Publication No. 7380
May, 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.

Printed in USA

CONTENTS

ILLUSTRATIONS

TABLES

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-CT. 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 AP-CT is referred to as the "AP".

1.2 SCOPE

This manual provides hardware information related to the AP. 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. 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 hardware. Additional information on the AP, 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
Programmer's Reference Manual Parts One and Two	7319
APDBUG Manual	7364-01
AP Diagnostic Software Manual	7284-02
AP Math Library Parts One and Two	7288-03

1.3 GENERAL DESCRIPTION

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

* floating-point adder (FADD) - 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 225ns although it takes 450ns 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 cyles to complete the multiplication (675ns), the pipeline method permits a new multiply operation to be started every 225ns.

  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 contains five registers used for addressing the system's memories, peripherals and scratchpad (SPAD) registers. These five registers are:

* table memory address register - 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.

* scratchpad destination - contains the address of the scratchpad register that is to receive the output of the scratchpad ALU.

Interface

The AP 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 and the host computer. The interface elements are:

* switch register - used by the host to load data and/or addresses into the AP. 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 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 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 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 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 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 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 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. 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 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 configuration.

1.4.1.1 Power Panel

One 5V/120A or 150A DC power supply, a +12V/3A DC power supply, and a -5V/3A DC power supply are to be mounted on the power panel. A line box is already mounted. 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.

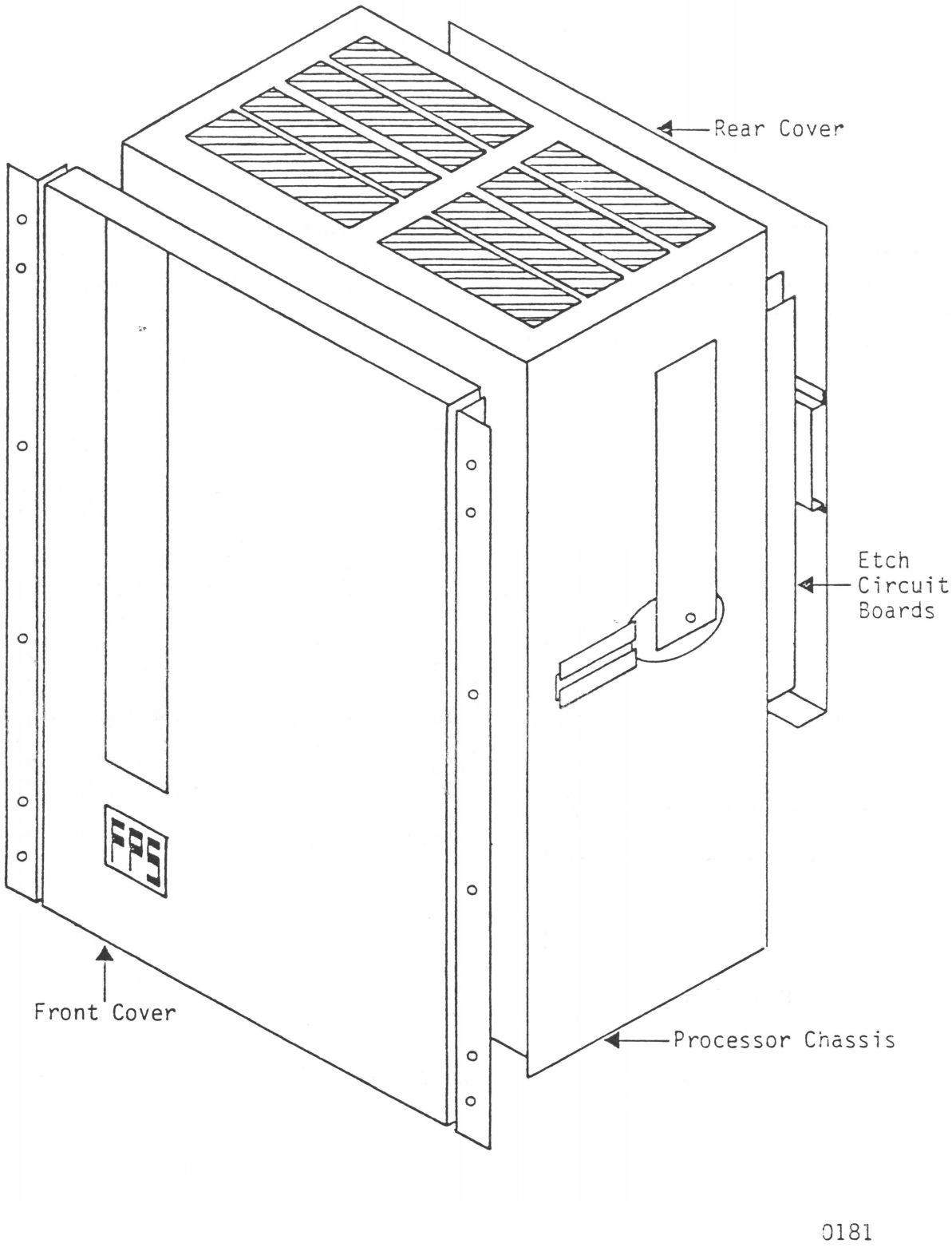
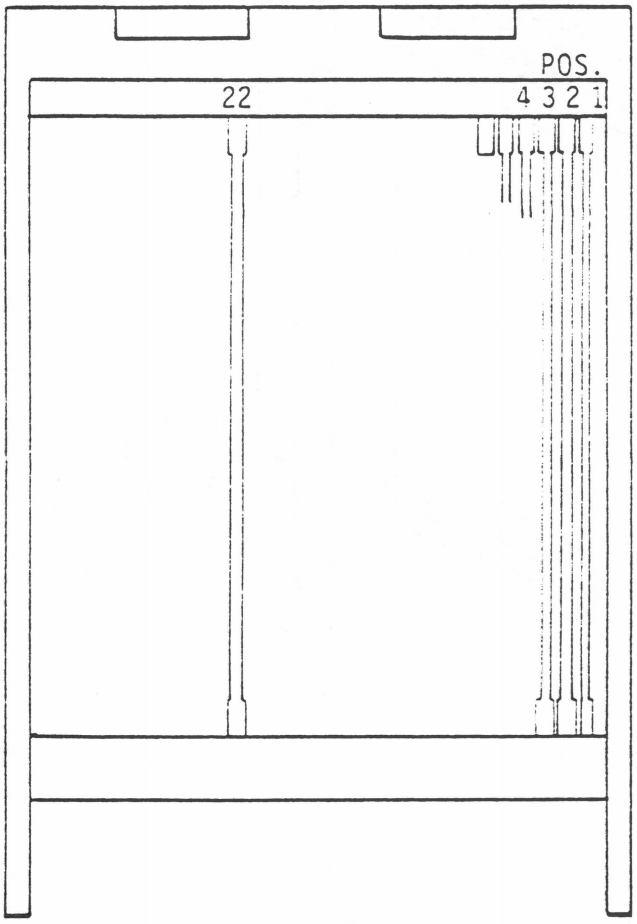


Figure 1-1 The Processor



POS.	BOARD	TYPE
1	263	PDP INTERFACE
2	227	FORMATTER
3	226	
4	215	MAIN DATA
5	216	PROGRAM SOURCE
6	201	SCRATCH PAD
7	212	TM ADDRESS REGISTERS
8	214	INST. & FRONT PANEL REG.
9	210	DEST. ADDRESS REGISTERS
10	211	DATA PAD ADDR. REGISTER
11	202	DATA MEMORY OUT. REGISTER
12	217	TM ROM
13	209	TM OUTPUT REGISTER
14	213	DATA MEMORY IN. REGISTER
15	200	DATA PAD RIGHT
16	200	DATA PAD LEFT
17	203	
18	204	FLOATING POINT ADDER
19	205	
20	206	
21	207	FLOATING POINT MULTIPLIER
22	208	
23-31	NULL	

AP-CT FRONT VIEW 0297

Figure 1-2 AP-CT Configuration

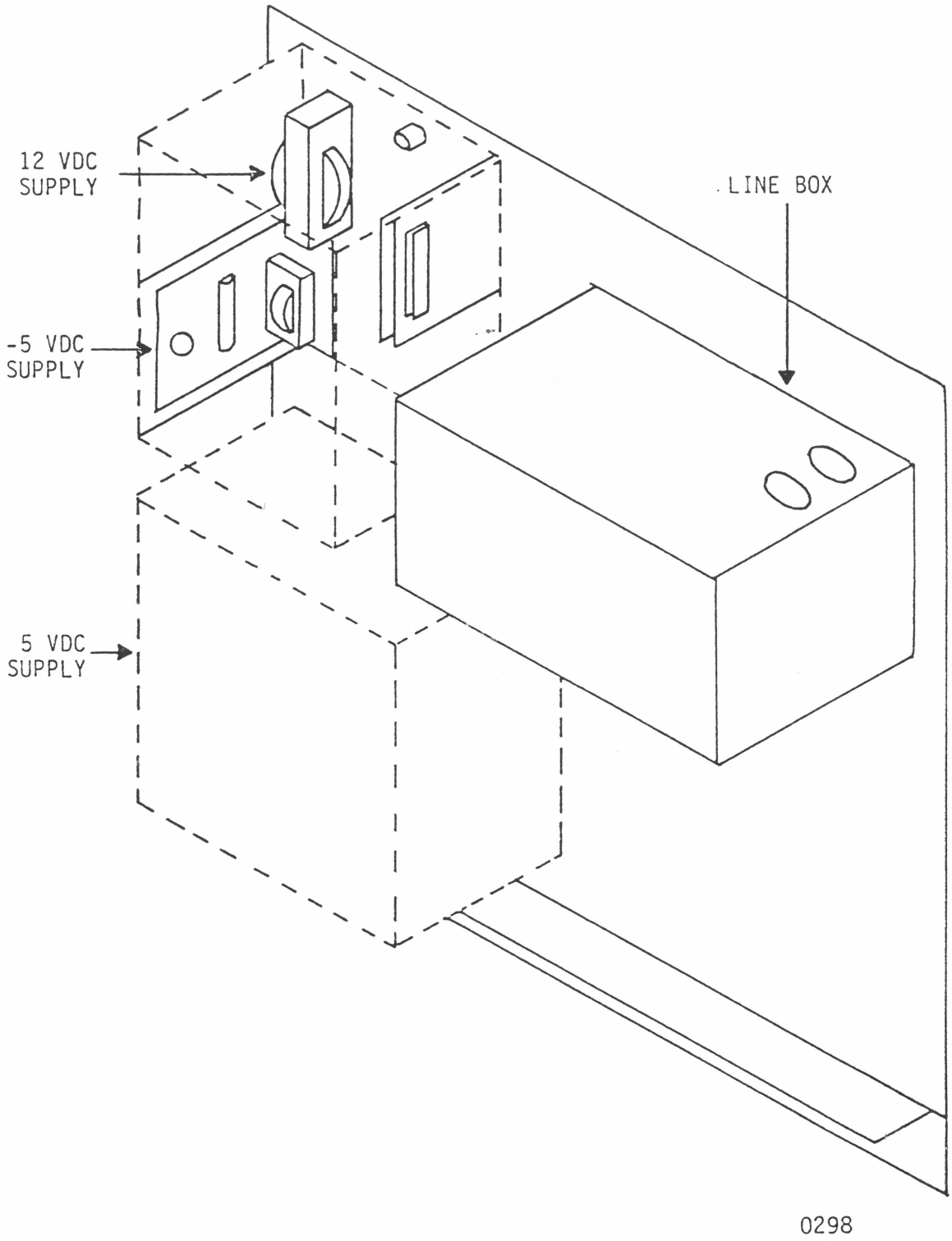


Figure 1-3 Power Panel

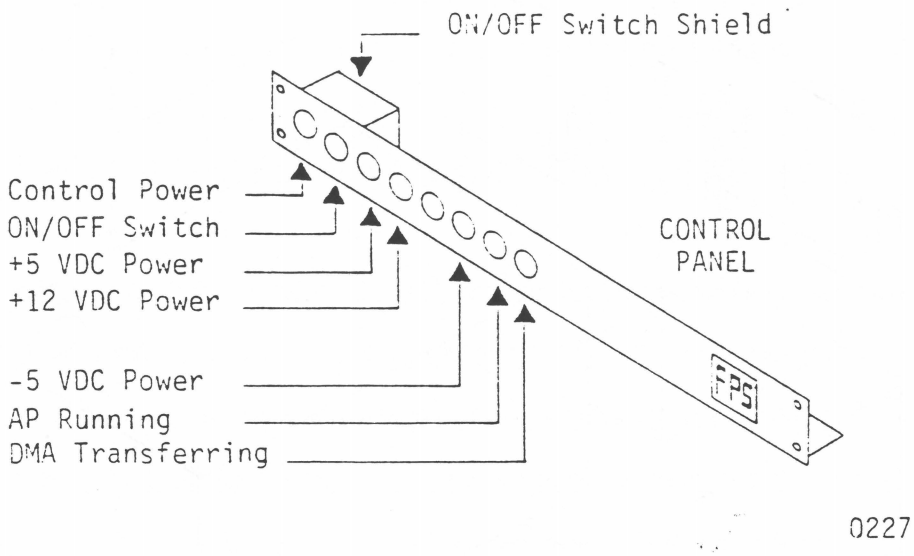


Figure 1-4 Control Panel

CHAPTER 2

SITE PLANNING AND PREPARATION

2.1 INTRODUCTION

This section provides a summary of the equipment characteristics of the AP 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 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 vertical installation of the AP. FPS recommends that the user allow 36.83 cm. (14-1/2 inches) of free vertical rack space for installation of an AP modified for horizontal installation.

2.2.1 Processor

The AP physically requires 62.23 cm. (24-1/2 inches) or vertical rack space when mounted vertically; it requires 36.83 cm. (14-1/2 inches) when mounted horizontally. It 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 when the AP is mounted vertically. This can be be occupied by the control panel, blank panels, or other equipment which does not fully occupy the space adjacent to the AP.

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. 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. 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. 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 FPS Schematic #680-2150-102 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 orginates 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 is visible. Rear mounting minimizes total rack space and reduces the chances of accidentally switching the AP 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 can be supplied in three different power configurations as specified in Table 2-1. The total power required for the AP 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 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 (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 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.

FPS specifies the AP 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. The AP, 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 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.

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.

3.2 ACCEPTANCE OF DELIVERY

The acceptance of delivery of the AP, unless otherwise negotiated, occurs at the factory. For insurance purposes, however, the shipping carton that contains the AP 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 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 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 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 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

Functional unit	Present	Absent
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 supplied) should be verified. Again, if shortages exist, these should be recorded and FPS should be notifed 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 voltge option is the one that was ordered.

3.4 RACK INSTALLATION

The rack installation of the AP follows a logical sequence. After checking the area, set aside for the AP 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 AP Mounting Hardware

(1) Set Rackslides.
(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 Rackslides.
(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 x 1/2 BHS	10-32 x 3/8 BHS	10-32 X 1/2 FHS	10-32 x 3/4 OHS	Lock Washers #10	Cup Washers #10	Flat Washers #10	10-32 Nut w/Lock Washer	Nut Plate 10-32	Captive Nut 10-32
Processor				12		12				12
Rackslides		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
w/Loss Allowance	7	22	17	22	30	22	60	7	5	26

These are to be packaged in plastic bags and packages with the rackslides 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 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 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 neccessary 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.

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 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.

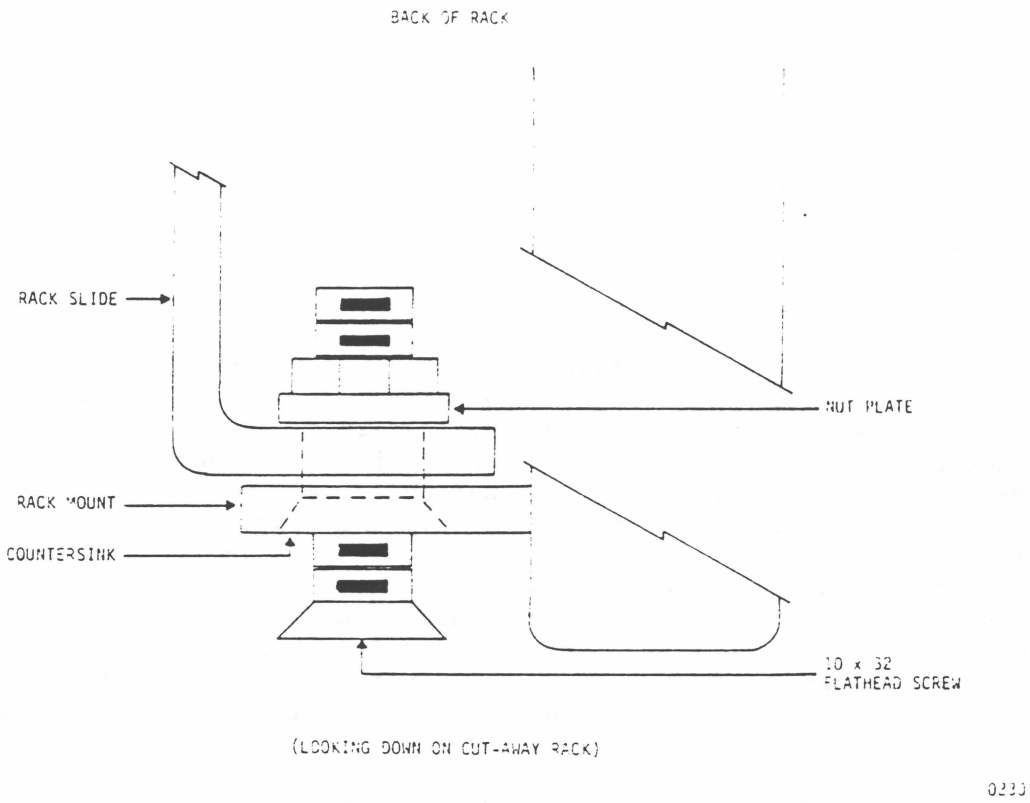


Figure 3-1 Rack Mounting Detail (Untapped)

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.

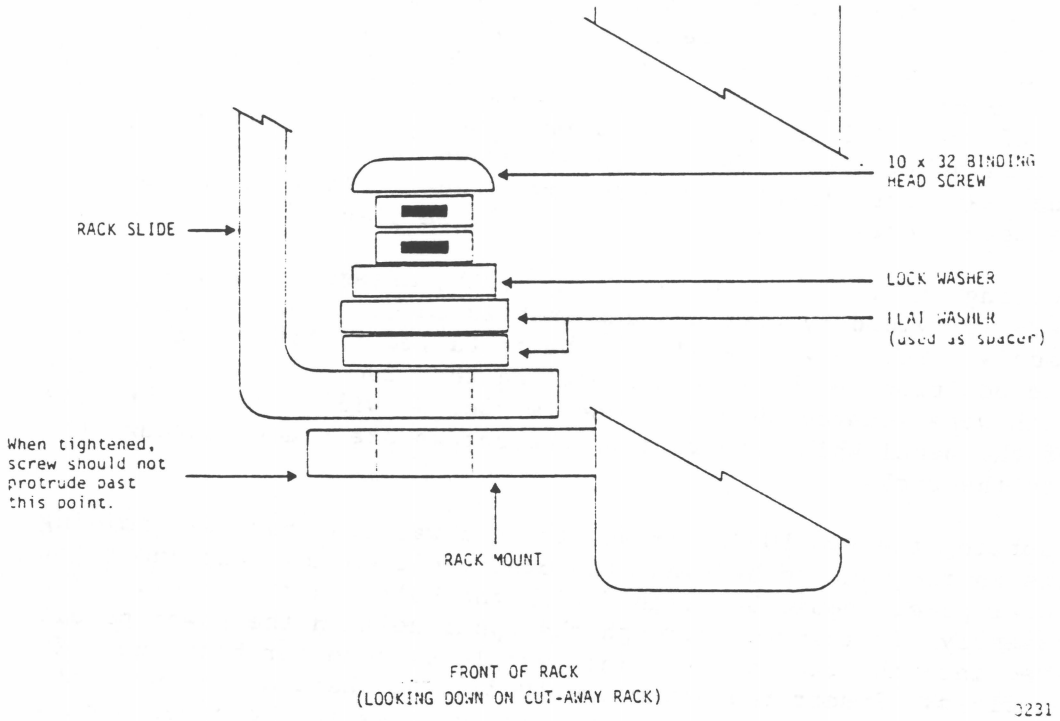


Figure 3-2 Rack Mounting Detail (Tapped)

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 attempting the 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 asembling 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 of the equipment groups 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. FPS Schematic #680-2150-102 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.

3.5.2 Power Panel to Processor

To prepare for the power panel hookup:

1) Slide out the processor.
2) Remove the AP retaining screws.
3) Lift off the front panel.
4) Slide the processor forward until the rack slides lock.

CAUTION

Once the front panel is removed, the user must be careful not to damage the wire-wrap pins on the mother board.

The power supplies for the AP-CT are user-supplied. These are a +5V, a -5V, and a +12V supply, as shown on FPS Schematic #680-2150-102. The line box is FPS-supplied. To connect these supplies:

1) Connect socket J11 on the I/O bracket at the bottom rear of the processor to the line box, as shown in FPS Schematic #680-2150-102. (From the front of the rack, socket J11 is on the left side of the processor near the fans.)

2) Connect socket J1, which is located just to the right of socket J11, to the power supplies, also as shown in FPS Schematic #680-2150-102.

3) Connect the four processor cables found on the lower part of the mother board to the +5VDC/150A or 120A power supply on the power panel. Note that the two cables marked red on the ends are for connection to the positive terminal; the two solid black cables are for connection to the negative terminal. Place one cable lug on each side of the appropriate bus-bar and secure it with a 10/32 binding head screw, lock washer, flat washer, and nut. Place a lock washer and a flat washer between the screw head and the first lug. Insert assembly through the bus-bar. Then place the second lug, flat washer and lock washer over

the appropriate bus-bar. Secure it in place by placing a lock washer between the 10/32 binding head screw mounting bracket and fastening it with a hex nut. Figure 3-3 shows how the cables are bolted to the bus-bar. After all cables are connected, tighten them securely and check to see that the positive (+) is not shorted or close to the negative (-) at this connection.

After the power supplies are connected, the host-to-AP cable is 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. However, general guidelines follow.

The host-to-AP cabling connects, 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. However, 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 is connected to the AP and not connected to the host until after the voltage levels are checked on the AP, as specified in Section 3.6. Further details and directions for the host-to-AP connection are contained in Section 3.6.3.

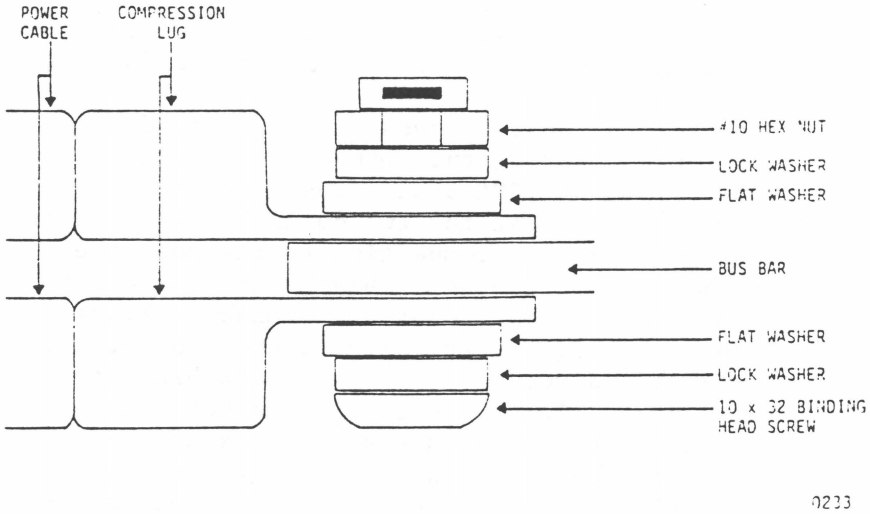


Figure 3-3 Bus Bar Bolting Detail

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 mother board. The four heavy cables on the mother board 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 mother board 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.

The -5 volts should then be checked. Directly to the left of the left +5 VDC and down slightly on the mother board, 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 + 0.05 volts. If this voltage does not fall within this specification it should be adjusted until it does.

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 of the mother board. If the voltage does not meet the above specification, it should be adjusted.

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 eight 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 PDP-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 has been installed into the users system, the system should be checked to verify that the integration of the AP 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 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 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 DIAGNOSTICS

FPS supplies a diagnostic software package with each AP. 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, 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 system operation, but are also used in isolating and correcting system deficiencies. This manual will give the user enough information to verify the AP system. If a further description of these programs and their input commands is necessary, FPS Manual #7284 AP Diagnostic Software Manuals should be consulted.

For verifying the operation of the AP, 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 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 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 and should be run for one hour to verify the system.

4.3.2 APPATH

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

4.3.3 APARTH

APARTH runs the arithmetic elements of the AP 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 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 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 memory, program source address 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 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 is shown in Figure 5-1.

Before attempting to understand detailed theory of operation of the AP 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.

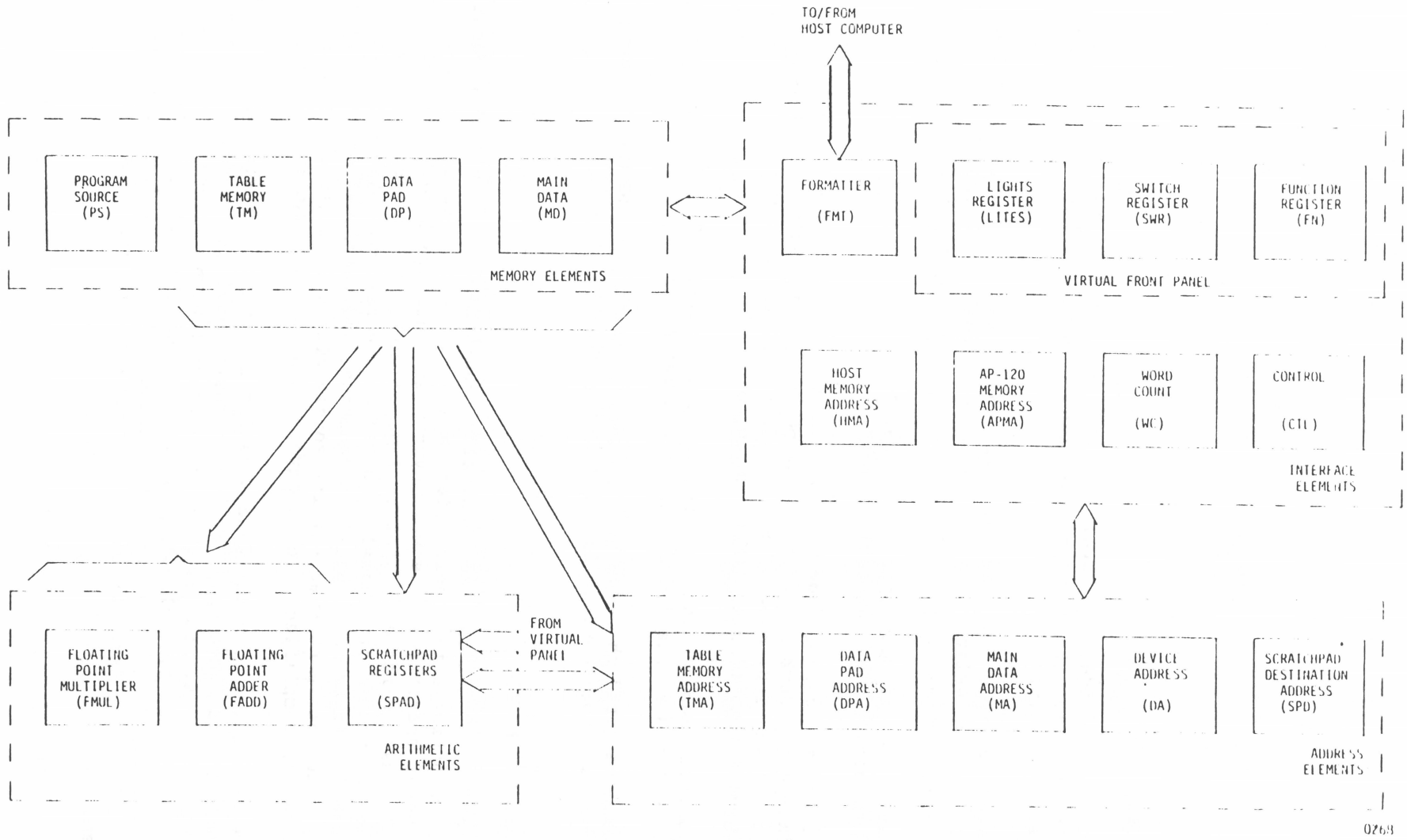


Figure 5-1 AP Simplified Block Diagram

5.2.1 Functional Units

The main units of the AP 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 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. 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 512-word bi-polar RAM.
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.

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 which normally handles the 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.

The MD memory is a MOS memory and is available in a standard 2 clock cycles interleaved version.

5.2.1.2 Arithmetic Elements

The AP 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 two clock cycles to perform an add operation, but a new add can be started every clock cycle.

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 225ns). 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 225ns, the result is not ready until 675ns 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 mutliple 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 and 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 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 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. The register may be incremented or decremented, or a new value may be loaded in from the scratchpad unit or the data pad bus.
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. 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 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 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. The host computer can either write into or read from this register. The program within the AP 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 registers. The lights register can be loaded, but not read by the AP. 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.

The DMA portion of the interface permits block transfers of data from the host computer to the AP 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 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, 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 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 always communicate with the main data (MD) memory, and only the main data 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 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 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 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 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 paths that are used to load the floating-point multiplier (FMUL) and the floating-point adder (FADD). The multiplier and adder paths 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 multiplier register M1.

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 multiplier register M2.

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 x 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 x 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 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.

The output of the SPAD can be enabled on to the DPBS. The program source output can be enabled on to the bus through the control buffer.

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)
- to the input of SPAD

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 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 detination address register (SPD)
AP 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 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 registers because the AP can use the PNL bus to move the contents of its memories or 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 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 (MD 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.

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 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.

5.2.2.6 Input/Output Bus

The input/output bus (IOBS) is a 38-bit, multiple-source, multiple-destination bus that permits communication between external devices and the AP internal elements. Data is driven tri-state on to the bus.

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 to an external device.

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 memory elements.

5.2.3 Examples of Data Flow

In order to understand the AP 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, DMA transfer and AP to host DMA transfer.

5.2.3.1 DMA Transfer - Host Computer to AP

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 Memory Address Register (APMA)	Must contain the starting address of the AP 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 first placing the data on the the host data bus (HD), setting up the multiplexer properly enabling the register, and clocking data into the register.

Once the registers have all been loaded, the AP 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. 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 DMA stops when the word count reaches zero. This indicates that the desired number of words has been transferred. If desired, the word count can be monitored during the transfer. 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 to Host Computer

A DMA transfer from the AP 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 memory elements. Once the registers are all loaded, different buses are used to send the register contents to appropriate locations.

The AP 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 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 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 (other than the main data and program source memory boards).

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 225nsec 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 225ns instruction cycle. This is accomplished by using the read address (DPA+XR—>Xnn) during the first portion of the cycle and the write address (DPA+XW—>Xnn) 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 Xnn.

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, 225ns 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.

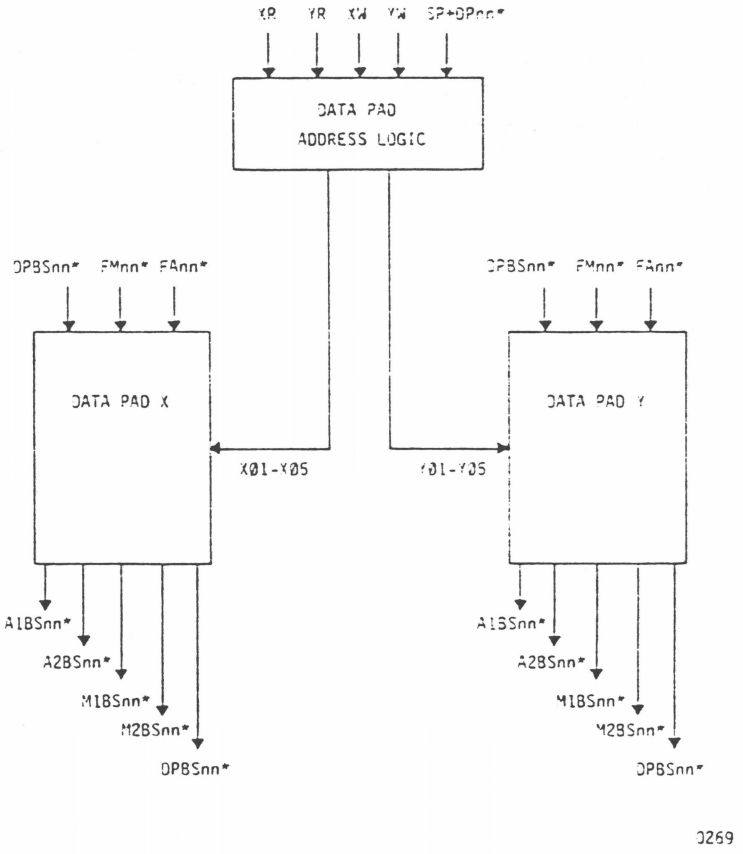


Figure 5-2 Data Pad System Block Diagram

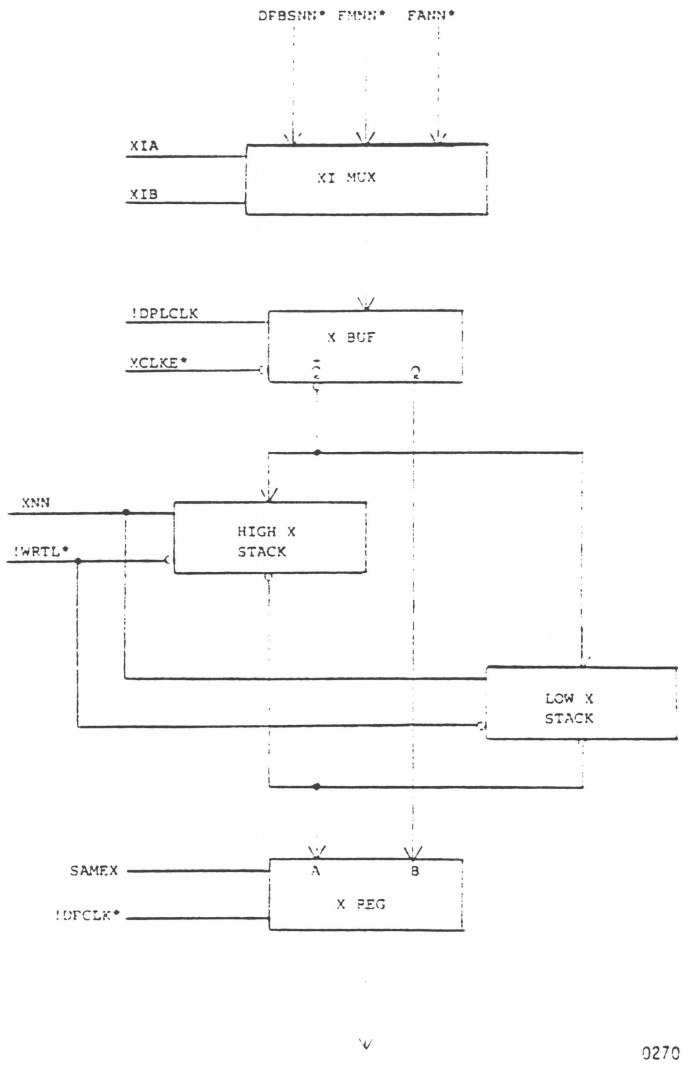


Figure 5-3 Block Diagram of DPX

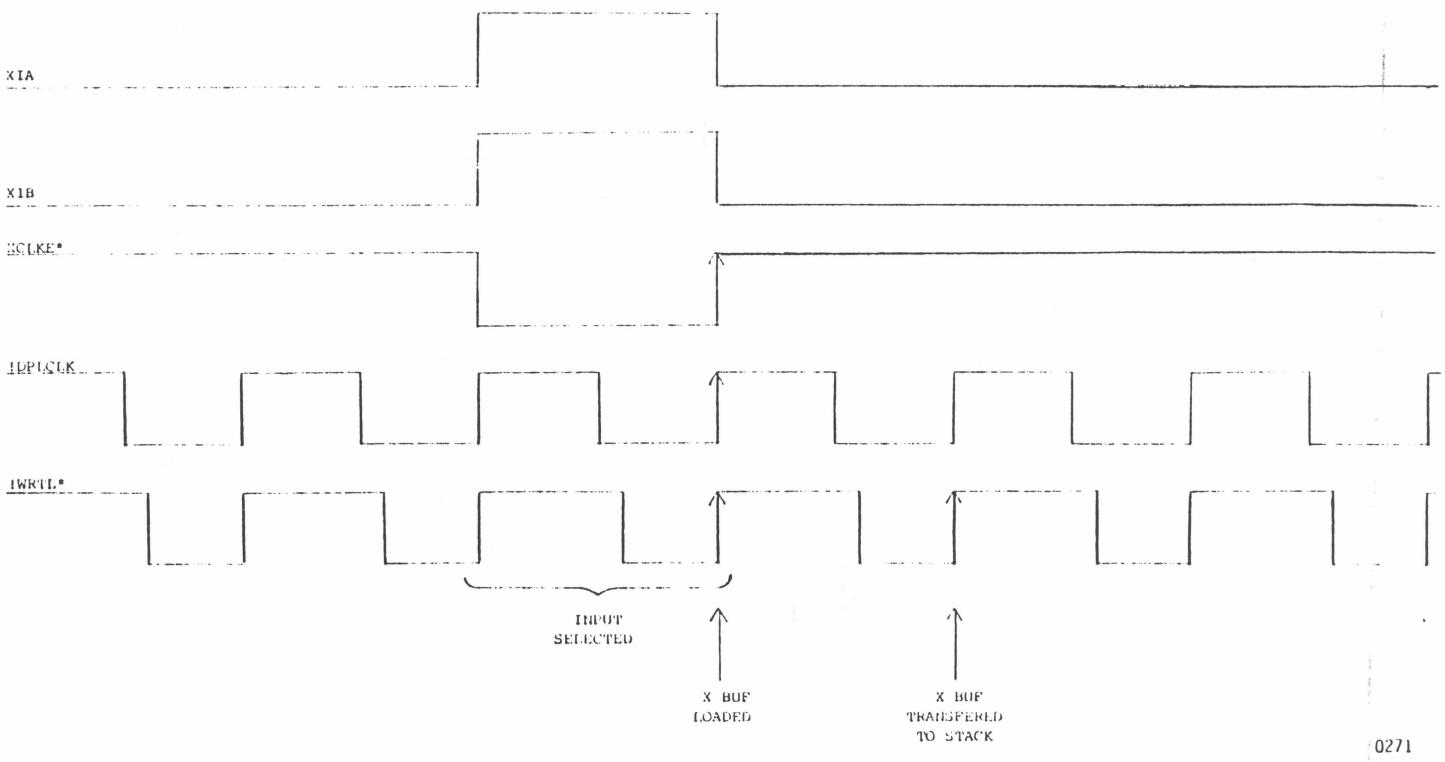


Figure 5-4 Timing Diagram of Writing DPX

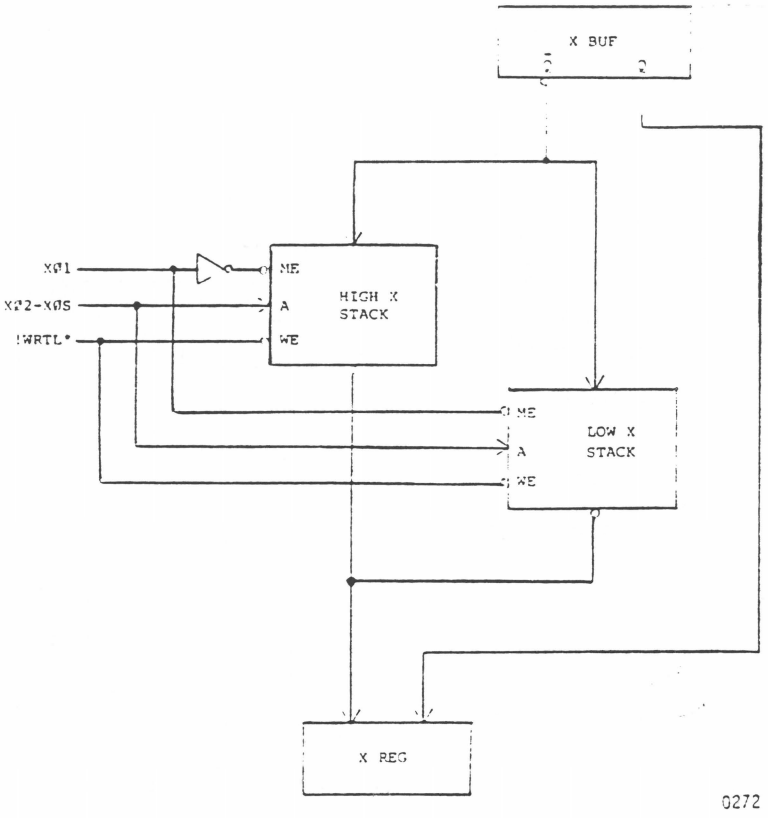


Figure 5-5 Detail Showing Stack Address

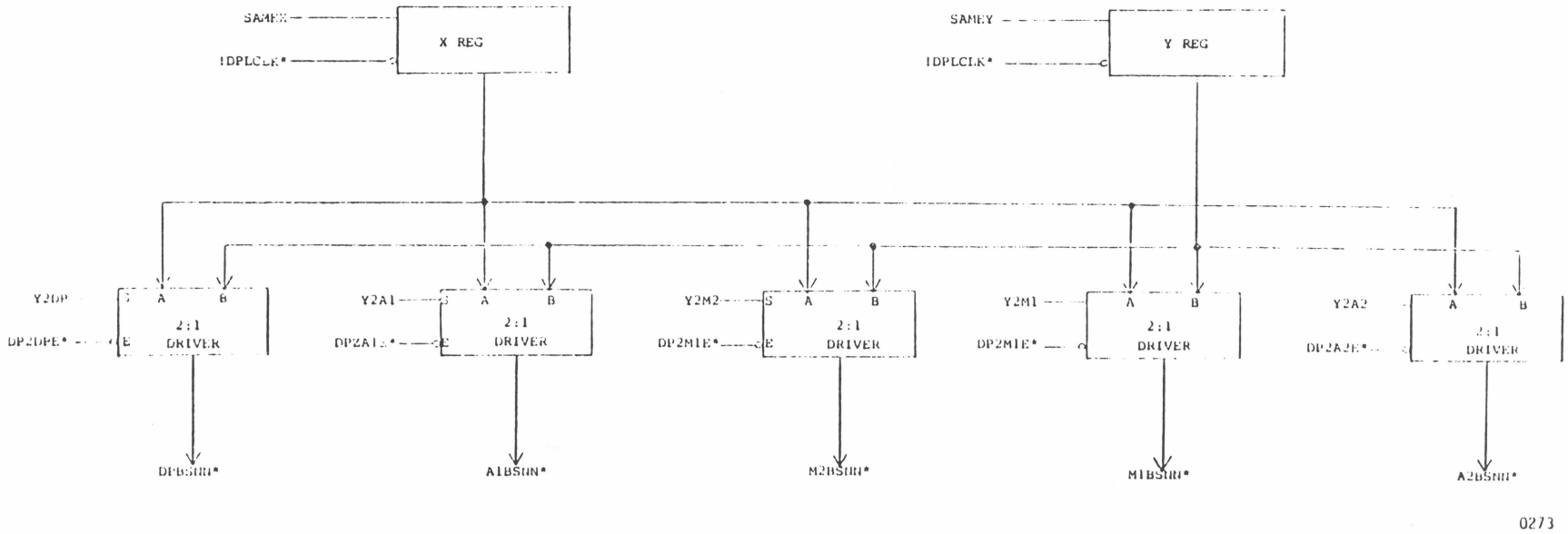


Figure 5-6 Data Pad Output Logic

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. Main data is available in 8K. Main data is available as standard memory. The figure shows how the main data system is interconnected in the AP.

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 225ns 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) 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.

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 MDnn* 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 MDnn* 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'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 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 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 MDCA1. 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 CB1 (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 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.

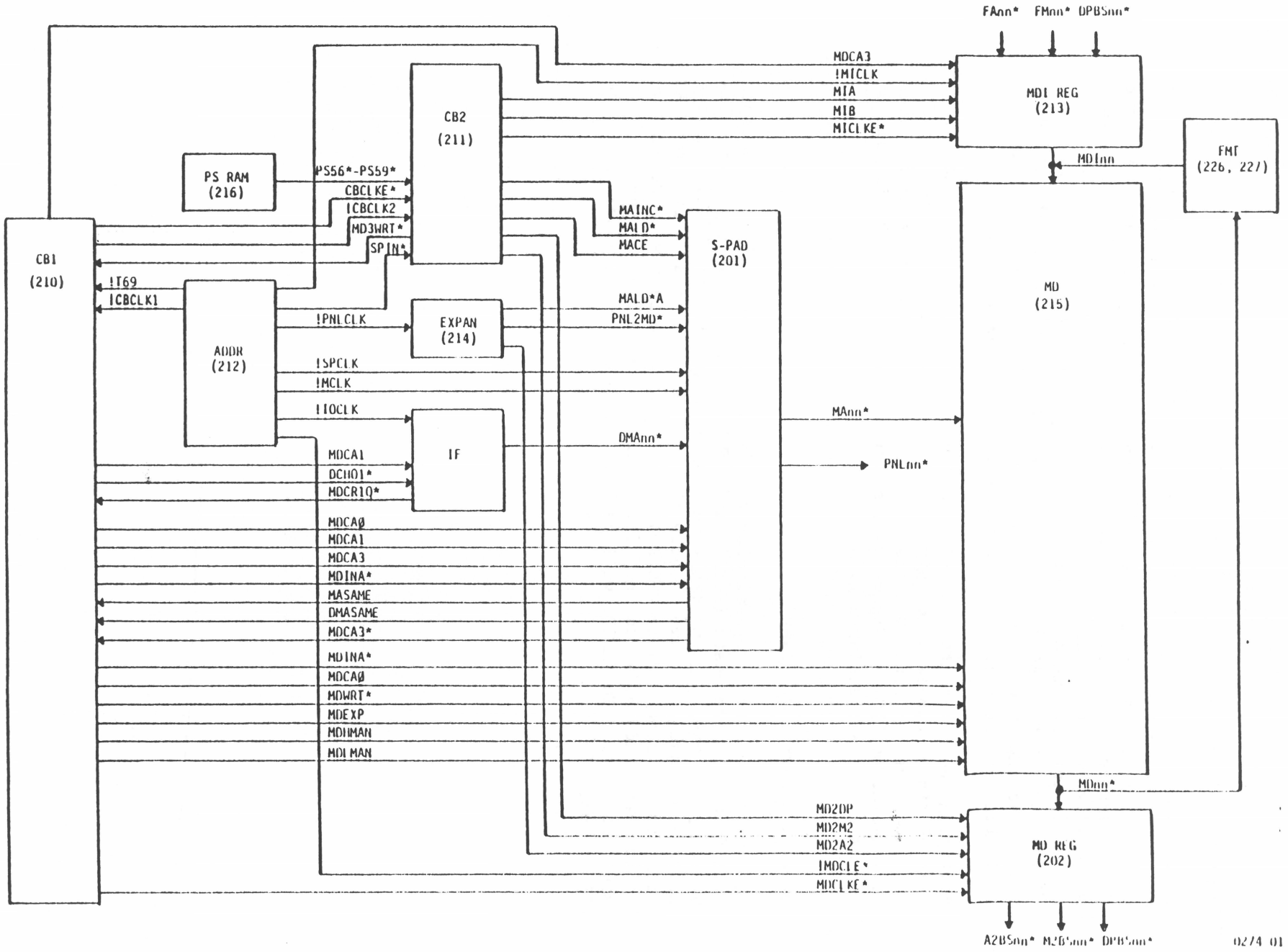


Figure 5-7 Main Data System Interconnection Block Diagram

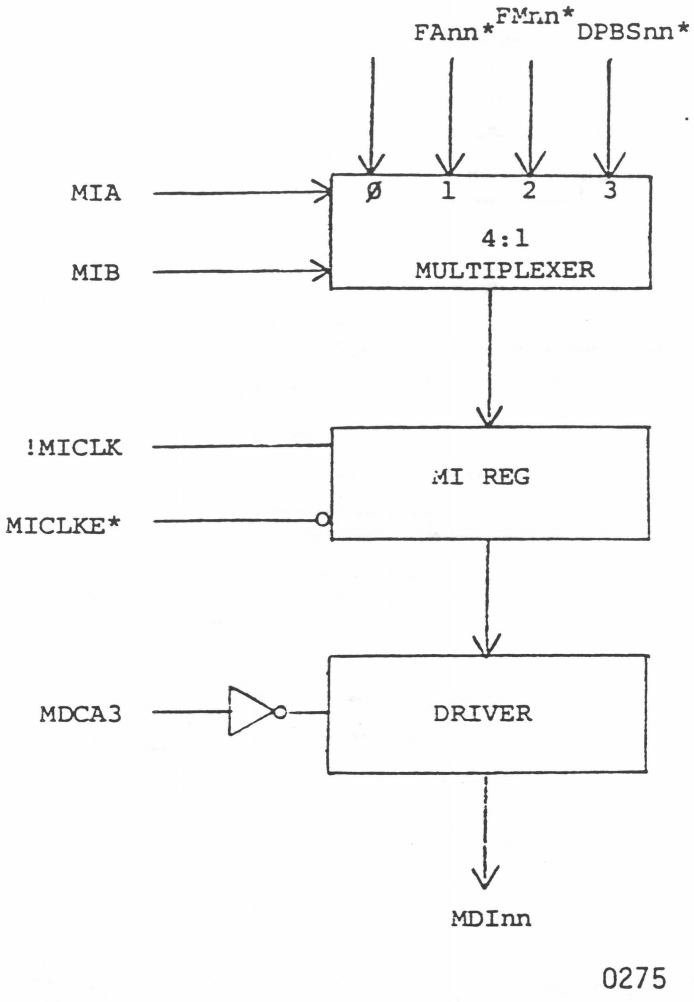


Figure 5-8 MI Reg Block Diagram

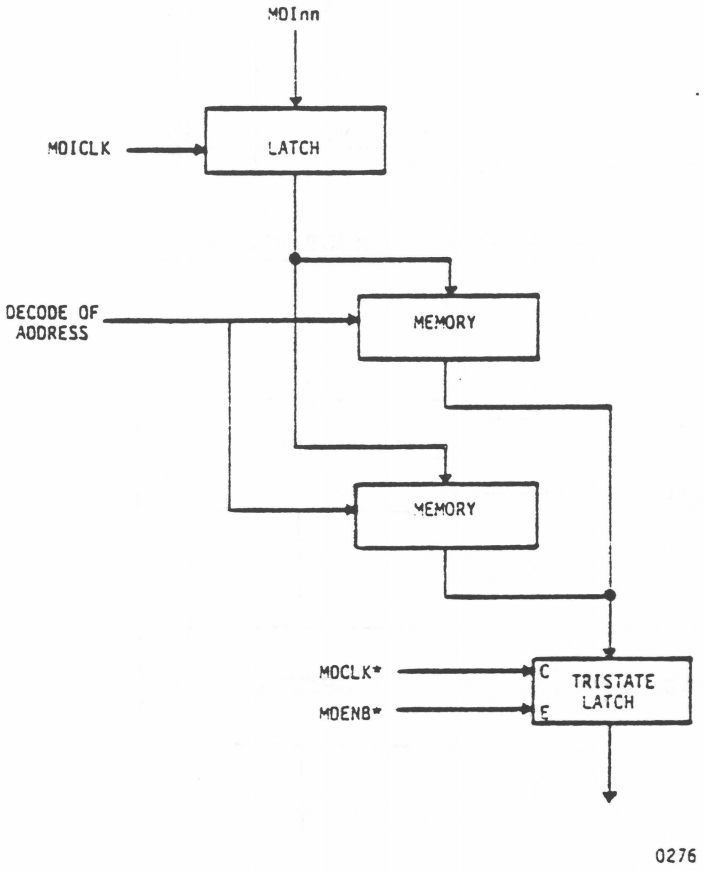


Figure 5-9 Main Data Memory Element Block Diagram

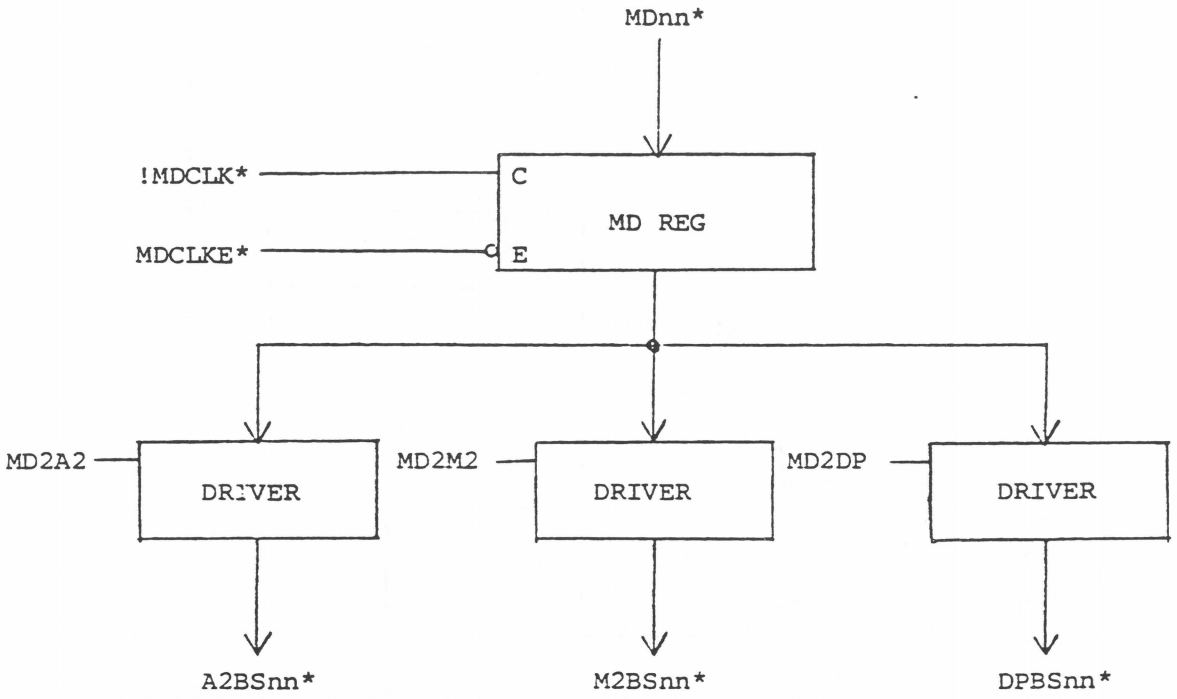


Figure 5-10 Main Data Register Block Diagram

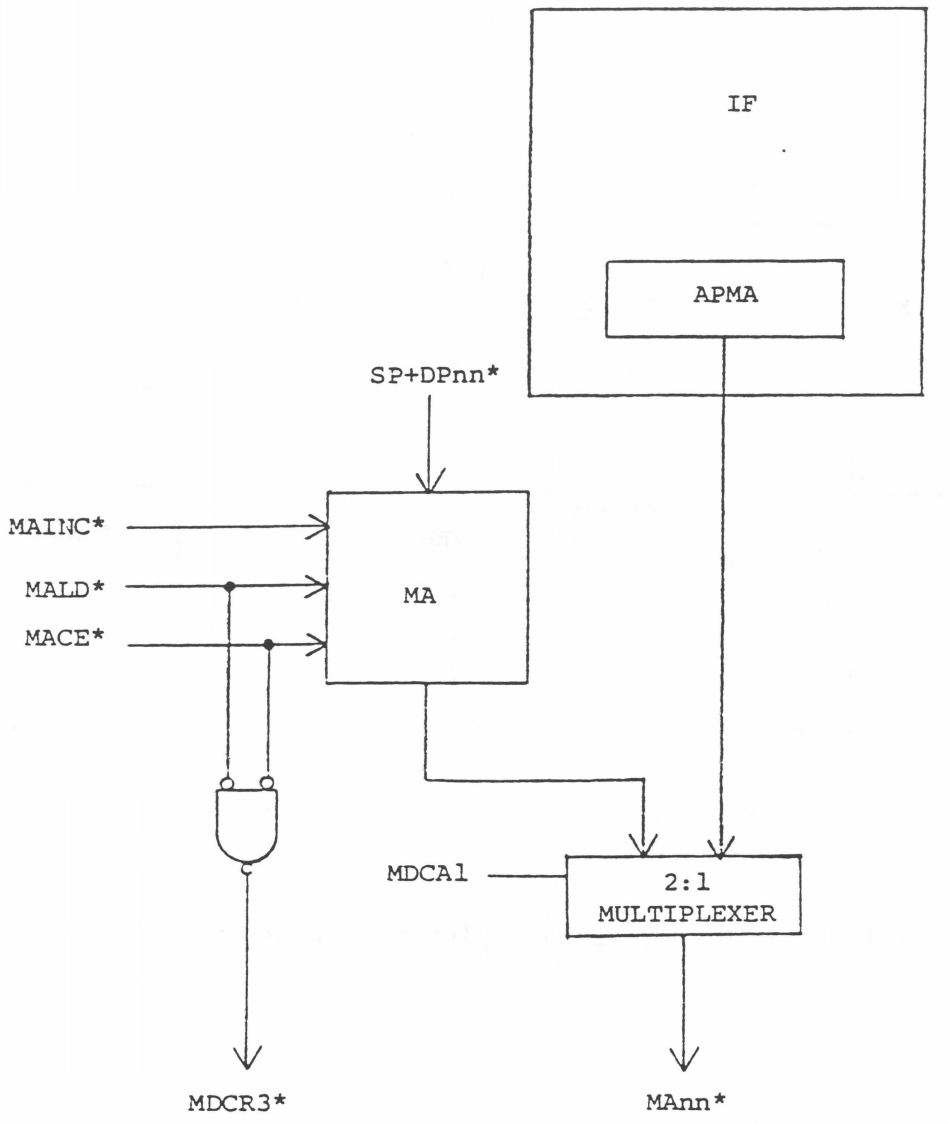


Figure 5-11 Memory Address Simplified Block Diagram

5.3.3 Table Memory

The memory element in the AP 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).

The output of the table memory is loaded in TMreg and can be enabled onto the M1BS, the A1BS, and the data pad bus.

Figure 5-13 shows how the table memory is interconnected in the AP 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 TMrom. 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↑

0279

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.

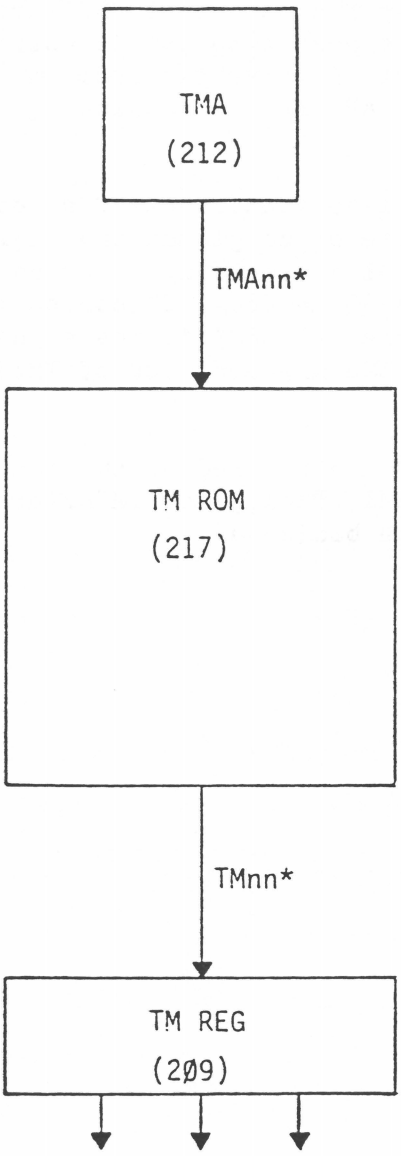
The read only memories (ROMs) are programmed prior to leaving the FPS factory. Therefore, utilizing TMrom is nothing more than selecting the data in the desired address. The output of the table memory address logic is applied to ECB 217 where it is decoded to enable and address the appropriate location 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, 225ns clock that is generated from and in phase with the system clock. As long as the AP 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(ses).



0280-01

Figure 5-12 Table Memory System Block Diagram

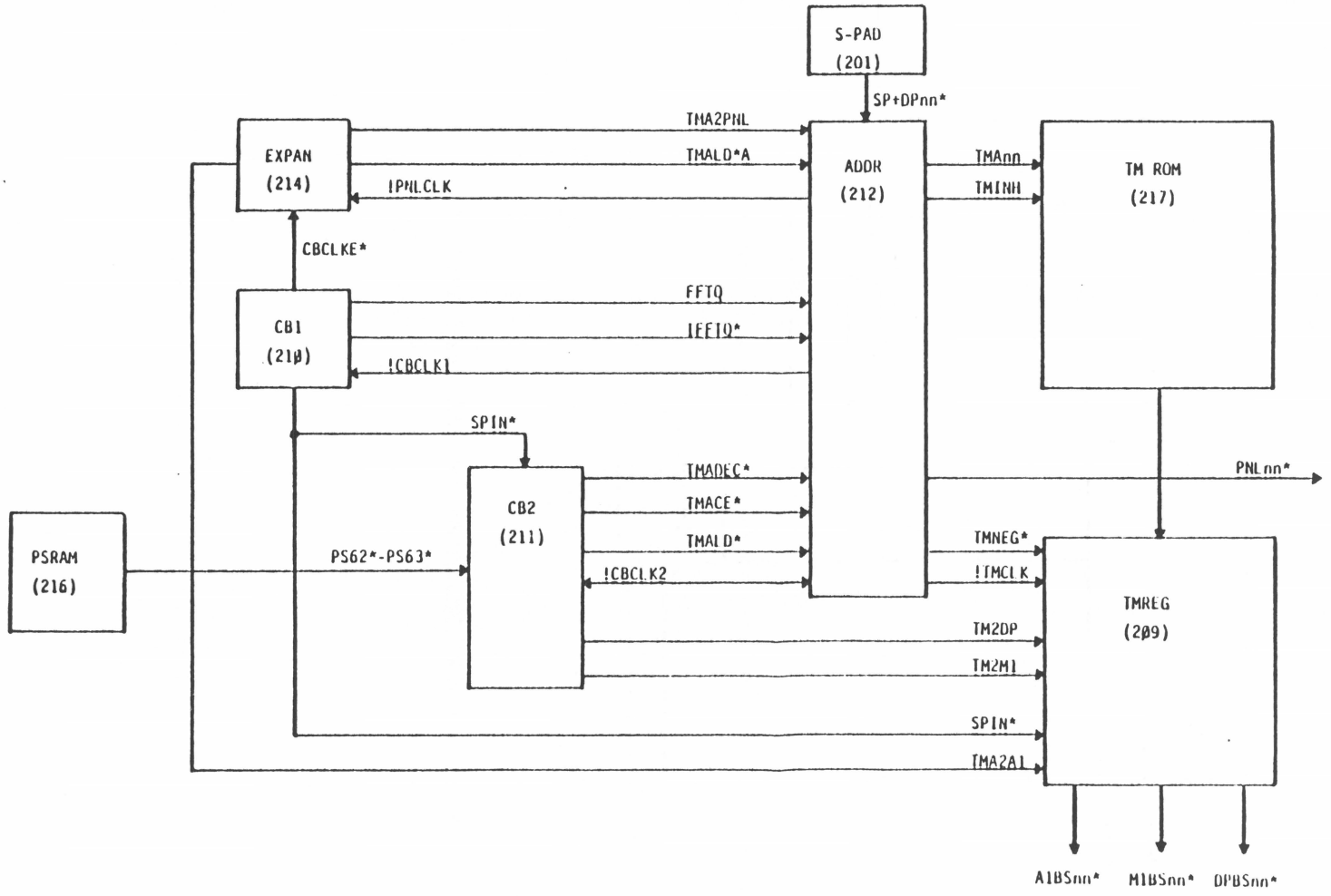


Figure 5-13 Table Memory ROM Interconnection Block Diagram

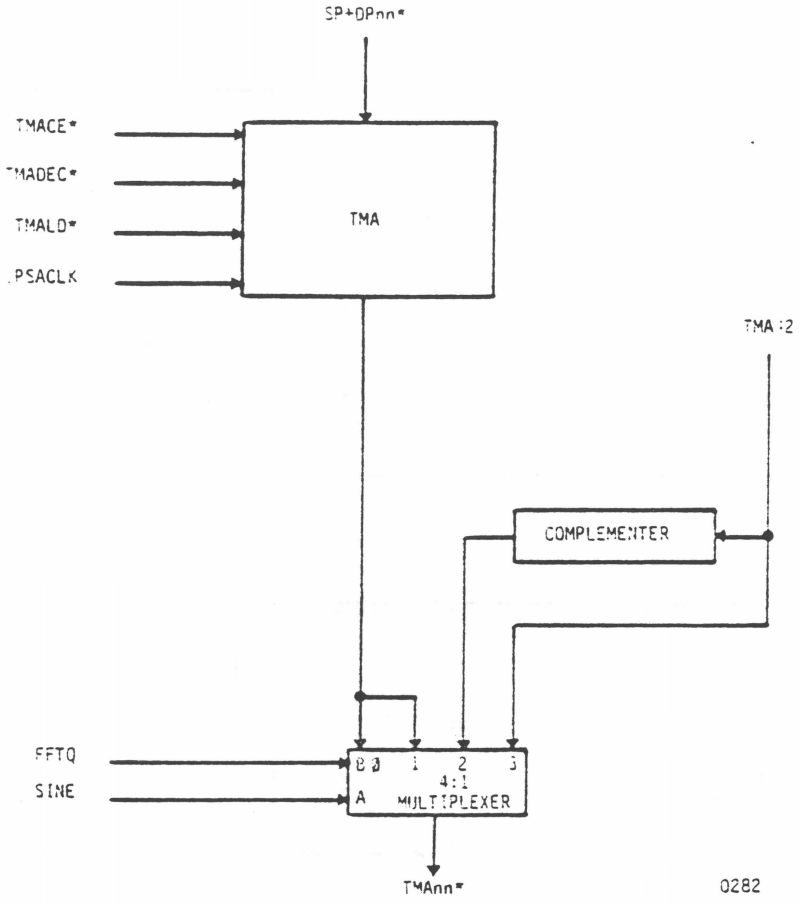


Figure 5-14 Table Memory Address Logic

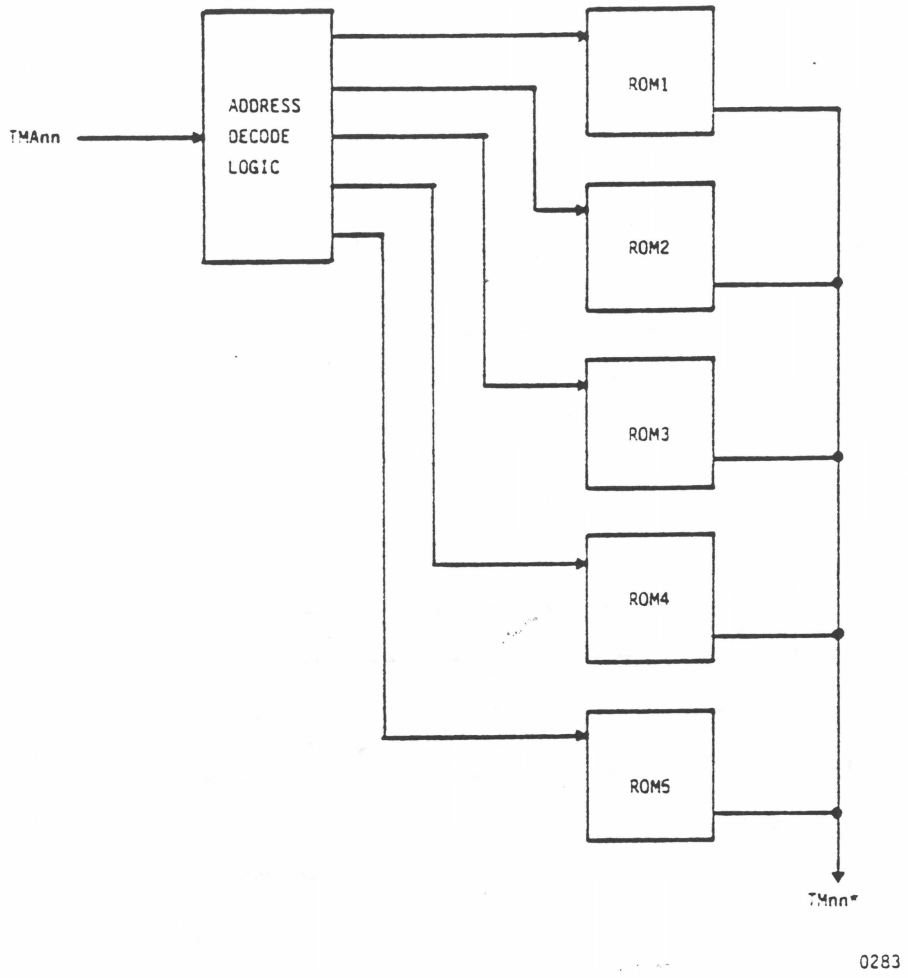


Figure 5-15 Table Memory ROM Block Diagram

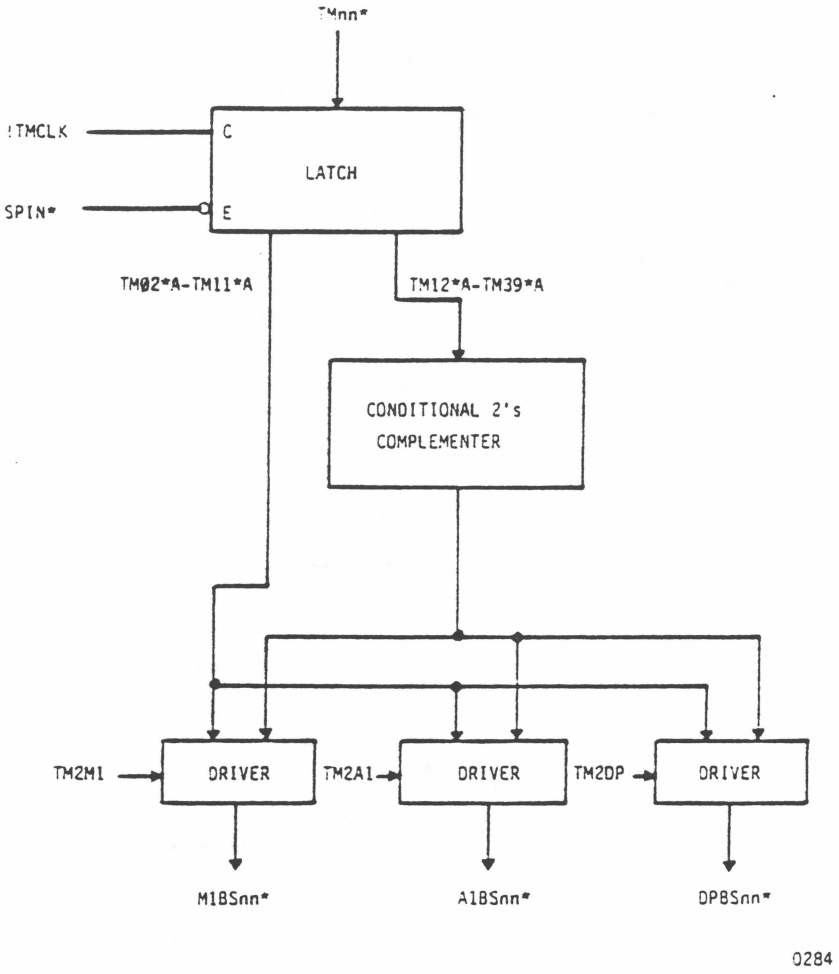


Figure 5-16 Table Memory Register Block Diagram

5.4 ARITHMETIC

The AP 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, 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 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:

$$2 \times 10^{6}$$

6 power (exponent)

significant integer

radix (base)

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

$$483 = 48300 \times 10^{-2}$$

$$483 = 4830 \times 10^{-1}$$

$$483 = 483 \times 10^{0}$$

$$483 = 48.3 \times 10^{1}$$

$$483 = 4.83 \times 10^{2}$$

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 x 2(7) or 0.11001 x 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:

$$FPN = x.M * B^{E}$$

The above expression can be broken down as follows:

x.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 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 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 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 x 21 or 1.11 x 22), 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 0 0 0 . 0 0 0 0

                       trailing
                        zeros

small number        0 0 0 0 . 0 0 0 1

                                 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:

$$0.1 \times 2^{7}$$

$$0.100001 \times 2^{6}$$

$$0.1 \times 2^{0}$$

$$0.1 \times 2^{-3}$$

Note that in each of the above cases, the most signficant 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:

$111. \times 2^{0}$	original number (unnormalized)
11.1×2^{1}	after first shift
1.11×2^{2}	after second shift
$.111 \times 2^{3}$	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:

$.0001 \times 2^{7}$	original number (unnormalized)
$.001 \times 2^{6}$	after first shift
$.01 \times 2^{5}$	after second shift
$.1 \times 2^{4}$	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, 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 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:

```
 +5
 -3
----
```

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:

```
 +5
 +7 (complement of -3)
----
```

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.

```
 +5
 +7 (complement of -3)
----
 1 2

carry
```

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
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)
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, 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 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 Sign	Magnitude	1's Complement Form 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 -2^3 (-8)	2^2 (4)	2^1 (2)	2^0 (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:

$$TV = (-d0 * 2^{N-1}) + \left(\sum_{i=1}^{n-1} di * 2^{N-(i+1)}\right)$$

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 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:

```
  0111    +7
+ 0111    +7
------    ---

  1110   +14
```

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.

	Sign bit			
Sign	00111	+7	11001	-7
Extension	00111	+7	11001	-7
	01110	+14	10010	-14

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 tyical 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, 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 (29)	256 (28)	128 (27)	64 (26)	32 (25)	16 (24)	8 (23)	4 (22)	2 (21)	1 (20)	
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 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 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 x 23, 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.

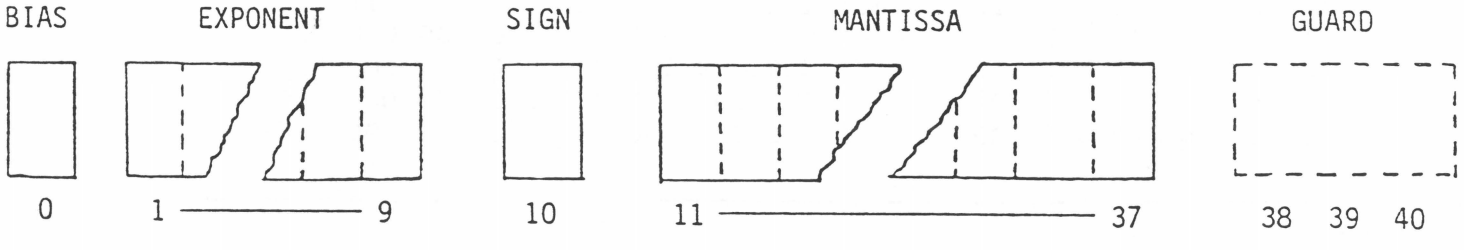


Figure 5-17 Internal Floating-Point Format

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	$-.9999 * 10^{99}$	-.9999E99
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. 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 Internal Floating-Point Numbers

	Exponent	Sign	Mantissa
Zero	0 000 000 000	0	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
-1.0	1 000 000 001	1	100 000 000 000 000 000 000 000 000 000
-1.0	1 000 000 000	1	000 000 000 000 000 000 000 000 000 000
AP Max	1 111 111 111	0	111 111 111 111 111 111 111 111 111 111
-AP Max	1 111 111 111	1	000 000 000 000 000 000 000 000 000 001
AP Nmax	1 111 111 111	1	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
AP Nmin	0 000 000 000	1	100 000 000 000 000 000 000 000 000 000

As stated in a previous section, biasing is used in the AP 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 adds a constant of 512 to its exponents. This 512 is the bias value.

Unfortunately, the bias used in the AP 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, 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 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.

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 4-4 as two equivalent representations for -1.0. Because the largest positive mantissa is all 1's (approximately .9999999 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 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 format (-1/2). This can be seen in the table as the first representation of -1.0.

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

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

$X1 = 165.2 \text{ (or } .1652 \times 10^{3} \text{)}$

$X2 = 21.00 \text{ (or } .2100 \times 10^{2} \text{)}$

	Fixed-Point Method	Floating-Point Method: Before Alignment	Floating-Point Method: 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 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 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 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) 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 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

$$X1 = .9 \times 10^{9}$$

$$X2 = .9 \times 10^{9}$$

Addition:

$$\begin{array}{ll} X1 & .9 \times 10^{9} \\ X2 & .9 \times 10^{9} \\ \hline \text{sum} & 1.8 \times 10^{10} \end{array}$$

$$\text{corrected} \quad .18 \times 10^{10}$$ (mantissa right shifted; exponent incremented)

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:

$$1.8 \times 10^{9}$$ (mantissa overflow)

$$0.8 \times 10^{9}$$ (no mantissa overflow)

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.

0.111×2^{4} (normalized)

1.110×2^{3} (unnormalized)

0.0111×2^{5} (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:

1.000×2^{-4} (normalized)

0.001×2^{-3} (unnormalized)

1.1000×2^{-5} (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 x 10x is an overflow condition that is corrected by a right shift (.18 x 105).

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 mantissa).

When normalizing mantissas, it is possible to create an exponent overflow or underfow 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

$$X1 = .9 \times 10^{99}$$
$$X1 = .9 \times 10^{99}$$

Addition:

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

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:

$$.18 \times 10^{100}$$

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, 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:

$$.99 \times 10^{99}$$

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

$$X1 = .90 \times 10^{-99}$$
$$X2 = .89 \times 10^{-99}$$

Subtraction:

$$\begin{array}{rl} & .90 \times 10^{-99} \\ -X2 & .89 \times 10^{-99} \\ \hline \text{sum} & .01 \times 10^{-99} \end{array}$$

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:

$$0.1 \times 10^{-100}$$

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, the system forces the mantissa to zero and the exponent to its largest negative value. In this example, the result would be:

$$.00 \times 10^{-99}$$

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 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 opertion is first normalized. Once normalized, the result can then be rounded.

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

$$X1 = .9999 \times 10^{5}$$

$$X2 = .5100 \times 10^{1}$$

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:

$$\begin{aligned} X1 &= .9999 \times 10^{5} \\ X2 &= .000051 \times 10^{5} \\ \text{sum} &= .999951 \times 10^{5} \end{aligned}$$

At this point, the problem states that the sum is to be rounded. Becuase, 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:

sum of X1 + X2	$.999951 \times 10^{5}$	
rounding constant	.0000499	
	1.0000000×10^{5}	
result		

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:

$$.10000 \times 10^{6}$$

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.

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 mutiplication 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:

0's added	0'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 shifted two places
1 1 0 0	1 1 0 0
----------	----------
1 0 0 1 1 1 0 0	1 0 0 1 1 1 0 0

5.4.5.2 Mutliplication of Floating-Point Numbers

Whenever two numbers with exponents are multiplied, the numerical values are multiplied and the exponents are added. Thus, 32 x 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 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 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^{5}
0 0 1	0 1 1 0	6×2^{1}
-----	-------	
	0 0 0 0	
	0 1 0 1	
	0 1 0 1	
	0 0 0 0	
-----	-----------	
1 1 0	0 0 1 1 1 1 0	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	1 1 0	0 1 0 1	5×2^{6}
complemented	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	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.
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

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

Find 1st partial product (add multiplicand)

Shift left and add 0 to previous product

Shift left and add 0 to previous product

Shift left and add 0 to previous product

Shift left and add 0 to previous product

Shift left and add 0 to previous product

Shift left and add multiplicand to previous product

Faster Method

1 0 1 1 1 0 1

1 0 0 0 0 0 1

Find 1st partial product (add multiplicand)

Shift six places to left and add multiplicand to the previous product

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
---  ---  ---        added starting bit

 1    0    1    0
```

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

```
0 1 1 0      6
1 0 1 1     -5
```

Add 0 to the right of the multiplier

```
0 1 1 0
1 0 1 1 0
```

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

```
0 1 1 0
1 0 1 1 0
```

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

```
0 1 1 0
1 0 1 1 0
1 1 1 1 0 1 0      1st partial product
```

sign extend — 2's complement

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

```
0 1 1 0
1 0 1 1 0
1 1 1 1 0 1 0
0 0 0 0 0 0        2nd partial product
```

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

```
0 1 1 0
1 0 1 1 0
1 1 1 1 0 1 0
0 0 0 0 0 0
0 0 1 1 0          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

```
0 1 1 0
1 0 1 1 0
1 1 1 1 0 1 0
0 0 0 0 0 0
0 0 1 1 0
1 0 1 0            4th partial product
```

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

```
0 1 1 0
1 0 1 1 0
1 1 1 1 0 1 0
0 0 0 0 0 0
0 0 1 1 0
1 0 1 0
1 1 0 0 0 0 1 0    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:

```
                          0 1 1 0    6
 comparisons      notation
                          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:

```
                          0 1 1 0      6
                          1 0 1 1 0   -5
                          ---------

    1 0              K-X   1 1 1 1 0 1 0
    1 1              K+0   0 0 0 0 0 0
    0 1              K+X   0 0 1 1 0
    1 0              K-X   1 0 1 0
                           -------------

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

                           Shift left the number of bit posi-
0 0 0 0 0 1 1              tions required by the string of 0's
--------------------       then add multiplicand to running
                           partial product when a 1 is en-
0 0 0 0 0 1 1 0 1 1   +27  countered. This provides the final
                           product.
```

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 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 complementer, 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. 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 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 225ns (one cycle time), although the add operation itself requires 450ns. 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.

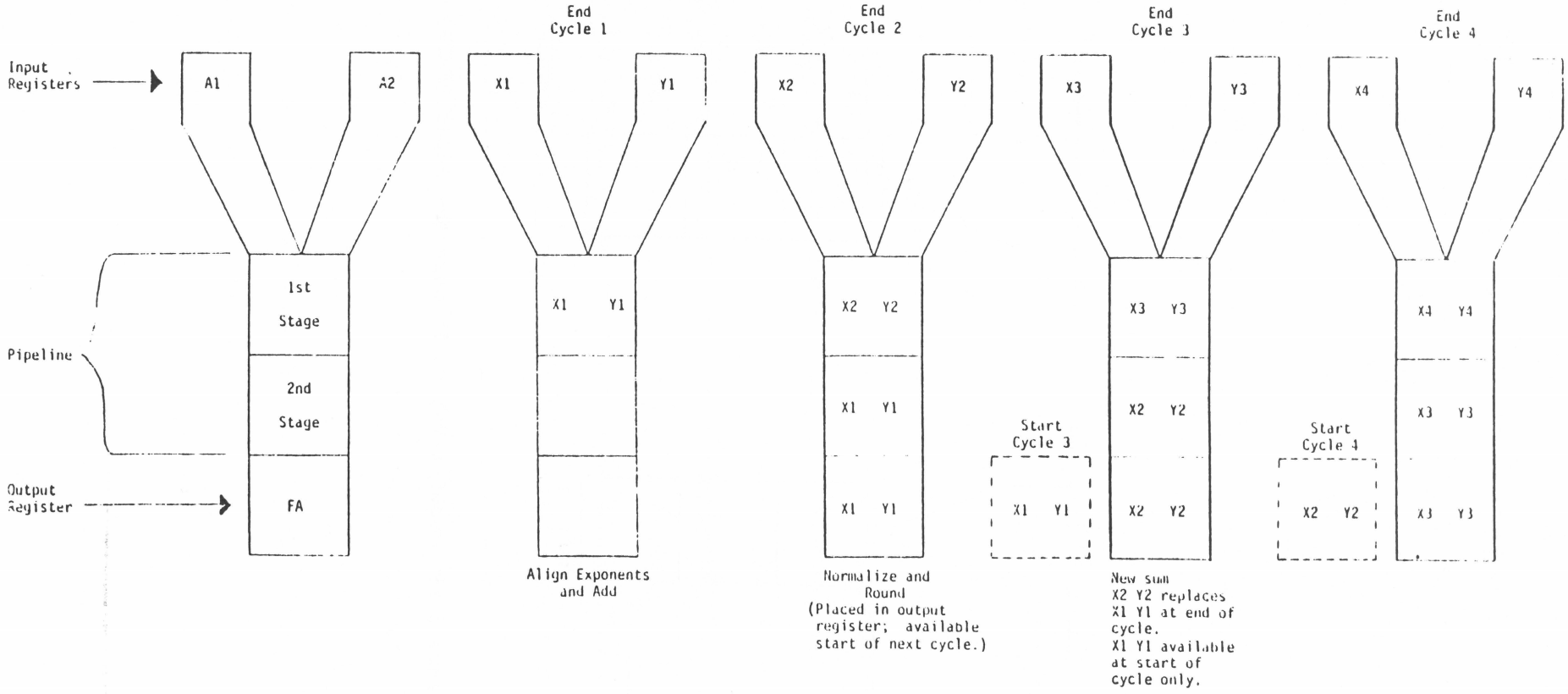


Figure 5-18 Adder (FADD) Pipeline Operation

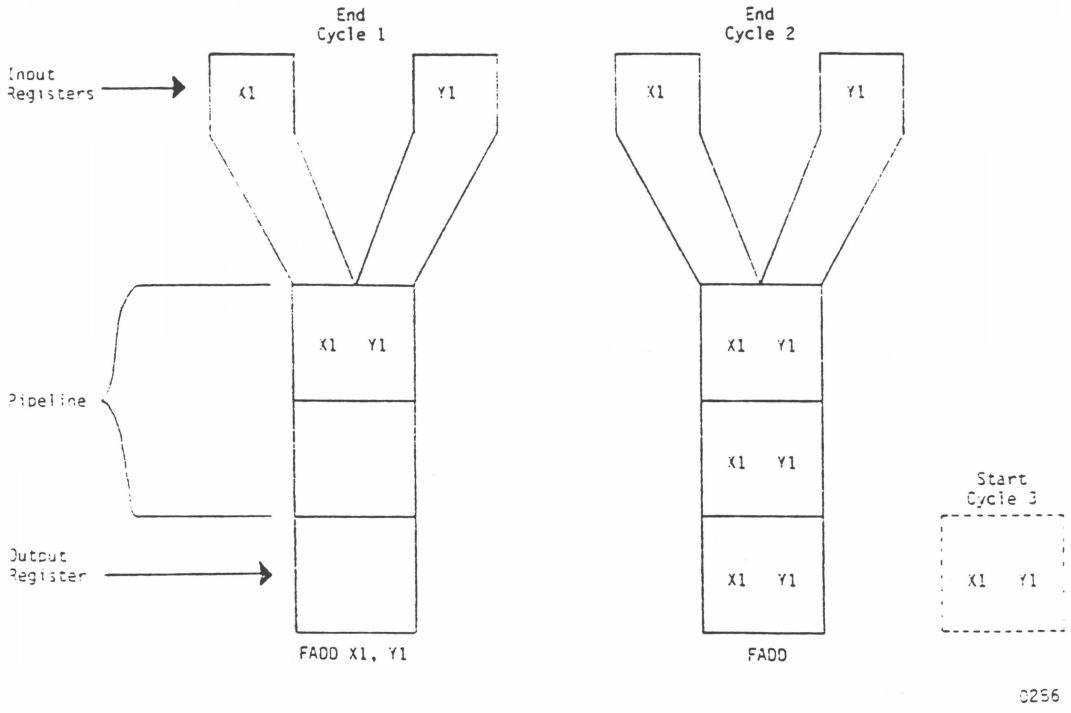
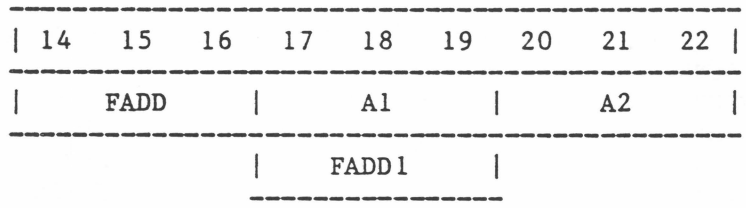


Figure 5-19 Pushing Values Through the Adder (FADD)

5.4.6.2 FADD Instructions

The floating-point addition element in the AP (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		A 2	
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 (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 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 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 FAnn* 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 diagramatically 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 225nsec 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.

SELA1 is the sign of the difference between A1 and A2. If the difference between A1 and A2 (A1-A2) is positive, then SELA1 will be high and IM2* will be high, and IM2* 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 SELA1 is low, the sign of A1-A2 is negative and IM1* is selected as the positive difference between the exponents and used as the shift count.

SELA2 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. SELA2 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 SELA2 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 Dnn* in two stages. The first stage conditionally right-shifts Dnn* 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 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.)

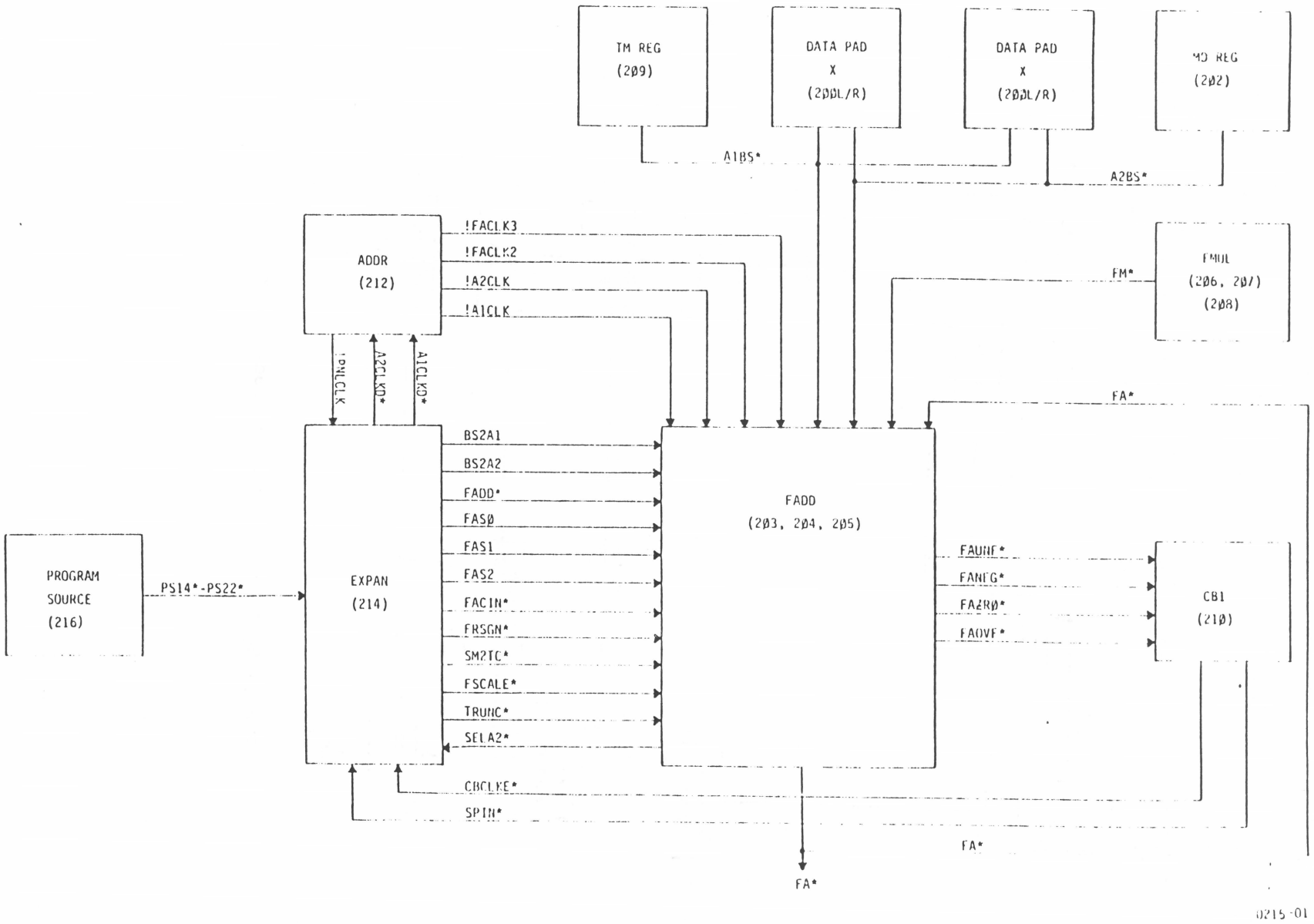


Figure 5-20 Interconnection Block Diagram

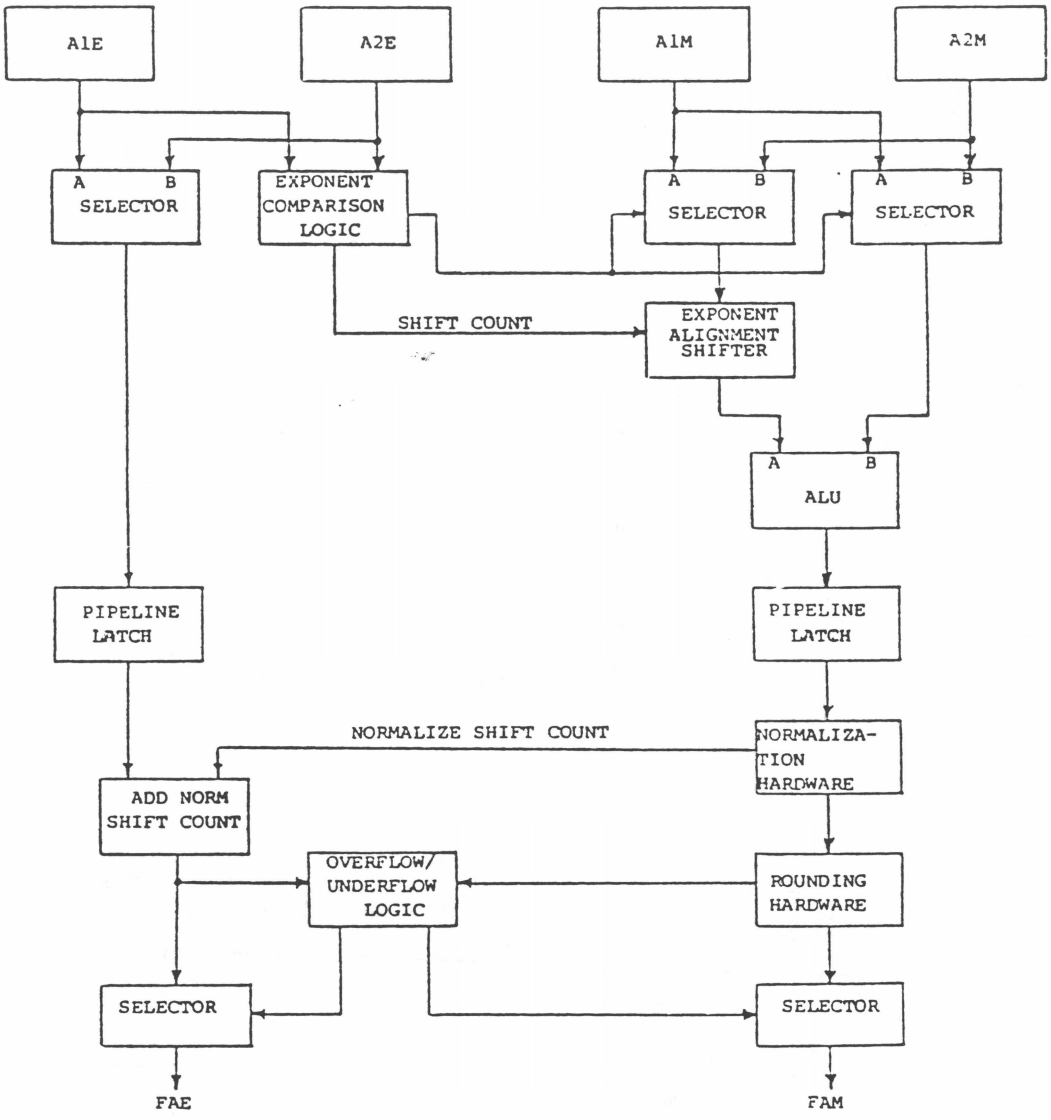


Figure 5-21 FADD Hardware Block Diagram

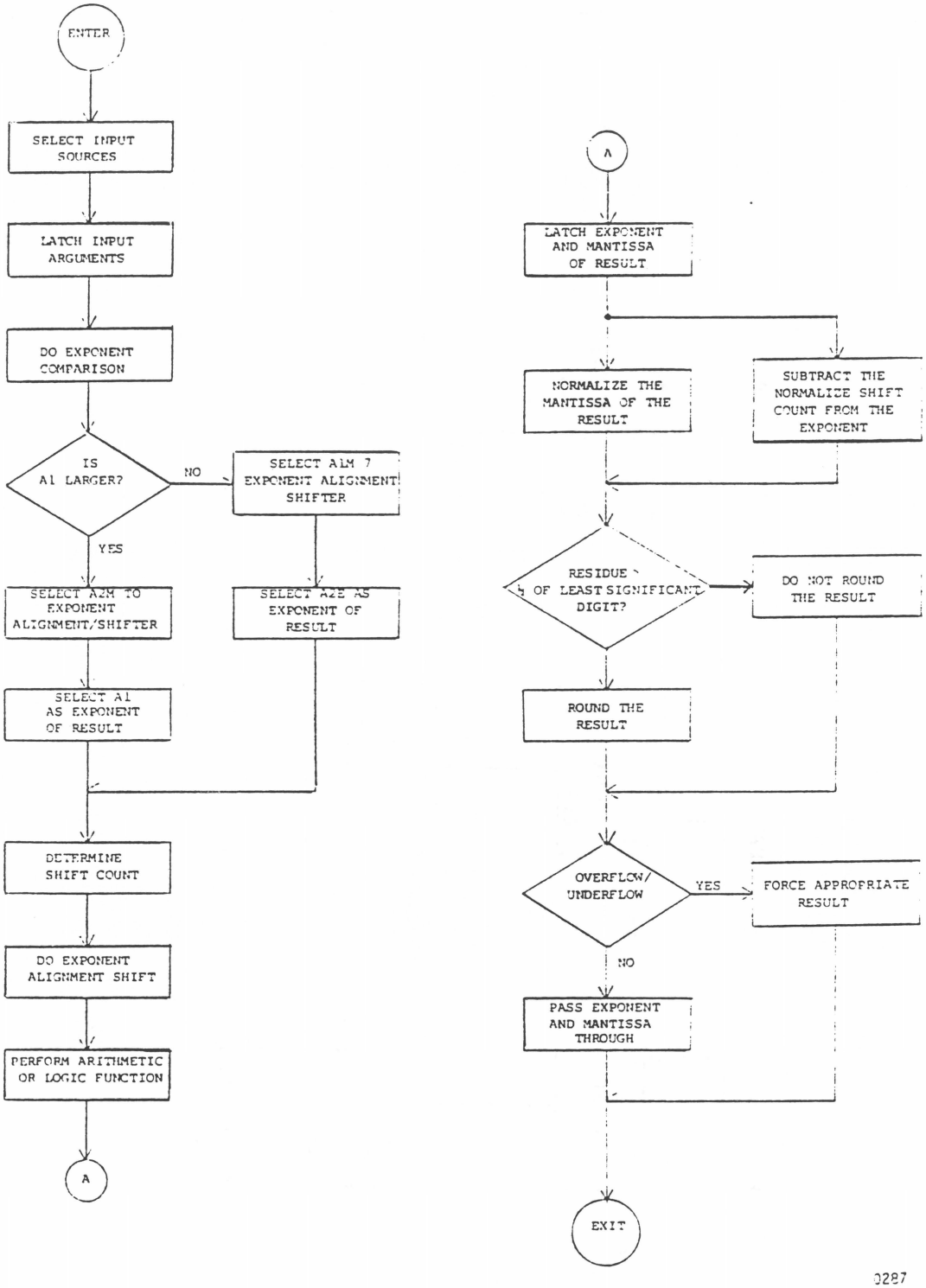


Figure 5-22 Flow Chart AP Floating Adder Logic

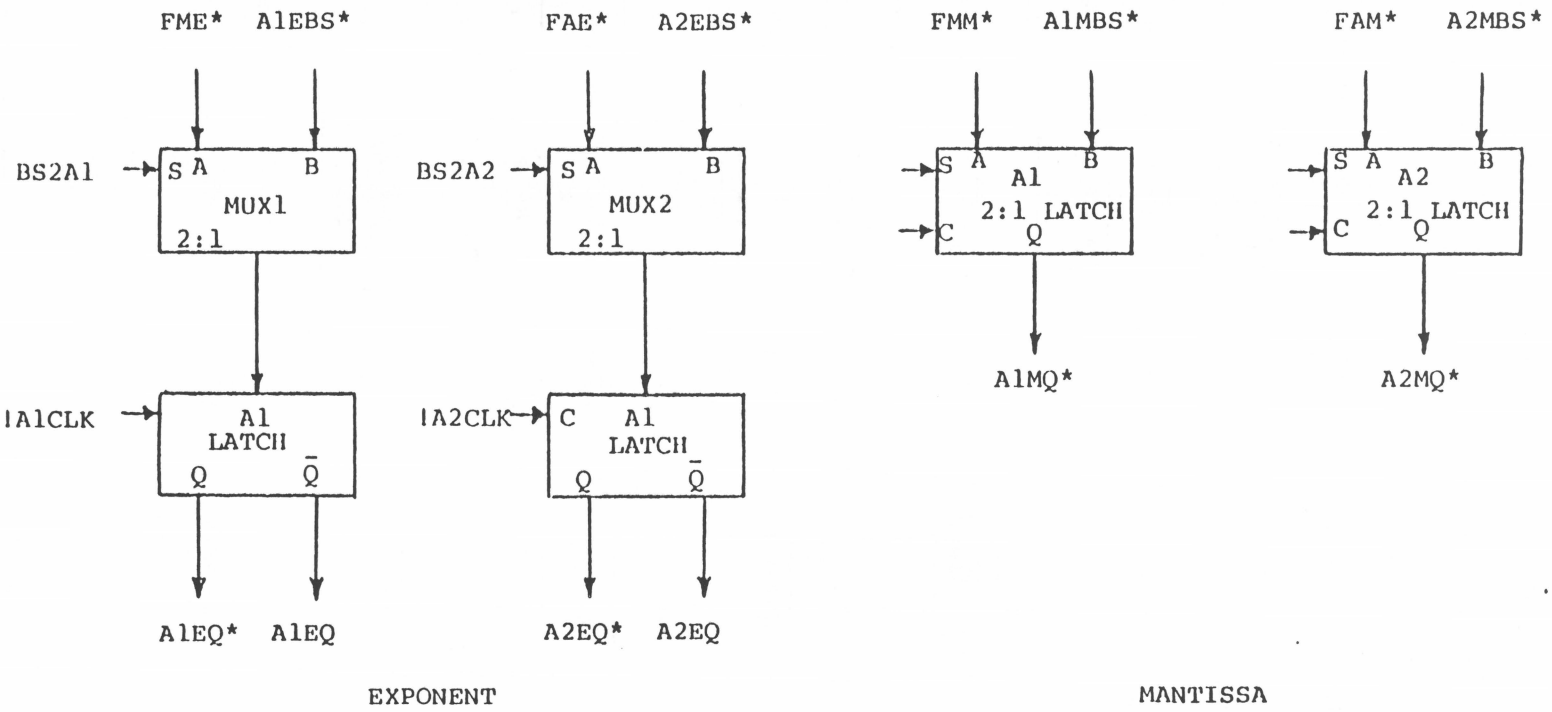


Figure 5-23 Floating Adder Input Latches

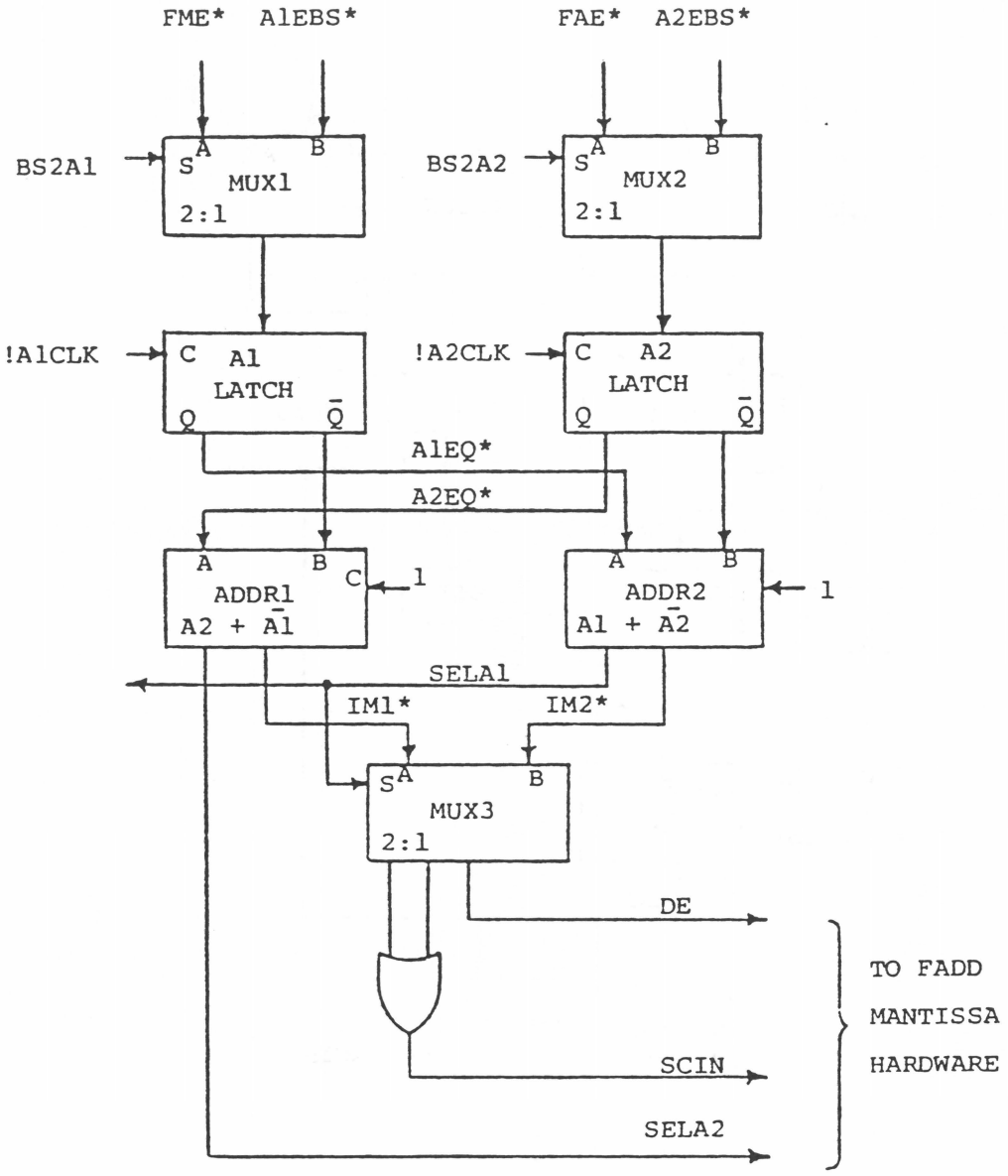


Figure 5-24 Input Latches and Exponent Comparison Logic

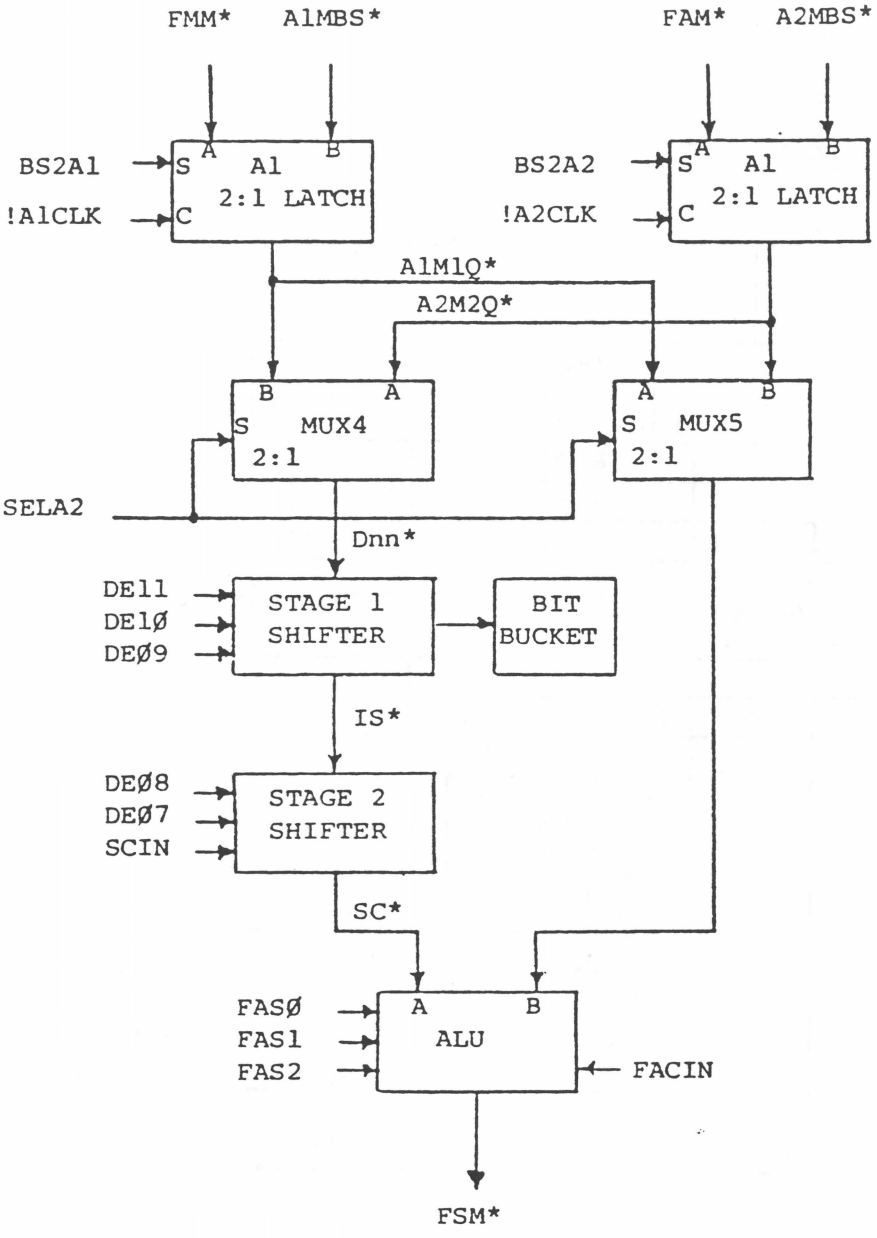


Figure 5-25 Exponent Alignment Logic

5.4.7 FMUL Hardware

The AP 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 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'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 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, FMOUF* and FMUNFL*. These signals are sent to the AP 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.

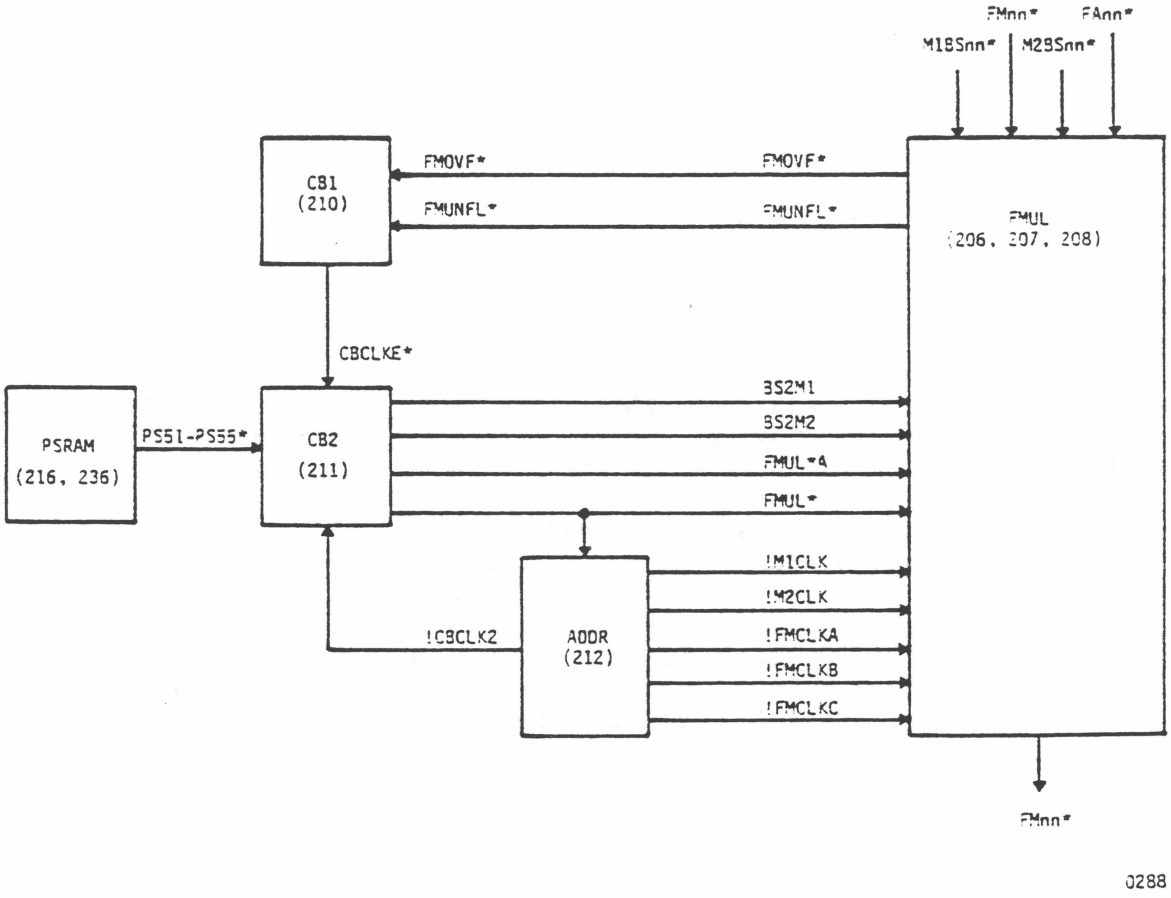


Figure 5-26 Floating Multiplier Interconnection Block Diagram

5.4.7.1 FMUL Overview

The floating-point multiplier (FMUL) used in the AP 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 225ns (one cycle time), although the multiply operation itself requires 675ns. 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.

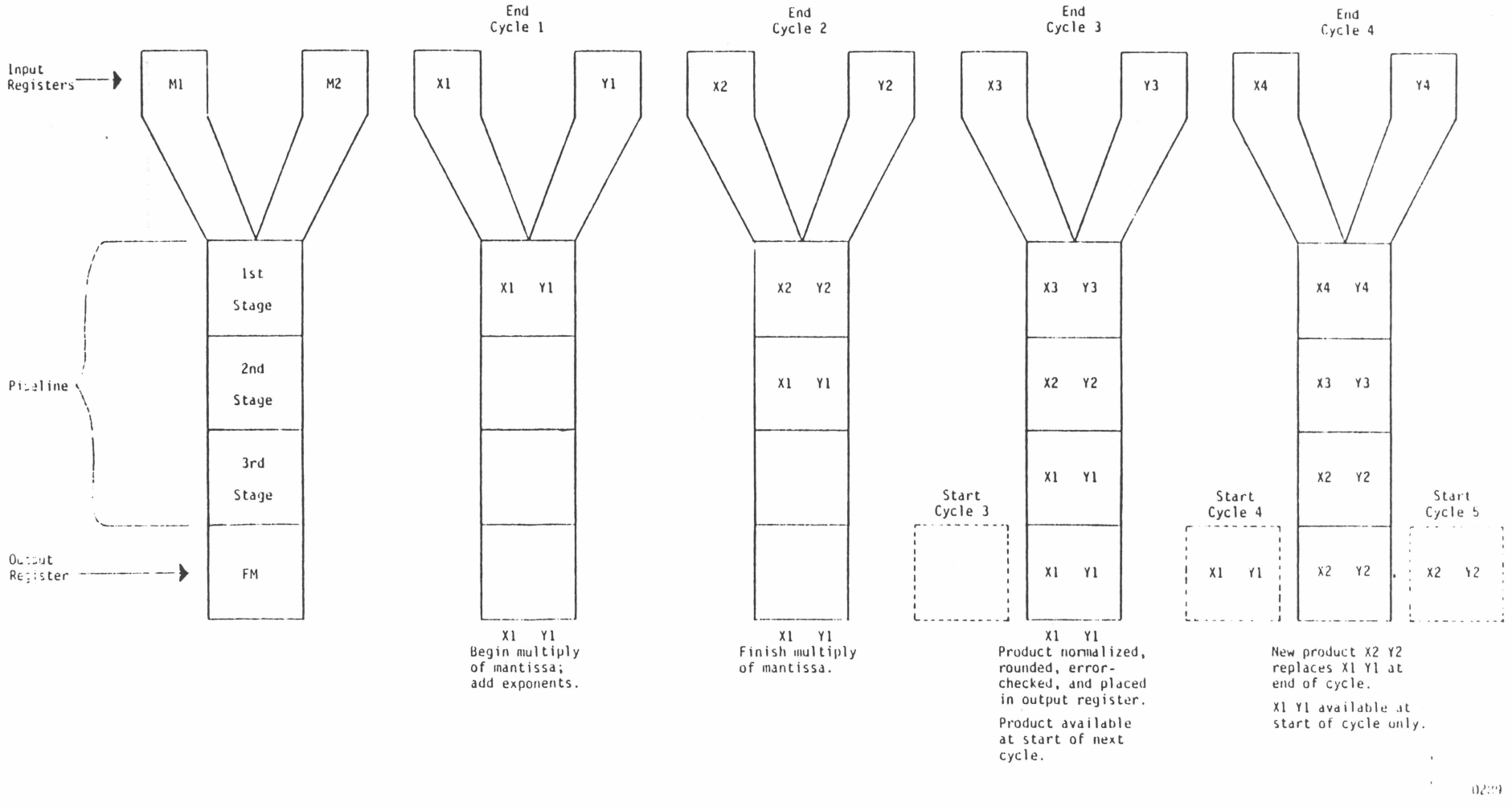


Figure 5-27 Multiplier (FMUL) Pipeline Operation

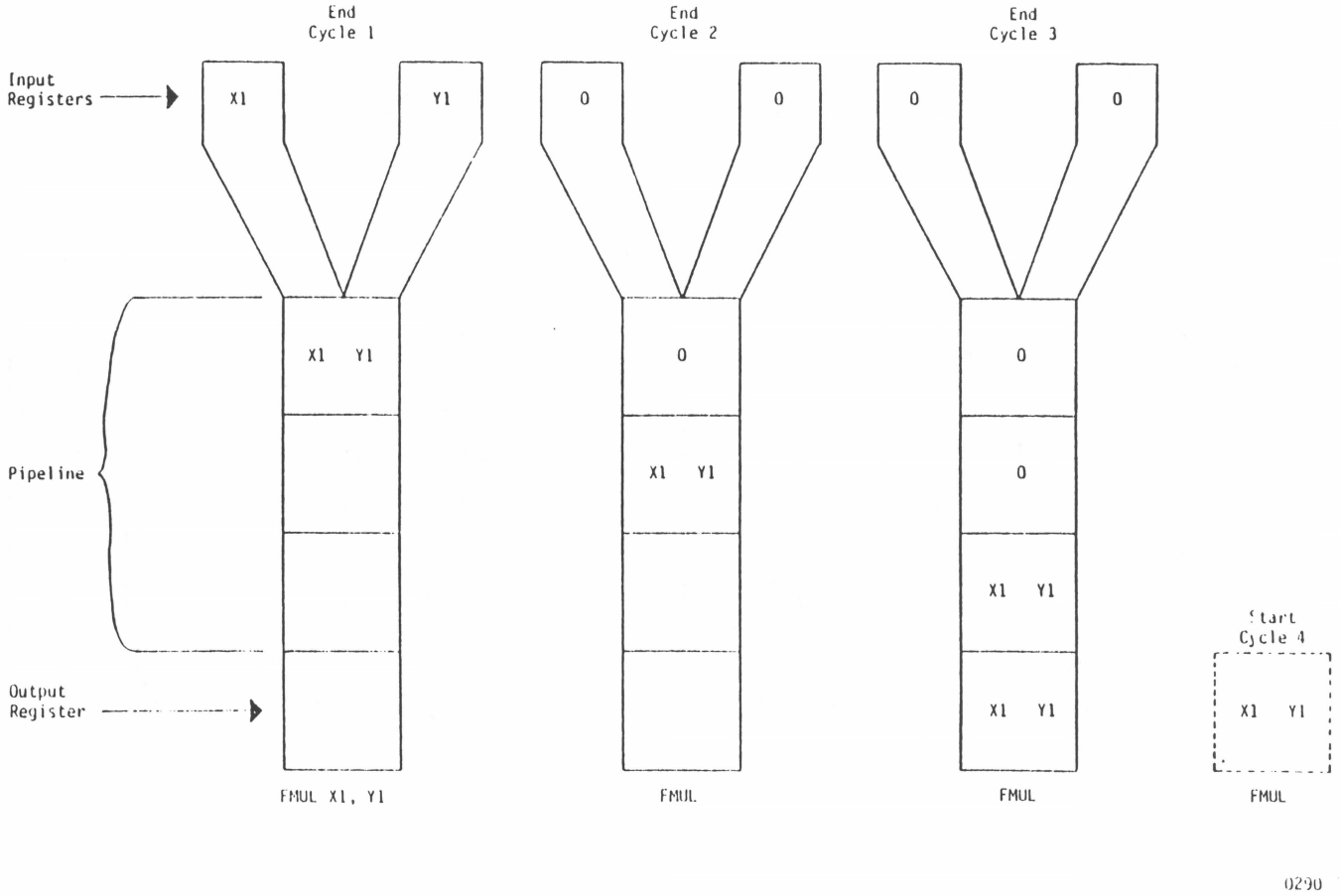


Figure 5-28 Pushing Values through the Multiplier (FMUL)

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 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 (225ns) 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 225ns. 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.

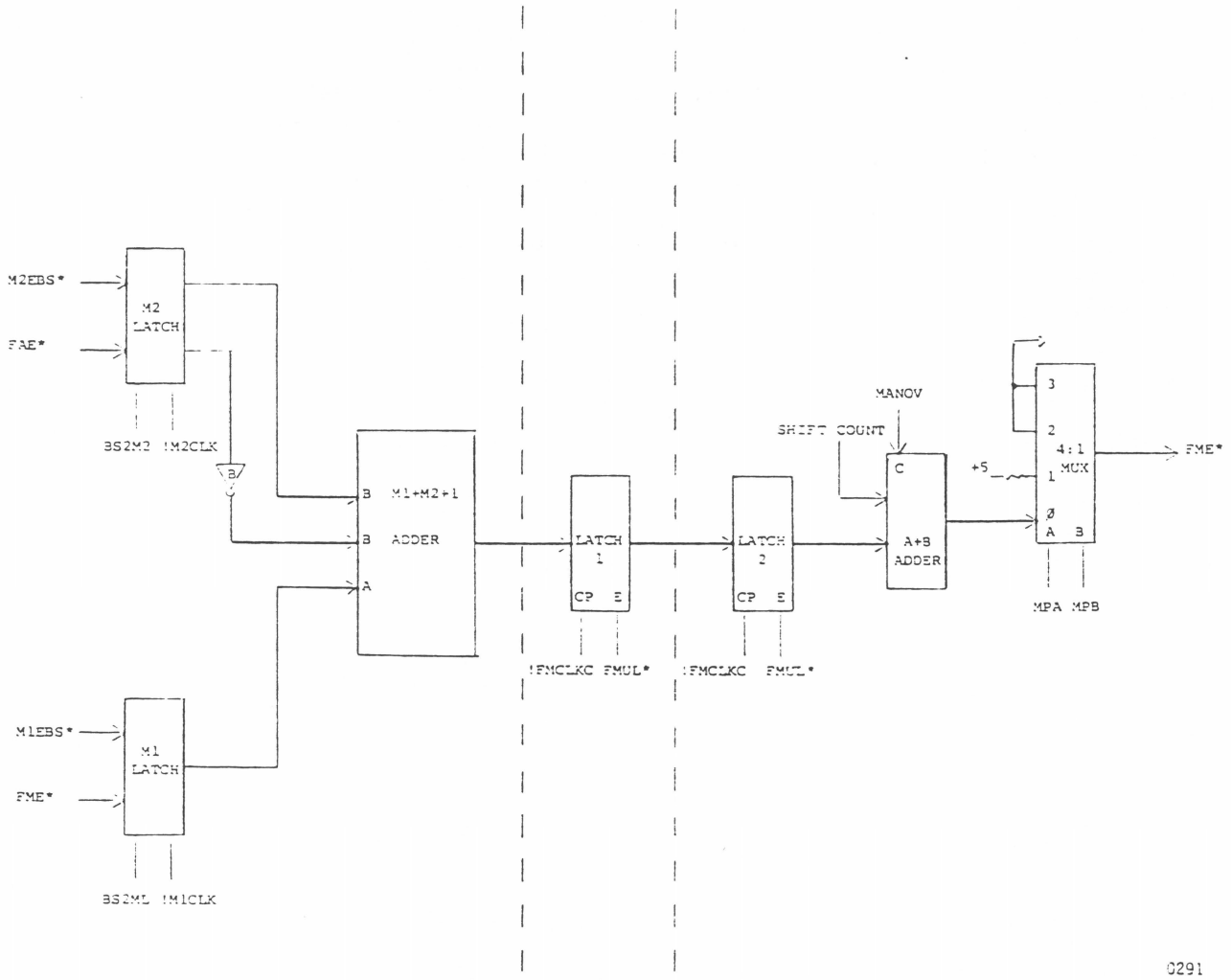


Figure 5-29 Floating Multiplier Exponent Logic

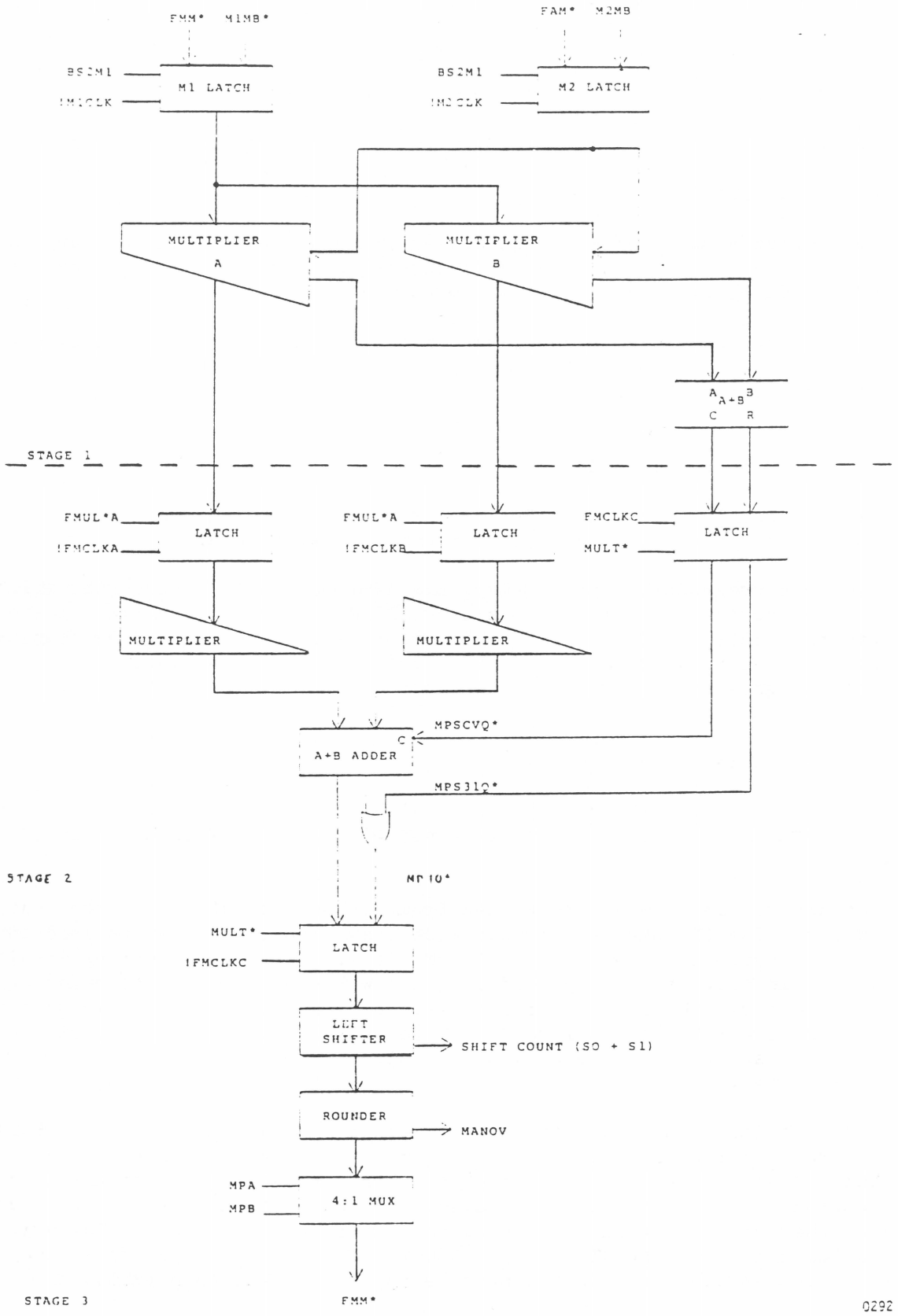


Figure 5-30 Floating Multiplier Mantissa Logic

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 ECB 216 can have up to 512 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 PS0WRT 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 225ns, 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.

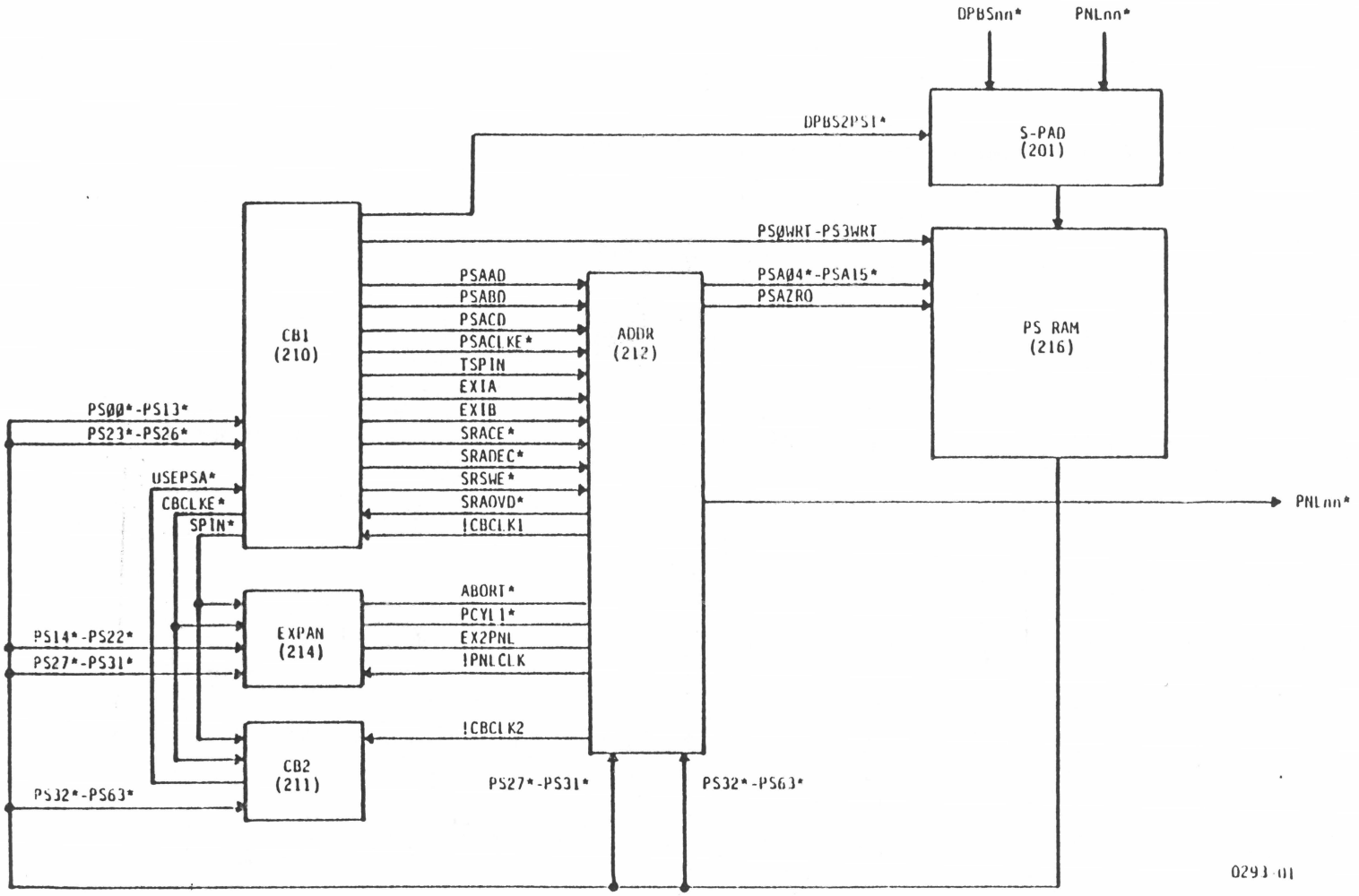


Figure 5-31 AP-120B Block Diagram

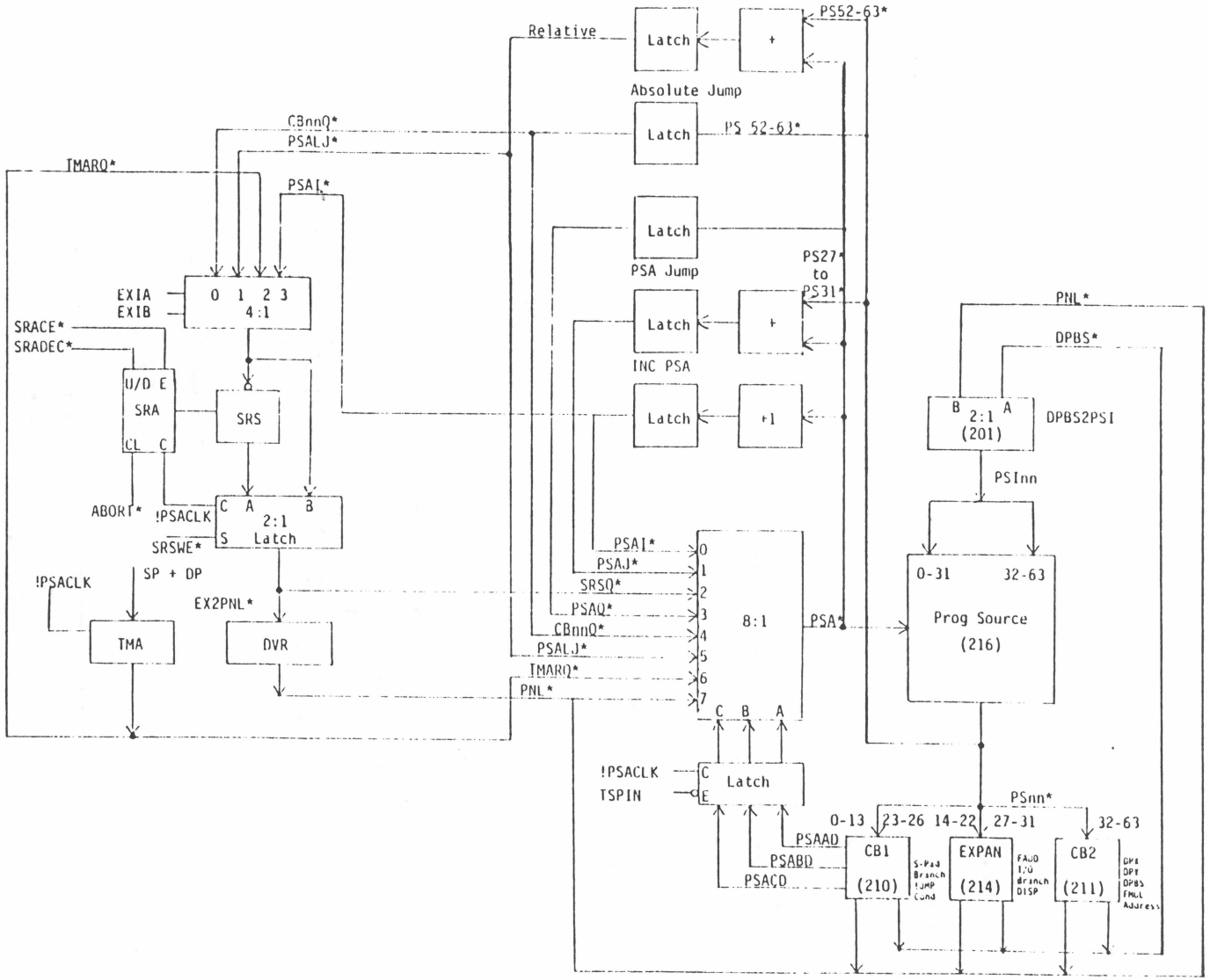


Figure 5-32 Program Source Address Logic

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 (SPsps) 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.

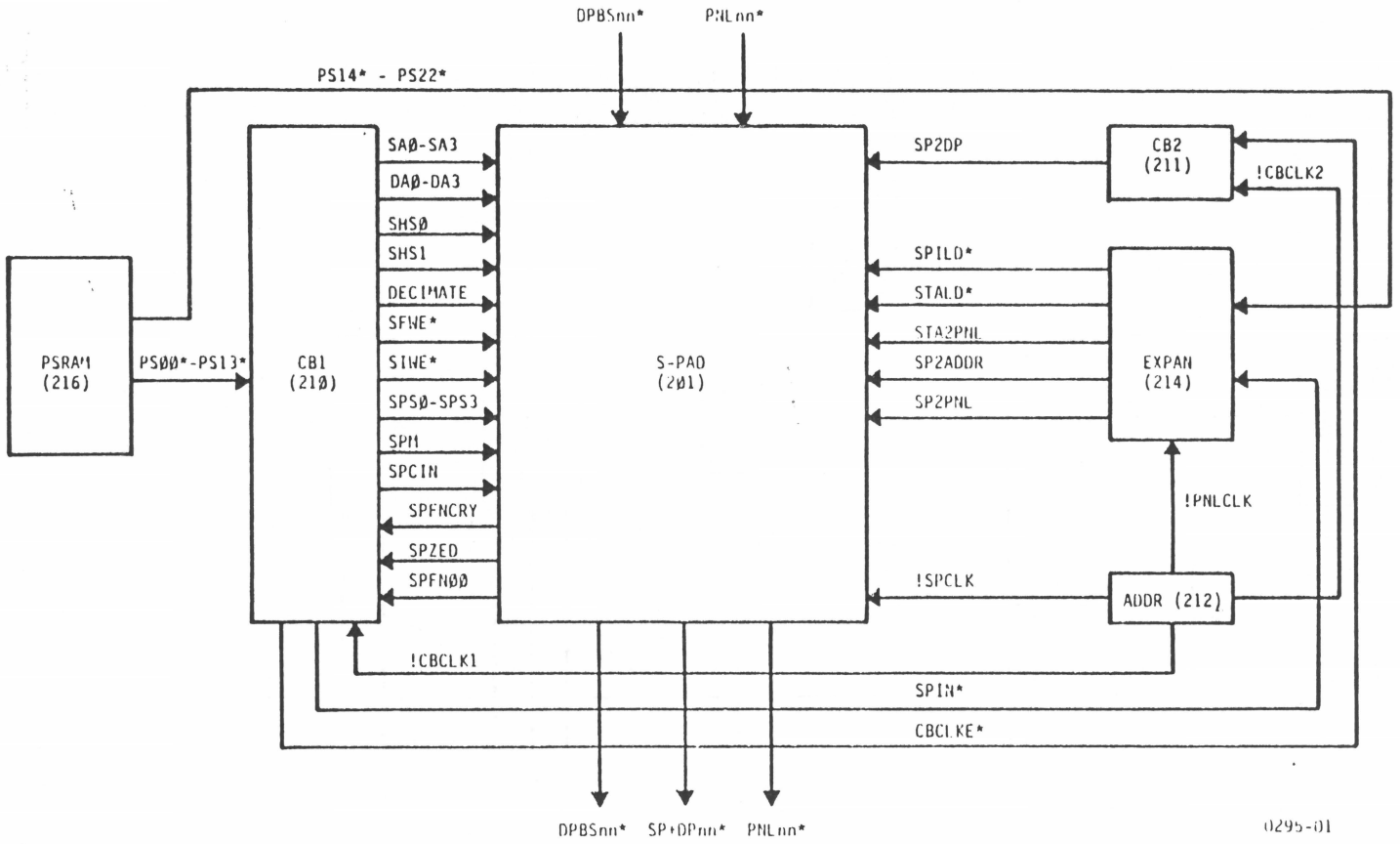


Figure 5-33 SPAD Interconnection Block Diagram

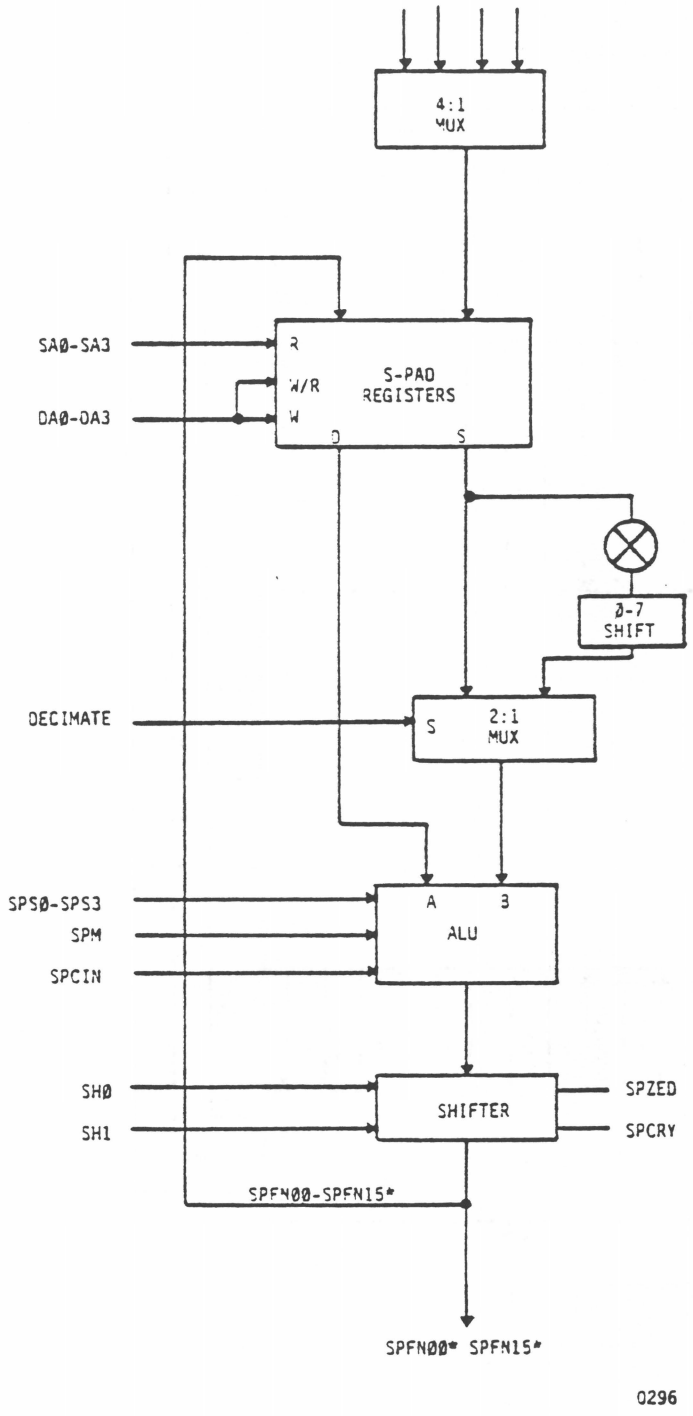


Figure 5-34 SPAD Block Diagram

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 and the host.

There are actually two interfaces that are part of the AP. The first is a general-purpose interface containing the various registers and the bus required for communication between the AP 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 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-impedence 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 driver 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 15ns 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 driven 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 or HWCCLK*.

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 B0CLK*.

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 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 450ns 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 HWCCLK* decrements the WC register. In either register, the count settles a maximum of 15ns 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.

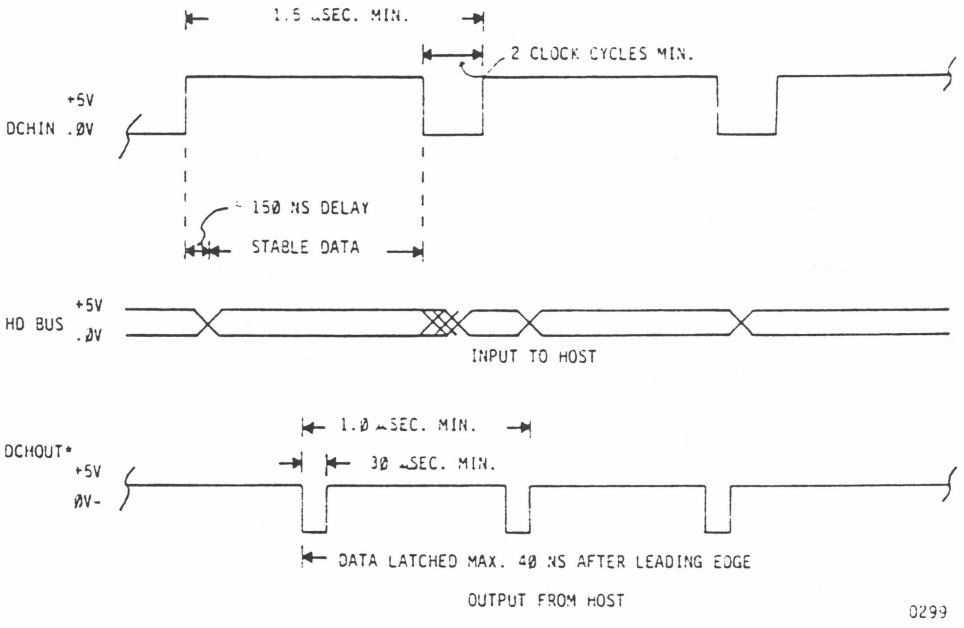


Figure 6-1 Timing Diagram

Table 6-1 Controller to AP Signals

Signal	Backplane Con. Board	Backplane Con. Pin #	Jumper Con.	Signal Type ++	Load Type+	Description (see also test)
BØCLK*	219	A94	J1-3Ø	Pulse	A	Loads FMTH from HD bus
B2CLK*	219	A96	J1-4Ø	Pulse	A	Loads FMTL from HD bus
BH2HD*	219	BØ6	J1-36	Level	A	Drives FMTH to HD bus
BL2HD*	219	BØ8	J1-44	Level	A	Drives FMFL to HD bus
CLINT*	219	B23	J1-2Ø	Pulse	D	Clears AP interrupt to host
DCHIN	219	BØ7	J1-Ø6	Level	A	Drive DMA input to host
DCHOUT*	219	B35	J1-Ø4	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
HOSTRSØ	219	A77	J1-49	Level	A	HMA, APMA, WC, or CTL
HOSTRS1	219	A79	J1-5Ø	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	BØ5	J1-46	Level	A	Drive HMA, WC, APMA, CTL reg to HD bus
REQ	219	B37	J1-Ø2	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-Ø8	SS	A	Interrupt generated by AP
CTLØ8*	219	B27	J1-18	SS	A	Indicates consecutive cycle DMA transfer
CTL1Ø	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	J1-1Ø	Pulse	1k pull-up	Set DMA request to host
WC=Ø*	219	B65	J1-12	SS	A	Indicates WC reg has reached Ø

++ 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	
HDØØ	219	A74	J2-43	
HDØ1	219	A76	J2-45	
HDØ2	219	A78	J2-47	Tri-state, bi-directional 16-bit data bus
HDØ3	219	A8Ø	J2-49	
HDØ4	219	B24	J2-5Ø	Terminated 1k to Vcc and ground on 219 board
HDØ5	219	B26	J2-48	Drive with 8Ø95 or equivalent
HDØ6	219	B28	J2-44	Should be terminated with 1k to Vcc at host end of cable.
HDØ7	219	B3Ø	J2-46	
HDØ8	219	B36	J2-42	
HDØ9	219	B38	J2-4Ø	
HD1Ø	219	B4Ø	J2-Ø6	
HD11	219	B42	J2-Ø4	
HD12	219	B58	J2-Ø2	
HD13	219	B6Ø	J2-Ø1	
HD14	219	B62	J2-Ø3	
HD15	219	B64	J2-Ø5	

Table 6-4 Host Memory Address Lines

Signal	Backplane Con. Board	Pin #	Jumper Con.	
HMAØØ	219	A61	J2-38	
HMAØ1	219	A63	J2-36	
HMAØ2	219	A67	J2-34	The HMA lines always display the current contents of the HMA register
HMAØ3	219	A69	J2-32	
HMAØ4	219	BØ9	J2-3Ø	
HMAØ5	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
HAMØ6	219	B15	J2-26	B13
HMAØ7	219	B17	J2-24	B51 B91
HMAØ8	219	B47	J2-22	and are tied directly to the 18 grounds of jumper B
HMAØ9	219	B49	J2-2Ø	
HMA1Ø	219	B53	J2-18	
HMA11	219	B55	J2-16	
HMA12	219	B87	J2-14	
HMA13	219	B89	J2-12	
HMA14	219	B93	J2-1Ø	
HMA15	219	B95	J2-Ø8	
GND	219	A65		
		B13		
		B51		
		B91		

A)

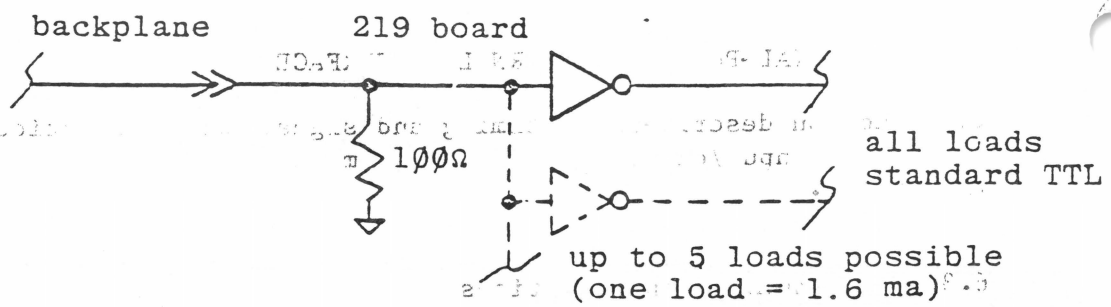


Figure 1

B)

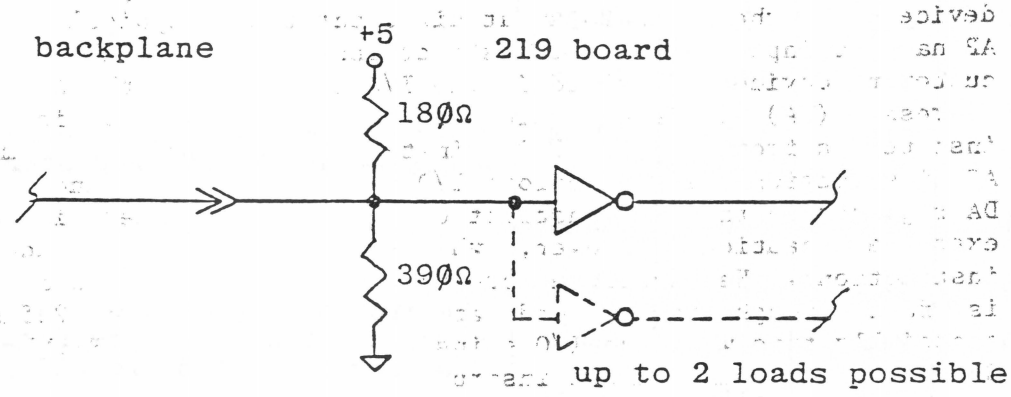


Figure 2

C)

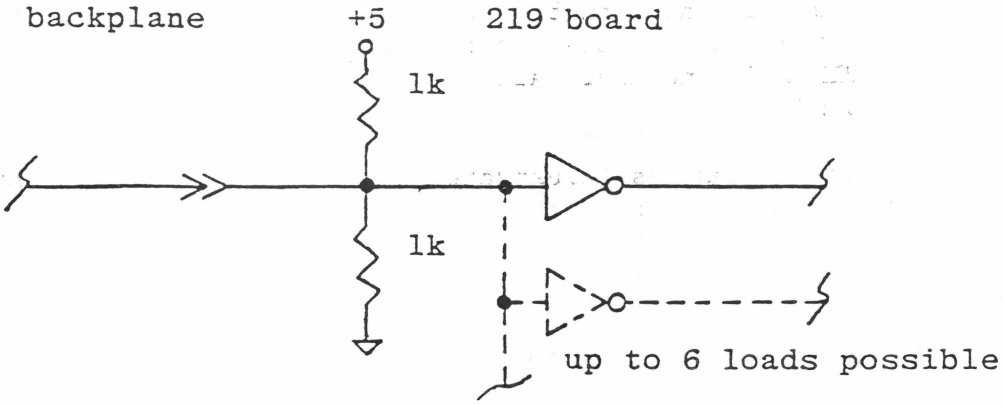


Figure 3

D)

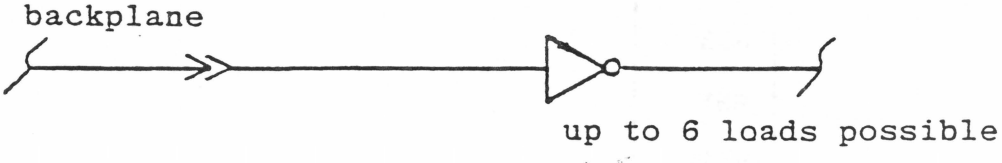


Figure 6-2 Signal Load Variations

6.3 AP SPECIAL-PURPOSE (INTERNAL) INTERFACE

This section describes the timing and signal characteristics of the AP 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 processor, the programming of I/O devices on the AP is somewhat different than a typical host CPU. The AP has I/O input and output instructions. However, in place of the customary device code field in the I/O instruction, the AP 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 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 clock rate of 225ns. 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 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 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 I/O instructions in the initialization and control of the host interface to the AP.

```
1APAL  REV 2
 PASS 1
 PASS 2
                          $TITLE APAL1
                          $ENTRY APAL1
                        "PROGRAMMING EXAMPLE
                        "APDMA 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

                                                            "DA < 1
000002  001103                OUTDA; INC 0  DB=100          "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
```

```
1APAL  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           BINTRQ DONE          "IF INTERRUPT  GO
        000162
        000000
        000000

                                                     "TO SERVICE ROUTINE
000001  000000                  BR .-1               "IF NOT  BRANCH BACK OR
        000117
        000000
        000000

                                                     "DO OTHER USEFUL THINGS.
000002  000000          DONE    NOP                  "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 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 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.

```
1APAL  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;             "SET DA = 0 (WC)
        107000          LDSPI 17                "SET SP(17) = 0
        002000
        000000

000001  001177          OUTDA; DB=20;           "SET WC = 20
        142000          INC 17                  "ADVANCE DA TO 1
        002000
        000020

000002  001177          OUTDA  DB=100;          "SET HMA = 100
        142000          INC 17                  "ADVANCE DA TO 2
        002000
        000100

000003  000003          OUT  DB=201             "SET CTL = 201
        140000
        002000
        000201

                                                "CONSECUTIVE CYCLE
                                                "START DMA
                                                "AP PROGRAM CONTROL
                                                "32-BIT INTEGER FORMAT
000004  000003          LDDA; DB=4              "SET DA = 4(FMT)
        107000
        002000
        000004

000005  000000          NOP                     "NOP TO WAIT FOR
        000000
        000000
        000000
```

```
                                                        "OTHER DEVICES
000006  000003          LOOP    SPININ; DB=INBS         "WAIT FOR DMA
        145000                          DPX(0)<DB       "TO LOAD FMT
        041004
        000000

                                                        "AND THEN READ FMT
                                                        "TO DPBS.
000007  041777                  SPINDA; DB=INBS;        "GET NEXT WORD
        147000                          DPY(0)<DB;INCDPA;"FROM FMT TO DPY
        011000                          MOV 17,17       "ADVANCE DPA
        100004

                                                        "SET DA TO 2
000010  001673                  IN; DB=INBS;            "READ CTL TO SP(16)
        144000                        LDSPI 16          "TO TEST FOR DONE
        001000
        000000

000011  047673                  MOVR 16 16  LDDA;       "SET UP TEST OF LSB
        107000                          DB=4            "RESET DA TO 4
        002000
        000004

000012  010010                  BNC LOOP                "DMA DONE?
        000014
        000000
        000000

000013  000003                  HALT
        170000
        000000
        000000

                        $END


****      0 ERRORS    ****


SYMBOL  VALUE
APAL3   000000 ENT
LOOP    000006
```