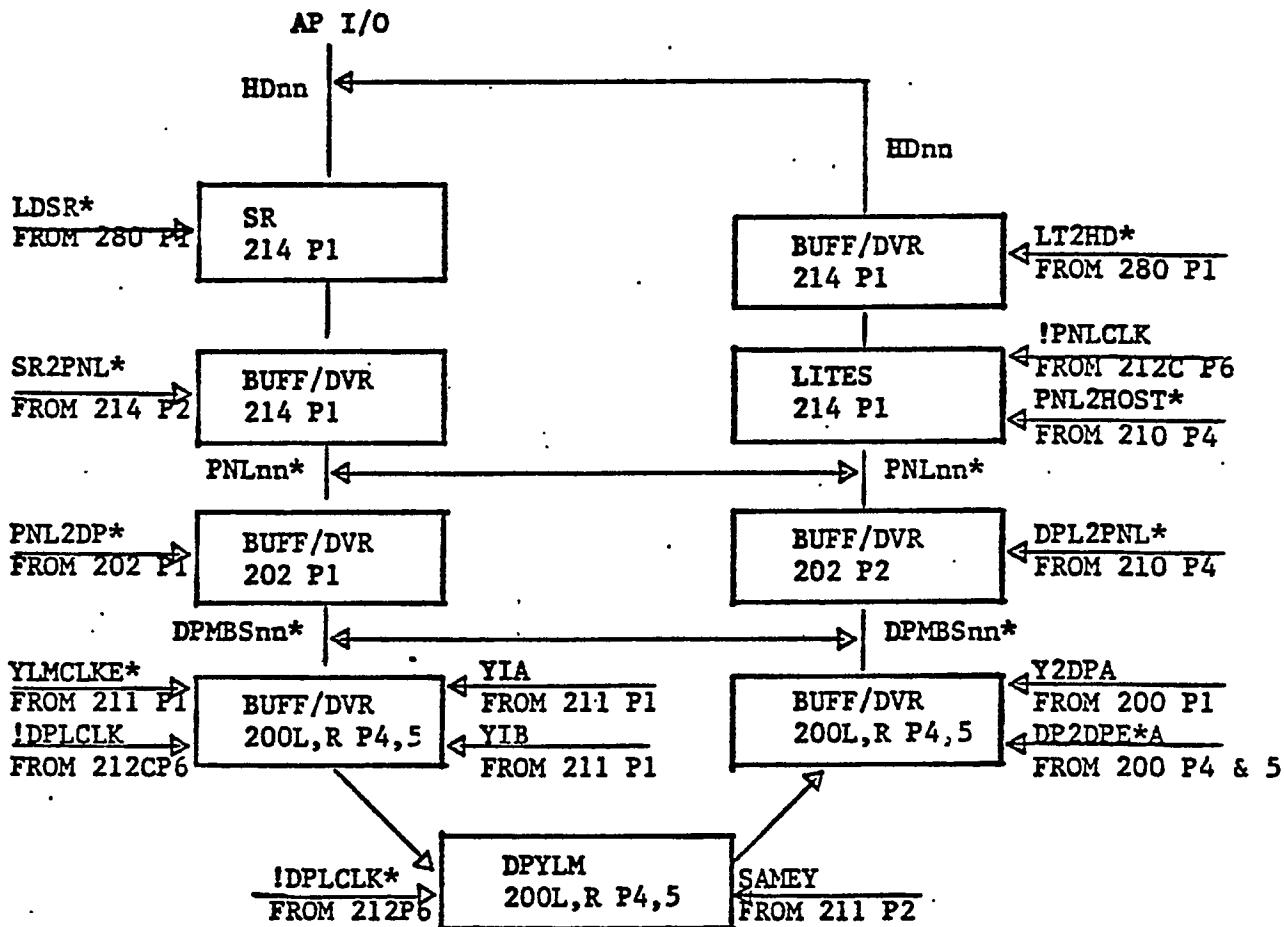
*Tests 22 and 23 use the same data paths and signals as tests 25 and 26 respectively except that X replaces Y in the control signals.

CODE 25 SR → PNL → DPBS → (YHMCLKE*) DPYHM
 DPYHM → DPMBS00*-11* → PNL → LITES



GE MEDICAL SYSTEMS INSTITUTE

CODE 26 $SP \rightarrow PNL \rightarrow DPBS \rightarrow (YLMCLKE*) DPYLM$
 $DPYLM \rightarrow DPMBS12*-27* \rightarrow PNL \rightarrow LITES$



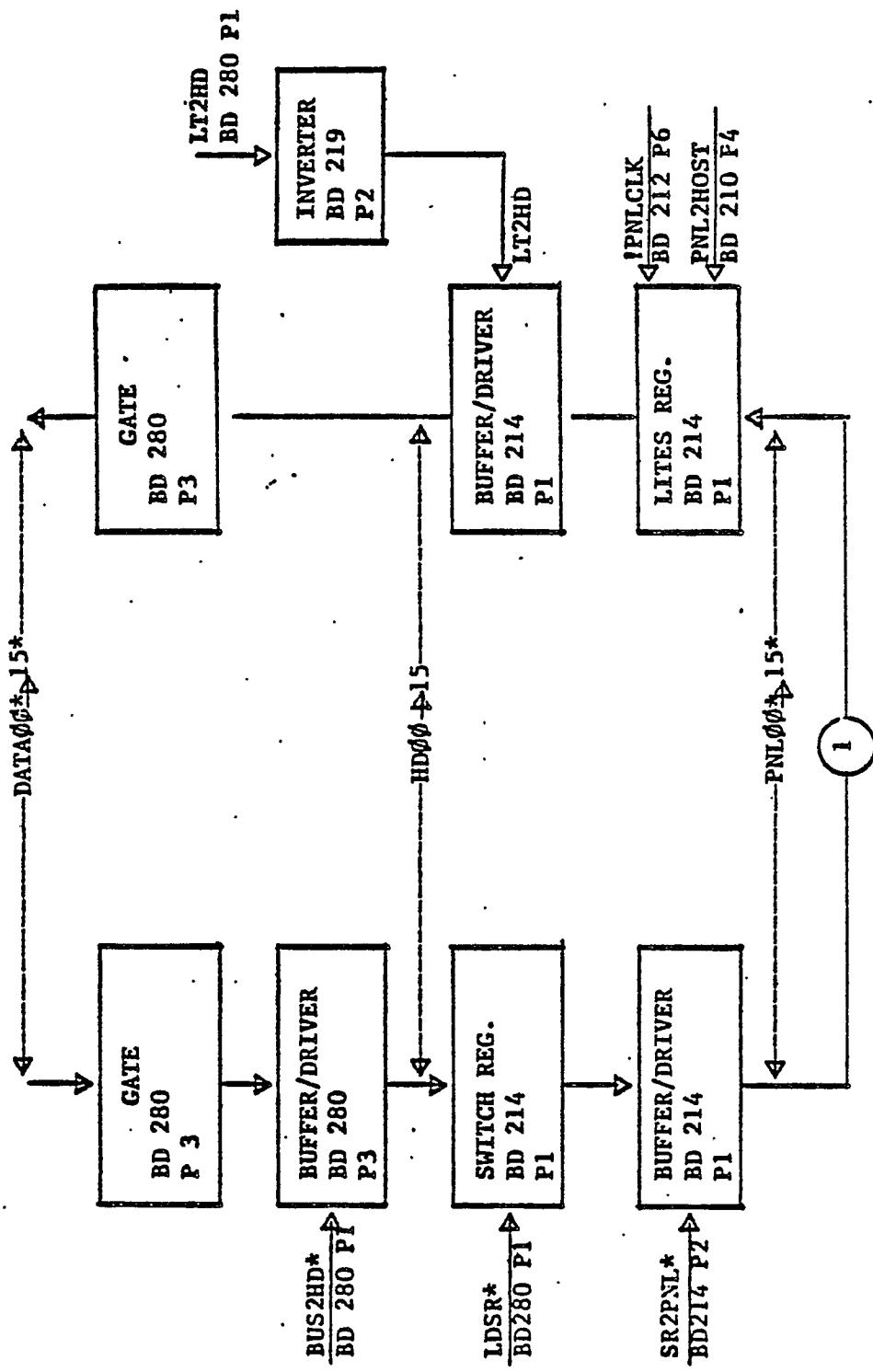
NOTE: Both DP memory boards are referenced

200R: bits 12-19
 200L: bits 20-27

GE MEDICAL SYSTEMS INSTITUTE

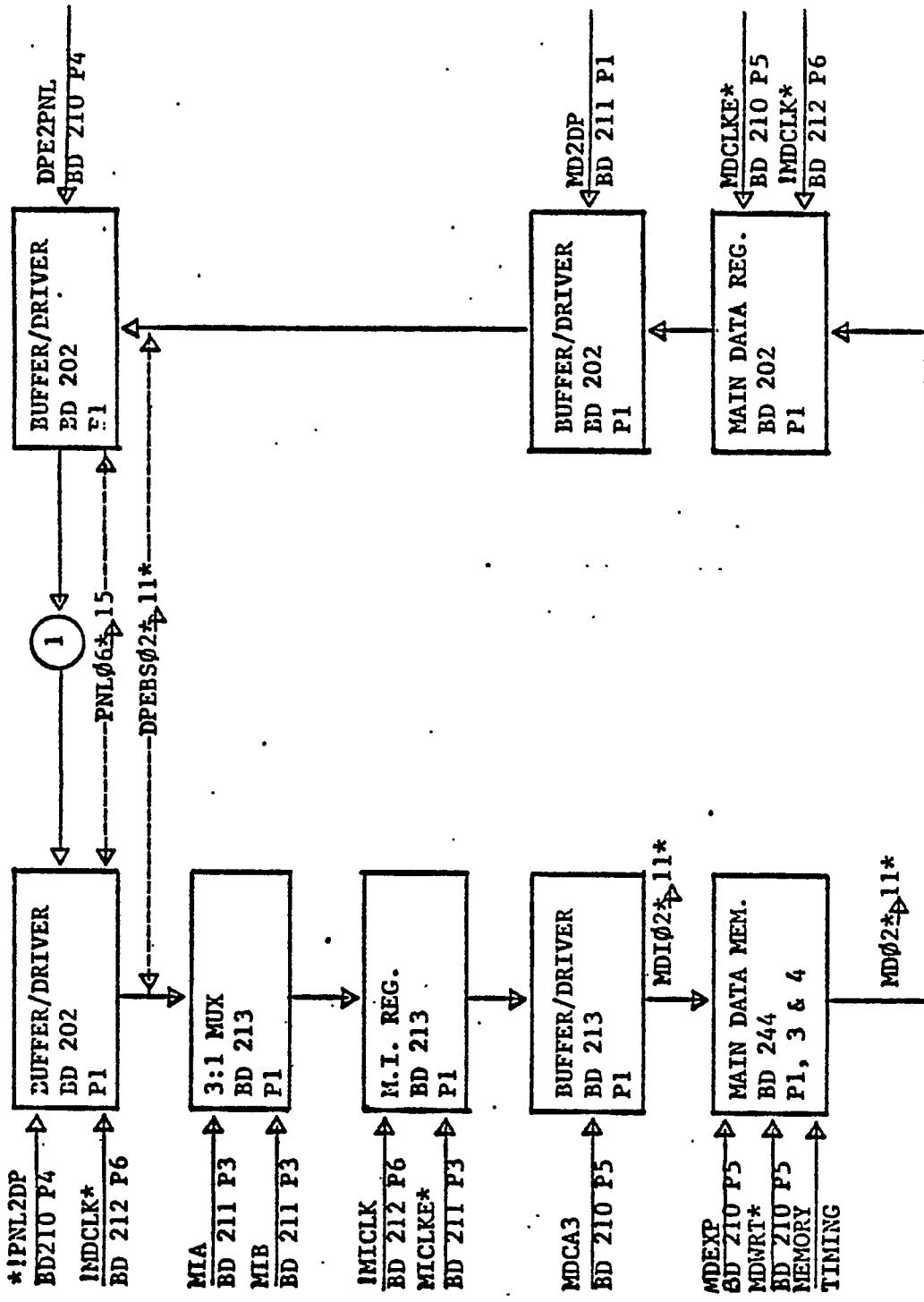
CODE 27, 30, 31

(COMMON ELEMENTS)

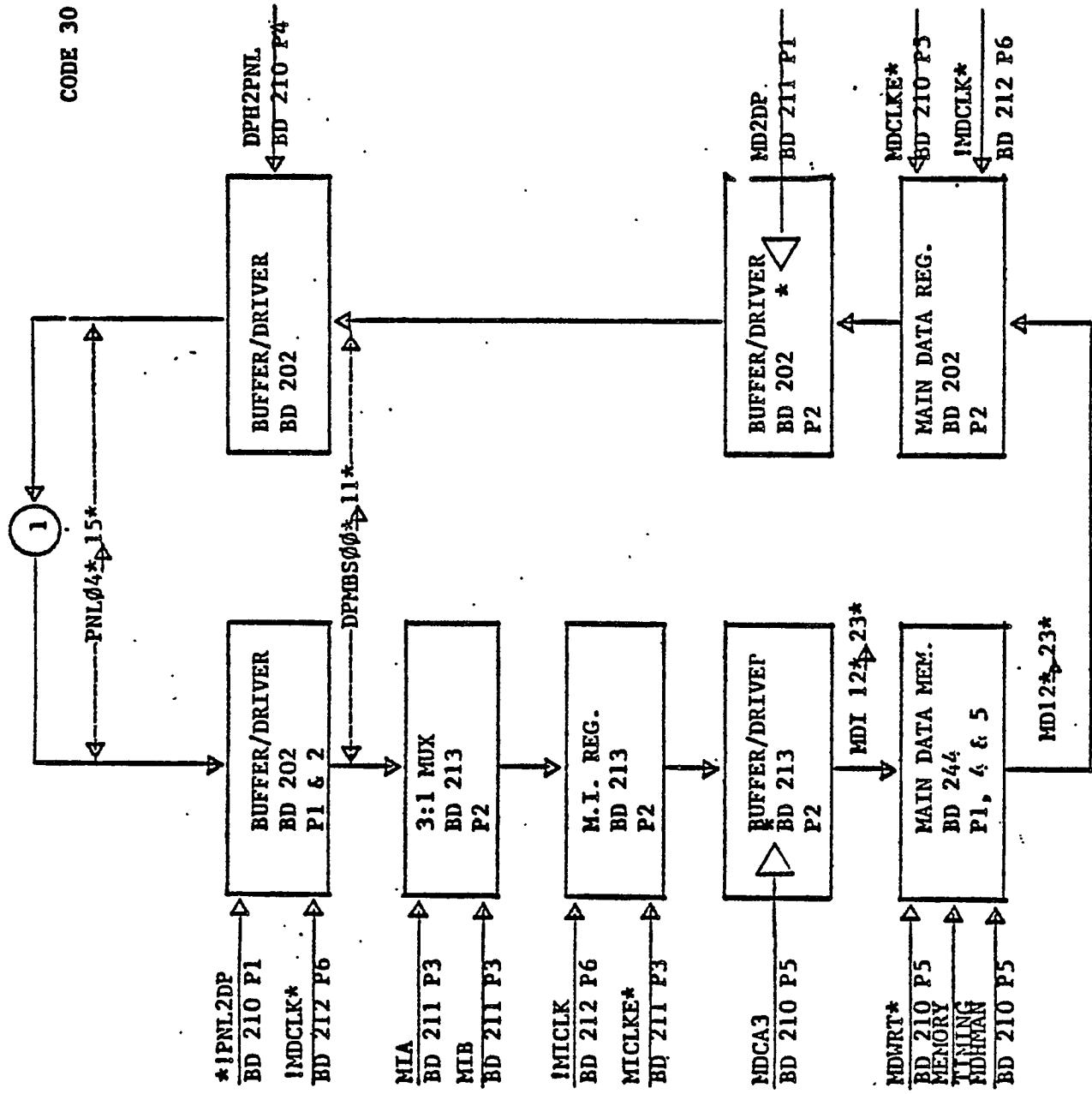


GE MEDICAL SYSTEMS INSTITUTE

CODE 27

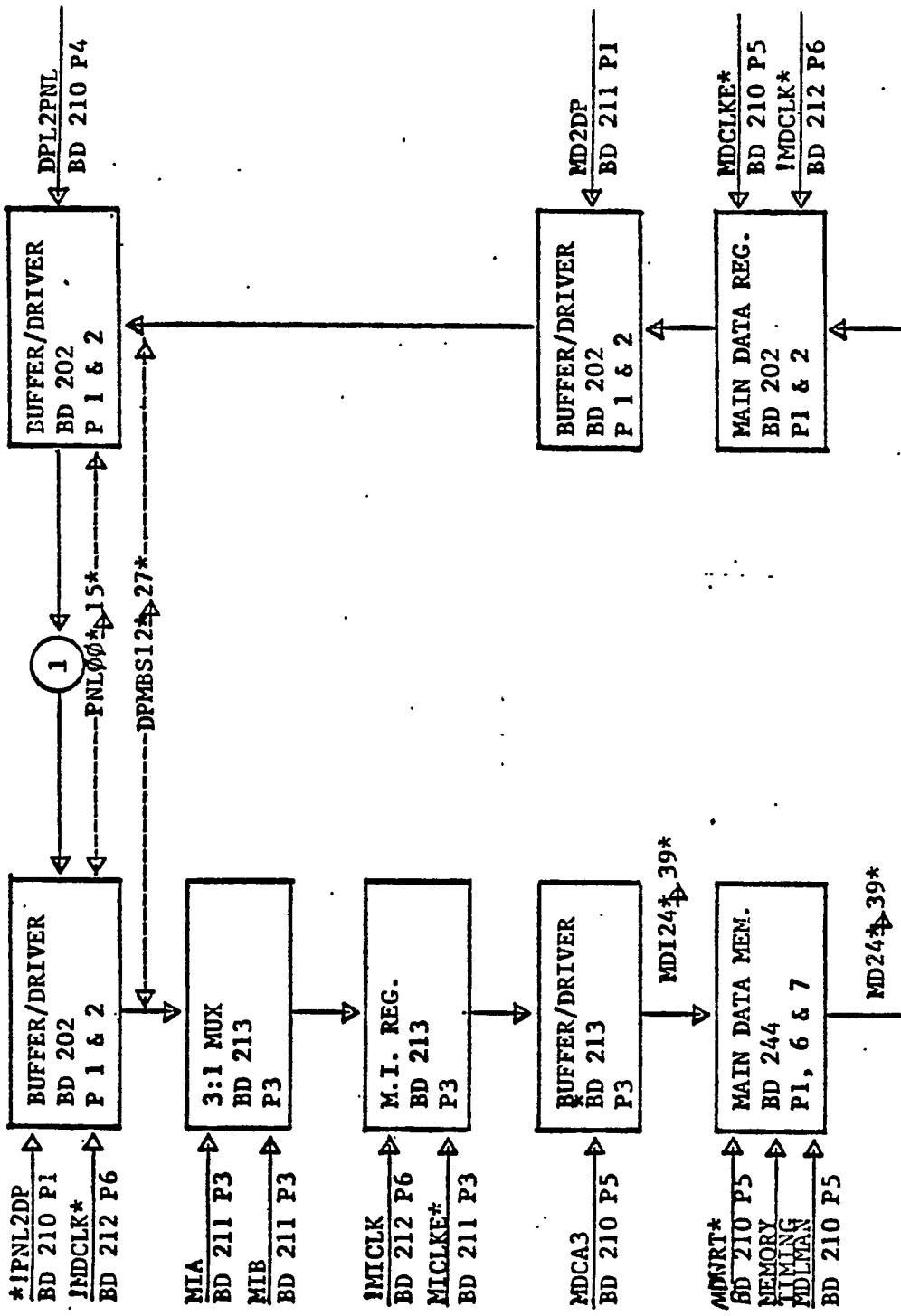


GE MEDICAL SYSTEMS INSTITUTE



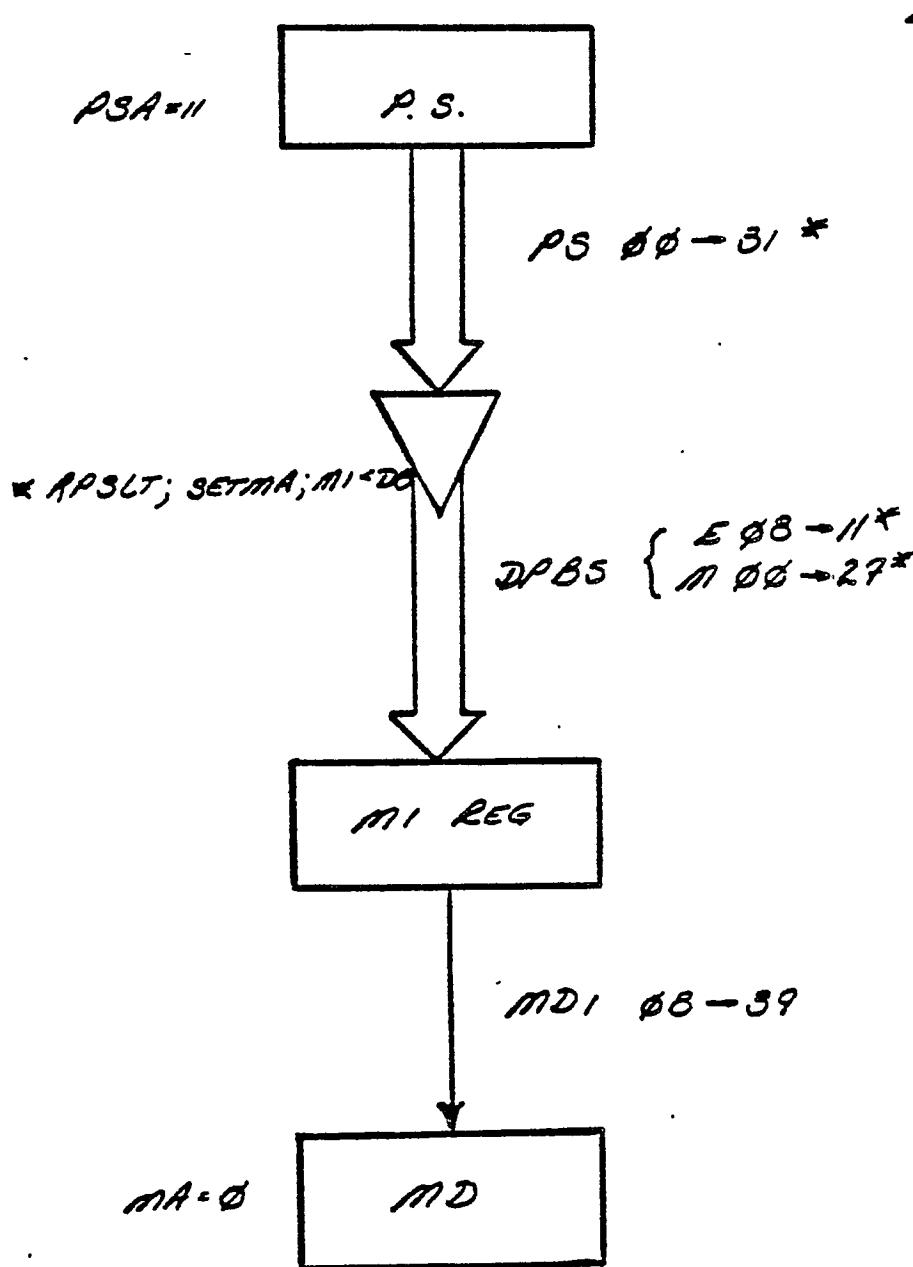
GE MEDICAL SYSTEMS INSTITUTE

CODE 31



GE MEDICAL SYSTEMS INSTITUTE

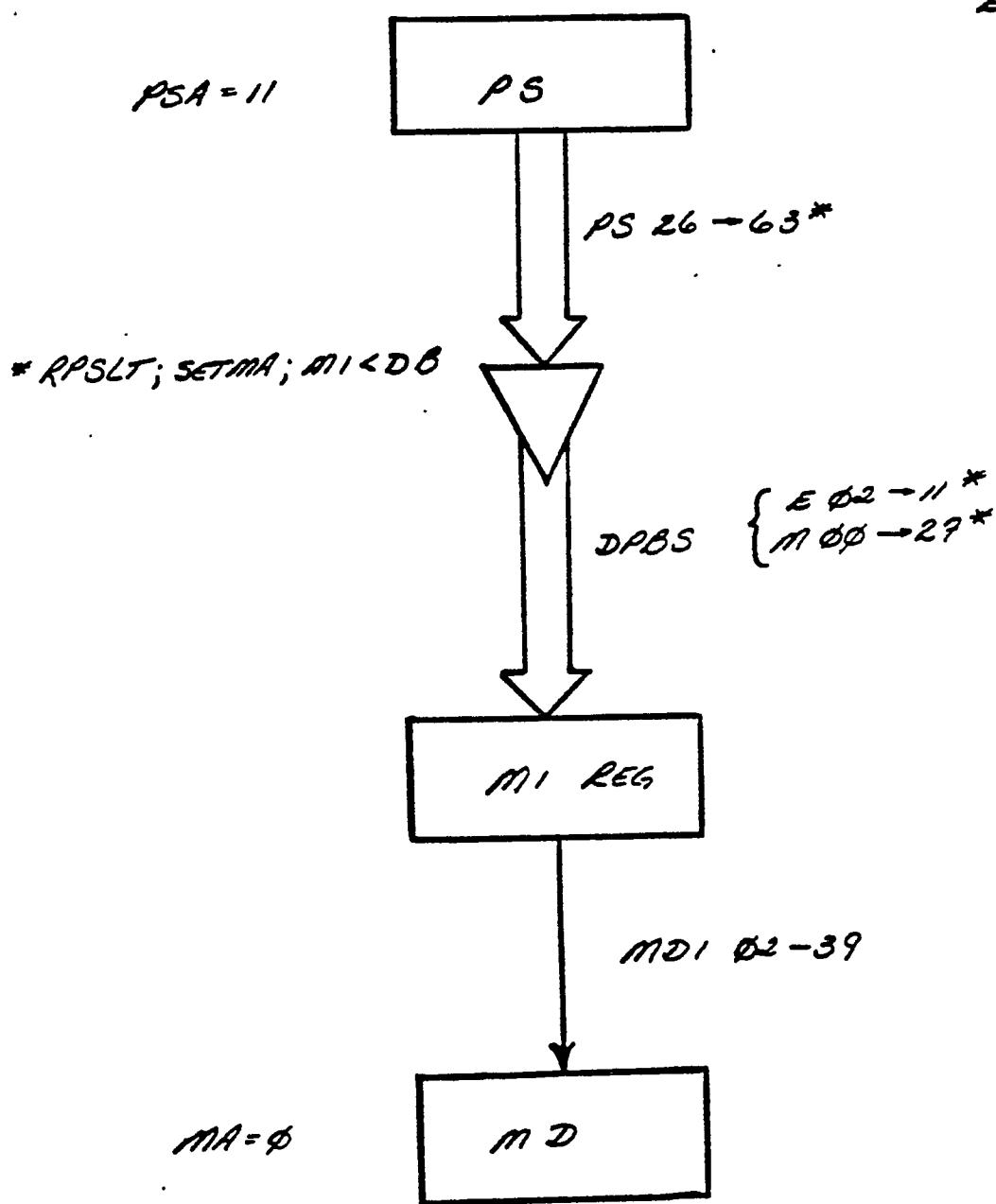
ERROR 100



THIS TESTS PS 00-13 AND 23-26 ON BOARD 210
AND PS 22, 27-31 ON BOARD 214

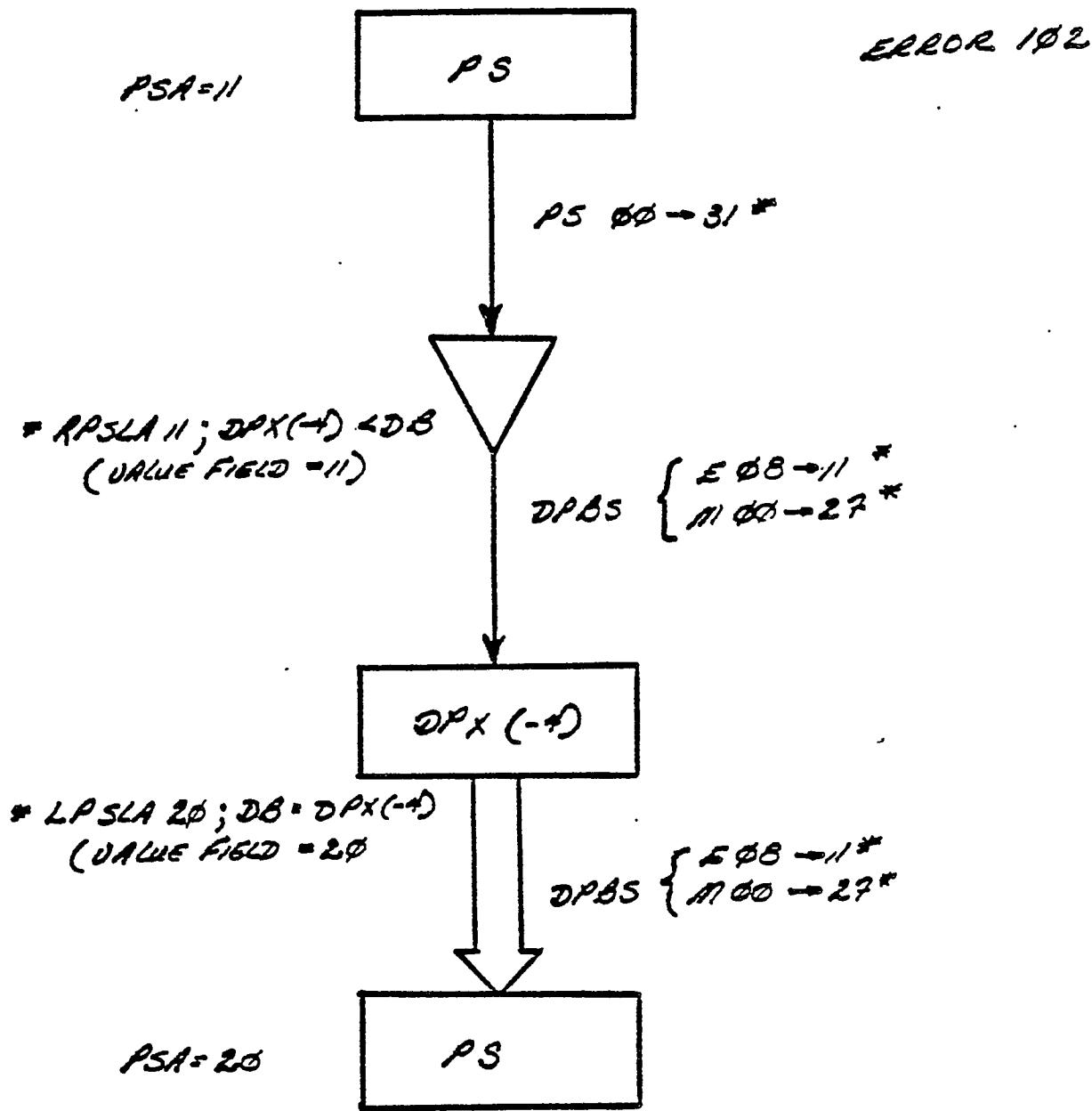
GE MEDICAL SYSTEMS INSTITUTE

ERROR 101



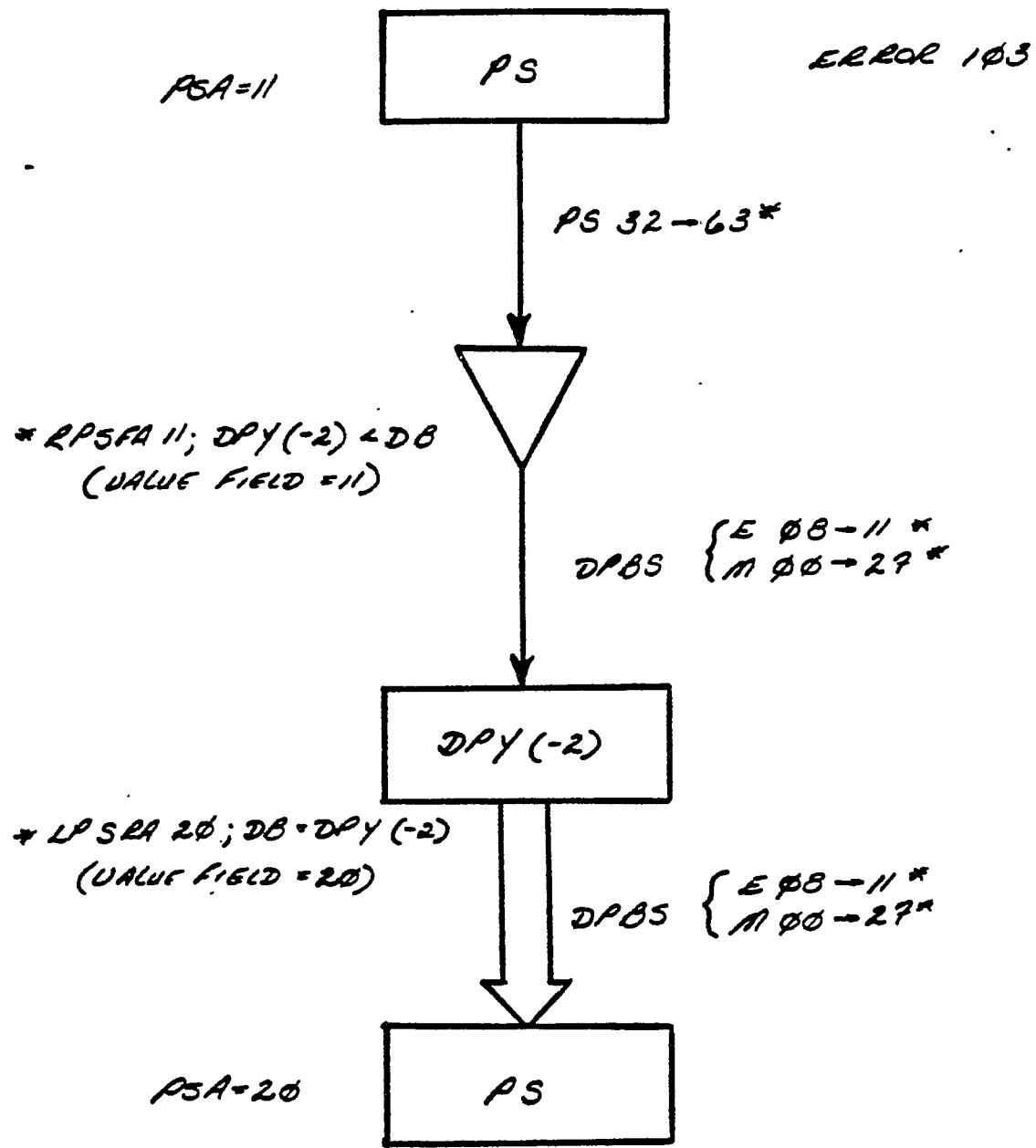
THIS TESTS PS 26 ON BOARD 210
PS 27 + 31 ON BOARD 214
AND PS 32 - 63 ON BOARD 211

GE MEDICAL SYSTEMS INSTITUTE



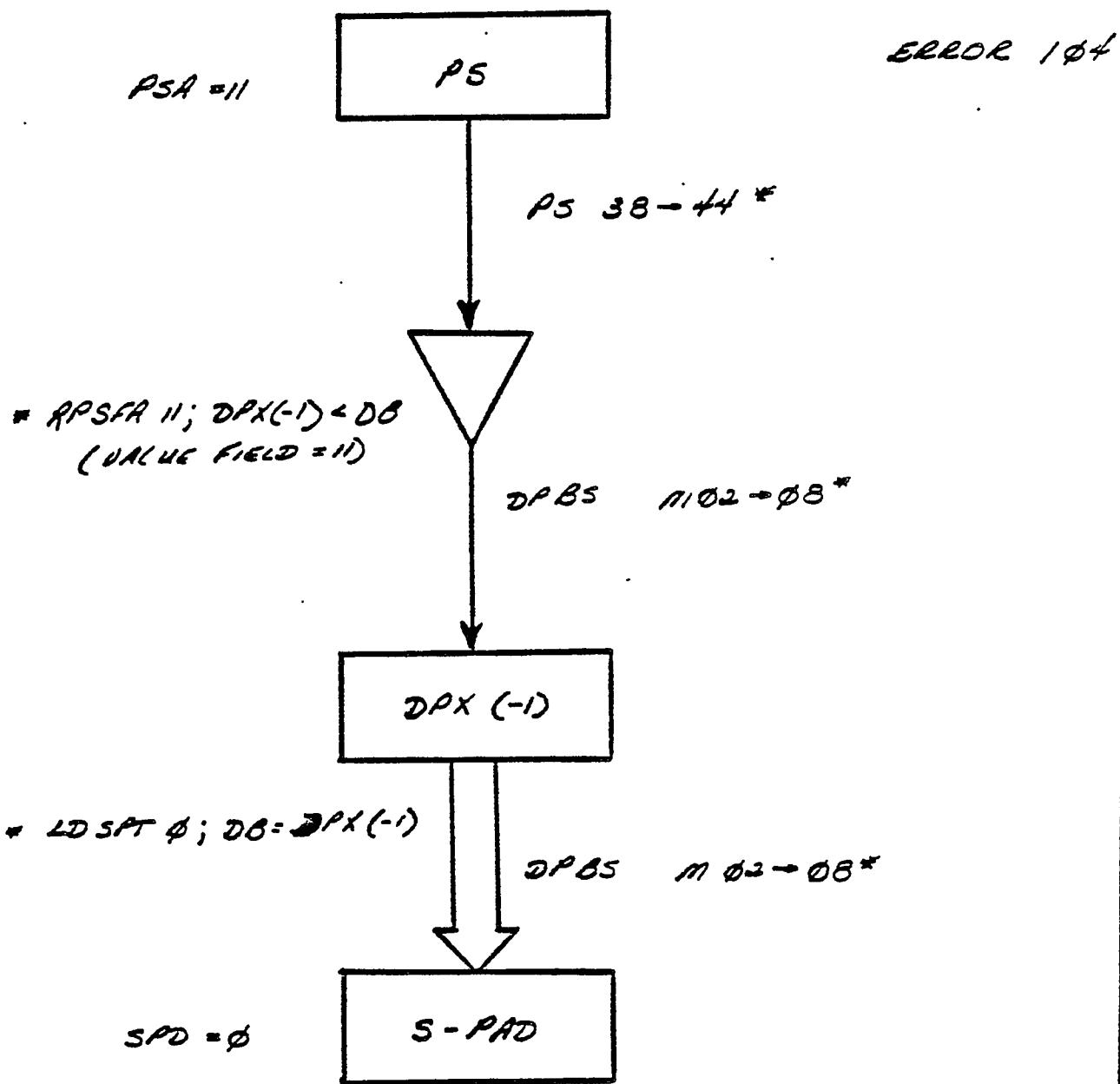
THIS TESTS PS 00→13 AND 23→26 ON BOARD 210
 PS 14→22 AND 27→31 ON BOARD 214
 AND PS 32→63 ON BOARD 211

GE MEDICAL SYSTEMS INSTITUTE

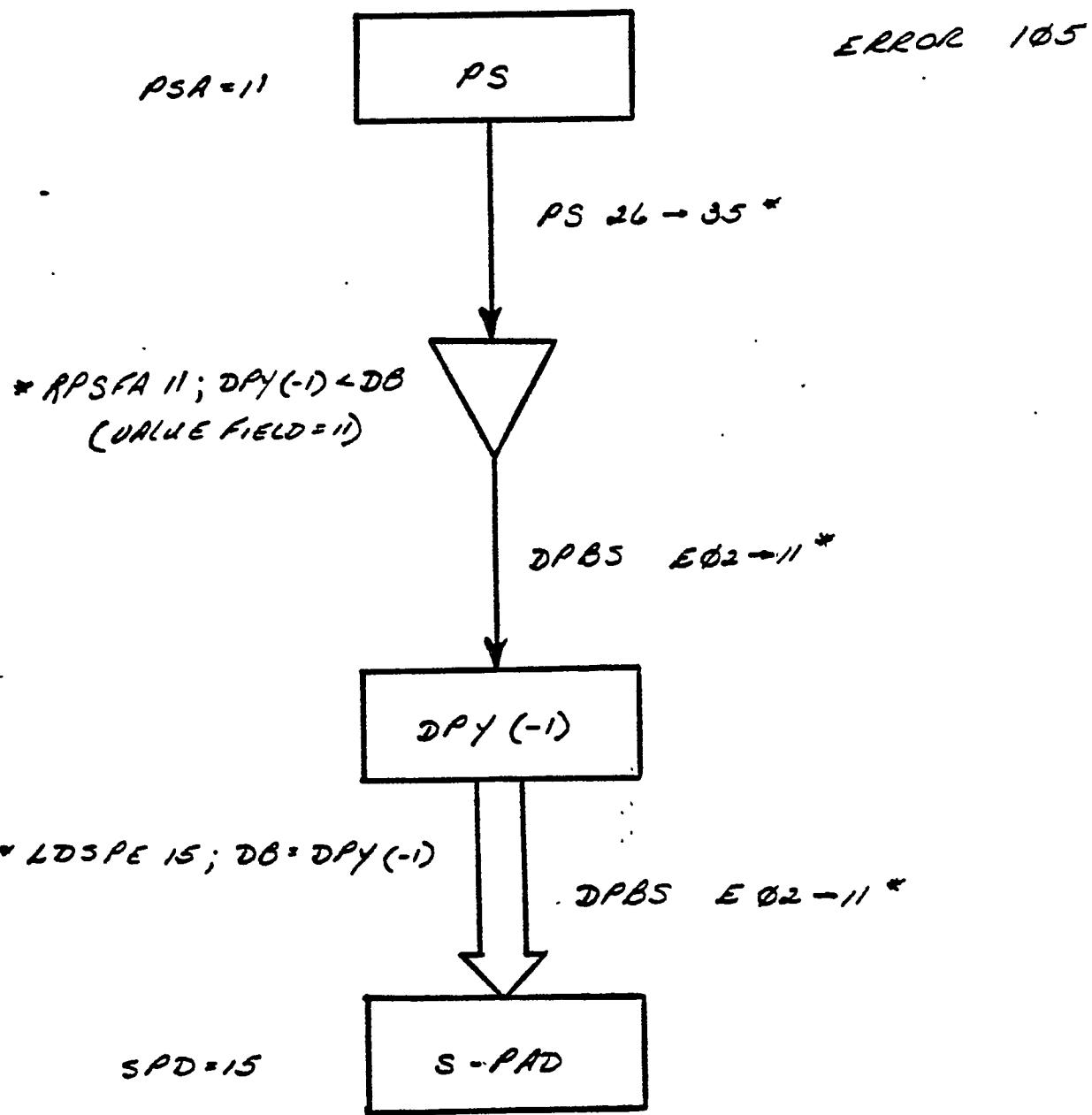


THIS TESTS PS 26 ON BOARD 210
AND PS 27-31 AND PS 32-63 ON BOARD 211

GE MEDICAL SYSTEMS INSTITUTE

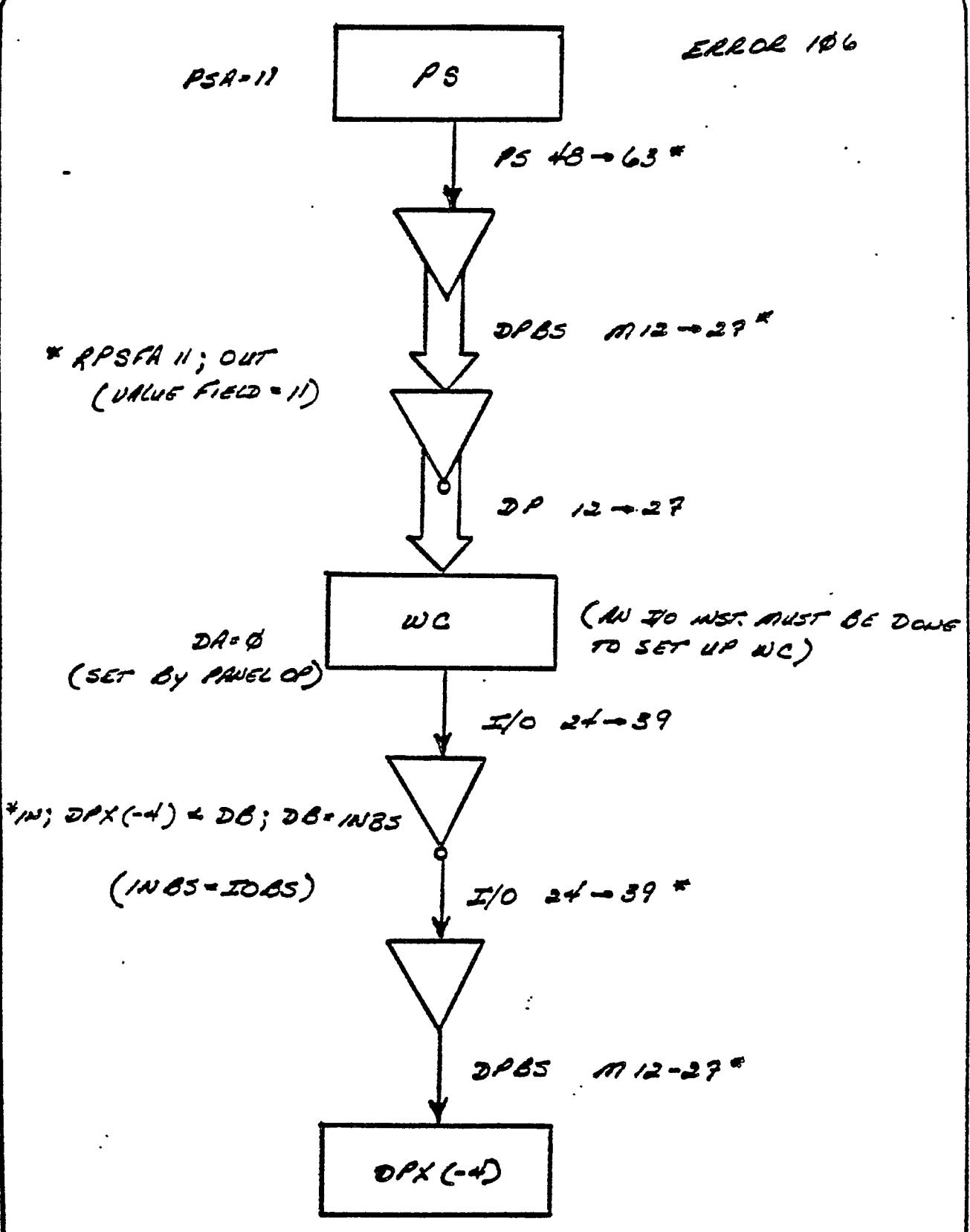


GE MEDICAL SYSTEMS INSTITUTE

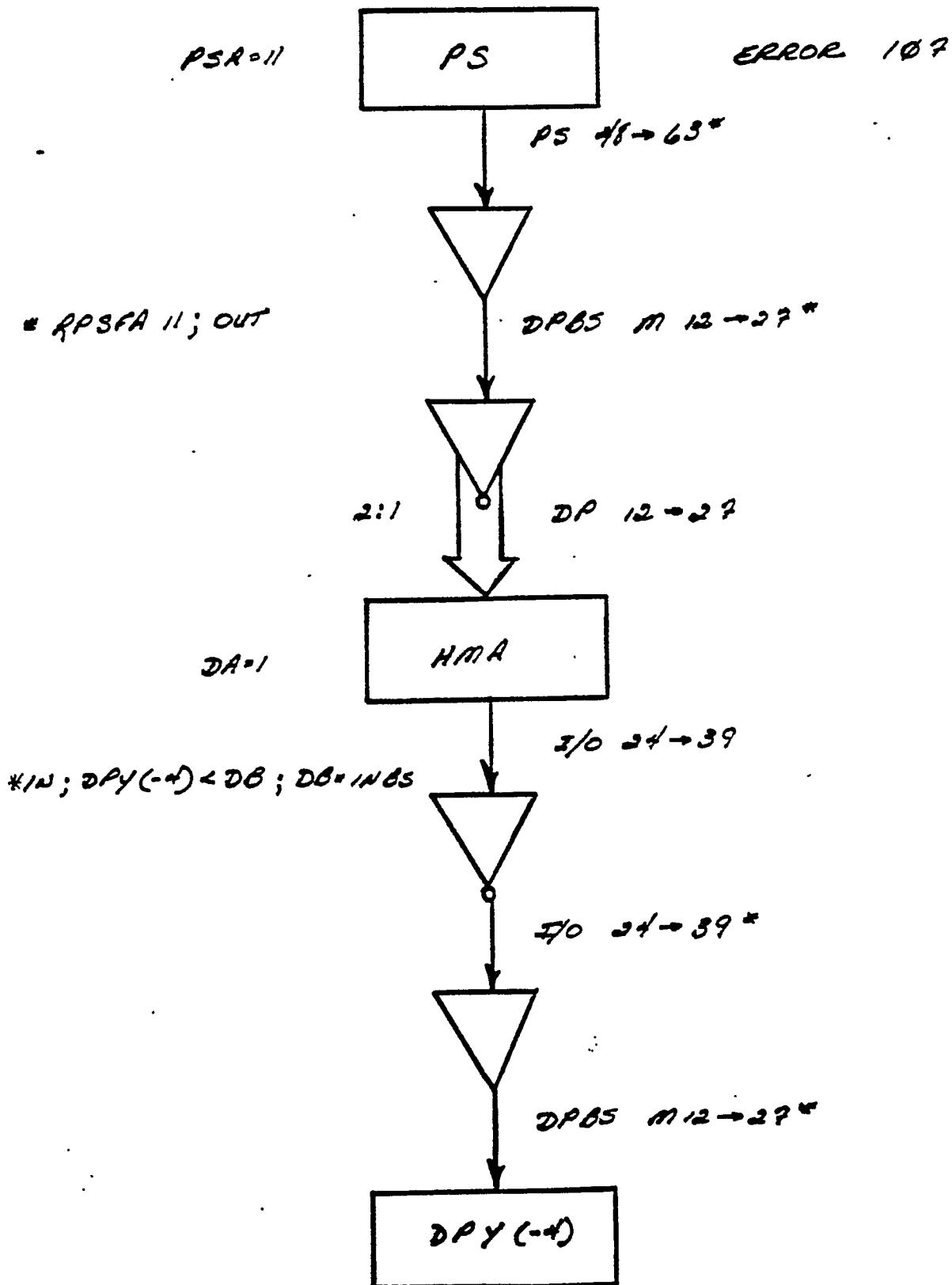


THIS TESTS PS 36 ON BOARD 210
PS 29-31 ON BOARD 214
AND PS 32-35 ON BOARD 211

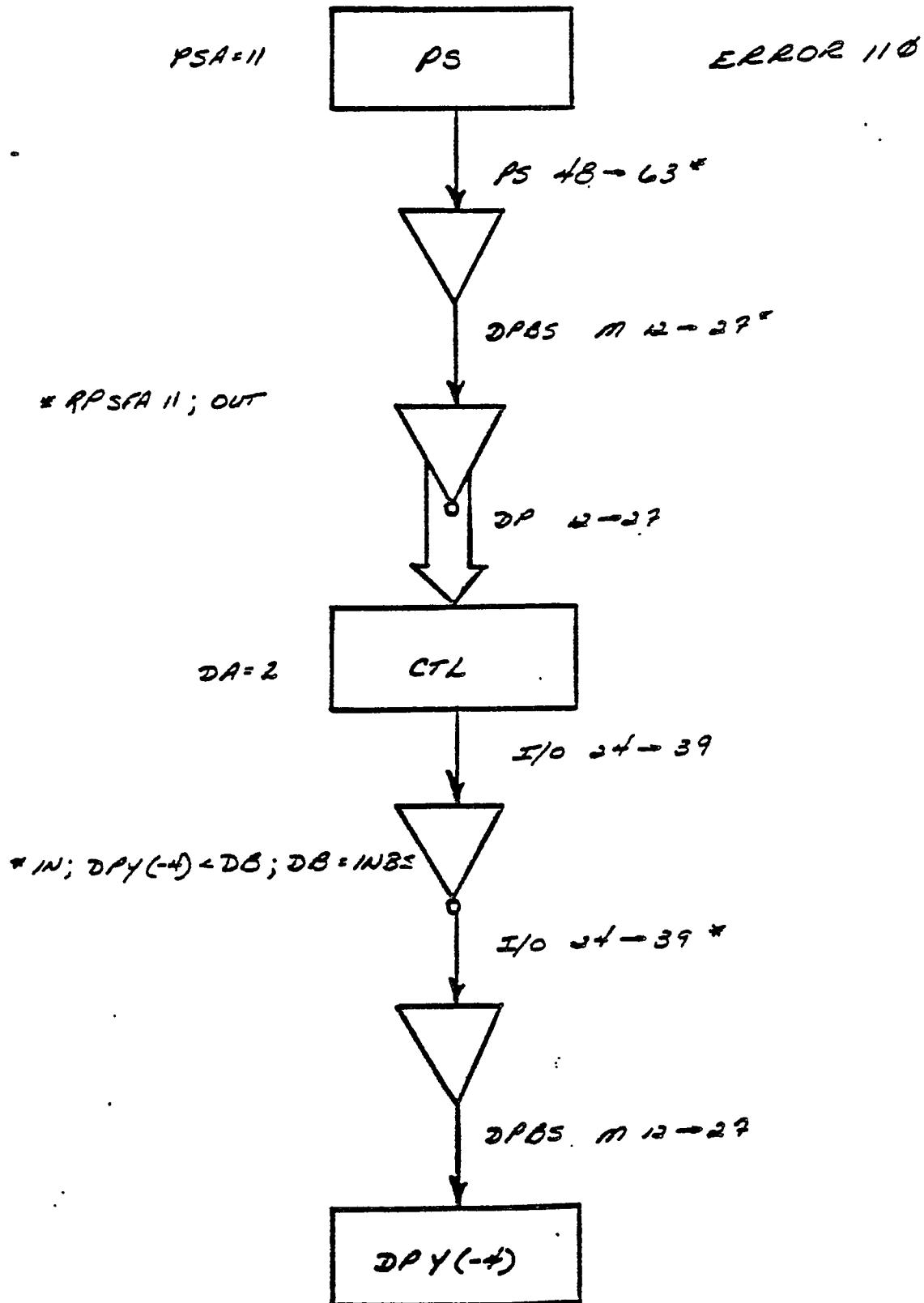
GE MEDICAL SYSTEMS INSTITUTE



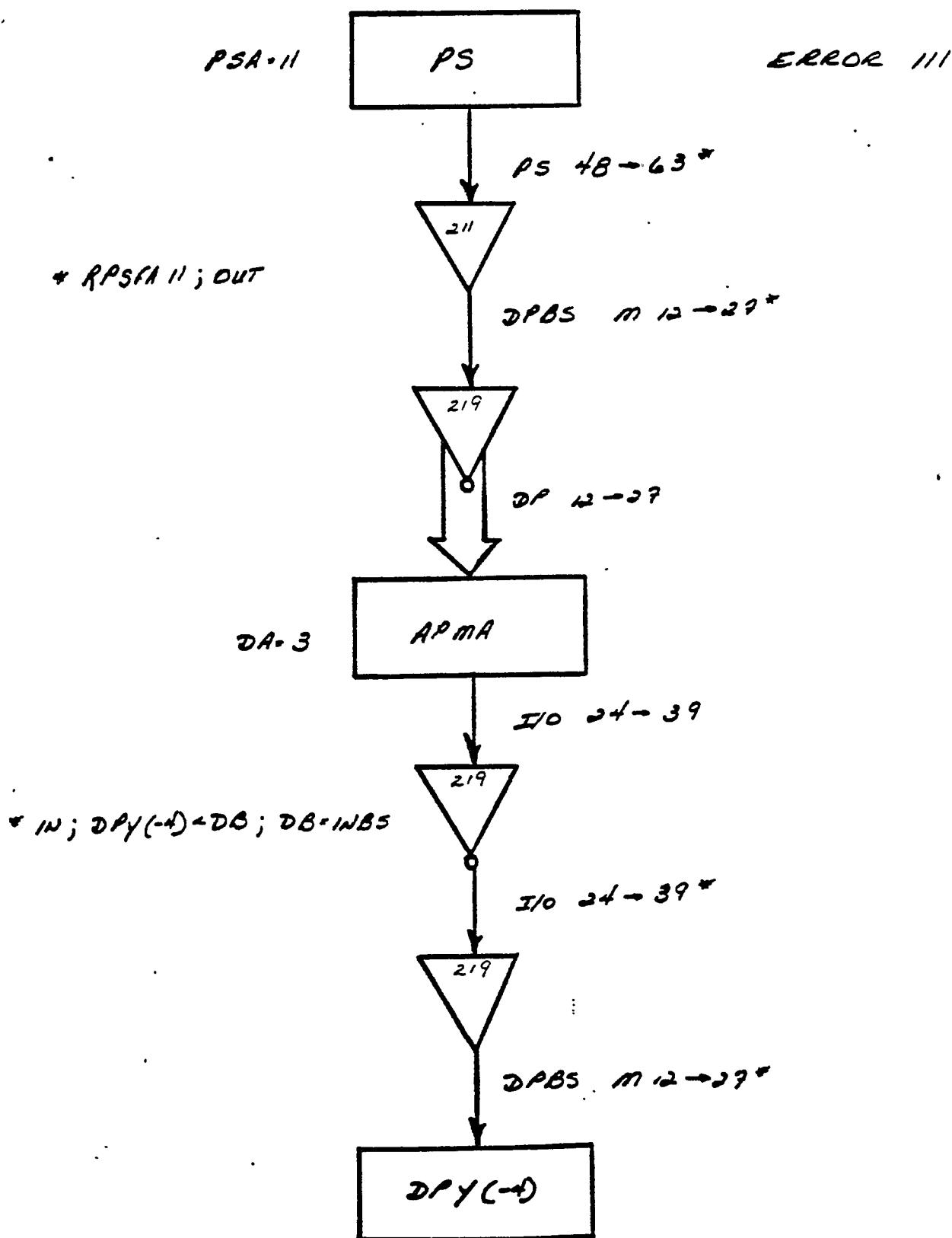
GE MEDICAL SYSTEMS INSTITUTE



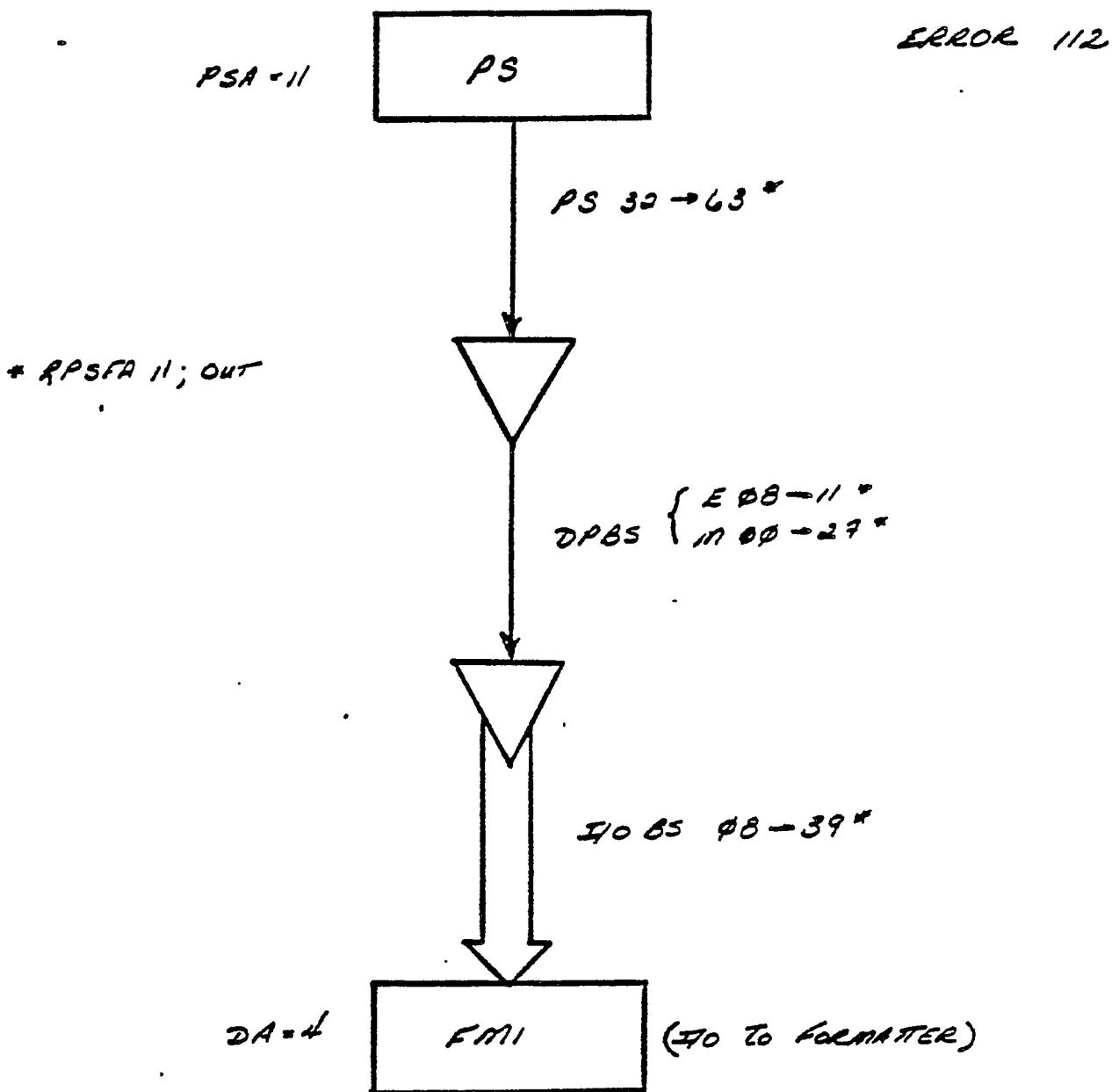
GE MEDICAL SYSTEMS INSTITUTE



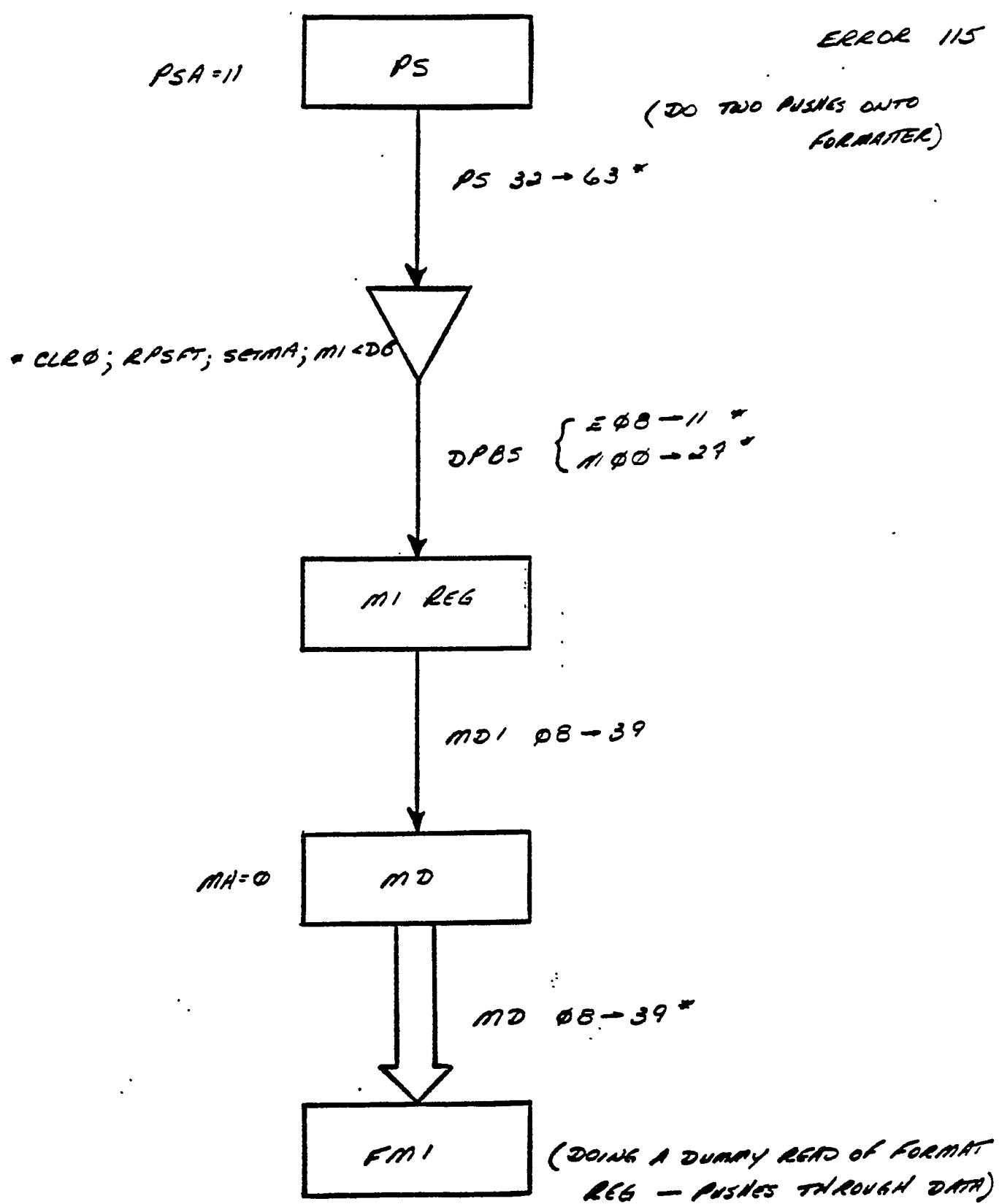
GE MEDICAL SYSTEMS INSTITUTE



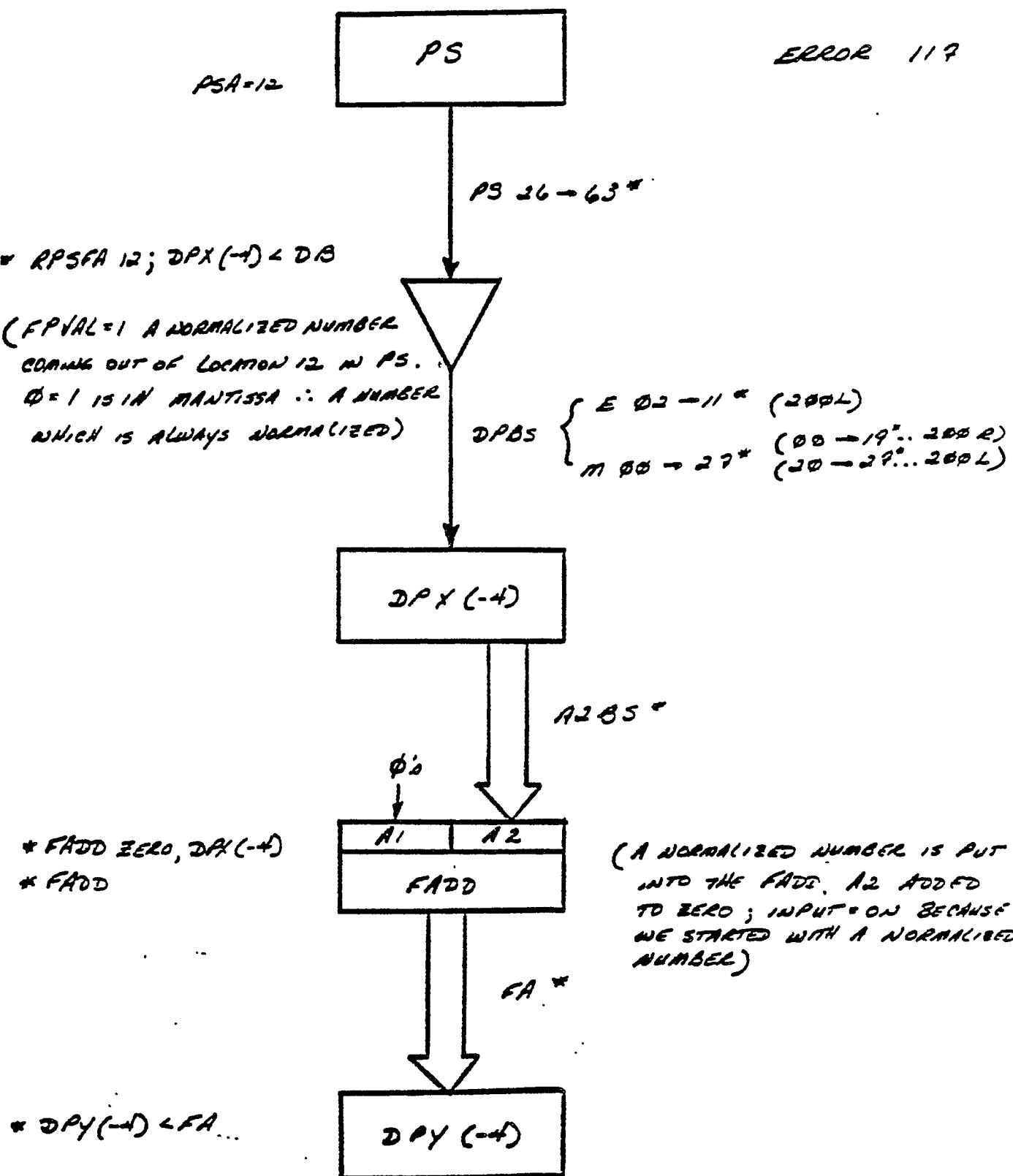
GE MEDICAL SYSTEMS INSTITUTE



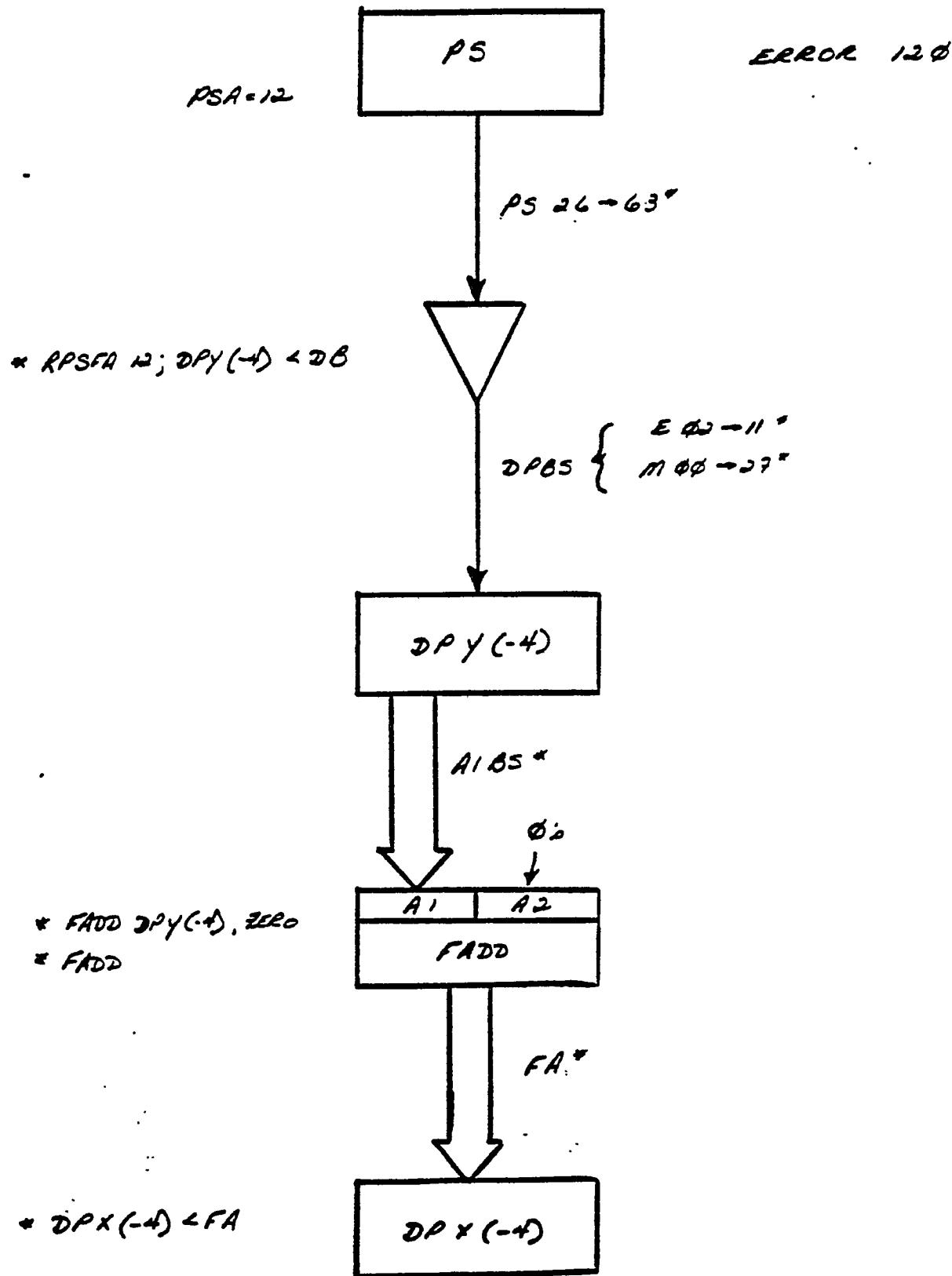
GE MEDICAL SYSTEMS INSTITUTE



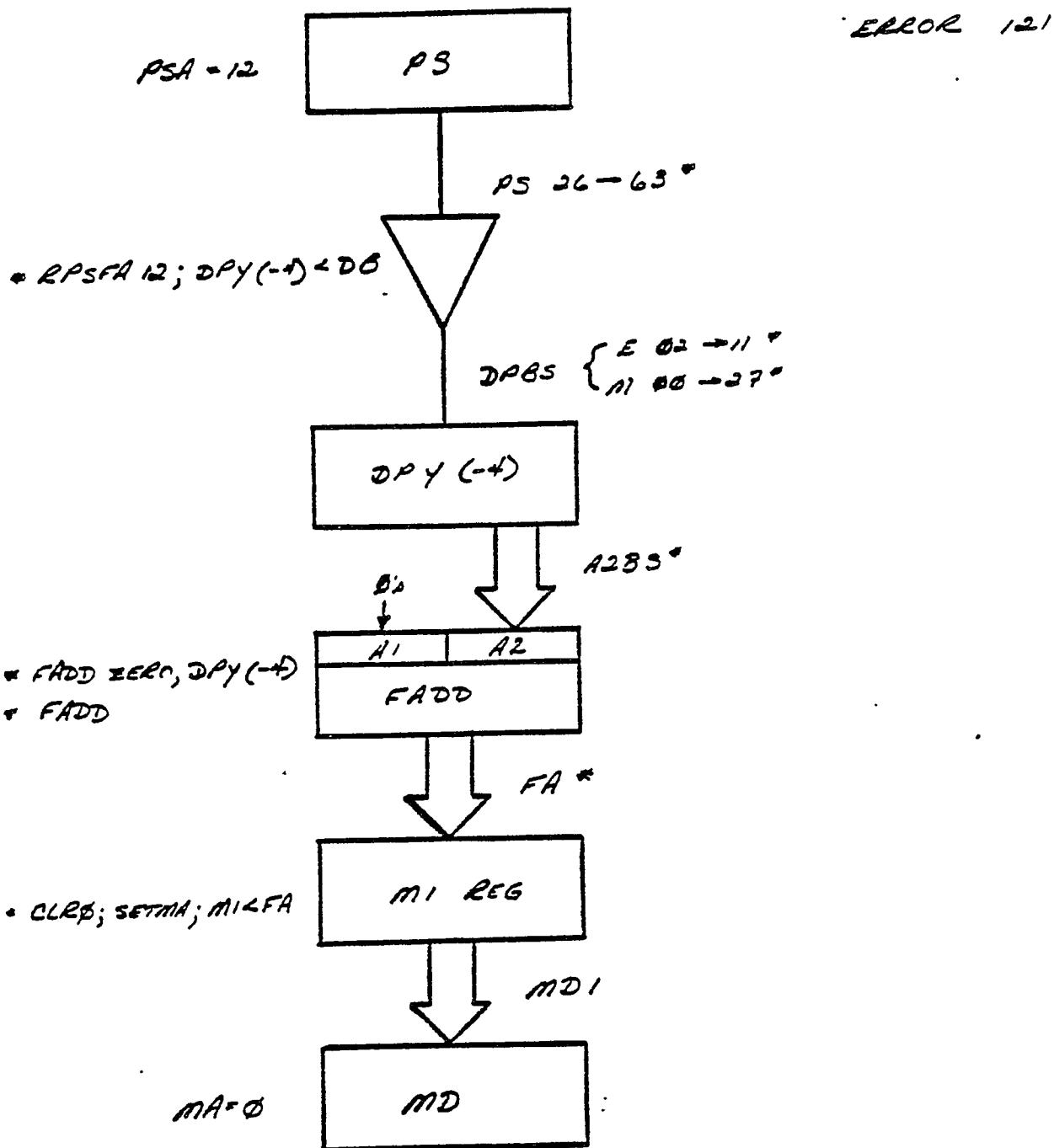
GE MEDICAL SYSTEMS INSTITUTE



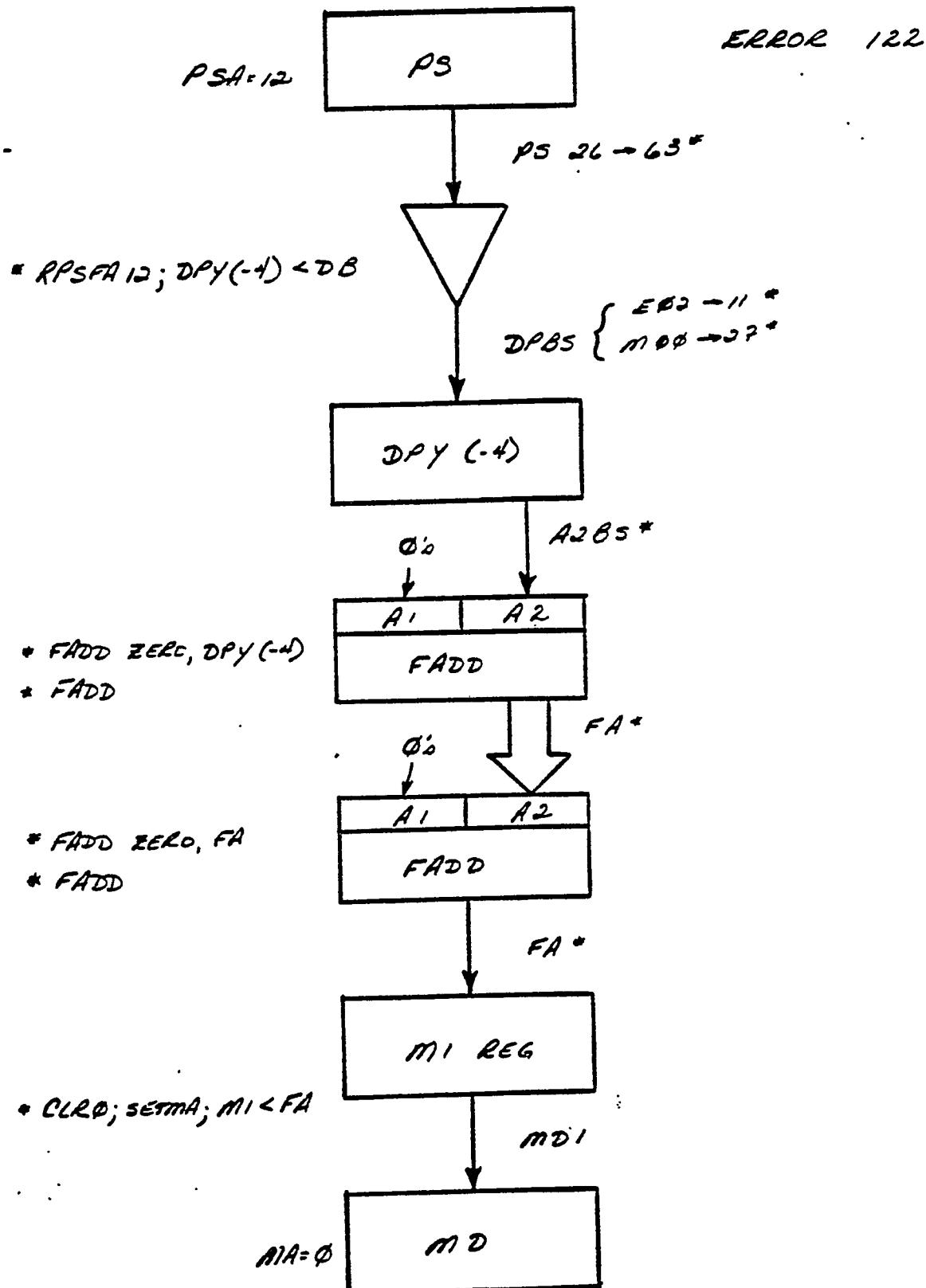
GE MEDICAL SYSTEMS INSTITUTE



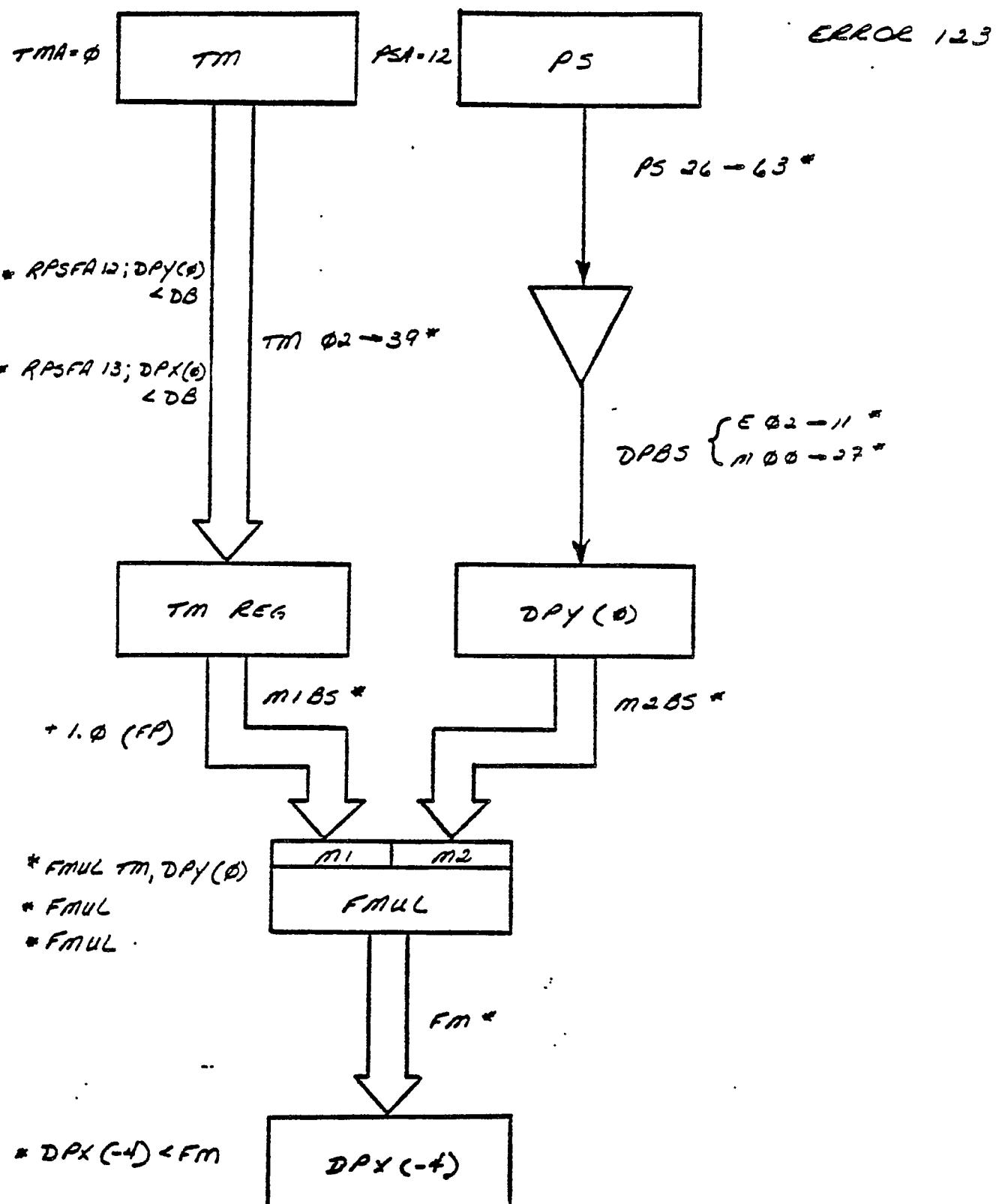
GE MEDICAL SYSTEMS INSTITUTE



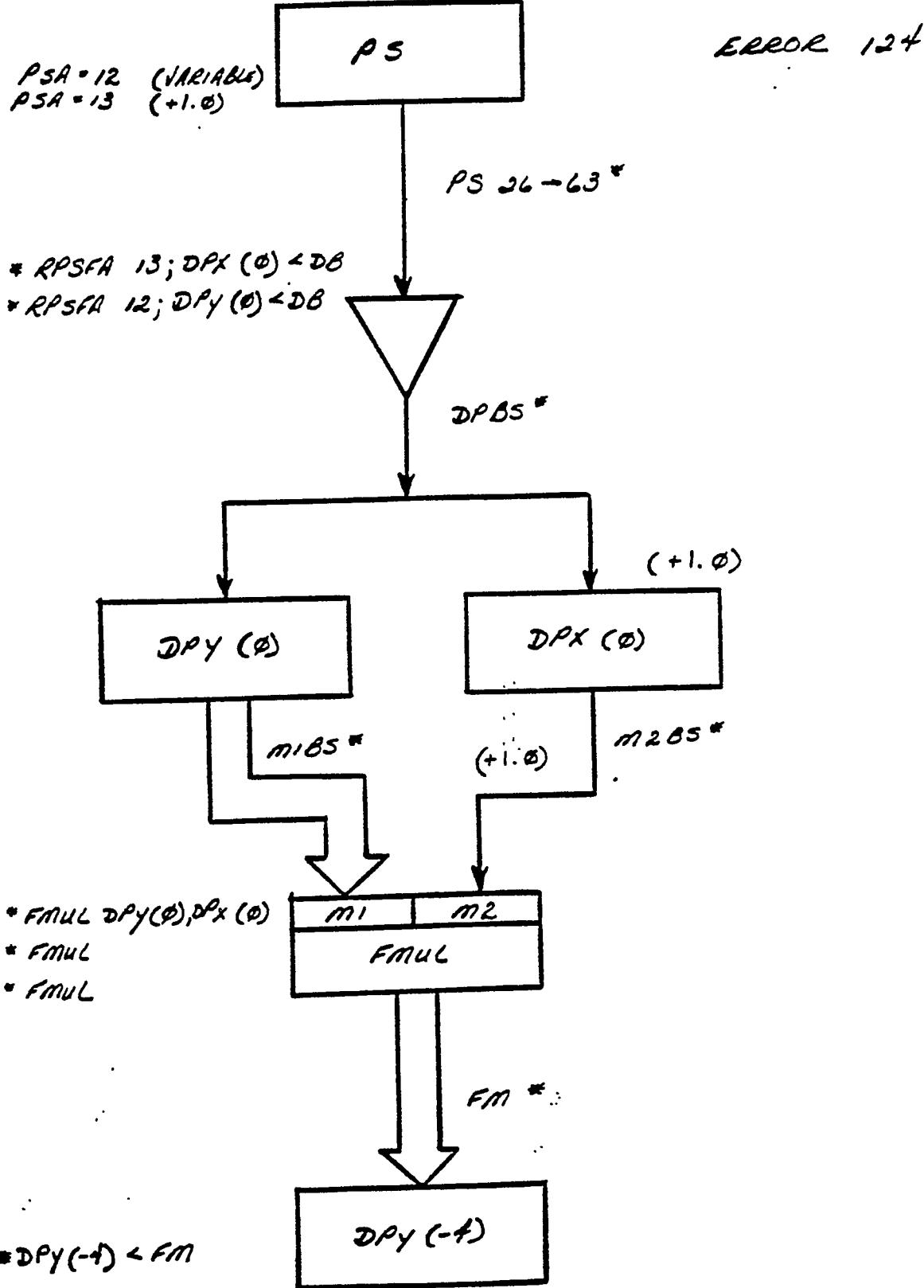
GE MEDICAL SYSTEMS INSTITUTE



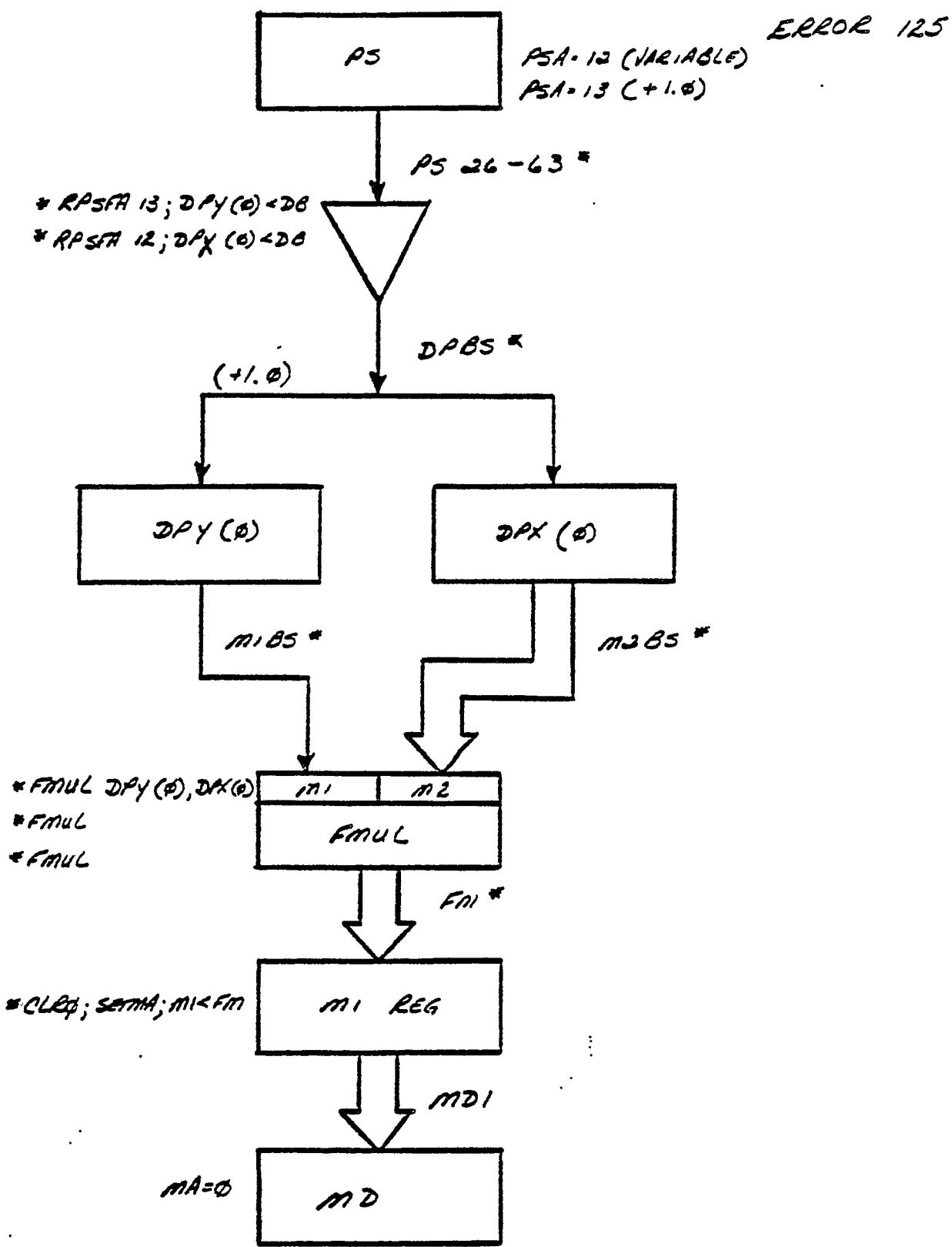
GE MEDICAL SYSTEMS INSTITUTE



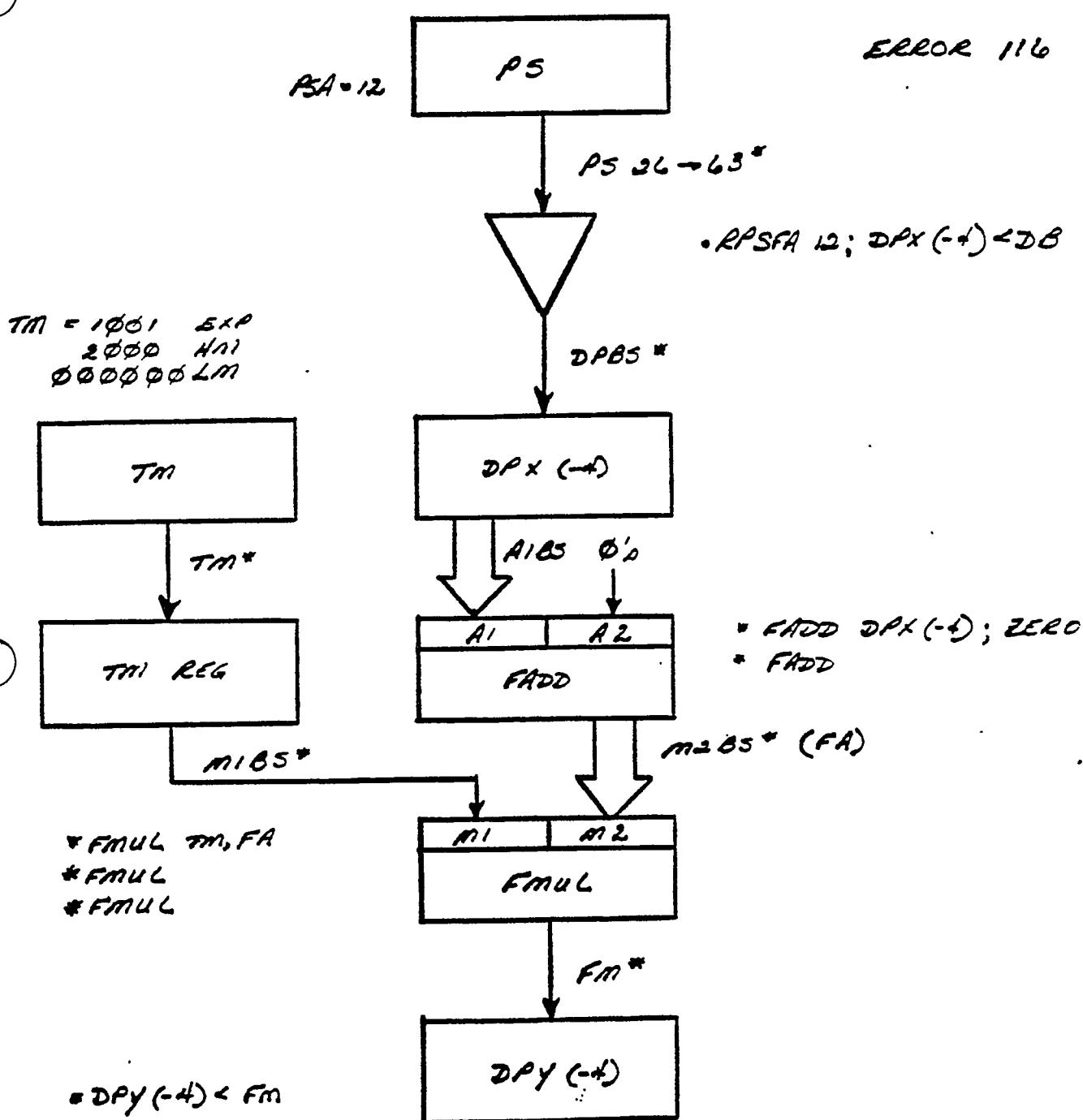
GE MEDICAL SYSTEMS INSTITUTE



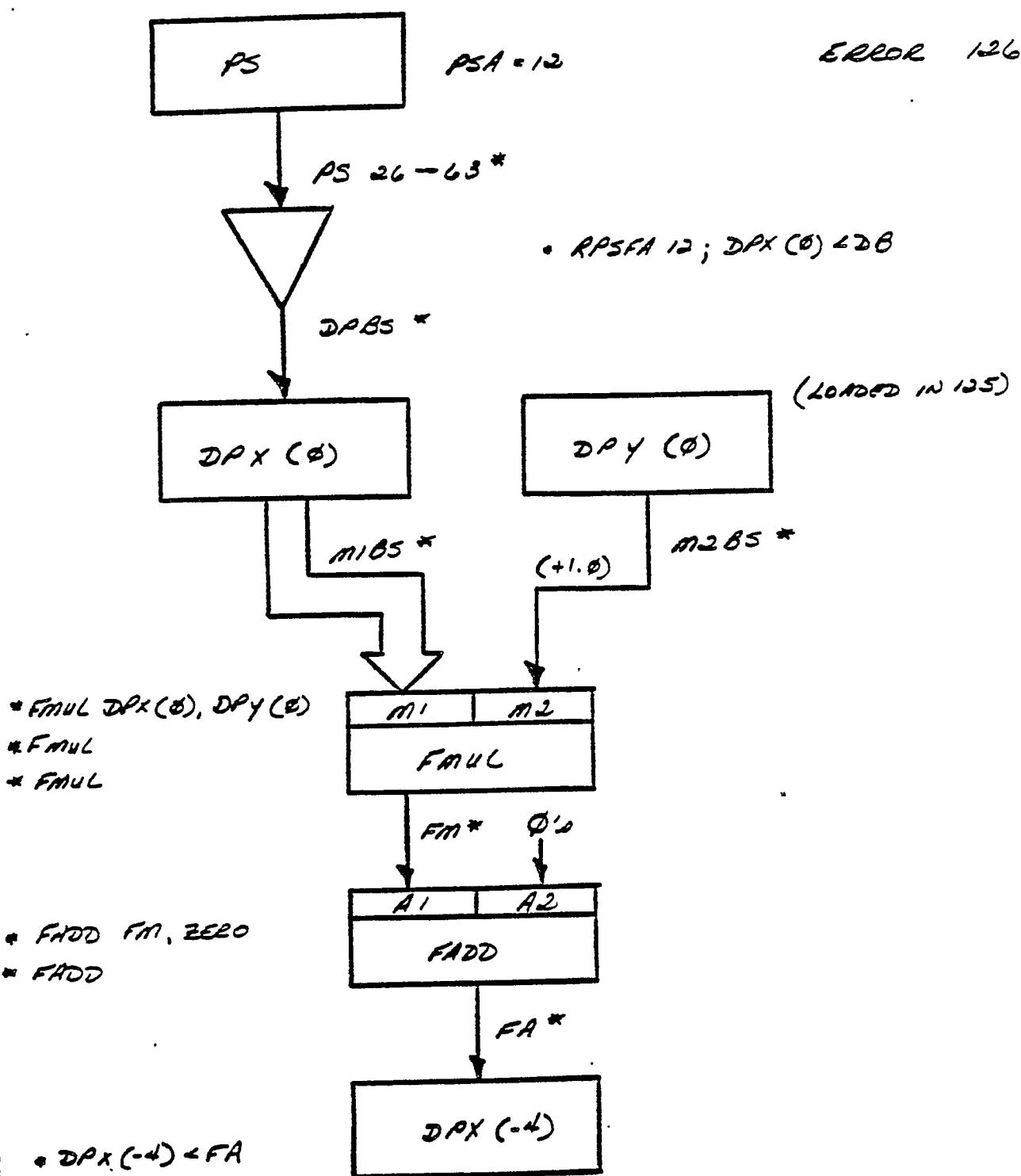
GE MEDICAL SYSTEMS INSTITUTE



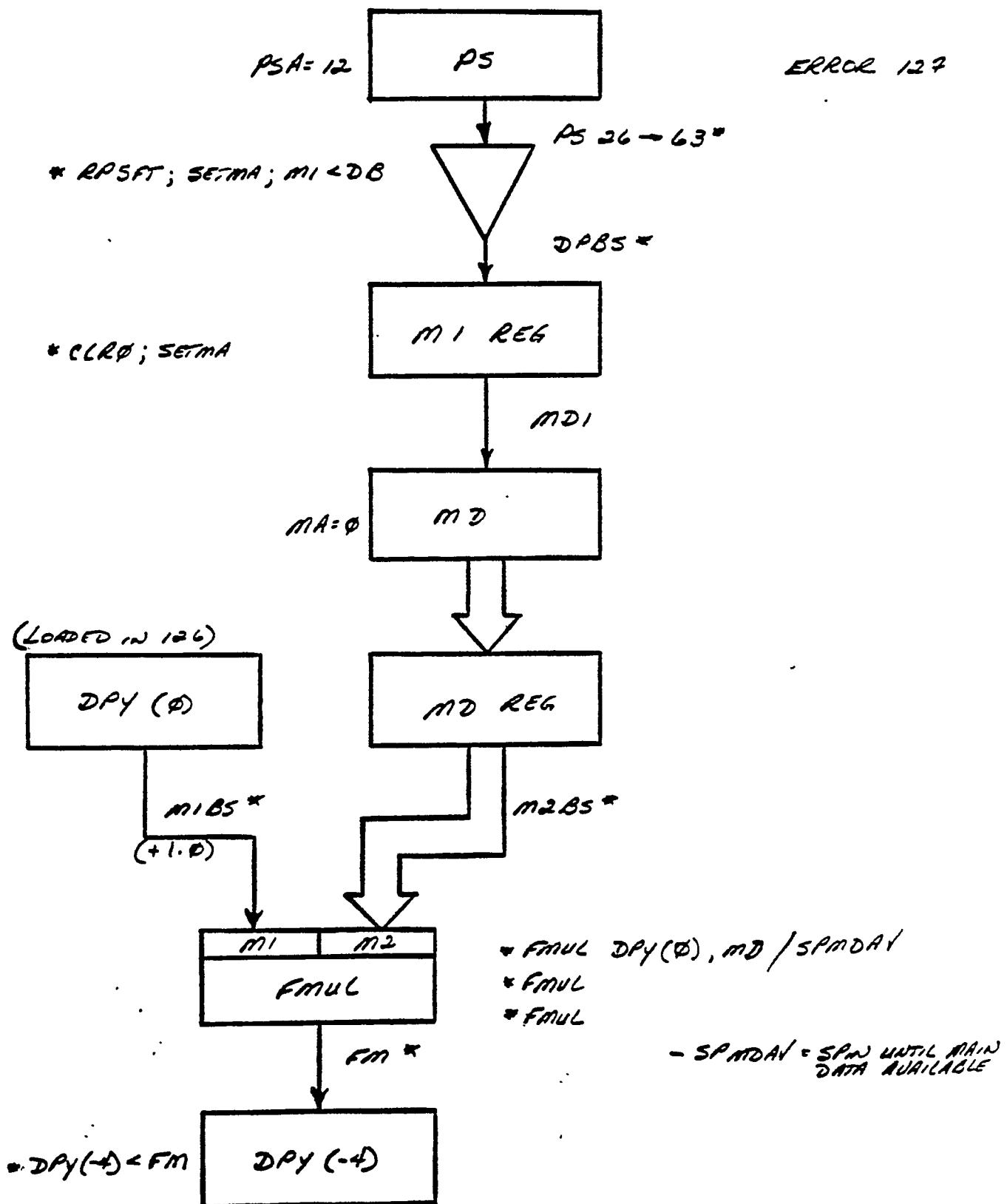
GE MEDICAL SYSTEMS INSTITUTE



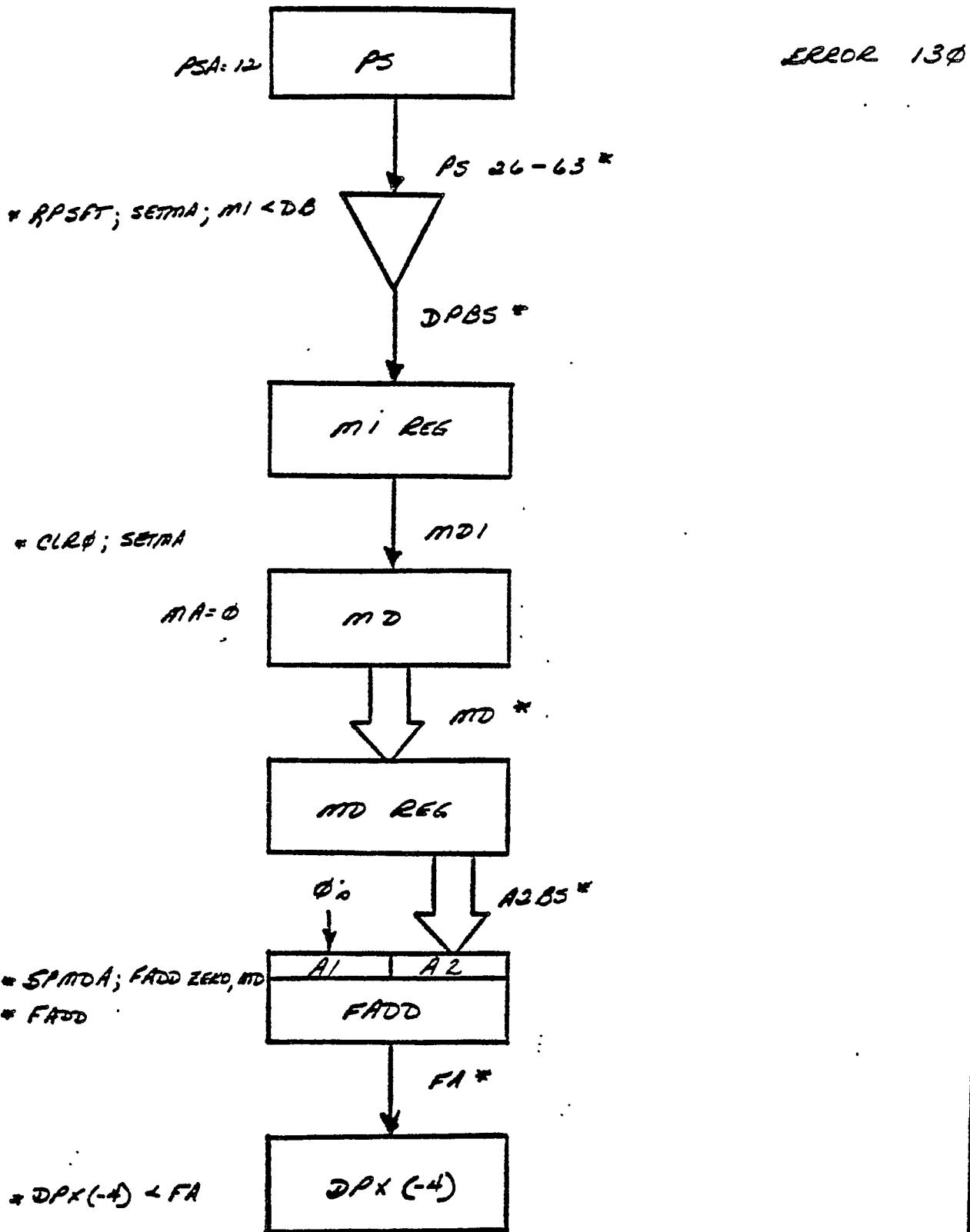
GE MEDICAL SYSTEMS INSTITUTE



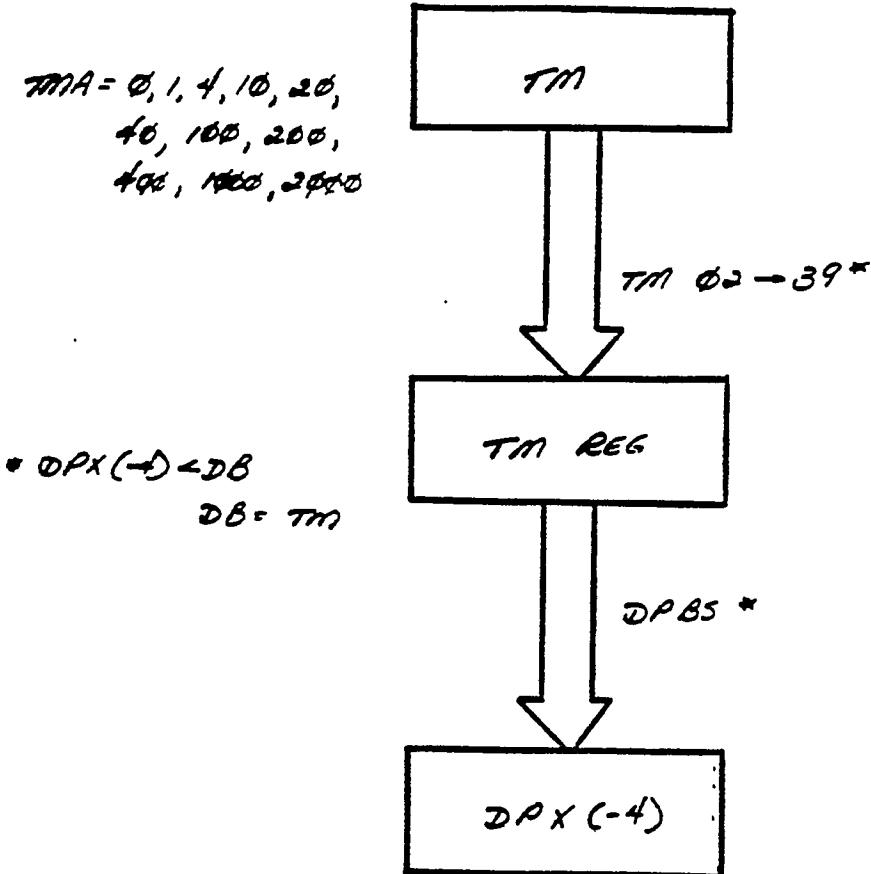
GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

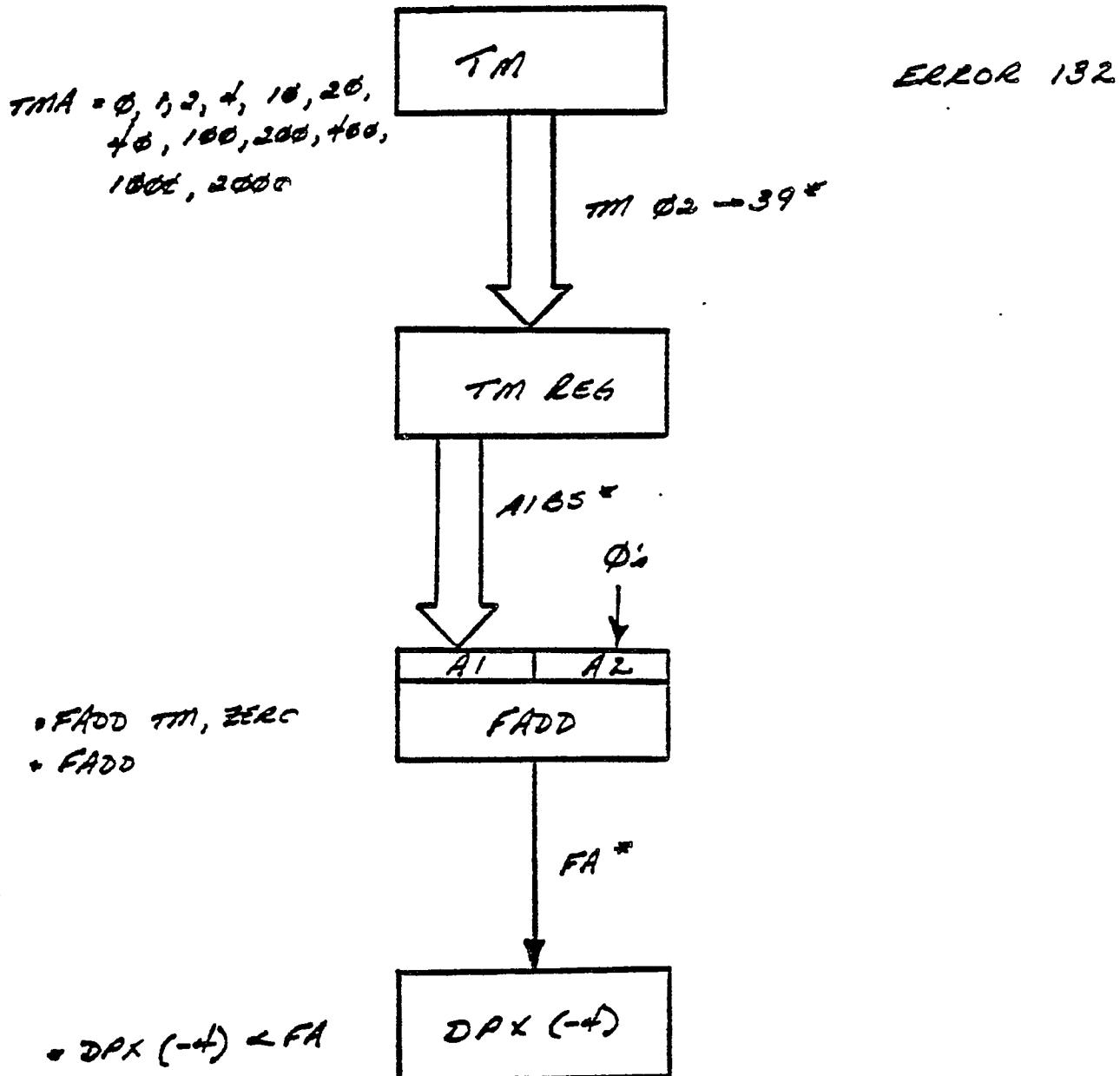


ERROR 131

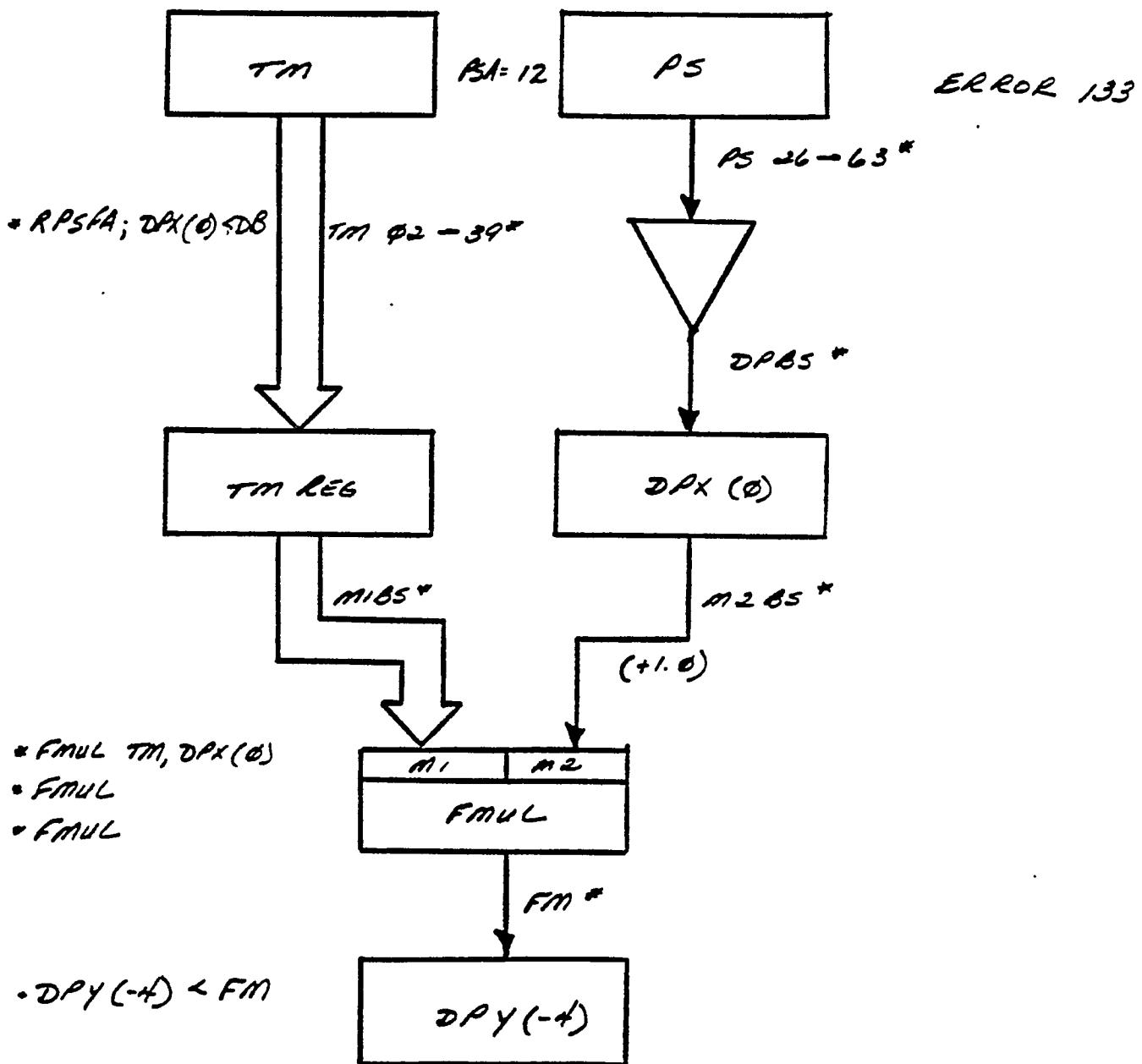
THERE IS A TABLE IN THE PROGRAM WHICH HAS THE CONTENTS OF TM LOC Ø, 1, ..., 2000. IT PULLS DATA OUT OUT ONE OF THREE DATA PATHS AND COMPARES THE VALUE PULLED OUT WITH THE VALUE IN THE TABLE

TM1 = Ø - 400	1st Row	} ON BOARD
1000 - 1999	2nd Row	
2000 -	3rd Row	

GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

VALUE FIELD PLACED
ONTO LOWER 16 BITS OF
DPBS

VALUE

ERROR 134

* LDSP1 ϕ ; DD = ϕ
(PUTS 5-PAD INTEGER ONTO
LOWER 16 BITS OF DPBS)

PS 48-63°

DPBS M12 - 27°

SPI = ϕ

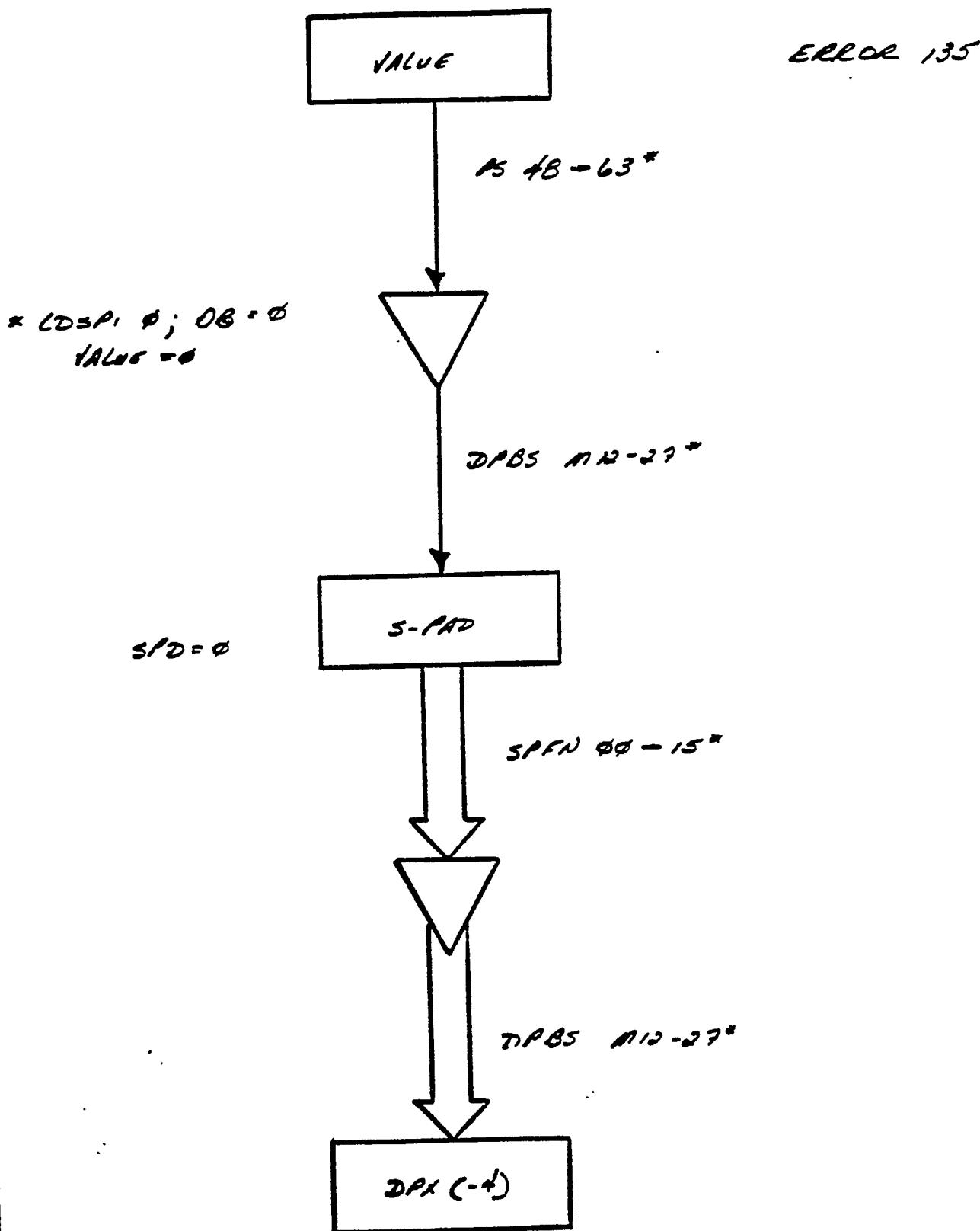
SPAD

SPCN $\phi\phi - 15^{\circ}$

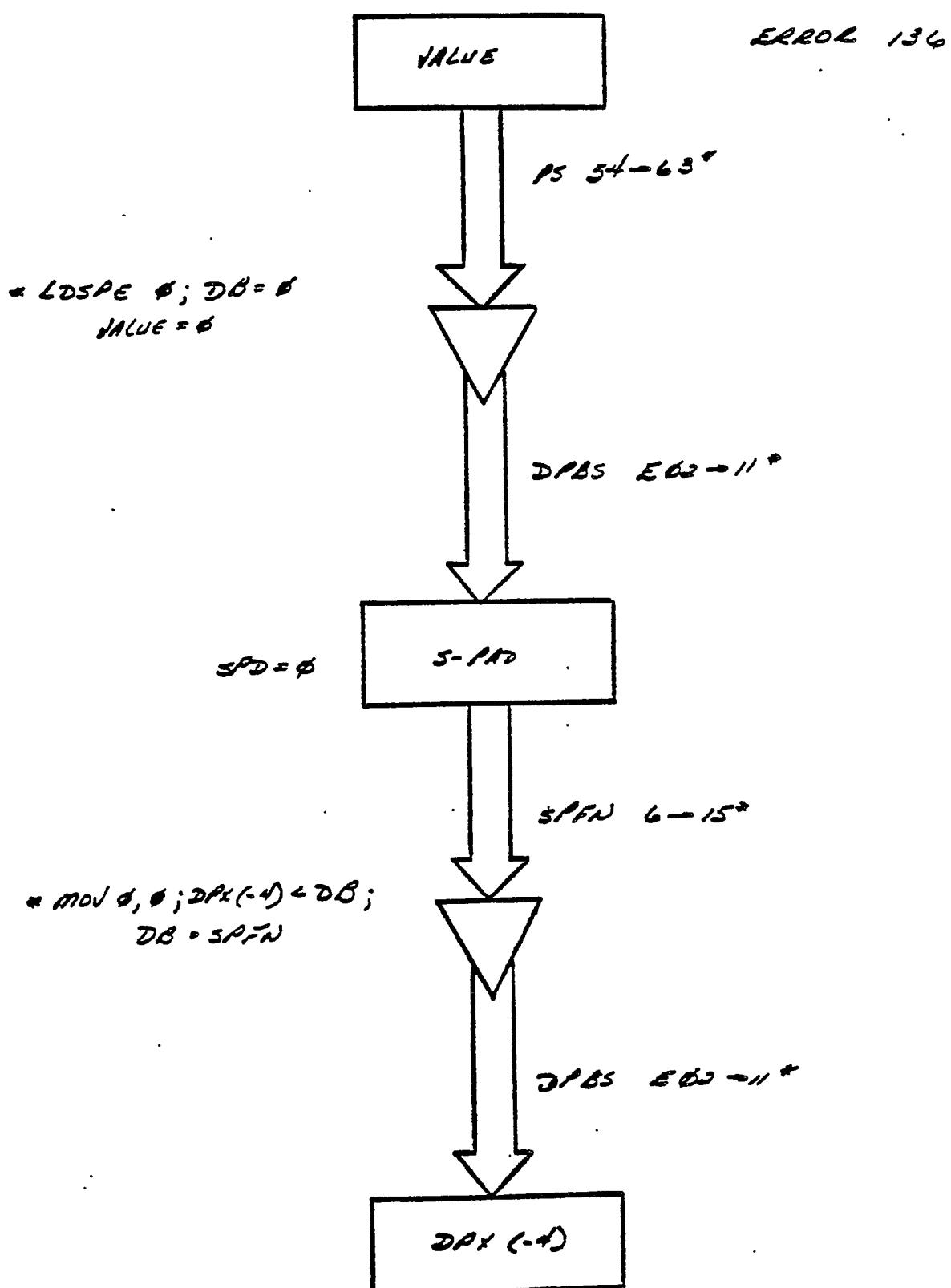
* MOV ϕ, ϕ ; SETMA

MA

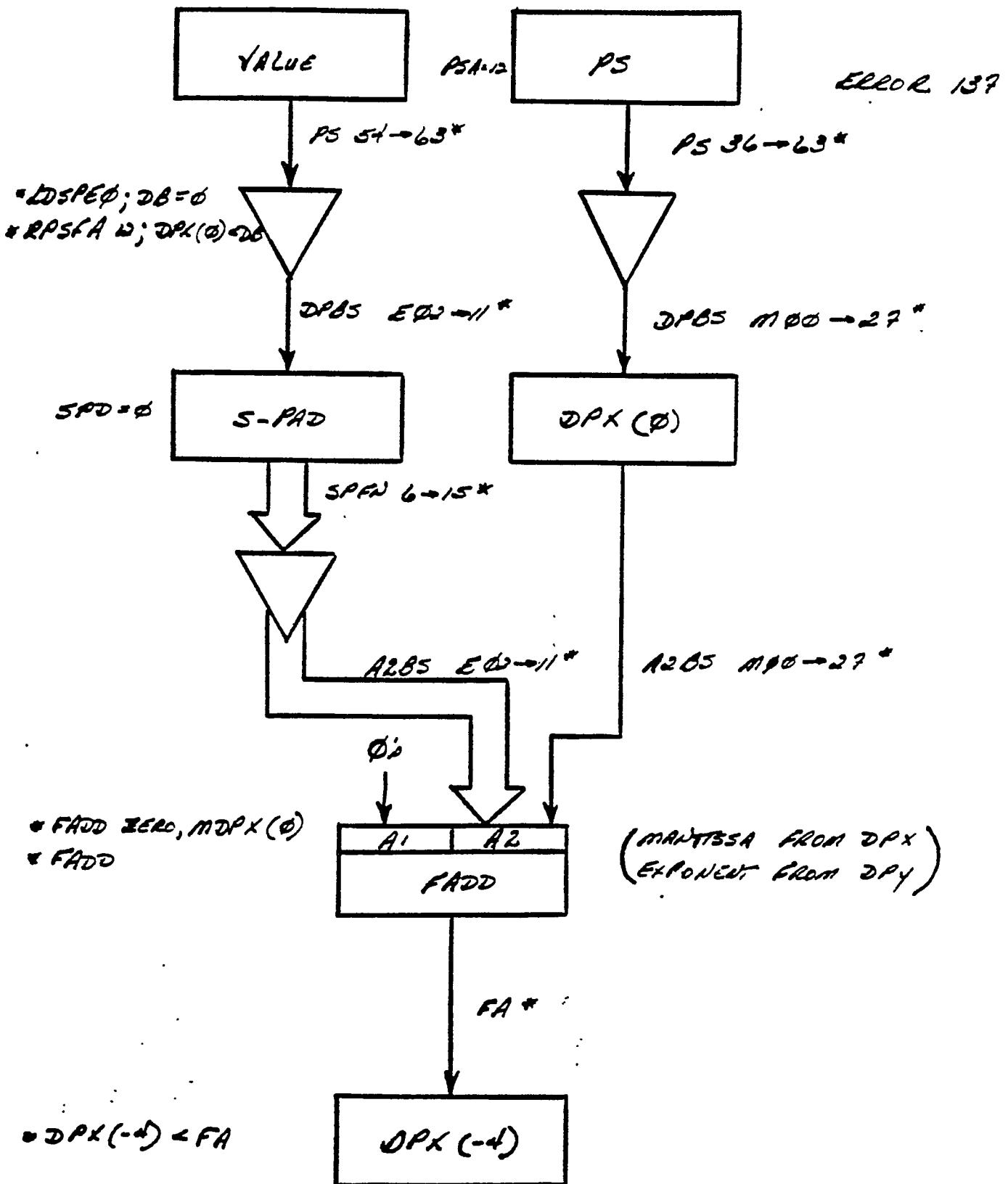
GE MEDICAL SYSTEMS INSTITUTE



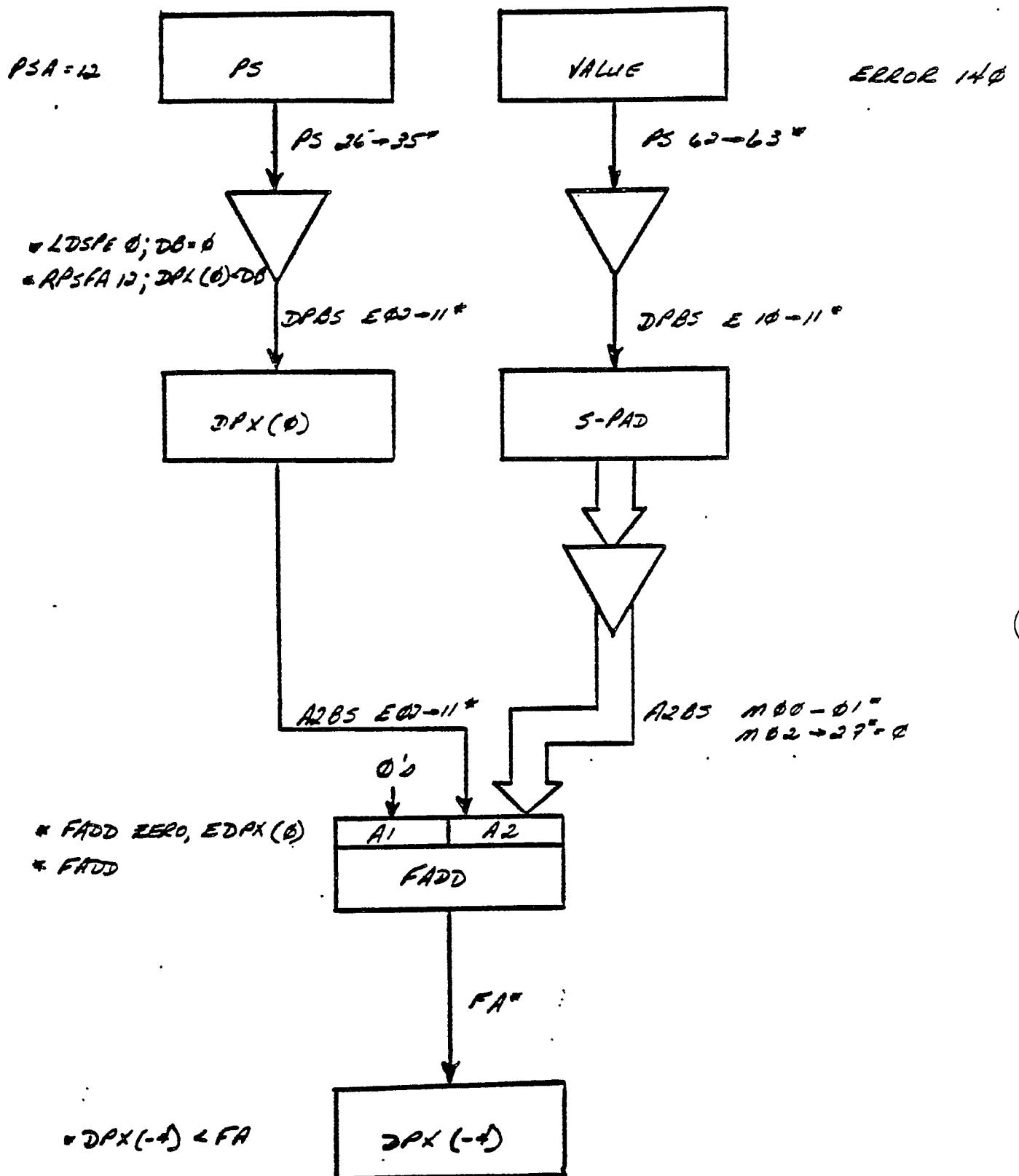
GE MEDICAL SYSTEMS INSTITUTE



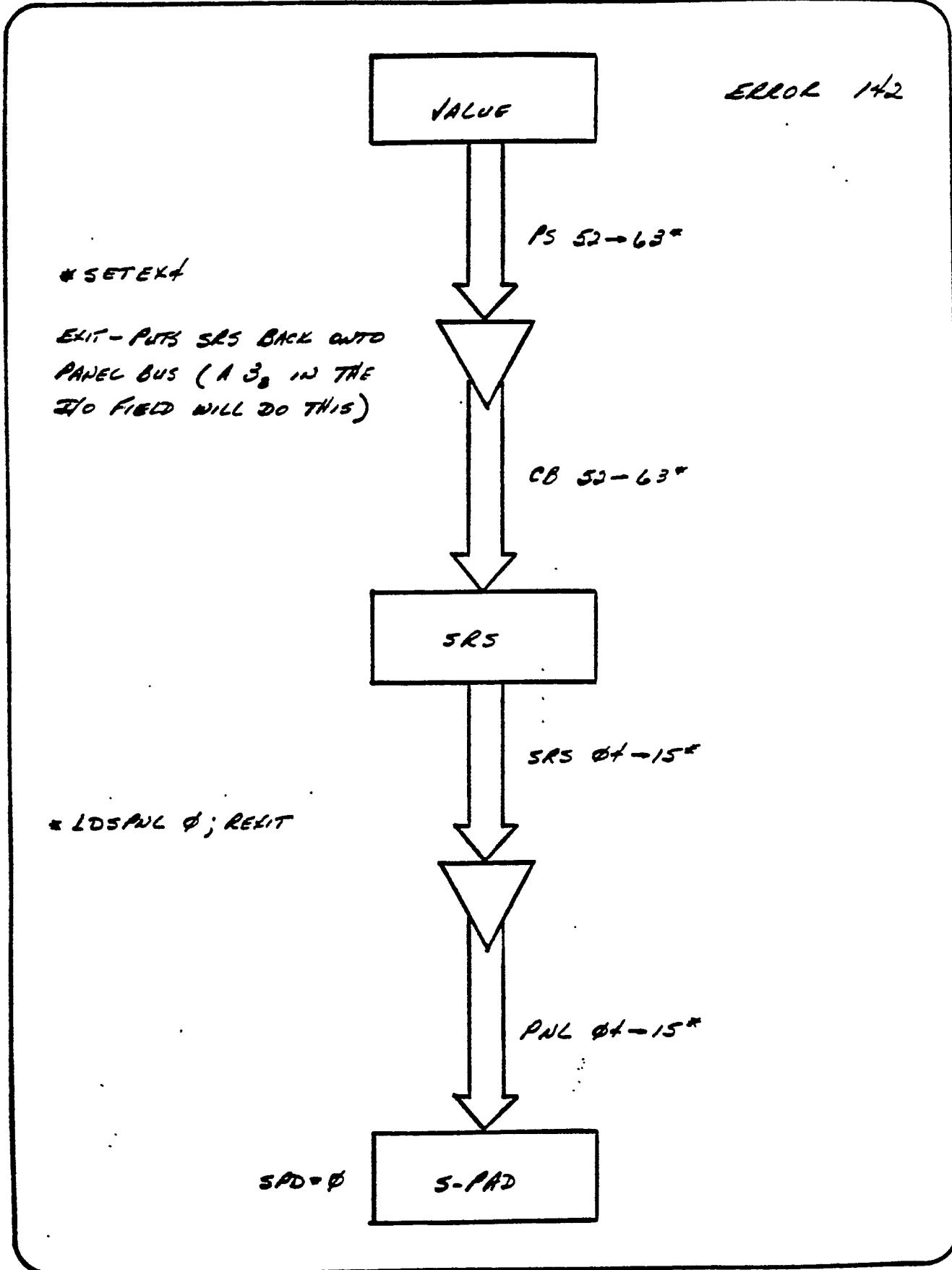
GE MEDICAL SYSTEMS INSTITUTE



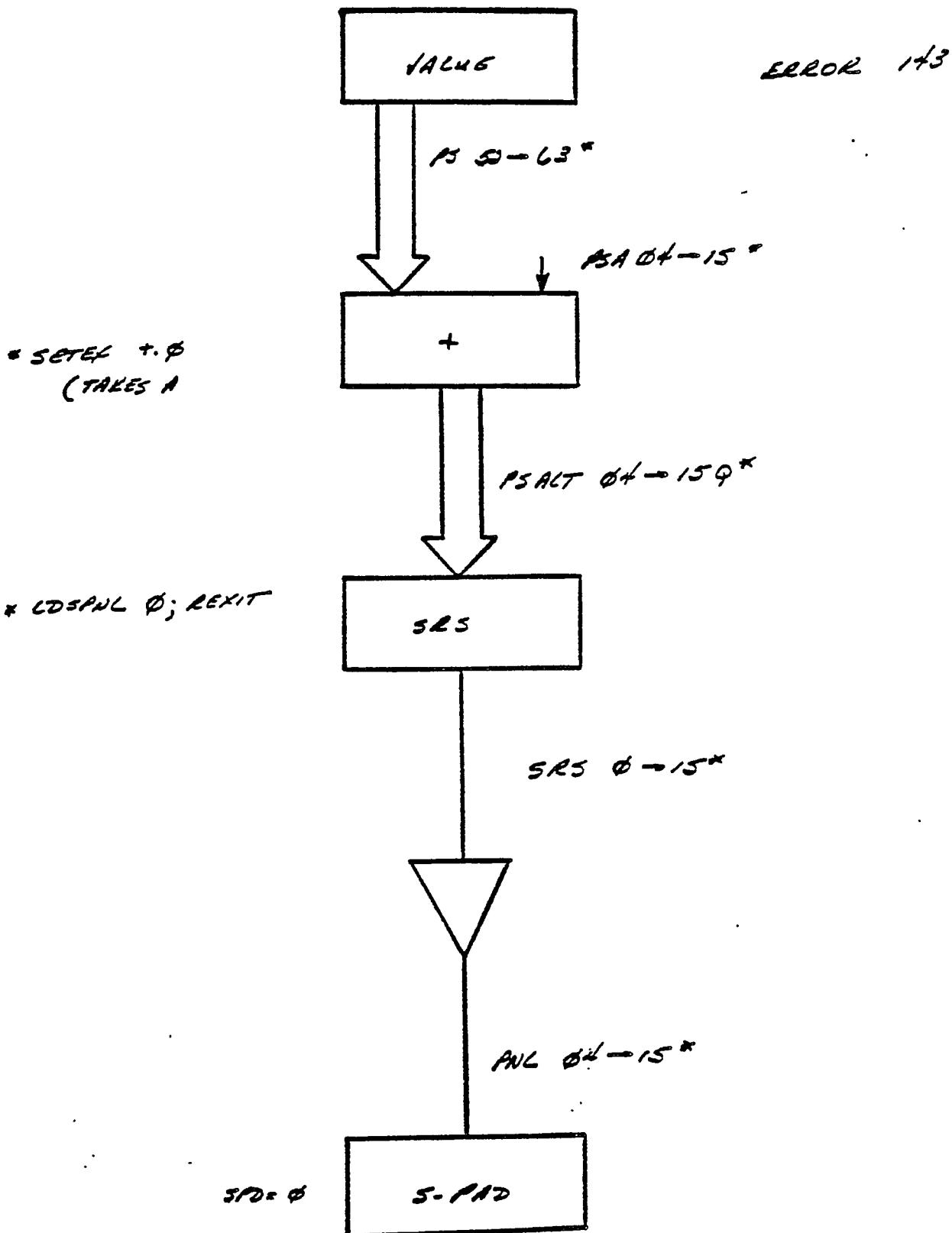
GE MEDICAL SYSTEMS INSTITUTE



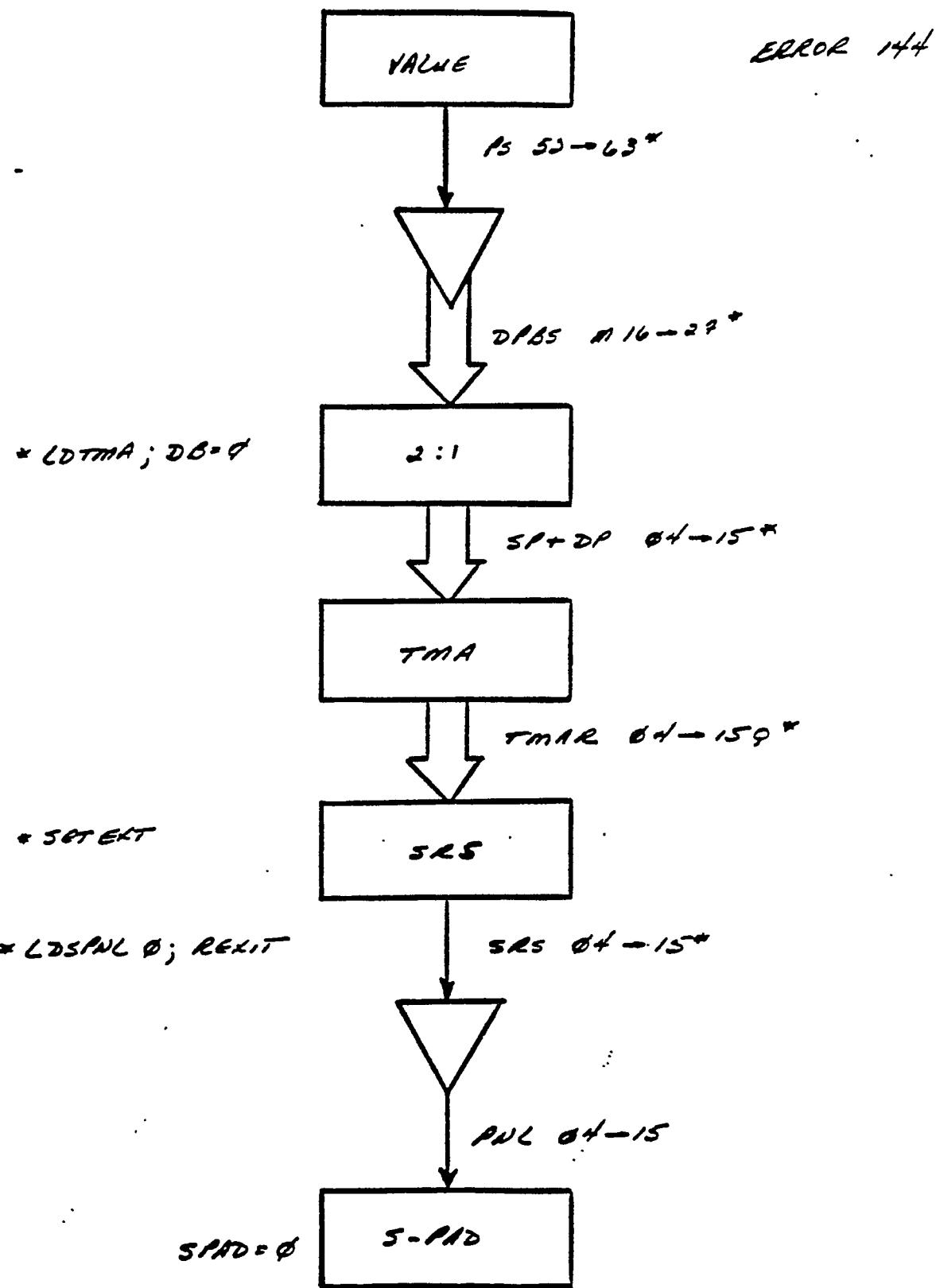
GE MEDICAL SYSTEMS INSTITUTE



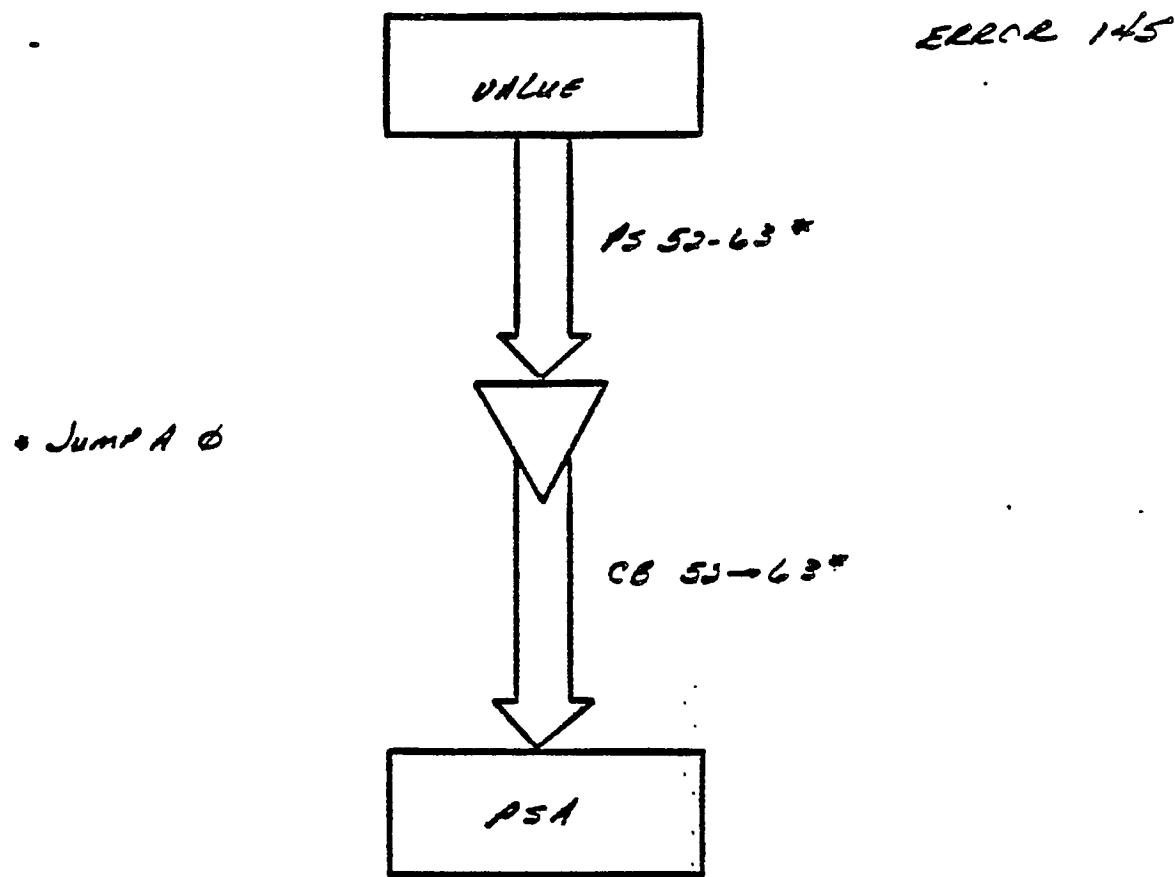
GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

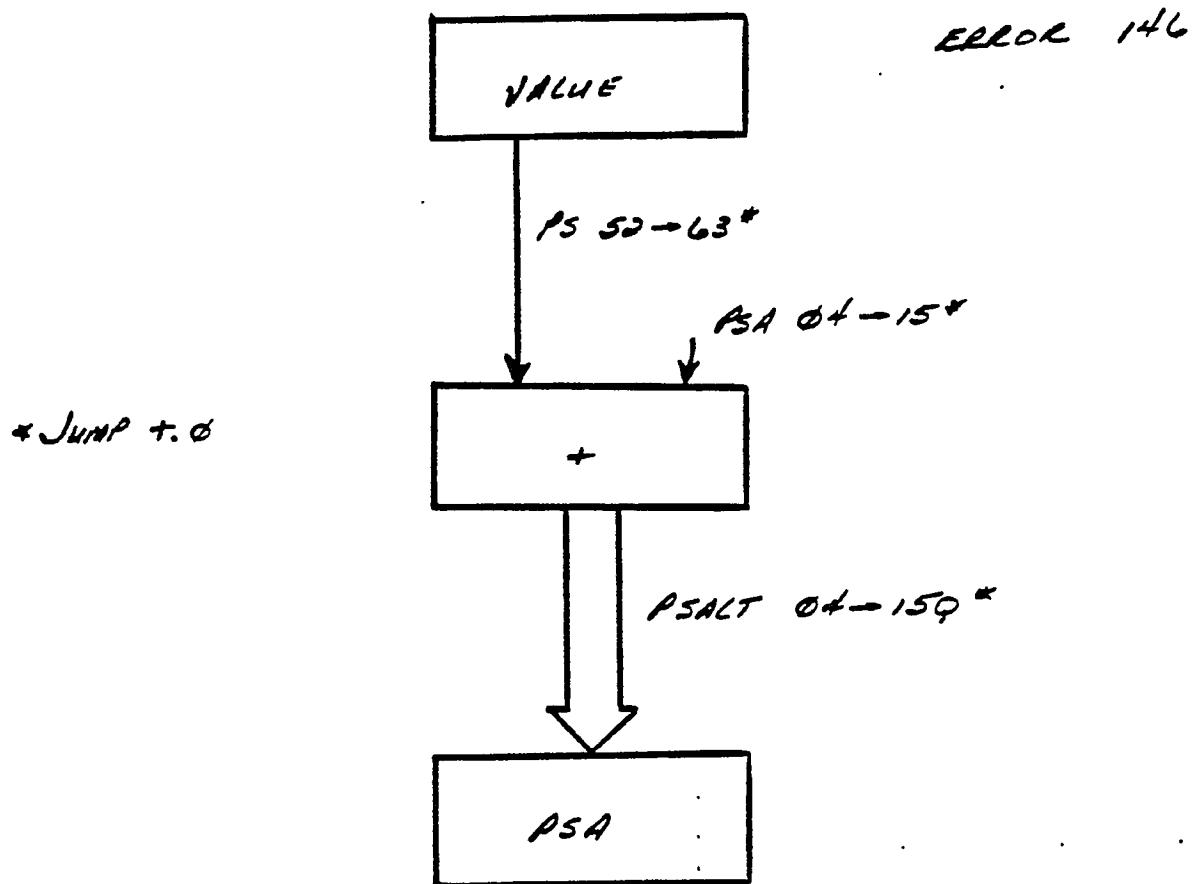


GE MEDICAL SYSTEMS INSTITUTE

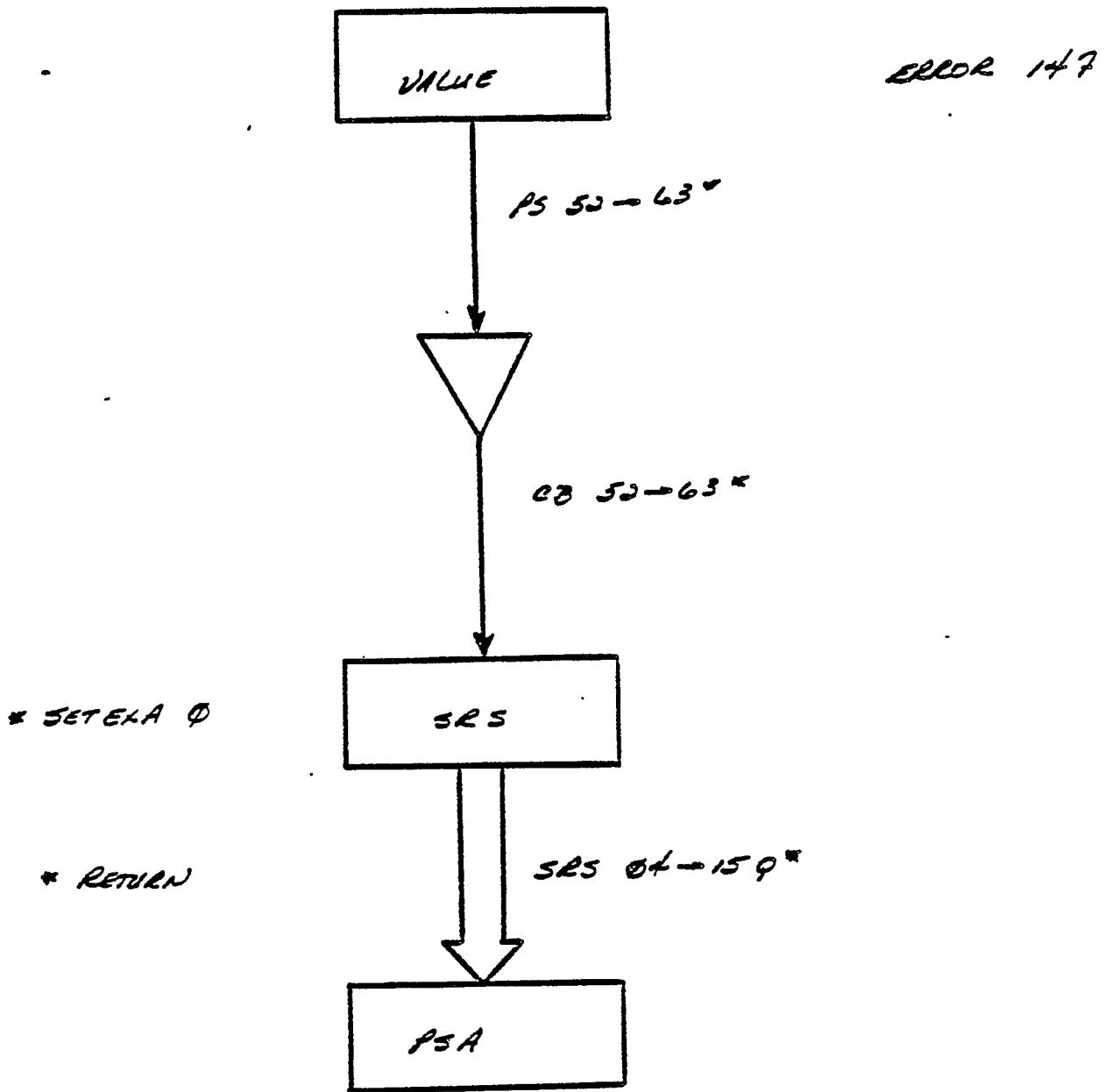


LAST FOUR (*) TESTS : 145, 146, 147, 150
ACTUALLY STEP THE AP NOT RUN IT

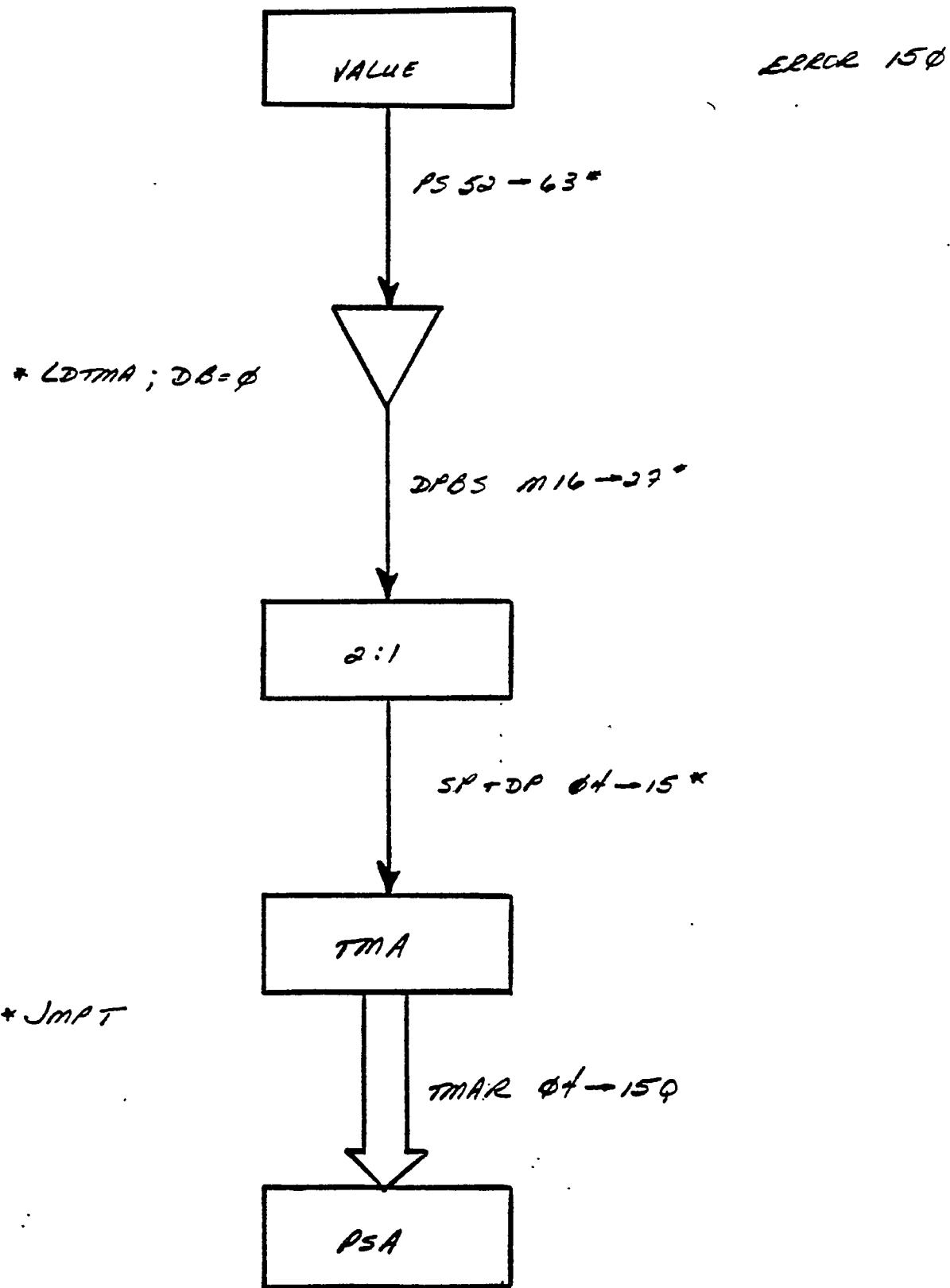
GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE



GE MEDICAL SYSTEMS INSTITUTE

SECTION A

Troubleshooting Hints

The following Flowcharts and guides are provided as a means to help the service engineer troubleshoot the memories and temporary storage registers of the array processor. It is not the intent of this section to be an absolute solution to any particular problem, however, it is useful as a memory or register chip replacement guide. Also, it provides the setting of the HEX SWITCHES of the TMROM, TMRAM(FULL), TMRAM(1/2), Program Source, and Main Data.

IMPORTANT CHECK

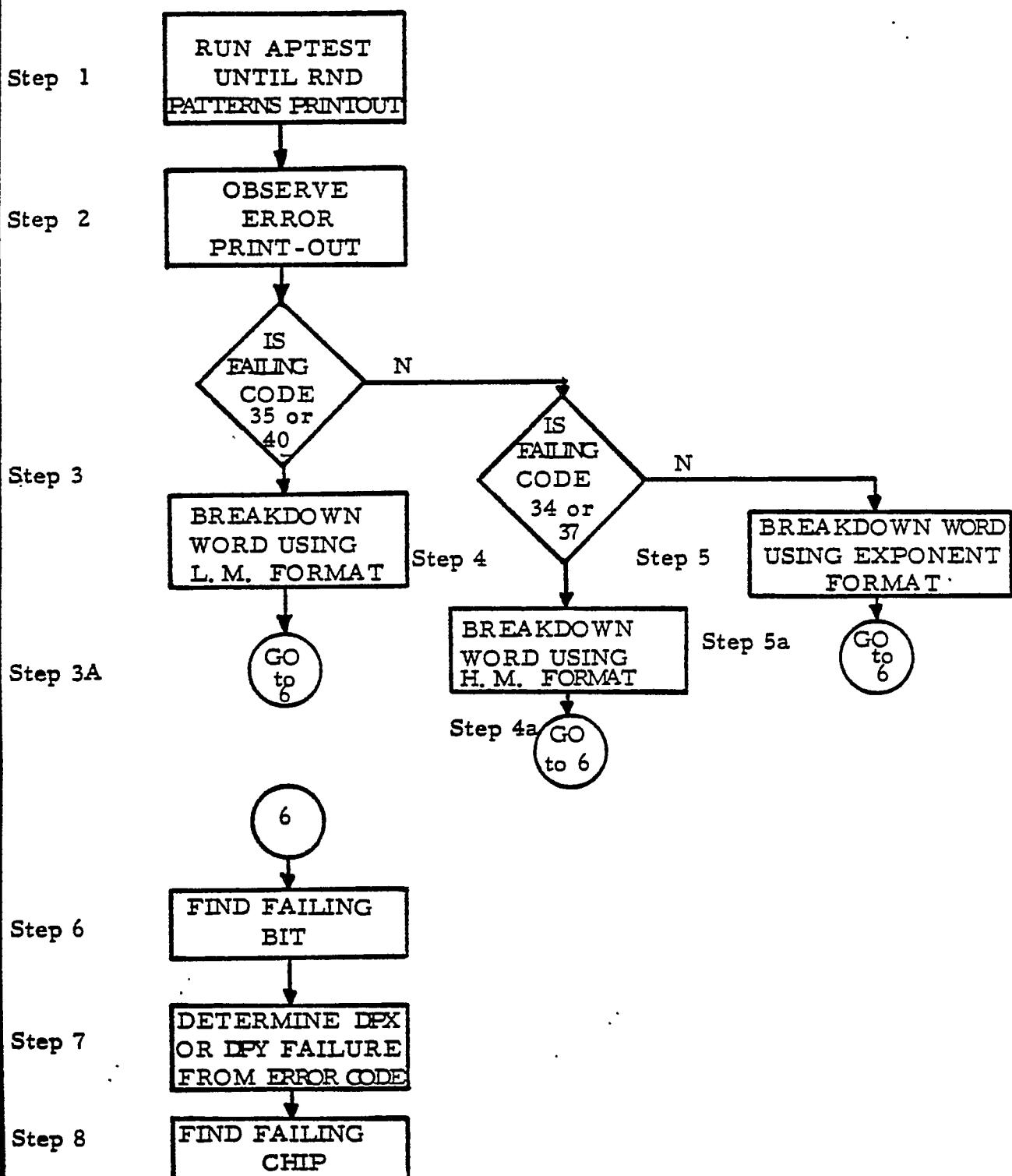
HEX SWITCH SETTINGS

<u>BOARD</u>	<u>SETTING</u>
*TMROM	0
TMRAM(FULL)	1
TMRAM (1/2)	2
Program Source	1
*Main Data	0

* NOTE: Main Data and TMROM have two switches that should be set to the same position.

GE MEDICAL SYSTEMS INSTITUTE

**FLOWCHART OF
DATA PAD STACK REGISTER REPLACEMENT**



GE MEDICAL SYSTEMS INSTITUTE

TROUBLESHOOTING GUIDE FOR DATA PAD STACK REGISTER REPLACEMENT

1. Run AP test until the failure occurs in the random patterns test.

2. Observe error message print-out.

Rnd Patterns E eeeeeee A aaaaaaa Code ccccccc Addr mmmmmmm

3. If failing code is 33, 34, 36, or 37, go to Step 4. Break the word down using the following low mantissa format.

E	e	e	e	e	e	e	A	a	a	a	a	a	a	a
Low Mantissa	M	MMM	MMM	MMM	MMM	MMM	M	MMM						
Bits	12	13-15	16-18	19-21	22-24	25-27	12	13-15	16-18	19-21	22-24	25-27	25-27	25-27

3A. Go to Step 6.

4. If failing code is 33 or 36, go to Step 5. Break the word down using the following high mantissa format.

E	e	e	e	e	e	A	a	a	a	a	a	a	a	
High Mantissa	X	XXX	MMM	MMM	MMM	MMM	X	XXX	MMM	MMM	MMM	MMM	MMM	MMM
Bits	012	345	678	9-11			012	345	678	9-11				

Note: The X's represent 0's not used in the breakdown.

4A. Go to Step 6.

5. Break the word down using the following exponent format.

E	e	e	e	e	e	A	a	a	a	a	a	a	a	
Exponent	X	XXX	XXE	EEE	EEE	X	XXX	XXE	EEE	EEE	EEE	EEE	EEE	EEE
Bits	2	345	678	9-11		2	345	678	9-11					

Note: The X's represent 0's not used in the breakdown.

5A. Go to Step 6.

6. Find the octal number in the expected and actual word that is different. From the breakdown of those numbers, determine which bit is different. This is the failing bit.

7. Determine if the failure occurred in DPX or DPY by looking at the failing code of the error message print-out and the following chart.

<u>Failing Code</u>	<u>Failing Data Pad</u>
33)	DPX
34)	"
35)	"
36)	DPY
37)	"
40)	"

GE MEDICAL SYSTEMS INSTITUTE

8. Determine the failing chip by using the results obtained in Steps 2, 6, and 7 with the Data Pad Stack Register Matrix following.

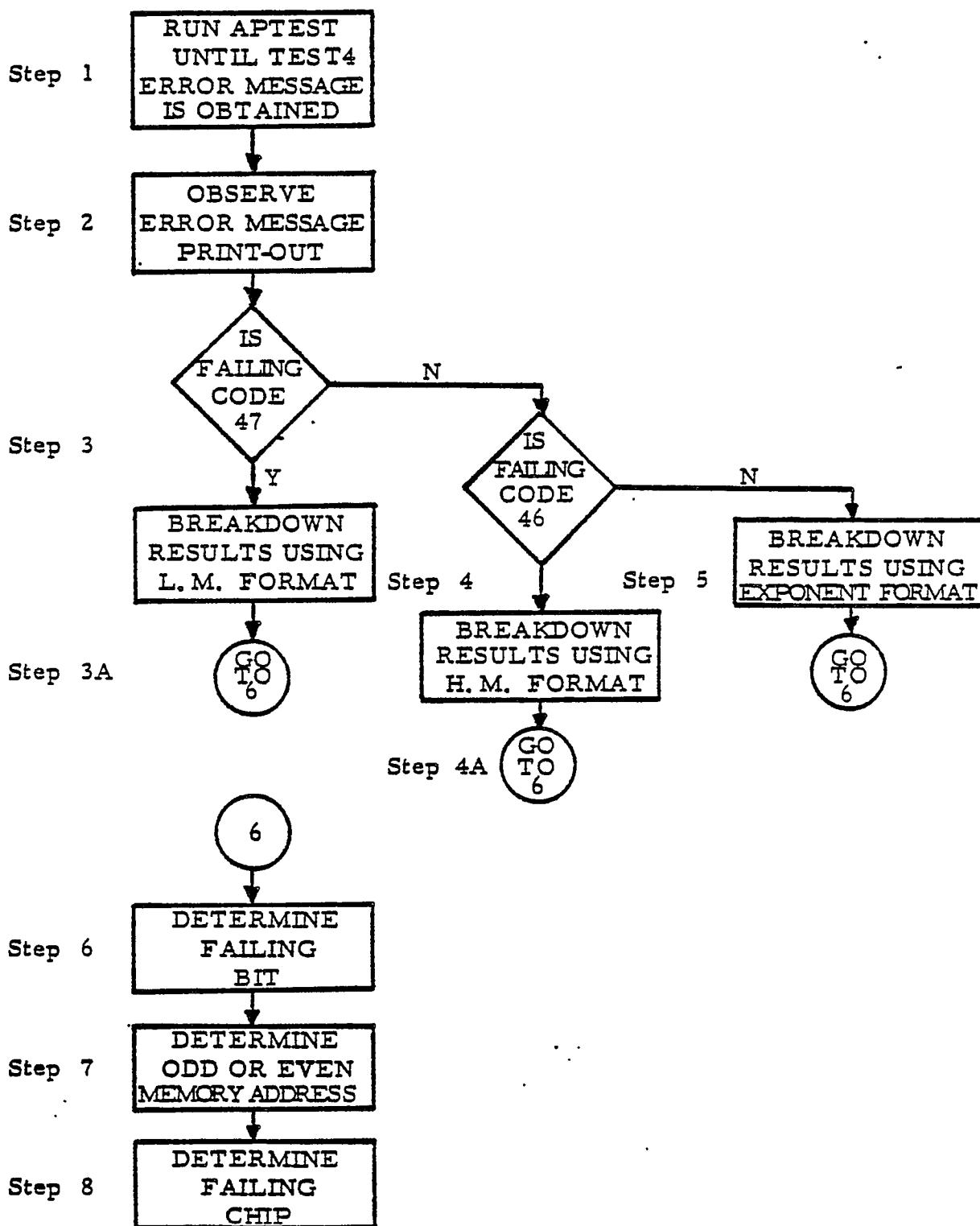
DATA PAD STACK REGISTER MATRIX

Failing Bit Position	Failing DP Address Range ==> 00 - 17	20 - 37	
	DPX	DPY	DPX
	L-D2	L-D3	L-E2
E2 - E3	L-D4	L-D5	L-E4
E4 - E7	L-D6	L-D7	L-E6
E8 - E11	R-D2	R-D3	R-E2
M00 - M03	R-D4	R-D5	R-E4
M04 - M07	R-D6	R-D7	R-E6
M08 - M11	R-D8	R-D9	R-E8
M12 - M15	R-D10	R-D11	R-E10
M16 - M19	L-D8	L-D9	L-E8
M20 - M23	L-D10	L-D11	L-E10
M24 - M27			L-E11

NOTE: L = Data Pad Board left which is next to the "203" board.
R = " " " right which is next to the "213" board.

GE MEDICAL SYSTEMS INSTITUTE

FLOWCHART OF MAIN DATA MEMORY TROUBLESHOOTING GUIDE



GE MEDICAL SYSTEMS INSTITUTE

TROUBLESHOOTING GUIDE FOR MAIN DATA MEMORY CHIP REPLACEMENT

1. Run AP test until the failure occurs in the random patterns test.
2. Observe error message print-out.

RND Patterns E eeeeeee A aaaaaaa Code ccccccc Addr mmmmmmm

3. If failing code is 000045 or 000046, go to Step 4. Break the word down using the following Low Mantissa format.

E	e	e	e	e	e	e	A	a	a	a	a	a	a	a	a
Low Mantissa	L	LLL	LLL	LLL	LLL	LLL	L	LLL	LLL	LLL	LLL	LLL	LLL	LLL	LLL
Bits	24	25-27	28-30	31-33	34-36	37-39	24	25-27	28-30	31-33	34-36	37-39			

- 3A. Go to Step 6.

4. If failing code is 000045, go to Step 5. Break the word down using the following High Mantissa format.

E	x	x	e	e	e	e	A	x	x	e	e	e	e	e	e
High Mantissa	X	XXX	HHE	HHE	HHE	HHE	X	XXX	HHE	HHE	HHE	HHE	HHE	HHE	HHE
Bits		12-14	15-17	18-20	21-23			12-14	15-17	18-20	21-23				

Note: The X's represent 0's not used in the break down.

- 4A. Go to Step 6.

5. Break the word down using the following exponent format.

E	x	x	e	e	e	A	x	x	a	a	a	a	a	a	a
Exponent	X	XXX	XXX	EEE	EEE	EEE	X	XXX	XXX	EEE	EEE	EEE	EEE	EEE	EEE
Bits	2	345	678	9-11			2	345	678	9-11					

Note: The X's represent 0's not used in the break down.

6. Find the octal number in the expected and actual word that is different. From the breakdown of those numbers, determine which bit is different. This is the failing bit.
7. Determine from the print-out if the memory address is even or odd.
8. Determine the failing chip by using the results obtained in Step 6 and Step 7 with the main data matrix on the next page. Replace the chip. Rerun test to insure fix.

GE MEDICAL SYSTEMS INSTITUTE

AP-120B MAIN DATA

AP TEST
ERROR CODE 27, 30, 31

or 45, 46, 47

FAILING
ADDRESS
RANGE = = =
(MA = XXXX)

EVEN

ODD

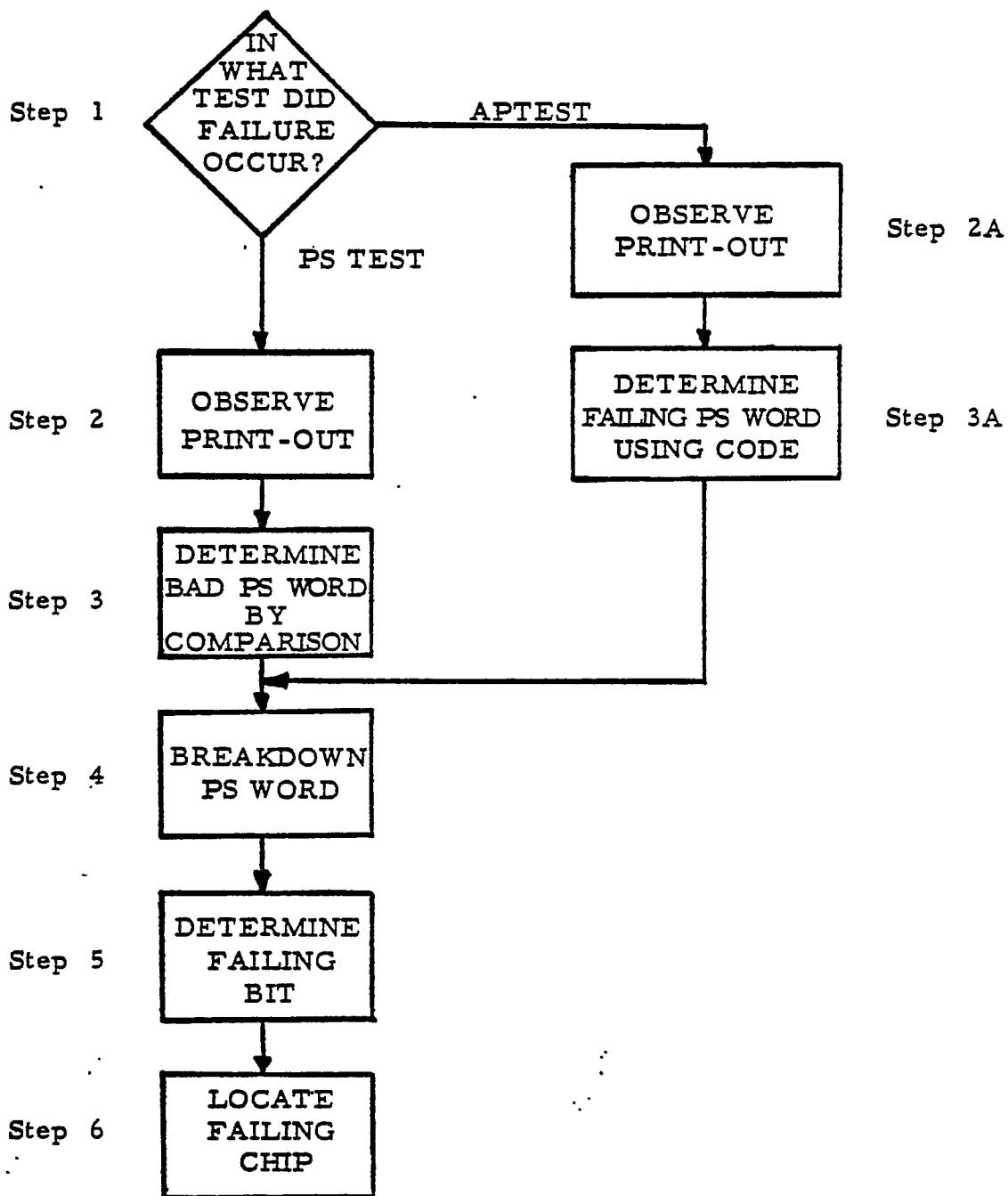
FAILING
BIT
POSITION

02	H4	F4
03	G5	E5
04	H5	F5
05	G6	E6
06	H6	F6
07	G7	E7
08	H7	F7
09	G8	E8
10	H8	F8
11	G9	E9
12	G10	E10
13	H10	F10
14	G11	E11
15	H11	F11
16	G12	E12
17	H12	F12
18	G13	E13
19	H13	F13
20	G14	E14
21	H14	F14
22	G15	E15
23	H15	F15
24	G16	E16
25	H16	F16
26	G17	E17
27	H17	F17
28	G18	E18
29	H18	F18
30	G19	E19
31	H19	F19
32	G20	E20
33	H20	F20
34	G21	E21
35	H21	F21
36	G22	E22
37	H22	F22
38	G23	E23
39	H23	F23

4116P-3 IS A 16K X 1 DYNAMIC RAM (FPS USES ALTERNATE RANK ADDRESSING HERE)

GE MEDICAL SYSTEMS INSTITUTE

FLOWCHART OF PS MEMORY TROUBLESHOOTING GUIDE



GE MEDICAL SYSTEMS INSTITUTE

TROUBLESHOOTING GUIDE FOR PS MEMORY CHIP REPLACEMENT

1. If failure occurred in AP test, use steps 2A, 3A, 4, 5, & 6.
If failure occurred in PS test, use steps 2, 3, 4, 5, & 6.
2. Observe error print-out.

PS LOC = AAAAAA

E	eeeeee	eeeeee	eeeeee	eeeeee
	PSWORD0	PSWORD1	PSWORD2	PSWORD3
A	aaaaaa	aaaaaa	aaaaaa	aaaaaa
	PSWORD0	PSWORD1	PSWORD2	PSWORD3

- 2A. Observe error print-out.

E eeeeeee A aaaaaaa C cccccc

3. Compare the expected and actual results. Determine which PS word is different. This is the failing PS word.
- 3A. Determine failing PS word using the code portion of the error print-out and the following table.

Code	Failing PS Word
000016 or 000041	PSW0
000017 or 000042	PSW1
100020 or 000043	PSW2
000020 or 000044	PSW3

4. Break the failing PS word down using the following format.

E e e e e e A a a a a a a a

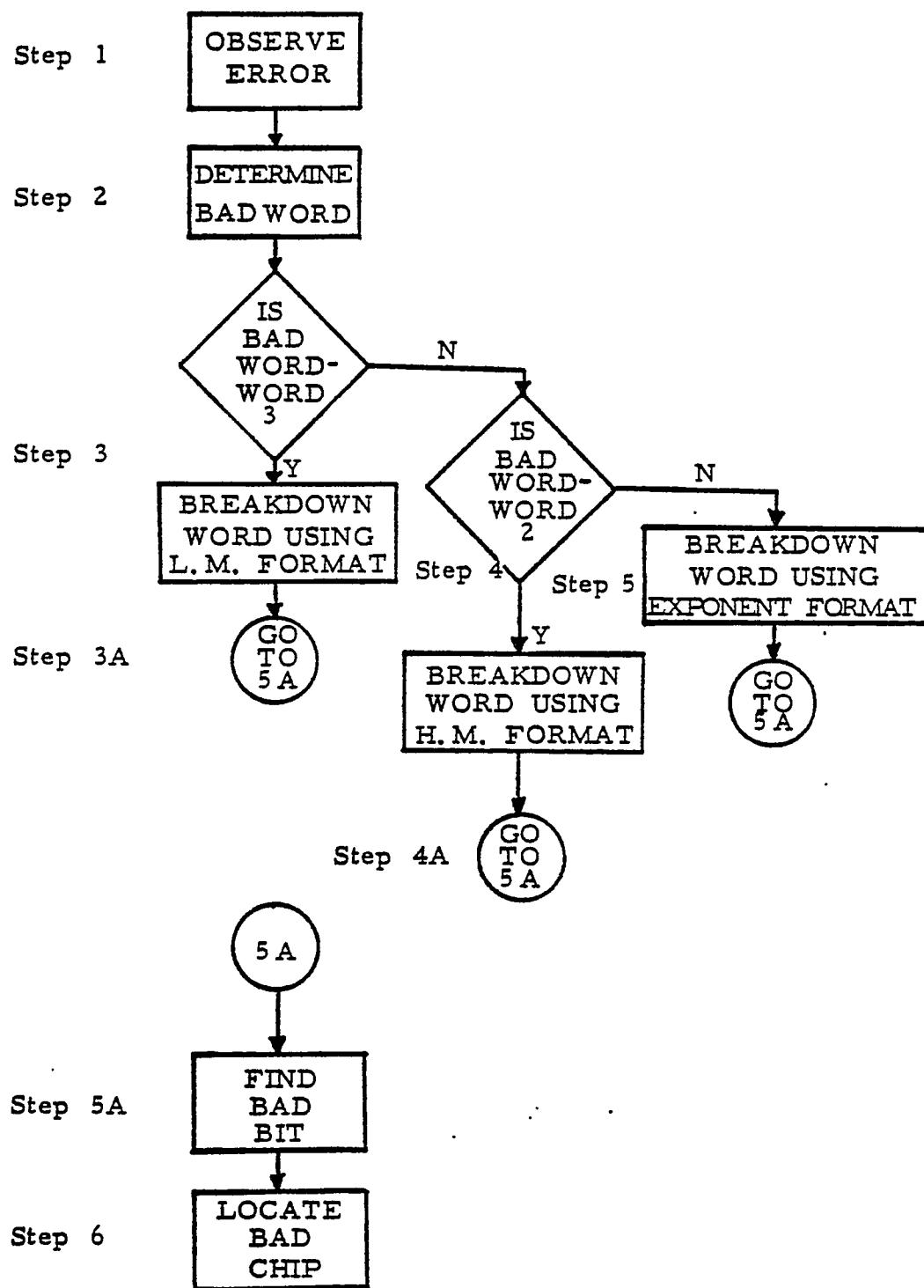
P.S. Bits K LMN OPQ RST UVW XYZ K LMN OPQ RST UVW XYZ

5. Find the octal number in the expected and actual word that is different. From the breakdown of those numbers determine which P.S. Bit is failing.
6. Determine the actual failing chip from the following table using results from Step 3 or 3A and Step 5. Replace chip. Rerun test to insure fix.

	<u>PSW0</u>	<u>PSW1</u>	<u>PSW2</u>	<u>PSW3</u>		<u>PSW0</u>	<u>PSW1</u>	<u>PSW2</u>	<u>PSW3</u>
K	A1	C1	E1	G1	S	A9	C9	E9	G9
L	A2	C2	E2	G2	T	A10	C10	E10	G10
M	A3	C3	E3	G3	U	A11	C11	E11	G11
N	A4	C4	E4	G4	V	A12	C12	E12	G12
O	A5	C5	E5	G5	W	A13	C13	E13	G13
P	A6	C6	E6	G6	X	A14	C14	E14	G14
Q	A7	C7	E7	G7	Y	A15	C15	E15	G15
R	A8	C8	E8	G8	Z	A16	C16	E16	G16

GE MEDICAL SYSTEMS INSTITUTE

FLOWCHART OF TMRAM TROUBLESHOOTING GUIDE



GE MEDICAL SYSTEMS INSTITUTE

TROUBLESHOOTING GUIDE FOR TMRAM MEMORY CHIP REPLACEMENT

1. Observe error message print-out.

MA AAAAAA

E ooeeee ooeeee eeeeeee A ooaaaa ooaaaa aaaaaaa
word 1 word 2 word 3 word 1 word 2 word 3

2. Compare the expected and actual results of print-out.
Find the WORD that is different.

3. If word 1 or 2 is the word that is different, go to Step 4.
Break word 3 down using the following low mantissa format:

E	e	e	e	e	e	e	A	a	a	a	a	a	a	a
Low Mantissa Bits	L	LLL	LLL	LLL	LLL	LLL	L	LLL	LLL	LLL	LLL	LLL	LLL	LLL
	24	25-27	28-30	31-33	34-36	37-39	24	25-27	28-30	31-33	34-36	37-39		

- 3A. Go to Step (5A).

4. If word 1 is the word that is different, go to Step 5.
Break word 2 down using the following high mantissa format.

E	x	x	e	e	e	A	x	x	a	a	a	a	a	a
High Mantissa Bits	X	XXX	HHH	HHH	HHH	HHH	X	XXX	HHH	HHH	HHH	HHH	HHH	HHH
		12-14	15-17	18-20	21-23		12-14	15-17	18-20	21-23				

Note: The X's represent 0's not used in the break down.

- 4A. Go to Step 5A.

5. Break word 1 down using the following exponent format.

Exponent Bits	X	XXX	XXE	EEE	EEE	E	X	XXX	XXE	EEE	EEE	EEE		
	2	3-5	6-8	9-11		2	3-5	6-8	9-11					

Note: The X's represent 0's not used in the break down.

- 5A. Find the octal number in the expected and actual word that is different.
From the binary breakdown of those numbers, determine which bit is different this is the failing bit.

6. Knowing the failing bit of Step 5A and the memory address of Step 1,
locate the failing chip using the TMRAM memory matrix issued on
the next page and replace it. Rerun test to insure fix.

GE MEDICAL SYSTEMS INSTITUTE

FAILING
TM RAM MEMORY MATRIX
ADDRESS

RANGE ==>	0-1777	2000-	4000-	6000-	10000-	12000-
(MA = XXXX)	3777	5777	7777	11777	13777	

FAILING
BIT POSITION

02	1-G2	1-E2	1-C2	1-A2	2-G2	2-E2
03	1-H2	1-F2	1-D2	1-B2	2-H2	2-F2
04	1-G3	1-E3	1-C3	1-A3	2-G3	2-E3
05	1-H3	1-F3	1-D3	1-B3	2-H3	2-F3
06	1-G4	1-E4	1-C4	1-A4	2-G4	2-E4
07	1-H4	1-F4	1-D4	1-B4	2-H4	2-F4
08	1-G5	1-E5	1-C5	1-A5	2-G5	2-E5
09	1-H5	1-F5	1-D5	1-B5	2-H5	2-F5
10	1-G6	1-E6	1-C6	1-A6	2-G6	2-E6
11	1-H6	1-F6	1-D6	1-B6	2-H6	2-F6
12	1-G7	1-E7	1-C7	1-A7	2-G7	2-E7
13	1-H7	1-F7	1-D7	1-B7	2-H7	2-F7
14	1-G8	1-E8	1-C8	1-A8	2-G8	2-E8
15	1-H8	1-F8	1-D8	1-B8	2-H8	2-F8
16	1-G9	1-E9	1-C9	1-A9	2-G9	2-E9
17	1-H9	1-F9	1-D9	1-B9	2-H9	2-F9
18	1-G10	1-E10	1-C10	1-A10	2-G10	2-E10
19	1-H10	1-F10	1-D10	1-B10	2-H10	2-F10
20	1-G11	1-E11	1-C11	1-A11	2-G11	2-E11
21	1-H11	1-F11	1-D11	1-B11	2-H11	2-F11
22	1-G12	1-E12	1-C12	1-A12	2-G12	2-E12
23	1-H12	1-F12	1-D12	1-B12	2-H12	2-F12
24	1-G13	1-E13	1-C13	1-A13	2-G13	2-E13
25	1-H13	1-F13	1-D13	1-B13	2-H13	2-F13
26	1-G14	1-E14	1-C14	1-A14	2-G14	2-E14
27	1-H14	1-F14	1-D14	1-B14	2-H14	2-F14
28	1-G15	1-E15	1-C15	1-A15	2-G15	2-E15
29	1-H15	1-F15	1-D15	1-B15	2-H15	2-F15
30	1-G16	1-E16	1-C16	1-A16	2-G16	2-E16
31	1-H16	1-F16	1-D16	1-B16	2-H16	2-F16
32	1-G17	1-E17	1-C17	1-A17	2-G17	2-E17
33	1-H17	1-F17	1-D17	1-B17	2-H17	2-F17
34	1-G18	1-E18	1-C18	1-A18	2-G18	2-E18
35	1-H18	1-F18	1-D18	1-B18	2-H18	2-F18
36	1-G19	1-E19	1-C19	1-A19	2-G19	2-E19
37	1-H19	1-F19	1-D19	1-B19	2-H19	2-F19
38	1-G20	1-E20	1-C20	1-A20	2-G20	2-E20
39	1-H20	1-F20	1-D20	1-B20	2-H20	2-F20

BITS 2-11 are BITS 12-23 are BITS 24-30 are

Exponent High Mantissa Low Mantissa

XXEEEE XXHHHH LLLLLL ON FPS ERROR TYPEOUT.....

GE MEDICAL SYSTEMS INSTITUTE

The matrix below shows which diagnostic should be run per board type. Before a problem is declared "not found" by an AP diagnostic, the diagnostic must be run 10 minutes.

One Pass D.O.A. Each Designated (X)	APTEST	APPATI	APANTH	PCPT	FURIT	THROU	THRAU	PSTEST	INPUT 3	KID 511/269	E2D 511/269
Diagnostic - 10 Minutes Each											
200	X X	X X			X X						
201	X X	X X			X X						
202	X X	X X			X X						
203	X X	X X			X X						
204	X X	X X			X X						
205	X X	X X			X X						
206	X X	X X			X X						
207	X X	X X			X X						
208	X X	X X			X X						
209	X X	X X			X X						
210	X X	X X			X X						
211	X X	X X			X X						
212	X X	X X			X X						
213	X X	X X			X X						
214	X X	X X			X X						
215	X X	X X			X X						
216	X X	X X			X X						
217	X X	X X			X X						
218	X X	X X			X X						
219	X X	X X			X X						
220	X X	X X			X X						
221	X X	X X			X X						
222	X X	X X			X X						
223	X X	X X			X X						
224	X X	X X			X X						
225	X X	X X			X X						
226	X X	X X			X X						
227	X X	X X			X X						
228	X X	X X			X X						
229	X X	X X			X X						
230	X X	X X			X X						
231	X X	X X			X X						
232	X X	X X			X X						
233	X X	X X			X X						
234	X X	X X			X X						
235	X X	X X			X X						
236	X X	X X			X X						
237	X X	X X			X X						
238	X X	X X			X X						
239	X X	X X			X X						
240	X X	X X			X X						
241	X X	X X			X X						
242	X X	X X			X X						
243	X X	X X			X X						
244	X X	X X			X X						
245	X X	X X			X X						
246	X X	X X			X X						
247	X X	X X			X X						
248	X X	X X			X X						
249	X X	X X			X X						
250	X X	X X			X X						
251	X X	X X			X X						
252	X X	X X			X X						
253	X X	X X			X X						
254	X X	X X			X X						
255	X X	X X			X X						
256	X X	X X			X X						
257	X X	X X			X X						
258	X X	X X			X X						
259	X X	X X			X X						
260	X X	X X			X X						
261	X X	X X			X X						
262	X X	X X			X X						
263	X X	X X			X X						
264	X X	X X			X X						
265	X X	X X			X X						
266	X X	X X			X X						
267	X X	X X			X X						
268	X X	X X			X X						
269	X X	X X			X X						
270	X X	X X			X X						
271	X X	X X			X X						
272	X X	X X			X X						
273	X X	X X			X X						
274	X X	X X			X X						
275	X X	X X			X X						
276	X X	X X			X X						
277	X X	X X			X X						
278	X X	X X			X X						
279	X X	X X			X X						
280	X X	X X			X X						

GE MEDICAL SYSTEMS INSTITUTE

AP TROUBLESHOOTING USING TRIGGER SIGNALS

Problem: When using a scope to look at data in the array processor, it becomes increasingly important to know that the data being viewed is the data that you wish to see. As you progress further into the diagnostics, the need for a good reference trigger becomes almost mandatory.

Solution: Provided are some good reference signals that can be used as trigger signals and which diagnostics to use them with.

<u>Trigger Signal</u>	<u>Location</u>	<u>Diagnostic</u>
HD2REG	BRD219 PN A95	APTEST CODES 0-5
PCYLL1*	BRD210 PN B40	APTEST CODES 6-31
RUN*	BRD210 PN B38	APPATH CODES ALL
ABORT*	BRD210 PN B5	APTEST CODES 32-47

Note: ABORT* is generated by typing an "LDIE" when an error is encountered.

Note: All signals with a * are low true signals and must be synced using negative slope.

**BACKPLANE
SIGNAL
GLOSSARY**

FAST RECONSTRUCT PROCESSOR

AP120B

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY

A1CLK	Clock for A1 Register of FA
A2CLK	Clock for A2 Register of FA
CBCLK1	Clock for Bd. 210, CB-1
CBCLK2	Clock for Bd. 211, CB-2
DPLCLK	Clock for Bd. 200L DPAD
DPLCLK*	Inverted clock for Bd. 200L DPAD
DPRCLK	Clock for Bd. 200R DPAD
DPRCLK*	Inverted clock for Bd. 200R DPAD
FACLK2	Clock for Bd. 204, FADD2
FACLK3	Clock for Bd. 205, FADD3
FMCLKA	Clock for Bd. 206, FMULA
FMCLKB	Clock for Bd. 207, FMULB
FMCLKC	Clock for Bd. 208, FMULC
IOCLK	Clock for Interface Bd.
M1CLK	Clock for M1 Register of FM
M2CLK	Clock for M2 Register of FM
MCLK	Clock for Main Data Memory
MDCLK*	Inverted clock for MDREG Bd. 202
MICLK	Clock for MIREG Bd. 213
PNLCLK	Clock for Panel Logic on EPAN. Bd. 214
SPCLK	Clock for SPAD Bd. 201
T69	Clock delayed by 69ns for MD Timing Bd. 210
TMCLK	Clock for TMREG Bd. 209
WRT*	Low true write pulse used to write PS, and the Subroutine Return Stack

GE MEDICAL SYSTEMS INSTITUTE

AP-120E BACKPLANE SIGNAL GLOSSARY - Continued

WRTL*	Write pulse for DPL
WRTR*	Write pulse for DPR
GND	Extra Grounds not included in the standard set provided by the Motherboard
A1CLKD*	A1 Clock Data (low true) causes A1CLK
A1EBS02* to A1EBS11*	A1 Exponent Bus
A1MBS00* to A1MBS27*	A1 Mantissa bus
A2CLKD*	A2 Clock Data causes A2CLK
A2EBS02* to A2EBS11*	A2 Exponent Bus
A2M000*	A2 Register Bit 00 (sign bit)
A2MBS00* to A2MBS27*	A2 Mantissa Bus
ABORT*	Internal System Reset Line
B0CLK	Byte 0 Clock to FMT Bd.
B1CLK	Byte 1 Clock to FMT Bd.
B2CLK	Byte 2 Clock to FMT Bd.
B3CLK	Byte 3 Clock to FMT Bd.
B2I0	FMT Buffer to I/O Bus Enable
BH2HD	FMT Buffer High to Host Data Enable
BL2HD	FMT Buffer Low to Host Data Enable
BS2A1	A1BS to A1 input select line
BS2A2	A2BS to A2 input select line
BS2M1	M1BS to M1 select line
BS2M2	M2BS to M2 select line
BUF2CLK	FMT Buffer #2 Clock

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY - Continued

CAP2PNL*	Control Buffer AP to PNLBUS
CB2A1E*	Control Buffer to ALEBS Enable
CB2A2E*	Control Buffer to A2EBS Enable
CB2A2M*	Control Buffer to A2MBS Enable
CBCLKE*	Control Buffer Clock Enable

(BOTTOM HALF INTENTIONALLY BLANK)

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY - Continued

DA0 to DA3	SPAD Destination Address Bits
DALD*	Device Address Load Enable
DE07 to DE11	Floating Adder
DE07* and DE08* DE10A and DE11A	Delta Exponent Bits for shift of mantissa of smaller argument
DECIMATE*	Bit-Reverse enable to SPAD source
DMA00* to DMA15*	Direct Memory Address to MD from Host Interface
DMASAME	DMA Bank Address same as last bank
DP2A1E	DPAD to A1BS Enable
DP2A2E	DPAD to A2BS Enable
DP2DPE	DPAD to DPBS Enable
DP2M1E	DPAD to MIBS Enable
DP2M2E	DPAD to M2BS Enable
DPA2PNL*	DPAD Address to Panel Bus Enable
DPALD*A	DPAD Address Load Enable
DPBS2PSI*	DPBS to Program Source Input Select
DPE2PNL	DPBS Exponent to Panel Bus Enable
DPEBS02* to DPEBS11*	DPAD Exponent Bus Bits
DPH2PNL	DPBS HMAN to Panel Bus Enable
DPL2PNL	DPBS LMAN to Panel Bus Enable
DPMBS00* to DPMBS27*	DPAD Mantissa Bus
EOVCRY	FA Exponent overflow carry
EOVG*	FA Exponent overflow carry generated

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY - Continued

EOVP*	FA Exponent overflow carry propagate
EX2PNL*	Exit to PNLBUS
EXIA	Exit Input Select A
EXIB	Exit Input Select B
EXP*	Exponent Write Select
FAA*	FA answer select A
FAB*	FA answer select B
FACIN*	Floating Adder Carry Input
FADD*	Floating Add microinstruction decode
FAE00* to FAE11*	Floating Adder Exponent output
FAM00* to FAM27*	Floating Adder Mantissa output
FANEG*	FA result negative
FAOVF*	FA result overflow
FAS0 to FAS3	FA ALU mode select controls
FAUNF*	FA result underflow
FAZRO*	FA result = zero
FFTQ	FFT mode flag
FL07*A to FL09*A	FA normalization shift count
FL10* to FL11*	(Float number)
FLAG0 to FLAG3	Program selectable Flags
FME02* to FME11*	Floating Multiplier Exponent output
FMM00* to FMM27*	FM Mantissa output
FMOVF*	FM result overflow
FMUL*	Floating Multiply micro-instruction decode
FMUL*A	Floating Multiply micro-instruction decode

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY - Continued

FMUNF*	FM Result underflow
FN2HD*	Function Register to Host Data Enable
FRSGN*	FA Force Sign
FRSGNQ*	Force Sign Latch
FSCALE*	Floating Scale
FSCALEQ	Floating Scale Latch
FSCALEQ*	Floating Scale Latch
FSM(-1)* to FSM30*	Floating Summer Mantissa bits (Connection from Stage 1 to State 2 of FA)
HD00 to HD15	Host Data Bus
HD2DP	Host Data to DPBS Enable
HRSET*	Host Reset
I+H09	IO OR HOST Data Bit 09
I+H10	IO OR HOST Data Bit 10
I+H13	IO OR HOST Data Bit 13
I+H14	IO OR HOST Data Bit 14
IFFTQ*	Inverse FFT Flag
IN	Decode of IO Input instruction
INTEN	Interrupt Enable
INTR*	Interrupt Request
INTRQ	Interrupt Request Latch
IO00* to IO39*	IO BUS
IOACK*	IO Acknowledge
IODRDY*	IO Data Ready

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY - Continued

IODRDYQ	IO Data Ready Latch
IODRDYQ*	IO Data Ready Latch
IOSPMD*	IO Spin if MD Busy
LT2HD*	LITES to Host Data Enable
M1EBS02* to M1EBS11*	M1 Exponent Bus
M1MBS00* to M1MBS27*	M1 Mantissa Bus
MIR00Q* to M1R27Q*	M1 Register outputs
M2EBS02* to M2EBS11*	M2 Exponent Bus
M2MBS00* to M2MBS27*	M2 Mantissa Bus
M2R02Q* to M2R27Q*	M2 Register outputs
MA00* to MA15*	Memory Address (MD)
MA2PNL	Memory Address to Panel Bus Enable
MACE*	Memory Address Count Enable
MAINC*	Memory Address Increment Select
MALD*	Memory Address Load Enable
MALD*A	Memory Address Load Enable
MAN*	Mantissa Write Select
MANOV	Mantissa overflow
MASAME	MA Bank same as last Bank
MD02* to MD39*	Main Data outputs
MD2A2	MD to A2BS Enable
MD2DP	MD to DPBS Enable
MD2M2	MD to M2BS Enable
MDCA0	MD Cycle Acknowledge 0 (refresh)
MDCA1	MD Cycle Acknowledge 1 (Host interface)

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY - Continued

MDCA2	MD Cycle Acknowledge 2
MDCA3	MD Cycle Acknowledge 3 (AP Internal)
MDCLK*	MD Register Clock Enable
MDCR1Q*	MD Cycle Request 1
MDCR2*	MD Cycle Request 2
MDCR3*	MD Cycle Request 3
MDEXP	MD Exponent Write Enable
MDHMAN	MD high Mantissa Write Enable
MDI02 to MDI39	MD input bus
MDINA*	MD cycle initiate
MDLMAN	MD Low Mantissa Write Enable
MDWRT*	MD Write Enable
MDWRT3	MD Write Request 3
MIA	MD Input Select A
MIB	MD Input Select B
MICLKE*	MD Input Clock Enable
OUT*	IO OUT micro-instruction decode
OVFL*	Overflow status
PCYL1*	Panel cycle 1
PCYL2*	Panel cycle 2
PNL00* to PNL15*	Panel Bus
PNL2DP	Panel Bus to DPBS Enable
PNL2HOST*	Panel Bus to Host (Lites Load Enable)
PNL2MD*	Panel Bus to MD write request
PPA01* to PPA26*	FM Partial Product outputs of Array A
PPA27Q* to PPA30Q*	
PPA31* to PPA52*	

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY - Continued

PPAUSE	Panel Pause from CB-1 (210)
PPB(-1)* to PPB24*	FM Partial Product outputs of
PPB25Q* to PPB28Q*	ARRAY B
PPB29* to PPB	
PS00* to PS63*	Program Source outputs
PS02PNL*	PS Word 0 to Panel Bus Enable
PS0WRT	PS Word 0 Write Strobe
PS12PNL*	PS Word 1 to PNL Bus Enable
PS1WRT	PS Word 1 Write Strobe
PS22PNL*	PS Word 2 to PNL Bus Enable
PS2WRT	PS Word 2 Write Strobe
PS32PNL*	PS Word 3 to PNL Bus Enable
PS3WRT	PS Word 3 write Strobe
PSA04* to PSA15*	Program Source Address
PSA2PNL	PSA to PNL Bus Enable
PSAAD	PSA Select A Data
PSABD	PSA Select B Data
PSACD	PSA Select C Data
PSACLKE*	PSA Clock Enable
PSAZRO	PSA = Zero, PS Disable
PSH2DP*	PS High to DPBS Enable
PSI00 to PSI31	PS Input Bus
PSL2DP	PS Low to DPBS Enable
REFSYNC*	Refresh Sync
RUN*	AP-120B Running
S+C2*	Step OR Continue Cycle 2 (panel function)

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY - Continued

SA0 to SA3	SPAD Source Address
SAMEX	DPX Read and Wrtie Addresses equal
SAMEY	DPY Read and Wrtie Addresses equal
SC00*	Sign bit out of FA Stage 1 mantissa scaler
SCIN	FA Scaler inhibit
SELA1A	Select A1 as larger input to FA
SELA2	Select A2 as larger input to FA
SFWE*	SPAD Function Write Enable
SHS0	SPFN Shift Select 0
SHS1	SPFN Shift Select 1
SIWE*	SPAD Input Write Enable
SM2TC*	Sign Magnitude to two's complement
SNSA	IO sense A
SP+DP00* to SP+DP15*	SPFN OR DPBS Bus
SP2ADDR	SPFN to Address (SP + DP Bus) select
SP2DP	SPFN to DPBS Enable
SP2PNL	SPFN to PNL Bus Enable
SPA2PNL*	SPAD Address to PNL Bus Enable
SPALD*	SPAD Destination Address Load Enable
SPCIN	SPFN Carry Input
SPFN00*	SPFN Sign Bit
SPFNCRY*	SPFN Carry Output
SPILD*	SPAD INPUT LOAD Enable
SPIN*	Micro-processor SPIN (Hangs on current instruction)

GE MEDICAL SYSTEMS INSTITUTE

AP-120B BACKPLANE SIGNAL GLOSSARY - Continued

SPIOD0	SPIN if IODRDY DATA=0
SPM	SPFN Mode
SPS0 to SPS3	SPFN ALU Controls
SPZED	SPFN = zero
SR2HD*	Switch Register to Host Data Enable
SR2PNL	Switch Register to Panel Bus Enable
SRACE*	Subroutine Return Address Count Enable
SRADEC*	Subroutine Return Address Decrement Select
SRAOVD*	Subroutine Return Address Overflow Data
SRSWE*	Subroutine Return Stack Write Enable
STA2PNL	AP STATUS to PNL Bus Enable
STALD*	APSTATUS Load Enable
TM02* to TM39*	Table Memory Outputs
TM2A1	TM to A1BS Enable
TM2DP	TM to DPBS Enable
TM2M1	TM to M2BS Enable
TMA00 to TMA15	Table Memory Address
TMA2PNL	TMA to PNL Bus Enable
TMACE*	TMA Count Enable
TMADEC*	TMA Decrement Select
TMALD*	TMA Load Enable
TMALD*A	TMA Load Enable
TMINH	Table Memory Inhibit
TMNEG*	Table Memory Negate

GE MEDICAL SYSTEMS INSTITUTE

AP -120B BACKPLANE SIGNAL GLOSSARY - Continued

TRUNC*	FA Truncate
TRUNCQ	FA Truncate Latch
TRUNCQ*	FA Truncate Latch
TSPIN	True Spin
UNFL*	Underflow Status
USECB*	Use Control Buffer Bits 48 to 63 as a Value
USEPSA*	Use PSAQ as Source for PSA
WRTEXP	MD Exponent Write Enable
WRTHM	HMAN Write Enable
WRTLM	LMAN Write Enable
X ₀₁ to X ₀₅ X _{02A} to X _{05A}	DPX Address
XECLKE*	DPX Exponent Clock Enable
XHMCLKE*	DPX HMAN Clock Enable
XIA	DPX Input Select A
XIB	DPX Input Select B
XLMCLKE*	DPX LMAN Clock Enable
Y ₀₁ to Y ₀₅ Y _{02A} to Y _{05A}	DPY Address
Y2A1	DPY to A1BS Select
Y2A2	DPY to A2BS Select
Y2DP	DPY to DPBS Select
Y2M1	DPY to M1BS Select
Y2M2	DPY to M2BS Select
YECLKE*	DPY Exponent Clock Enable
YHMCLKE*	DPY HMAN Clock Enable

OP120B Class Outline

I. Introductions)

A. Myself

B. Students)

1. Experience on C111
2. Experience on OP120B

C. Handout manuals)

1. OP120 Maintenance Manuals
2. OP120 Schematics)
3. OP120 Handouts)
4. Cookbook odds
5. MILP e

II. How does OP fit into system

A. Where does it go

1. own cabinet
2. on computer cabinet

overheads used:

Handouts Reader

located on
either
side of
computer
bay

\$1200

optional

located
where
drives
are in
computer
bay

\$1200

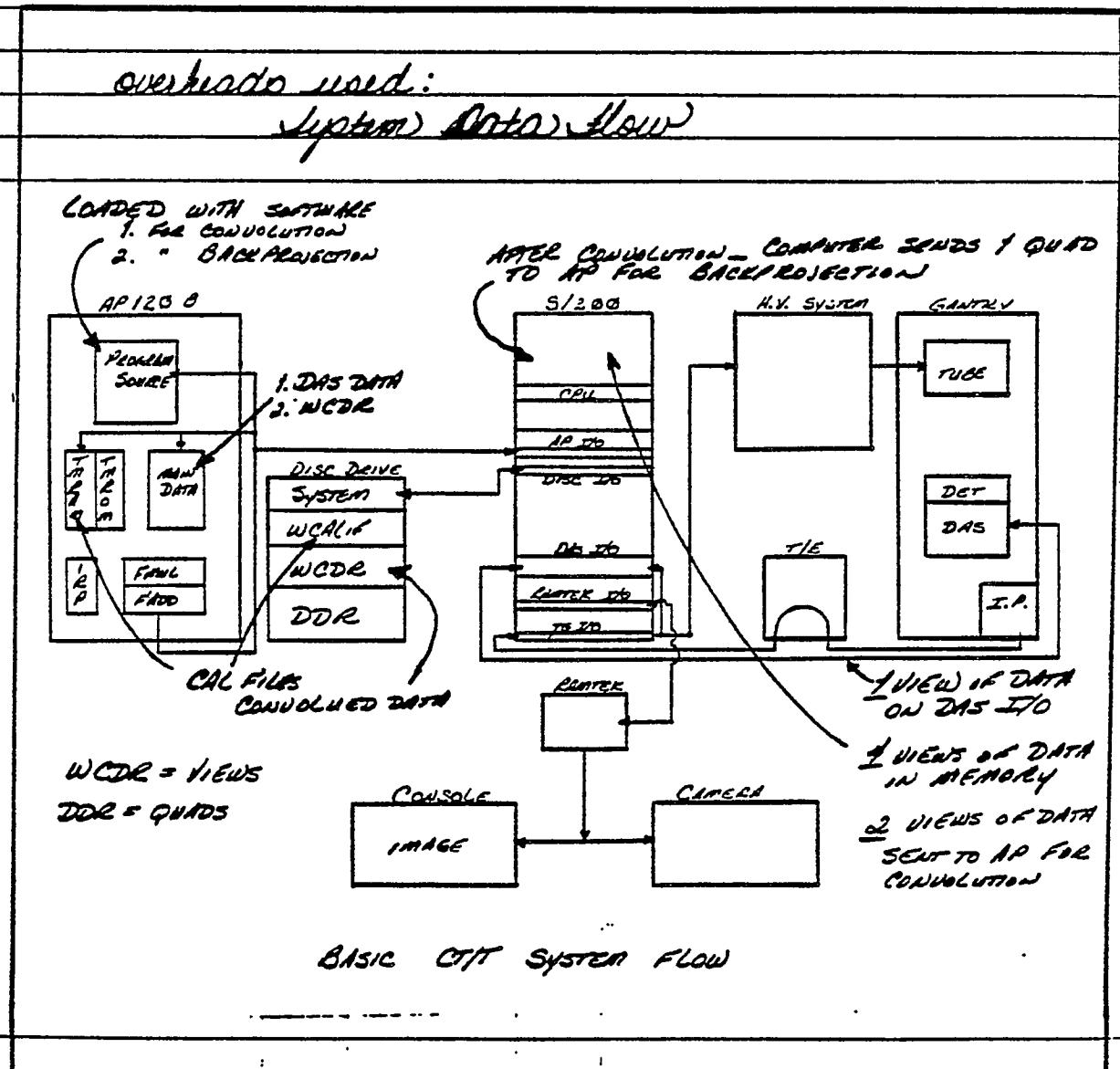
Special Order Engineering, instructions needed!

a what does it do

1. Convolution
2. Backprojection

overheads used:

(system) Data flow



III. Basic Structure

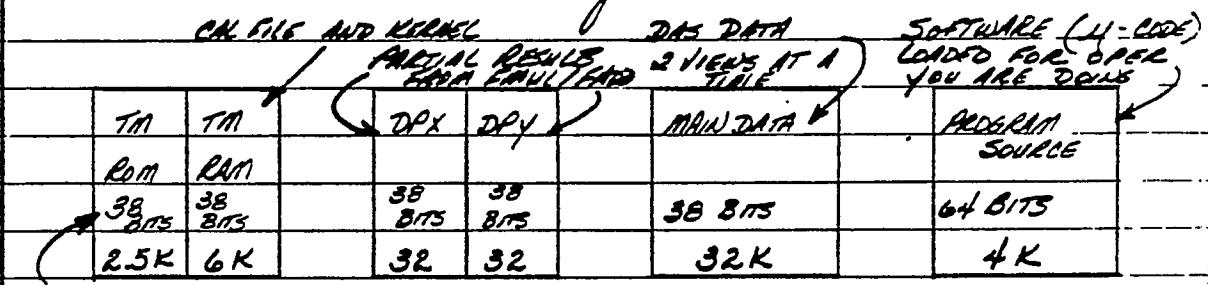
A. What is it made up of

1. three basic processors
- a. SPAD integer processor

- (3)
- b. FADD floating point adder
 c. FMUL floating point multiplier
 2. five basic memories
 a. Program source software (4-code)
 b. Mdin Data DOS Data / WCDR
 c. DPX, DPY scratch memory,
 d. TM RAM word / kernels
 e. TM ROM cosine constants / constants

overheads used:

OP Block Diagram



COSINE CONSTANTS = 3K
AND CONSTANTS = .5K

	FMUL	FADD	SAD
3 clk cycles needed to have the result	3 STAGE	2 STAGE	1 STAGE
	38 bits	38 bits	16 bits

Annotations: A bracket labeled "2 clk cycles needed" points to the FADD row. A bracket labeled "only compute address info. 1 clk cycle needed" points to the SAD row. A bracket labeled "(ALL, SHIFTER, 16 REG)" points to the SAD row.

3. Three types of buses

a. Single Source - Multi Destination

b. Multi Source - Multi Destination

c. Multi Source - Single Destination

overheads used:

OP Block Diagram

a. FAnn* → Single Source - Multi Dest.
FMnn* →

b. DPBSnn* → Multi Source - Multi Dest.

c. A1nn* → Multi Source - Single Dest.
M2nn* →

3. How do we talk to it

1. OP Device Codes

2. Unit Addresses in the OP

overheads used:

Appendix C pg. 1

Appendix C pg. 2

Unit Address Block

* add memory ADDRESS 5

c. What is a floating point number

1. typical

1.234×10^6

mantissa → base → exponent

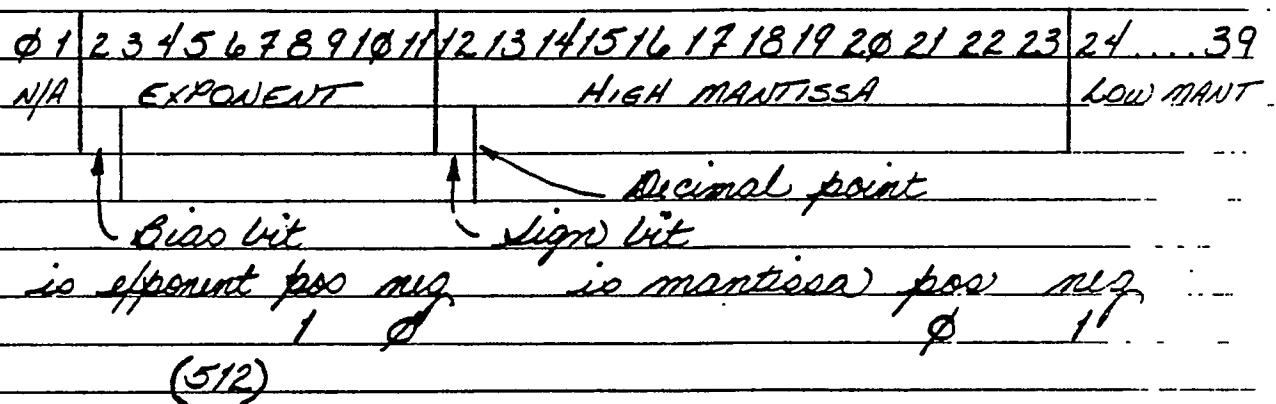
3

2. $\frac{1}{2} \times 10^0$ - 32 bit word
 0.0001234×10^{63}

leading zeros cause lost significance

3. AP - 38 bit word
 12345×2^{511}

4. Break down the 38 bit word



5. How do these 38 bit words appear in AP

a. In memory,

$E\phi 2 \rightarrow 11 \quad m12 \rightarrow 39$

b. In arithmetic units

$E\phi 2 \rightarrow 11 \quad m0\phi \rightarrow 27$

c. Common usage

$$\left. \begin{array}{l} E\phi 2 \rightarrow 11 \\ Hm\phi \rightarrow 11 \\ Lm 12 \rightarrow 27 \end{array} \right\} \text{some} \left\{ \begin{array}{l} E\phi 2 \rightarrow 11 \\ Hm 12 \rightarrow 23 \\ Lm 24 \rightarrow 39 \end{array} \right\}$$

d. the mantissa) is in 2's complement form
always

e. How does the AP sum

1. Control word (4-code)

a. 64 bit word (program source)

b. Executes in parallel up to 16 instructions
every 16.7 ns

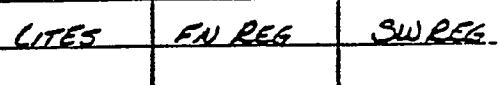
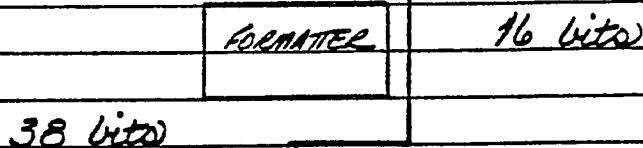
c. is loaded twice during normal scan

2. All clocks and going of dots controlled
by the 4-code.

f. Divide the AP into two (2) parts

overheads used:

OP Block Diagram



SPAD

ADD R REGISTERS

IV. Basic Operation

A. The 280 Board

1. Basic I/O Controller

2. Located in Slot 15 of Expansion Chassis

3. Controls data flow to and from CPU / AP

4. Connects to the AP120 by Mem bus

B. The 214 Board

1. Virtual Front Panel

a. Function Register

b. Switch Register (data)

c. Panel File Register

2. Directs data to the 219 Board (H.R.O.F.)

3. Directs data to the 236 Board (P.S.)

C. The 226/227 Boards

1. Formatter

a. Converts 32 bit data to 38 bit data

b. Converts 38 bit data to 32 bit data

2. Data is inputted thru the Formatter (DMA)

D. The 236 Board

1. Program Source

a. 1K by 64 bit Static Ram (93425A/2125)

b. 4K total stored

2. Bootstrap loaded thru VFP operation

3 instructions (locations)

3. Bootstrap loads P.S. thru DMA operation

a. loaded once for Convolution

b. loaded once for Backprojection

full programs

4. When loaded thru VFP

a. takes 4 cycles thru Paknn* bus to load

b. 0-15 one cycle / 16-31 one cycle / 32-47 / 48-63

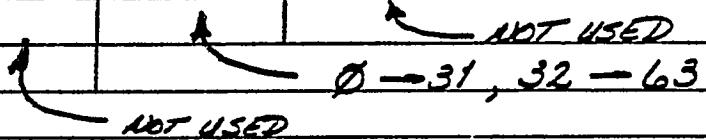
c. this is a time consuming process

5. When loaded thru DMA

a takes 2 cycles thru formatter

b formatter assembles two (2) 16 bit words
and converts to one (1) 38 bit word

c $\emptyset \rightarrow 34 \rightarrow 35, 36 \rightarrow 39$



d. DMA process used mostly, much faster

e. Addressing P.I.

a. interrupt Generator (201) used for address generation

b. Address only 12 bits long = 4K

c. 201 board has 16 registers, each 16 bits

V. How to run diagnostics

A. All diagnostics on the A/D tape

1. Boot A/D tape like always

2. type load test

d. Commands used to function

1 R = reset all go traps

2 W = wait on error (last)

3 E = execute test

b. Control C stops execution

c. A600B ; - restart Program Counter at location
600 (clear RWE)

d. A77377B ; - restart A/D Handler so you can
select the next test

e. LE ; = loop on errors

f. LDE ; = loop on error/no printout (for scoping)

g. to clear function specified "subset"

h. Patterns used are:

• all \emptyset s

• all 1s

• sliding 1 against \emptyset s

• random

3. Run diagnostics in this order

- AP Test
- AP Pack
- AP Arch
- PS Test
- IM ROM
- IM ROM
- PERIOD
- E1D XXX
- E2D XXX
- FCFT
- FDFT

4. If you run "Accept" from A101

a. Code 5,

1. all Eclipse tests will function
2. Proceed to AP tests

a. If you have a 511 (x2) will halt on 289

b. You must go around with a "CTRL C".

b. Code 6 / 7)

- All other AP tests will function

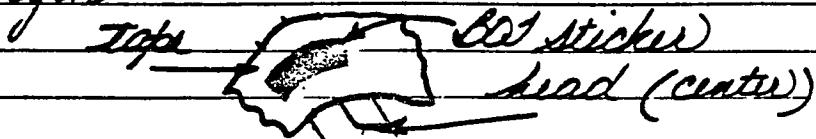
5. If you run "Run" or "Full" from A101

? same as above

for stand-alone system:

- boot tape from 000070
- type only test you want
- all other commands the same

* type must be a BOT (manually loaded) and tension on tape before functioning, can begin

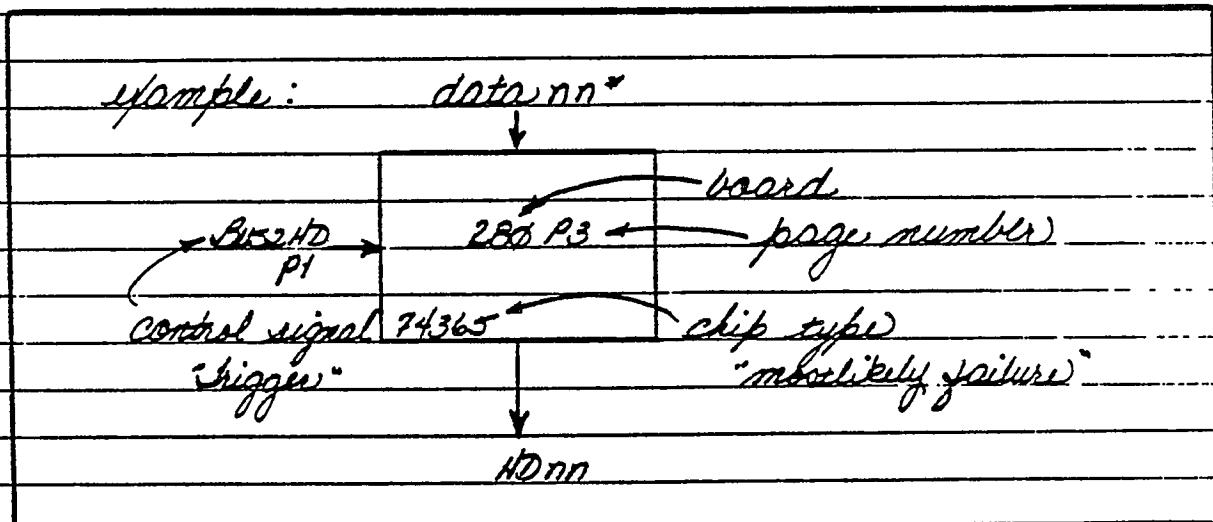


(10)

II AP Test

A AP test is broken into 10 individual tests

1. We will step thru these tests to understand the proper troubleshooting of failures
2. Error Codes or messages will be decoded.
3. The use of fixed flaws will also help
- B. Fixed flaws apply to error codes 0 - 31
1. How do you read them



2. When do you use them

a. when isolating your failures

b. When scraping

c. Test 1

1. Uses VFP operation

2. Error Codes 0 - 5

3. Triggered on Control Signal for slot register under test

d. Test 2

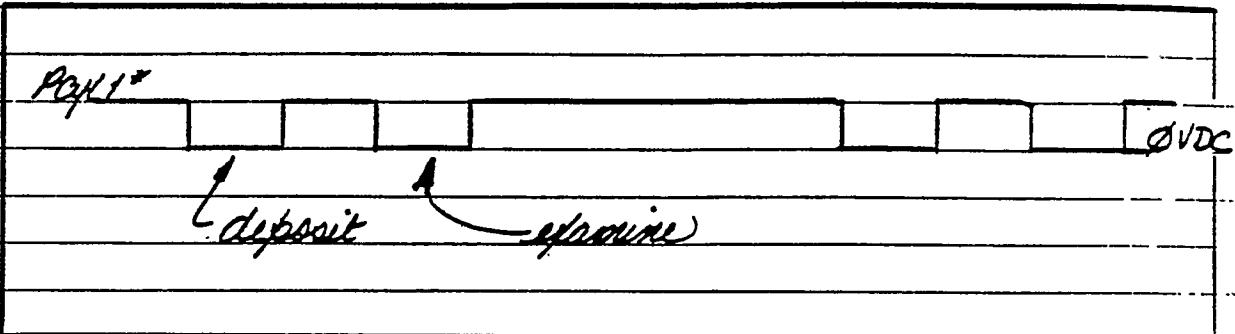
1. Uses VFP operation

2. Error Codes 6 - 26

3. Triggered on Control Signal for slot register under test

91

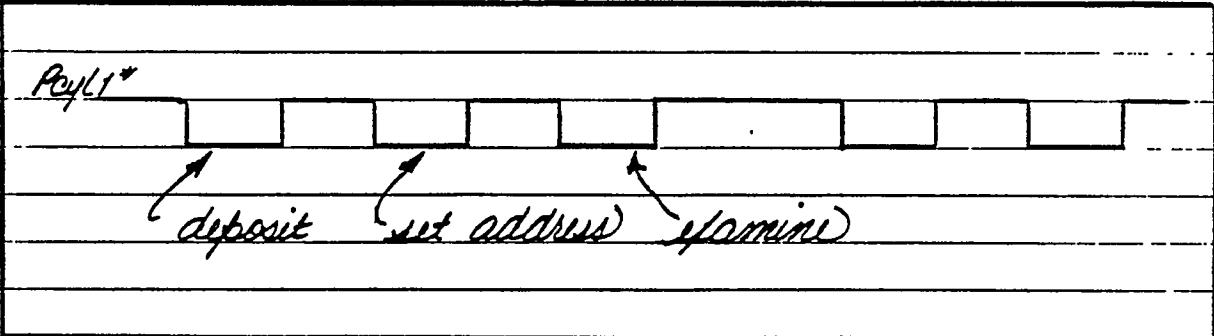
Pey1*



E. Test 2A

1. Uses VFP operation
2. Erase Codes 27 - 31
3. Triggered on Control signal for that section of word under test

Pey1*

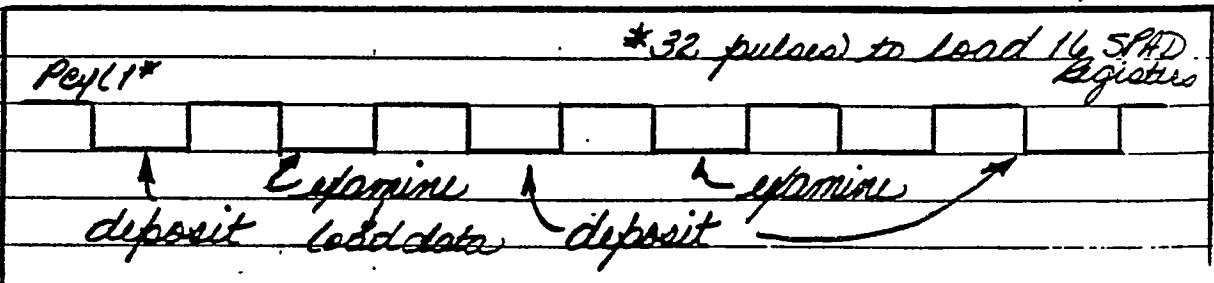


F. Test 3

1. Basic Address Test
2. Loads all locations first with their address (then erases all locations)
3. Erase Codes 32 - 47
 - codes 45 - 47 for Main Data Memory

Pey1*

* 32 pulses to load 16 SFD registers



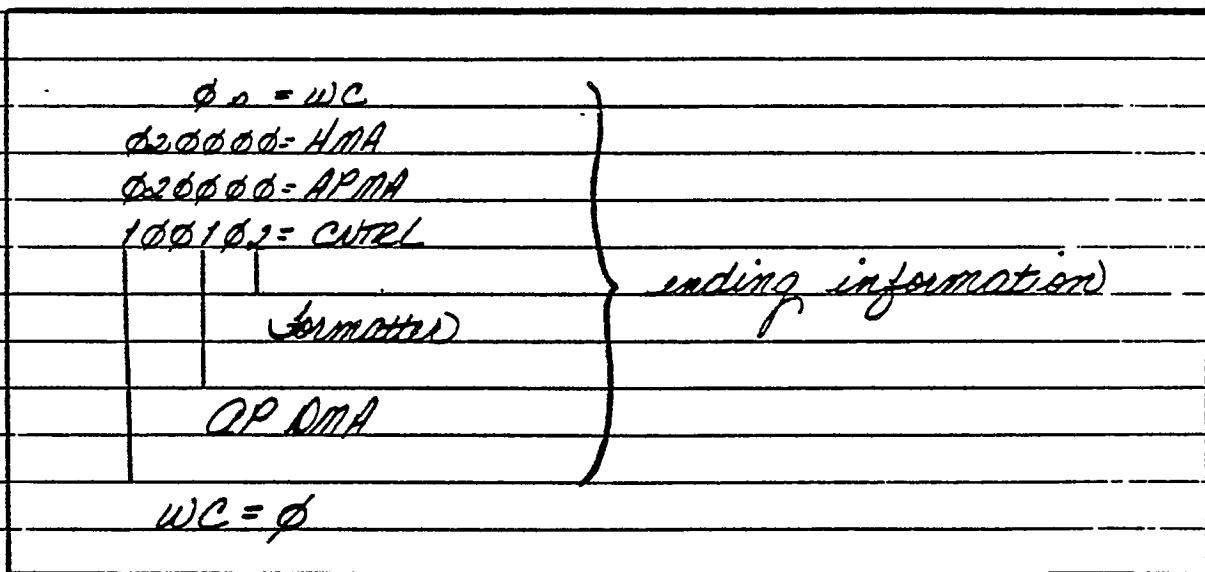
12

G. Test 4

1. Random Address Test
2. same operation as test 3 but uses random Data
3. same Error Codes

H. Test 5

1. Part 1 : transfers data from Host to AP Main Data Memory
checks Host's registers for correct DMA contents



2. Part 2: verifies memory data contents thru VPP operation

I. Test 6

1. runs as though Test 5, two parts
2. Data is transferred from AP to Host

J. Test 7/8

* Not used by our system

(13)

K. Test 9

1. Tests the 226/227 boards
2. Converts Nova format to AP format
3. Data transferred from Host to AP
4. Error messages

	TEST 9 E 001734 005264 113333 A 001730 000526 153333
--	---

(error messages will have 3 words)

L. Test 10

1. Tests the 226/227 boards
2. Converts AP format to Nova format
3. Data transferred from AP to Host
4. Error messages

	TEST 10 E 001734 005264 A 001730 0005060
--	---

(error messages will have 2 words)

5. Both test 9 and 10 are not all inclusive, they check about 20 number combinations

III. AP Path

A This test uses u-code to test the internal paths of the AP

1. AP test verified data can flow from Host to necessary parts of AP

2. AP Path verifies data can flow to almost every part of the AP

B This test uses VFP operation to check that the path under test is OK

C Use of clock slows will help to narrow down failure and where it is.

D Every message is packed format

1. Example

Code 0000115 E 000000 00001 A 000001 000001

Ploc 002545 ↗

E 000000 000000 000001

EXP=10 LM=12 LM=16

→ shift for new assignments

2. When dealing with DPX/DPY (200L/200R)

200 L = 20 BITS

200 R = 20 BITS

EXP 00-11-LM08-15 ↗

LM00-11-LM00-07 ↗

LM 20 → 23

LM 12 → 19

VIII. AP Arch

1. This test uses u-code to test the ability of the processor.

1. Header u-code

- a. 4 u-instructions long,
- b. Example

LOCATION	OPERATION	
000	EADD 0,0 ; EMUL TM,MD	
001	"	
002	"	
003	LDAPS ; DB - DECREMENT COUNT	} flush the pipelines
	Load AP Status	
	Data Bus	number of random instructions
		Bit reversal (unknown quantum)

2. trailer) u-code

- a. As long as necessary, depending on the number of random instructions that were executed (10)

b. Example

LOCATION	OPERATION	
004	first random instruction	
005		
006		} flush the pipelines and store results
007		

(16)

b. Error print-outs

1. Format is unpacked

2. OP (status) = hardware failure information

E 040000 A 040002

3. SPAD = address and data (16 bits)

SPAD E A

register address

4. DPX = address and data (38 bits)

DPX E A

register address

5. DPY = same as DPX format

6. When error messages start with

SPID = SPAD 201

DPX = FMUL 206/7/8

DPY = FADD 203/4/5

possible failed boards

c. Interpreting error messages

1. Look at error printout and decide what your attack plan will be

2. Example

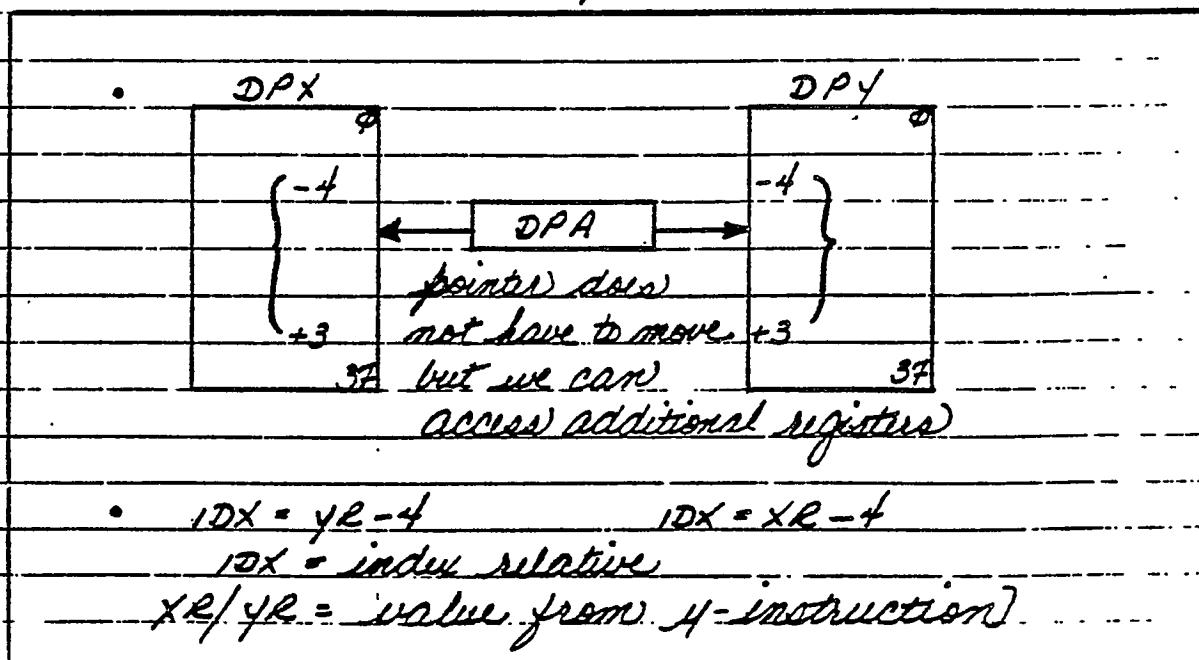
DPY 0000 E 001204 002171 0355265 A 001601 002171 0355264 1 00001 X,Y 001205 0373306 AP STATUS 040002 DPA 0036	Register/addr number of random instructions	base value of random generator	status content	(pointer) for failing address? FADD failure?
--	--	-----------------------------------	-------------------	---

(RE)

3940	4142	4344	4546	4748	4950	5152	5354	5556	5758	5960	6162	63
1	1	1	1	1	1	0	1	0	1	1	1	0
DPA GROUP							MUL OPER.	MEMORY GROUP				
XR	YR	XW	YW			M1	M2					
add	add	write	write			DPY	DPX					N/A
DPX	DPY	DPX	DPY			never change						
<u>used for relative addressing</u>												

- what operation are we concerned with
FADD
- what is the relative addressing of DPX/DPY
DPX(3) DPY(3)
- do we care about the remaining instruction bits
NO

c relative addressing



(F)

- i. Relative address computed from above formula
- ii. which instruction contains operation and inputs.
- iii. which instruction contains results of operation

Program source instructions

location

004 - always instruction / inputs - SAD results

005

006 - FADD results

007 - FMUL results

true when using 1 random number, always
decode errors with 1 random number

d. Actual Addressing = $DPA(DPA + IDX) / DPY(DPA + IDX)$

i. Actual address computed from above formula

ii. This verify address or not

DPX 0010

- error message -



→ $IDX = 1$

computed from PS words

DPA 0010

$DPA 10 + 1 = 11$

does this agree

4. What was data pattern on failure
a. Use the '*' command

* # 267	using charts from Handouts
IF	what are the 2 inputs
IF	what are the 2 results
* * 242	(used relative address IDX on the charts)
EXP IF	
HM IF	
LM	
*	

b. Ready to scope failure

i. You know the following

- what operation occurred
- what data patterns
- what logic boards to look

SPAD 201 operation

FADD 205 exponent 204/203 mantissa

→ EMUL 208 exponent 206/207/208 mantissa

SPAD 201 register

FADD 200L exponent 200L/R mantissa

EMUL 200L exponent 200L/R mantissa

if addressing problem
if data problem

ii. you want only 1 random number

* RNIX061260, 077306 WE,

(the x, y number from error message)

number of random numbers you want

New Error message (should be same.)

* F, decode 4-instructions

* #, read arguments

* LDE, loop error for scoping.

iii. Perform necessary board replacement or chip replacement

overheads used:

SPAD Block Diagram

FADD Block Diagram

FMULT Block Diagram

- Point out how boards are structured
- Briefly explain data flow and operation

(22)

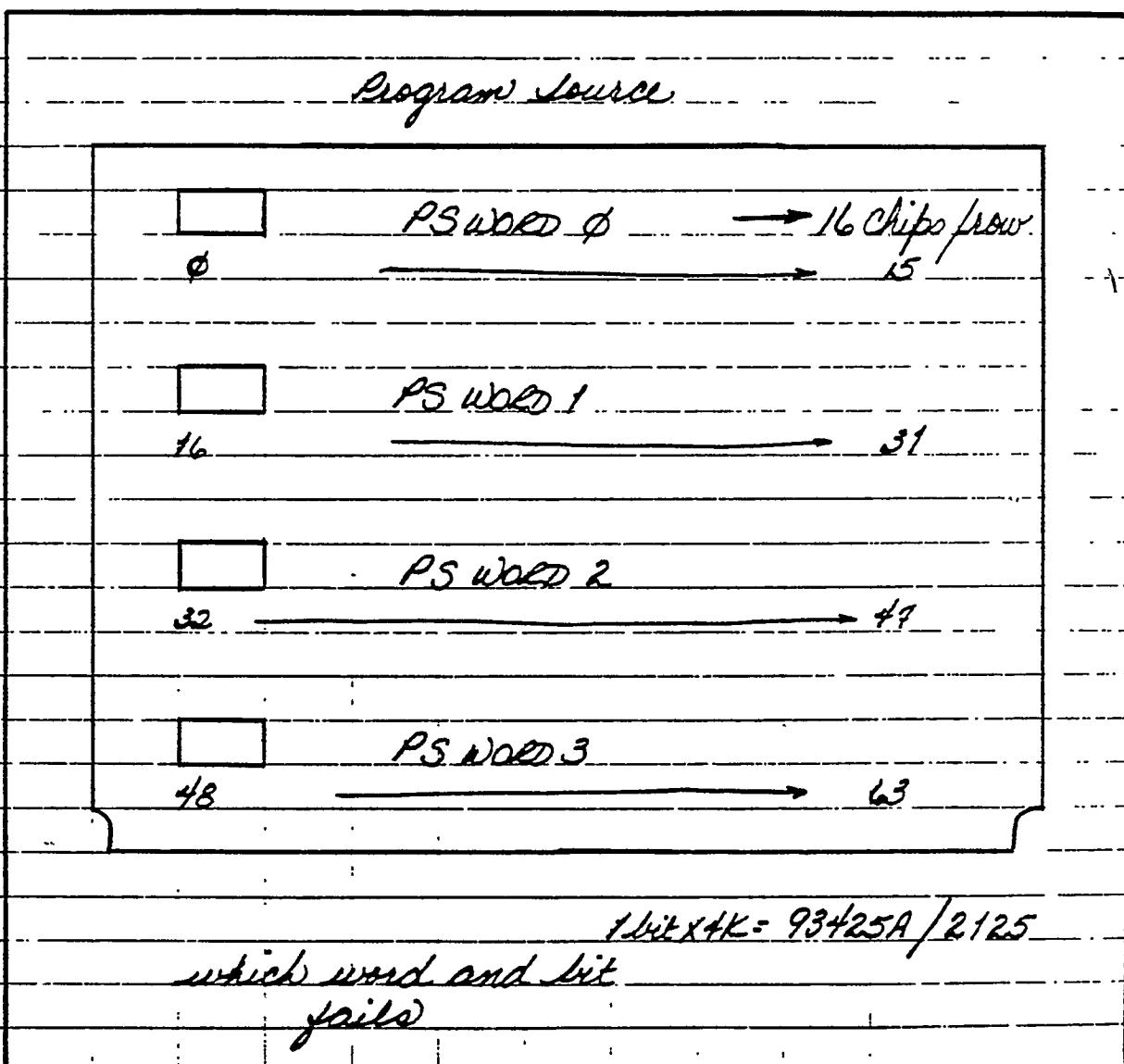
OP code Scoping Procedure

1. Read error message
2. Determine type of failure
3. Decode 4-instruction of failure group using "F" command
4. Determine relative data pad address of input arguments
5. Determine relative data pad address of result arguments
6. Look up the arguments using the "+" command
7. Do a "LDE";
8. Set data switches to desired 4-instruction step minus 1
9. Use "Trig Q" signal on 214 Board pin A13 to verify operation occurred
10. Use control signal for latches to scope data inputs
 - 10A. Inputs correct = redo step 8
 - Inputs not correct = addressing problem
 - 10B. Result matches actual result in message = board failure
 - Result matches expected result in message = address failure

IX. PS Text

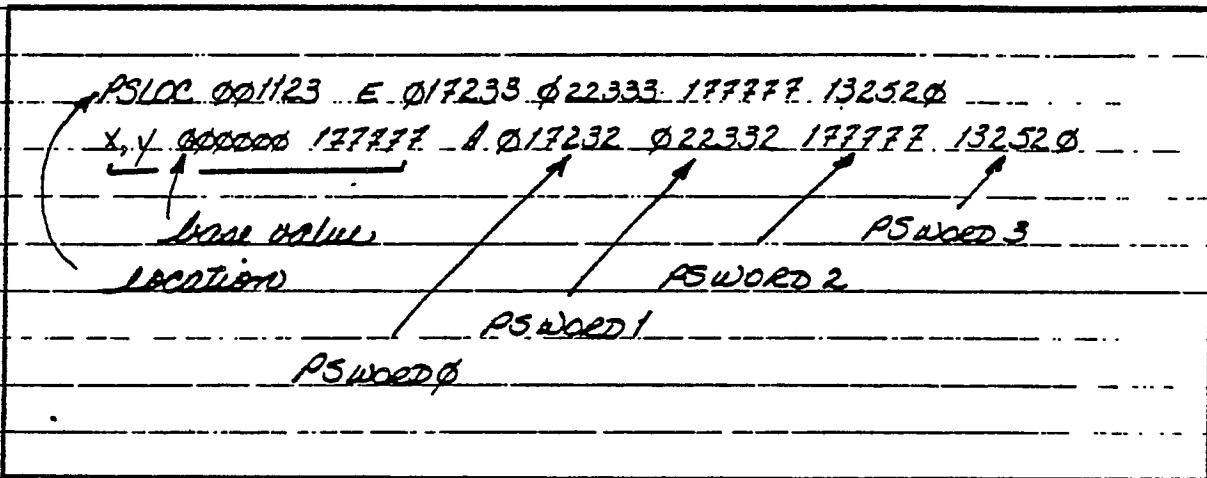
A. Board structure

1. 236 Board "4K"



2. 12 bits word for addressing (PSA)
- B. Error messages
1. Error format is unpacked

2. Example



3. Troubleshooting

a. of obvious data miscompare - addressing
failure

b. of just bit(s) then which bit(s) / word(s)

c. of failure on PS shown on earlier tests
then PS test should be run to verify
failure

4. In Rom

A. tests and verifies the 217 Board

B. If no pass printout, so let time go by, (~15-20 min)

C. If failure replace board

5. In RAM

A. Tests the 238 full and 238 half boards

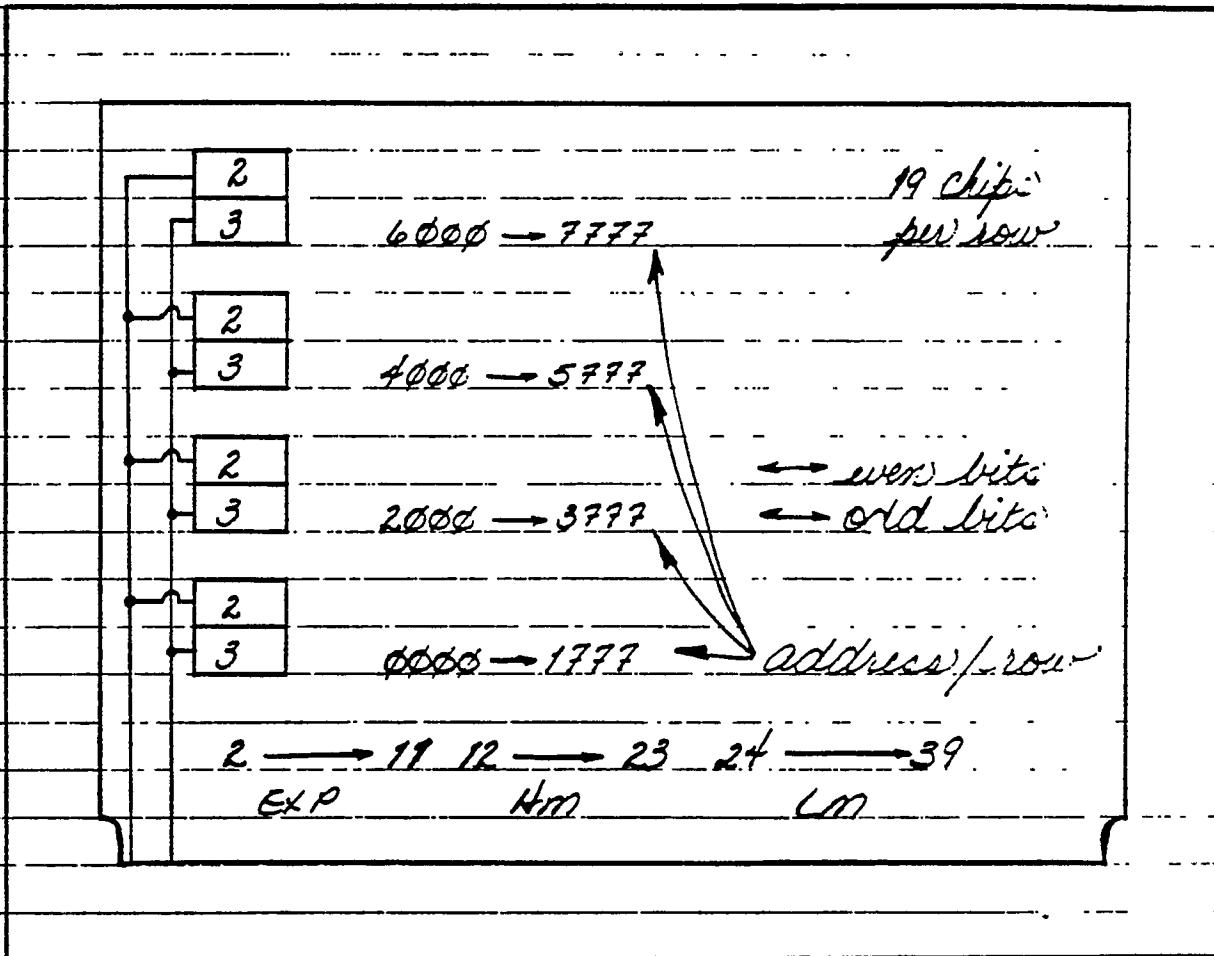
1. 238 full has 4K of RAM address =

2. 238 half has 2K of RAM address =

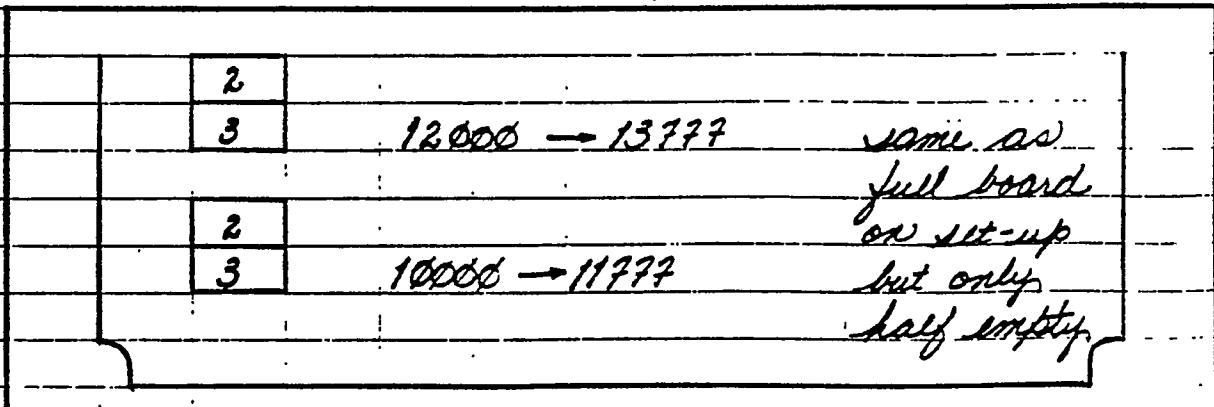
3. Pass printout = R

B. Board Structure

1. 238 full



2. 238 half



C Error messages

1. Error format is unpacked
2. Example

MA 0010372 E 001734 002731 124756
A 001734 002731 024756

address of failure
 what group and row
 error bit 24
 of mantissa

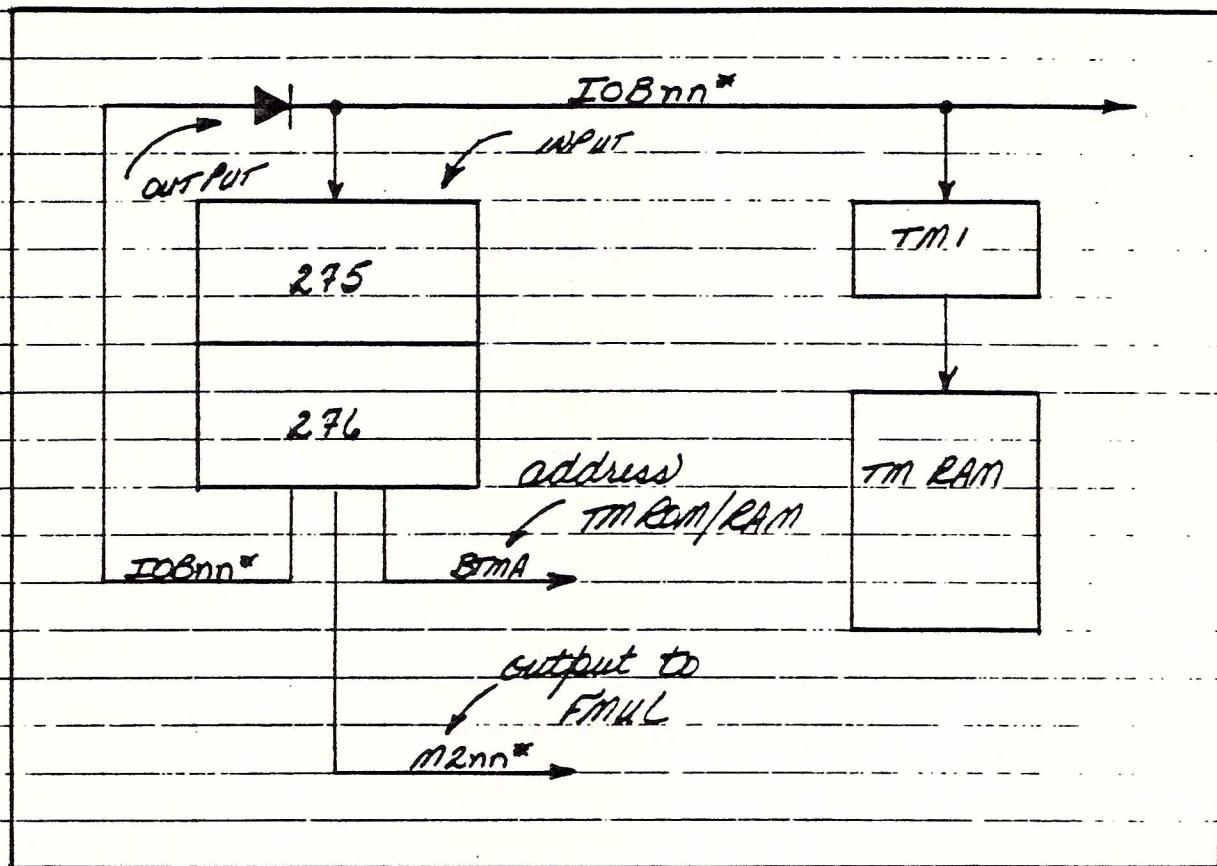
D Troubleshooting

1. of obvious data miscompare addressing
2. If bit cold
3. Use MA for rows and board for failure
4. Count over to chip for bit(s)
5. If bit hot cut failing common output bus to isolate failing chip

VII. CCP Boards

1. What do they stand for
2. Image Reconstruction Process
3. They only come into play during backprojection of deconvolved data
4. Final output of these boards = DDC by Quadro
5. Where do they fit into system
6. Add them to AP Block Diagram
7. How

(27)



c. Diagnostics

1. There are 3 tests which are divided into 9 individual tests that are spread across them
2. You must run all 3 test to verify all board function

a. ERPT53

- i. Tests mainly the 275 Board
- ii. All systems have the same 275 Board

7805	field upgrade C2 · AP (289)
7810/11	field upgrade X2 · AP (511)
8000	manufactured C2 · AP (289)
7900 (8800)	manufactured X2 · AP (511)

Tests 1, 2, 3 are called out in IRPT53

- verifies data can go into and out of boards
- verified & Reg. area

K Reg. = view being backprojected at this time

b. E1D 289/511

- i. Tests mainly the 276 Board
- ii. What type of system do you have
- iii. Only difference between boards is contents of Rom.

7805	} same 276 board	} difference, Rom contents
8000		
7810/11	} same 276 board	
7900 (8800)		

Tests 4, 5, 10 (8) are called out in E1D XXX

- 4 - PES = Proj. Sq = y axis value
- 5 - PAS = Proj. Sq = x axis value
- 10 = WGT = weighting factor = pixel value / data

really 8 but test errors print out 10

c. E2D 289/511

- i. Same as E1D XXX by definition
- ii. Use overhead of IRP Block Diagram to show data flow

Tests 6, 7, 11 (9) are called out in E2D xxx

• 6 = LPE = Log. Perp = log. of y axis

• 7 = LPA = Log. Paral = log. of x axis

• 11 = DET# = detector channel number = address
in ROM and in RAM

really 9 but test errors print out 11

XIII. FFT / FDT Tests

A. FFT

1. Fast Fourier transform

2. Uses VFP operation to verify operation

3. You need all but IOP / TM RAM Boards

B. FDT

1. Fast Fourier transform

2. Uses DNA operation to verify operation

3. You need all but IOP / TM RAM Boards

4. Best quick check of unit to see reliability

a. if failure seen other diag should pull out
faulting boards

b. other diag may not find failure - no surprise

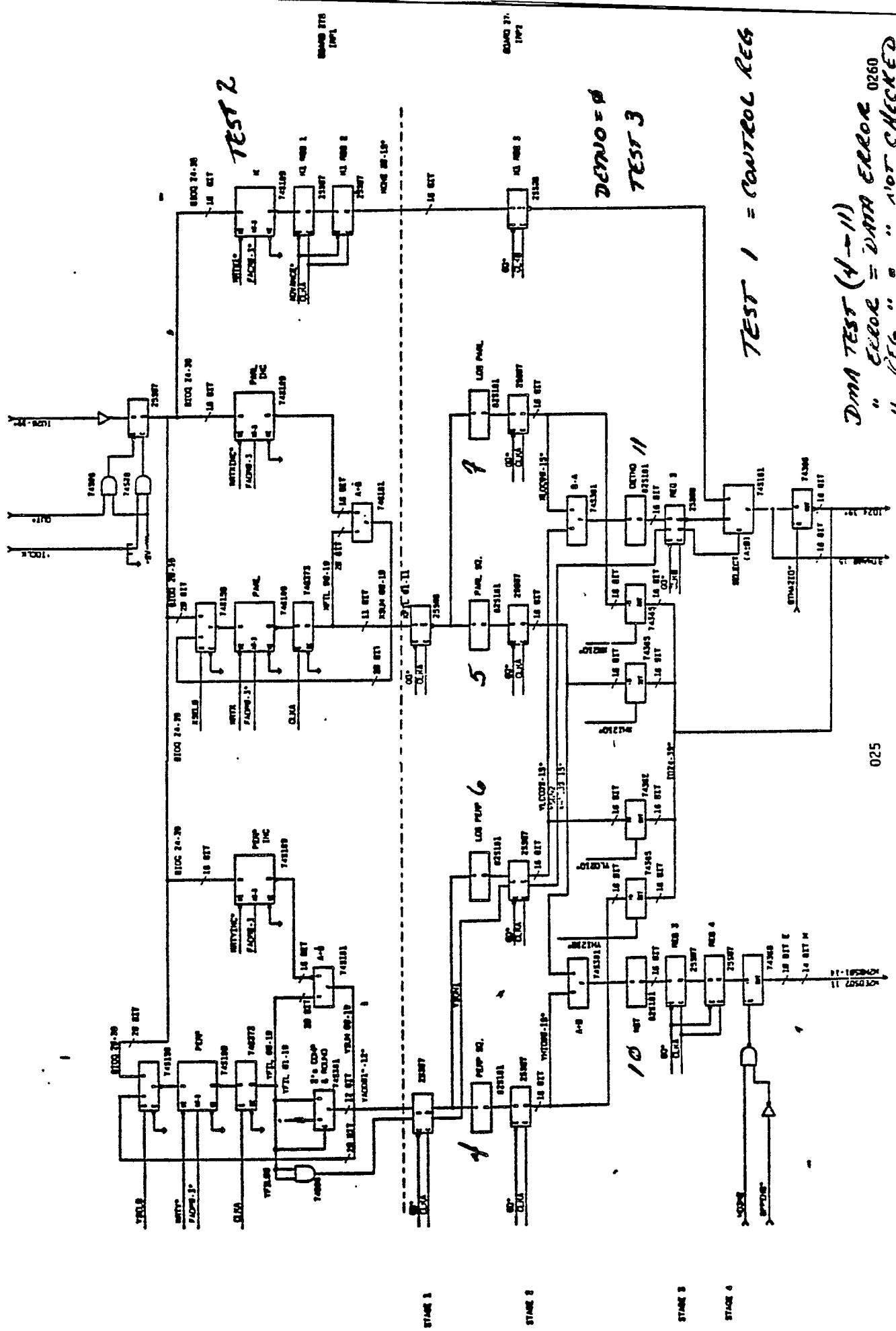


Figure 1-4 Block Diagram of IRP

Figure 1-4 Block Diagram of IRP (cont.)

025

117
W. H. T. K.

"close = DNA change 0260

" *C. E. G.* " " " *A. B. C. E. D.*

" *C. E. G.* " " " *A. B. C. E. D.*

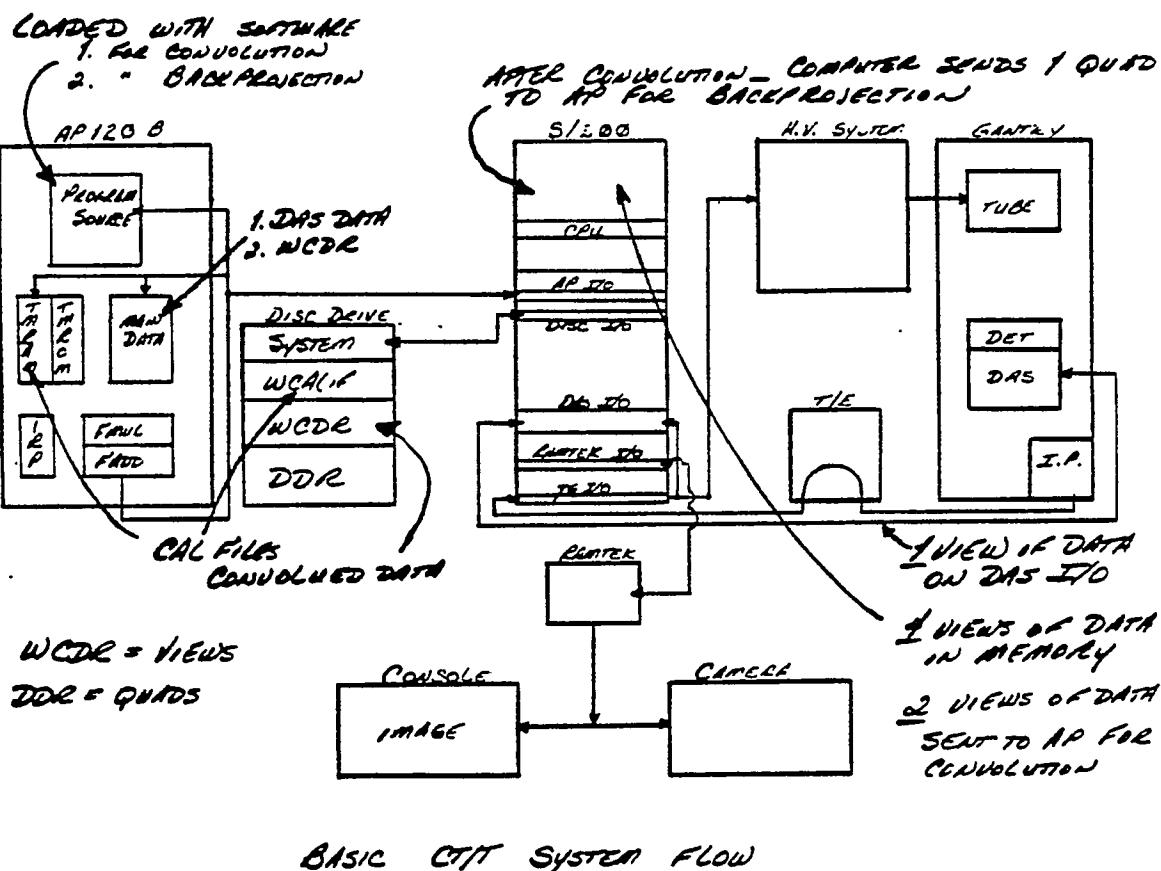
(31)

XII. System Data Flow

A. Review Data Flow

overheads used:

System Data Flow



B. Review Course

C. Student critiques

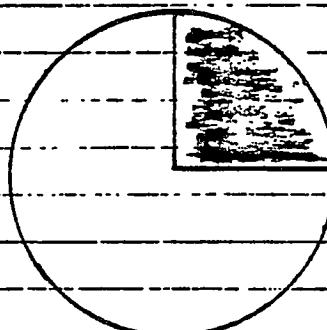
D. System failures

systems and other failures):

1. Missing Quad

238 TM RAM

201 SPAD

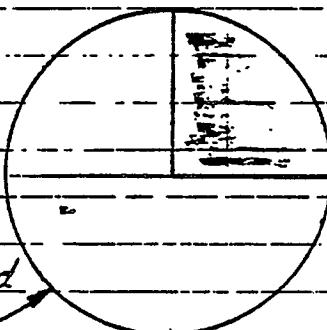


may not fail diag

2. Garbage Quad

238 TM RAM

may also show
garbage in associated
quad

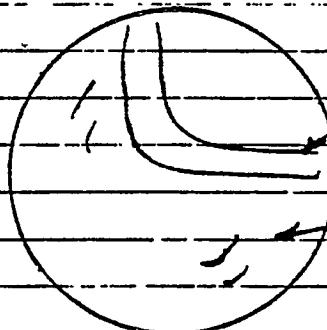


may not fail diag

3. Quad marks

76 cable
(AP → CPU)

201 SPAD

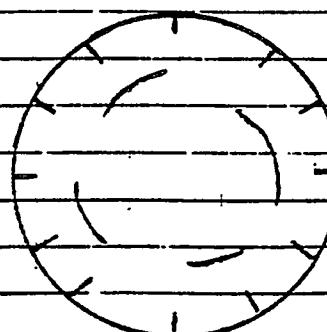


no diag fail
could look
like or

4. New marks/streaks

238 TM RAM

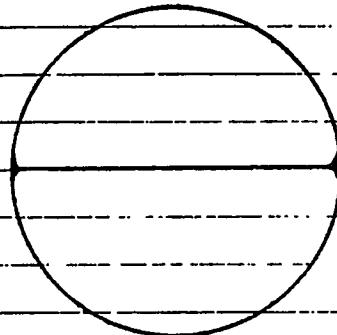
heat build-up



no diag fail

5. Point line in centre
of image

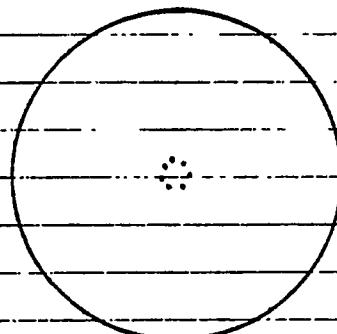
201 SPAD



no fail diag.

6. Dots in centre of image
(pattern)

201 SPAD



no diag. fail

7. Spectral errors
since "FD" software

no diag. fail

old 499 DAS I/O

tube split error
226/227 FORMATTER (catch all)

8. CT numbers binned by
1000

water = $\phi = (1000)$

no diag. fail

212

9. SIMOV error

no diag. fail
slice move error

201 SPAD

10. AP too slow / Data Channel too slow

no diag. fail

2 201 SPAD

54.

10. AP too slow / Data channel too slow no diag fail

1. gantry too fast !!

15-17 pulses

FD software slipped
phase lock = 1% slower

(must be)

3. DAS I/O

11. Total Reconstruction Error 898 no diag fail

219 GPIF you run scan and
no data starts to
flow

12. ABCODE Aabcd CODE

no diag fail

238 TMRAm you are bring system
up and before monitor
up

13. ABCODE

no diag fail

Software too slow you are scanning
"slow error" and message appears
"Recon error" during data collection

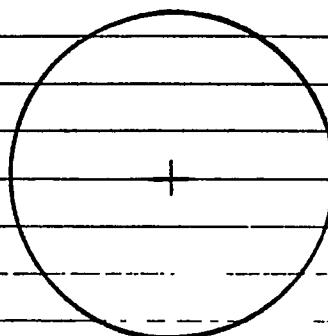
WCDR needs 2500 consecutive good blocks
but bad block found somewhere
in a CDR and does not flagged
before

(25)

14. Faint cross in image

no diag fail

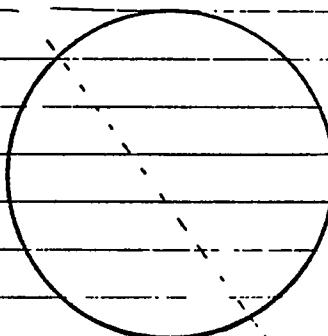
* one P/S Blower



15. Missing Pixels
(line)

no diag fail

* heat to off side



Date _____

FAST RECONSTRUCTION
Course

Mean Time to Repair (MTTR) Exercise No. 400

Student's Name

District

Situation: Listen to tapes 1 and 2 on the AP120B. The manuals were given to you before starting these tapes.

Symptoms: When you have finished with the first two (2) tapes, turn-in to the instructor the completed answer sheet from Book 1.

Describe Fix:

Parts Used:

Parts Needed:

Time of Dispatch:

Time of Fix:

Date _____

FAST RECONSTRUCT

Course

Mean Time to Repair (MTTR) Exercise No. 401

Student's Name

District

Situation: Listen to tape 3 on the AP120B. The manuals were given to you before starting this tape.

Symptoms:

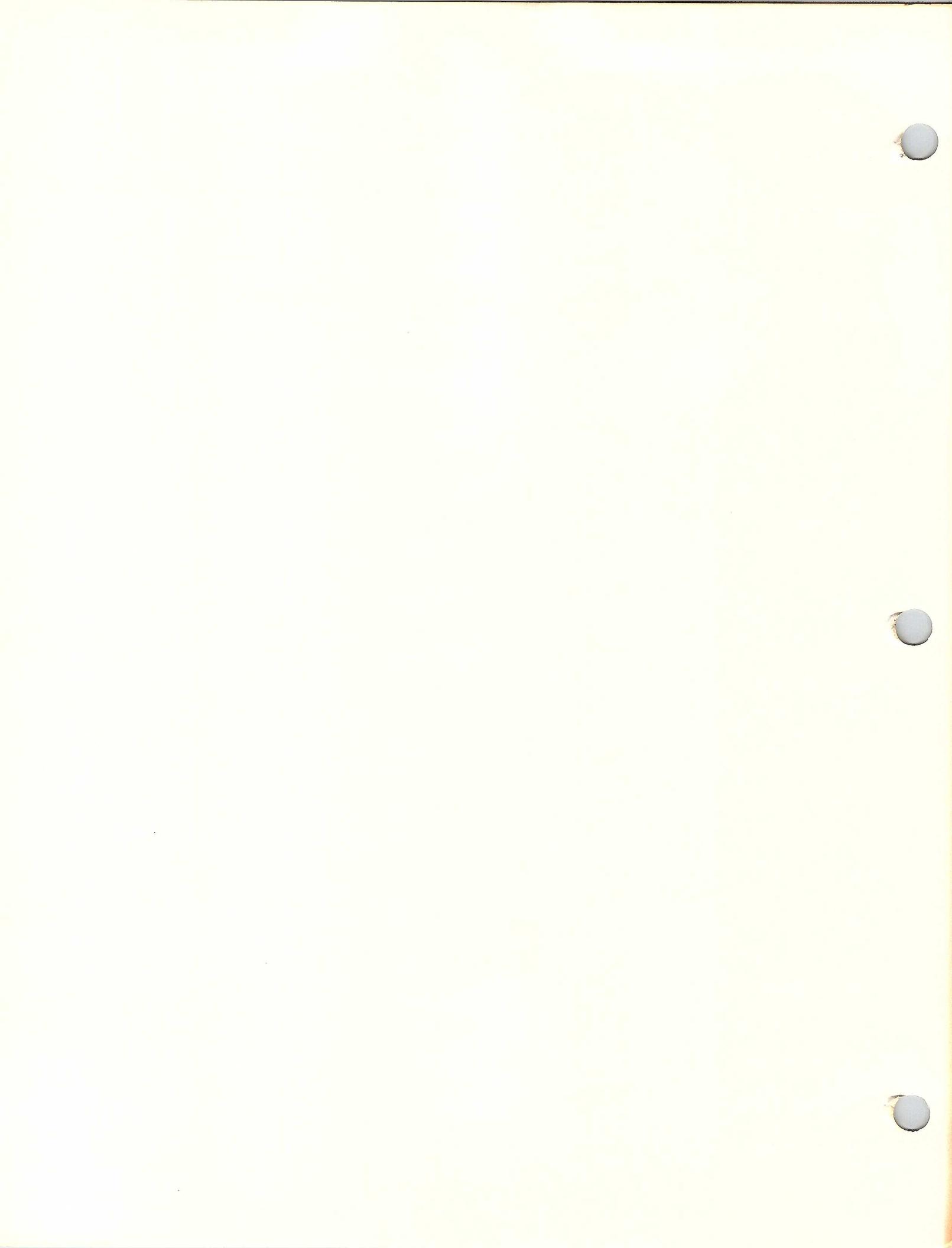
Describe Fix:

Parts Used:

Parts Needed:

Time of Dispatch:

Time of Fix:



Date _____

FAST RECONSTRUCT

Course

Mean Time to Repair (MTTR) Exercise No. 402

Student's Name

District

Situation: Read "Array Processor Maintenance Manual"
Chapter 5 - "Theory of Operation"

Symptoms:

Describe Fix:

Parts Used:

Parts Needed:

Time of Dispatch:

Time of Fix:

APPENDIX G

USE OF THE SWITCH REGISTERS IN DIAGNOSTIC TESTING

TYPES OF SWITCH REGISTERS

There are two types of switch registers in DTOS that the user can access:

- DTOS switch register
- Diagnostic-program switch register

The DTOS switch register is the switch register for the operating system. When DTOS is first loaded, this register has the default value of 000000. The user may make changes to this register any time that DTOS prompts with an asterisk (*) displayed on the terminal. Then, DTOS maintains this value for the DTOS switch register no matter how many times the different programs are loaded or reloaded.

The diagnostic-program switch register is a register for the currently loaded program. DTOS loads the program switch register with the value of the DTOS switch register whenever either a new program is loaded or the same program is reloaded. The user can then change the switch settings as desired by complementing the switches. However, DTOS maintains the current value of this register only while the program is loaded and running. (Refer to the following section titled "DIAGNOSTIC-PROGRAM SWITCH REGISTER" for information on how to complement the switches.)

SWITCH DESCRIPTIONS

The following list summarizes the switches in the DTOS switch register. The descriptions for switches 0 thru 11 are the same for both the DTOS and the diagnostic-program switch registers. However, the list does not give any descriptions for switches 12 thru 15 because DTOS does not use them. Nevertheless, an individual diagnostic program may use switches 12 thru 15 in its own switch register. (Refer to the text file of each program for information on how the specific program uses these switches.)

APPENDIX G (Continued)

SWITCH NUMBER	DEFINITION
Switch 0 = 0	All switches in the register are in the default condition.
Switch 0 = 1	Switches in the register are not in the default condition.
Switch 1 = 0	(Default condition) Loop on error.
Switch 1 = 1	Do not loop on error.
Switch 2 = 0	(Default condition) Print-out to console.
Switch 2 = 1	Do not print-out to console.
Switch 3 = 0	(Default condition) Do not print the percent of failure.
Switch 3 = 1	Print the percent of failure.
Switch 4 = 0	(Default condition) Print the pass count.
Switch 4 = 1	Do not print the pass count.
Switch 5 = 0	(Default condition) Do not print-out to the line printer.
Switch 5 = 1	Print-out to the line printer.
Switch 6 = 0	(Default condition) Do not halt on error.
Switch 6 = 1	Halt on error.
Switch 7 = 0	(Default condition) Do not print summary and/or passing of each subtest.
Switch 7 = 1	Print summary and/or passing of each subtest.
Switch 8 = 0	(Default condition) Print only the first error.
Switch 8 = 1	Print every error.
Switches 9-11	Reserved for future use by DTOS.
Switches 12-15	Not used by DTOS. Used only by the individual diagnostic programs.

DIAGNOSTIC-PROGRAM SWITCH REGISTER

Introduction

The user can change the switch settings in the program switch register during both of the following modes:

- Program-Run Mode
- Switch-Modification Mode

In both of these modes the user complements a switch by pressing a key that corresponds to the switch. This is the only way to change a switch in the program switch register because the "SWREG" command does not work while the diagnostic program is loaded and running.

Whenever the user wants to display the switch register value without changing the switches, he inputs the following command.

KEYS PRESSED	RESULTS
O	The program is now in the Switch-Modification Mode.
M	The terminal displays the current switch-register value spread out by bits. For example, if the value is an octal 000000, the terminal displays it as follows: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The following section titled "Complementing The Switches" describes the input process for both the Program-Run Mode and the Switch-Modification Mode.

Complementing The Switches

The Program-Run Mode

The user complements a switch in the diagnostic-program switch register while the program is running by pressing one of the keys listed below in the section titled "Keys Used To Complement Switches". The program accepts this input as a switch-complement input as long as it is not waiting for another input, such as when the following prompt is output to the terminal: "ENTER THE DEVICE CODE". The terminal then echoes the key that was pressed to verify that the related switch has been complemented.

If the user wants to change more than one switch, he just presses all the related keys with no spaces or commas between them. The terminal then echoes all the keys in the order that they were entered. If the user presses a key that is not listed below, the terminal just echoes the character without changing any switches.

It is important to note that in the Program-Run Mode, the program continues to run after complementing a switch.

Switch-Modification Mode

The Switch-Modification Mode allows the user to stop the normal flow of the diagnostic program while making changes to the program switch register. The user stops the program by inputting the CONTROL O command which is explained in the following list. He then complements switches in exactly the same manner that he does in the Program-Run Mode. However, the only difference is that the program is stopped during and after the changes are being made. The user starts the program again by inputting one of the commands listed below.

COMMAND	RESULT
O	This command locks the program in a mode called the Switch-Modification Mode. The user can input this command any time the program is not requesting a specific input (like "ENTER THE DEVICE CODE"). Refer to the above section titled "Program-Run Mode" for details on how to complement the switches by pressing the terminal keys.
Carriage Return	The user usually presses the Carriage Return key after he inputs a change. Using this key does the following: it keeps the switch register set to the most current value, exits the Switch-Modification Mode, and continues with the program from where it was when this mode was entered. If the user presses this key without inputting any changes, the program exits the Switch-Modification Mode, keeps the switch register value the same as it was on entering this mode, and continues with the program from where it left off.
CONTROL R	This is a system control character which normally keeps the switch register set to the current value and restarts the program from the beginning. However, when it is used during the Switch-Modification Mode, it causes the program to exit from this mode in addition to performing the above mentioned functions.
CONTROL D	This is a system control character which normally resets the switch register to the default value (all zeroes) and restarts the program from the beginning. However, when it is used during the Switch-Modification Mode, it causes the program to exit from this mode in addition to performing the above mentioned functions.

The following example shows how the user would change some switches in the switch register.

Example:

KEYS PRESSED	RESULTS
0	The program is now in the Switch-Modification Mode.
M	The terminal displays the current switch-register value spread out by bits. For example, if the value is an octal 000000, the terminal displays it as follows: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
46A	Switches 4, 6, and 10 are complemented. The switch-register in the above example is now octal 105040. (See note below.)
M	The "M" command is entered again just to see the changed bits as follows: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0
Carriage Return	The program exits the Switch-Modification Mode; the switch register has the new value of 105040; and the program continues where it left off.

NOTE The program sets bit 0 to a value of 1 whenever another bit in the switch register is set, otherwise it resets the bit. This explains why bit 0 in the above example (105040) is a 1 even though it was not set by pressing key 0.

Keys Used To Complement Switches

The following list shows the keys that are used to complement each switch.

KEY	SWITCH
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

<u>IC</u>	<u>PART NUMBER</u>	<u>PRICE</u>	<u>QTY</u>	<u>IC</u>	<u>PART NUMBER</u>	<u>PRICE</u>	<u>QTY</u>
930	46-136290P1			74163	46-203200P80		
936	46-136290P6			74164	46-203200P93		
944	46-136290P15			74170	46-203200P52	\$3.31	
6	46-136290P2			74172		5.29	
53	46-136290P11			74174	46-203200P51		
962	46-136290P3			74175	46-203200P58		
9324	46-136328P4			74176	46-136291P113		
9602	46-203200P95			74177	46-203200P99		
7400	46-203200P1			74179		1.48	
7401		\$.34		74180	46-203200P46	1.25	
7402	46-203200P7			74181	46-203200P59		
7403	46-203200P2			74182		1.25	
7404	46-203200P11			74192	46-203200P83		
7405	46-203200P12			74193	46-203200P84		
7406	46-136291P103	.45		74194		1.13	
7407	46-203200P63			74195	46-203200P40		
7408	46-203200P33			74197	46-203200P96		
7410	46-203200P4			74198		1.71	
7413	46-203200P56			74298		1.33	
7414	46-203200P73			75107	46-136327P32		
7420	46-203200P5			75108		3.22	
7427	46-203200P36			75109		3.38	
7430	46-203200P6			75110	46-136327P33		
7432	46-203200P53			75150	46-136327P34		
7433		.39		75154	46-203200P17		
7437	46-203200P35	.16		75232		1.94	
7438	46-203200P43	.52		75324		3.50	
7439		.75		75450		1.24	
440	46-203200P14			75452	46-136327P72		
445	46-203200P22			75453	46-136327P60		
447	46-203200P24			75463	46-203202P16		
7448	46-136291P67			74H00	46-203200P38		
7450	46-203200P8	.81		74H11		1.30	
7453		.27 - .31		74H21		.93	
7460		.34		74H50		.93	
7474	46-203200P17			74H52		1.32	
7475	46-203200P25			74H55		.93	
7482	46-203200P27			74H74	46-203200P42		
7484	46-203200P10			74500	46-203200P85		
7490	46-203200P19			74502	46-174586-P11	.57	
7492	46-203200P20			74503		1.28	3.00
7493	46-203200P21			74504	46-203200P86		
74107	46-203200P16			74505		.68	
74121	46-203200P18			74510	46-203200P87	.57	
74123	46-203200P65			74511		.97	
74132	46-203200P136			74515		.57	
74141		1.37		74520		1.39	
74145		.97		74522		.57	
74148	46-203200P67			74540		.85	
74151	46-203200P26			74551	46-174986-P7	.88	
74153	46-203200P31			74564		.85	
74155	46-203200P70			74574	46-174986-P4	1.71	
74157	46-203200P32			74585		8.85	3.02
74160	46-203200P68			74586		.85	
74161	46-203200P34			74S112	46-203200P9	1.96	

74S113	46-203200P131		2N3439	46-400052P33
74S114		\$1.19	2N3441	46-136350P3
74S133		.85	2N3614	57455
74S138	46-203200P88	2.75	2N3773	46-400052P41
74S139		2.51	2N3904	46-400052P43
74S140		1.06	2N3906	46-400052P44
74S153		2.39	2N3958	46-400052P47
74S157		2.39	2N4124	46-400052P36
74S163		6.13	2N4125	\$.40
74S174	46-203200P89	3.39	2N4126	46-400052P38
74S175		3.39	2N4416	2.45
74S181		6.91	2N5302	46-400052P54
74S182		4.57	2N5631	6.74
74S194		5.34	2N6518	46-400052P50
74S197	H6-174981-P6	3.41	2N6521	46-400052P48
74S251		2.86	2N6523	46-400052P49
74S257	H6-174981-P47	2.50	TD 400	46-400052P53
74S258		2.90	1N 484	.51
74S260		.63	1N 751	75084
74S280		7.54	1N 752	75085
74S289		3.28	1N 753	75245
74S301		4.55	1N 754	75270
74S387		4.41	1N 758	75272
74L75	46-203200P74		1N 759	46415
74L98		2.46	1N 823	75148
8885		3.85	1N4003	11519
9662		1.50	1N4005	.60
9300		1.10	1N4148	46-136314P1
TMS6011		7.45	1N4151	.75
3001P		.83	1N4157	46-400052P61
2N 685	46-400052P52		1N4760	59225
2N 914	85724		1N5245	53152
2N1613	46-400052P42		710	46-136322P35
2N2219	46-400052P35		C723	46-136284P1
2N2222		.95	EL723	46-136284P2
2N2905	46-400052P37		741	46-203204P14
2N3055	46-400052P39		749	2.93

74S189 2.43
 75234 1.94
 D45C5 .90
 LN4403 25 .30
 75324 3.50
 2N4400 0.35
 LA3046 0.95
 TIP 35 2.04
 TIP 36 2.06
 1.20
 UA711HC 4.70
 UA733 HC 2.60
 2N4393
 IN5228B
 201173

SA 250 3.50

