

FLOATING POINT
SYSTEMS, INC.

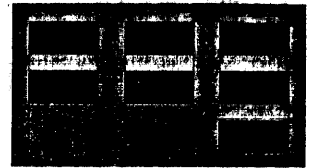
AP-120B

ARRAY TRANSFORM PROCESSOR



PROCESSOR
HANDBOOK

7259-02



FLOATING POINT
SYSTEMS, INC.

PROCESSOR HANDBOOK

7259-02

Form #7259
© Floating Point Systems 1976
All Rights Reserved
Printed in the United States of America
Rev 02, May 1976

PREFACE

Historically, array transform processors have been largely integer-arithmetic devices, since the slower processing rate of floating-point arithmetic was undesirable when working with large arrays of data. However, integer methods have problems which make programming awkward due to the limited dynamic range of integer arithmetic. Array scaling and "block" floating-point techniques have either allowed human and other errors to creep into the results, or were costly and time consuming. Further, as processing became more sophisticated, even 16-bit integer data words were insufficiently precise for preserving the accuracy of simple 8-bit analog-to-digital converted input data. This is because the many multiplications and additions in typical cascaded array processing can cause the propagation of truncation errors*.

With the advent of faster digital logic, many users realized that floating-point processing makes programming easier, virtually eliminates dynamic range problems, greatly alleviates the precision problem, and that it is potentially as fast as the last generation of integer processors. Floating Point Systems, Inc. recognized this trend in 1970, and was formed to specialize in floating-point peripheral processors.

The rush to floating-point processing was not a smooth one. Many floating-point formats sprang up, and Floating Point Systems became expert in format converting "on the fly", so processing time would not be lost during a format conversion. Why convert formats? Simple. Not all formats are mathematically clean. For example, it is unwise to use a hexadecimal-exponent format for serious number crunching because a hexadecimal normalization can cause as many as 3 leading zeros between the binary point of the mantissa and the first significant bit. This means as many as 3 least-significant bits may be lost, due to right-shifting the mantissa past the available word length (truncation), when an extreme hexadecimal normalization occurs (about 25% of the time), and of course, 2, or 1, or no bits may be lost (with equal probability) for other possible hexadecimal cases.

*A 16-bit integer multiplied by a 16-bit integer results in a 32-bit product. If the result is truncated to the 16 most-significant bits, then half the time the resultant's least-significant bit (lsb) is wrong since it should have been rounded up. Now the product of two of these potentially wrong lsb numbers results in the next lsb being wrong part of the time; thus cascaded operations propagate the errors leftward toward the more significant bits.

The FPS solution is to use a true 10-bit binary exponent, which has more dynamic range than the standard 7-bit hexadecimal or 8-bit binary exponent. FPS then uses a 28-bit mantissa, plus 3 guard bits, which provides enough bits to not only allow for hexadecimal in/out formats, but also to carry enough information to permit post-normalization and convergent-rounding after each arithmetic operation. Thus FPS can receive any reasonable floating-point format that is desired as the input format, convert it on-the-fly to the FPS format, process it in FPS format with minimal truncation error propagation, and then convert on-the-fly to the desired output format. This procedure allows transparent "no penalty" operation on the data, thus preserving the integrity of the input data.

In addition to the well chosen floating-point format, the AP-120B has a general-purpose, non-bus oriented architecture for the arithmetic units. This allows great flexibility in that operands and resultants can be moved simultaneously from any register in the AP-120B to any other. This rather generalized structure of the AP-120B allows it to execute specialized algorithms, such as the FFT, in times comparable to those achieved by hardwired special-purpose processors, but it also makes the AP-120B well suited to less highly organized computations.

In the matter of software, it should be noted that this machine is a *synchronous* monolithic multiprocessor, as opposed to an *asynchronous* multiprocessor. The practical significance of this is that programming by the user and/or FPS (Standard Algorithms, System and Test Software) is tremendously simplified, due to the predictability of data flow and timing considerations. There is no need for internal hand-shaking between arithmetic units, memories, and microprocessor; data and results are available at precisely determined times. The synchronous approach not only allows a non-stochastic simulator to be written for easy program debugging, but in addition programs may be single-stepped in the real processor, with execution identical to free-running programs. A further bonus of the synchronous design is the easy produceability, maintainability, interchangeability, and reliability (there is no need to explore an infinite number of possible timing conditions as one clock phases by another, as happens in an asynchronous machine). Convenient and rapid data-dependent branching, simple overlapping of data input, arithmetic processing, and data output are further examples of the care taken to assure a fast, accurate, convenient and reliable array processor.

TABLE OF CONTENTS

SECTION 1 GENERAL INFORMATION

1.1	Introduction	1-1
1.2	System Overview	1-1
1.3	Example AP-120B Application	1-3
1.4	Physical Description	1-5
1.5	Software	1-7

SECTION 2 FUNCTIONAL DESCRIPTION

2.1	Control Unit	2-1
2.2	S-Pad Unit	2-2
2.3	Floating Point Adder Unit	2-4
2.4	Floating Point Multiplier Unit	2-6
2.5	Data Pad Unit	2-8
2.6	Data Memory Unit	2-10
2.7	Table Memory Unit	2-11
2.8	Internal Floating Point Format	2-12

SECTION 3 PROGRAMMING CONSIDERATIONS

3.1	Floating Point Adder	3-1
3.2	Floating Point Multiplier	3-5
3.3	Data Pad	3-9
3.4	Data Memory	3-12
3.5	Table Memory	3-16
3.6	S-Pad	3-19

SECTION 4 INTERFACE

4.1	Front Panel	4-1
4.2	Use of Front Panel	4-6
4.3	Direct Memory Access	4-9
4.4	Format Conversion Register	4-12
4.5	AP-210B Internal	4-13
4.6	Loading Programs into the AP-210B	4-14

SECTION 5 PARTICULAR HOST INTERFACES

5.1	General Interface	5-1
5.2	DEC PDP-11	5-3
5.3	Data General NOVA/Eclipse	5-5
5.4	Raytheon R704/RDS-500	5-7
5.5	Texas Instruments TI980	5-10
5.6	Varian 620	5-20

TABLES

1.1	Floating Point Arithmetic Times	1-9
1.2	Basic Scalar Functions	1-10
1.3	FORTTRAN Callable Routines	1-11

FIGURES

1.1	General AP-120B Block Diagram	1-2
1.2	Physical Configuration	1-6
2.1	Control Unit	2-2
2.2	S-Pad Unit	2-3
2.3	Floating-Point Adder Unit	2-5
2.4	Floating-Point Multiplier Unit	2-7
2.5	Data Pad	2-9
2.6	Data Memory Unit	2-10
2.7	Table Memory Unit	2-11
4.1	AP-120B Panel and Host Interface	4-2
5.1	General AP-120B/Host Computer Interfaces	5-1

SECTION 1 GENERAL INFORMATION

1.1 INTRODUCTION

The AP-120B is a high-speed (167-ns cycle time) peripheral floating-point arithmetic Array Processor, which is intended to work in parallel with a host computer.

Its internal organization is particularly well suited to performing the large numbers of reiterative multiplications and additions required in digital signal processing, matrix arithmetic, statistical analysis, and numerical simulation.

The highly parallel structure of the AP-120B allows the "overhead" of array indexing, loop counting, and data fetching from memory to be performed simultaneously with arithmetic operations on the data. This allows much faster execution than on a typical general-purpose computer, where each of the above operations must occur sequentially.

The AP-120B achieves its high speed through the use of fast commercial integrated circuit elements and an architecture that permits each logical unit of the machine to operate independently and at maximum speed.

Specifically:

- 1) Programs, constants, and data each reside in separate, independent memories, to eliminate memory accessing conflicts.
- 2) Independent floating-point multiply and adder units allow both arithmetic operations to be initiated every 167 ns.
- 3) Two large (32 locations each) blocks of floating-point accumulators are available for temporary storage of intermediate results from the multiplier, adder, or from memory.
- 4) Address indexing and counting functions are performed by an independent integer arithmetic unit that includes 16 integer accumulators.

In a typical application, such as a Fast Fourier Transform, the above features allow nearly the entire computation to be overlapped with data memory access time.

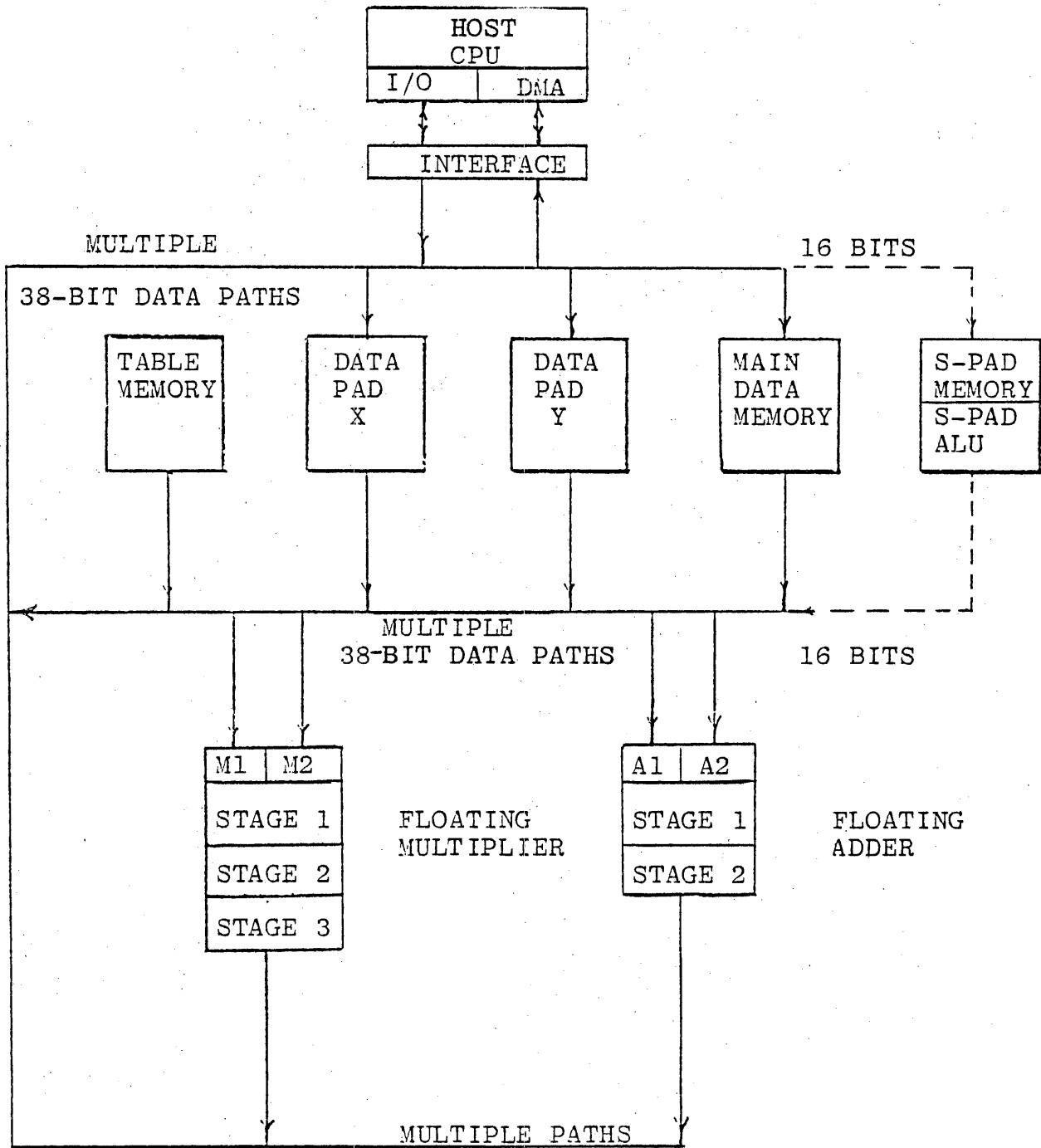
Effective processing precision is enhanced by 38-bits of internal data width, an internal floating-point format with optimum numerical properties, and a convergent rounding algorithm.

1.2 SYSTEM OVERVIEW

A general block diagram of AP-120B arithmetic paths appears in Figure 1.1.

Connection is made to the host in a manner that permits data transfers to occur under control of either the Host Computer or the AP-120B. For most host computers, this will mean that the AP-120B is interfaced to both the programmed I/O and DMA channels.

Figure 1.1 General AP-120B Block Diagram



The system elements are interconnected with multiple parallel paths so that transfers can occur in parallel. All internal floating-point data paths are 38-bits in width (10-bit biased binary exponent and 28-bit 2's complement mantissa).

Data Memory (MD) is organized in 8K-word modules of 38-bit words each, expandable up to 64K words in the main chassis. The effective memory cycle time (interleaved) is 333 ns.

Table Memory (TM) is used for storage of constants (FFT constants), and is tied to a separate data path so as not to interfere with Data Memory. It is bipolar, 167 ns read-only memory, and is organized in 512-word, 38-bit increments.

Data Pad X (DPX) and Data Pad Y (DPY) are two blocks of 32 floating accumulators each. Each is a two-part register block, wherein one register may be read and another written from each block in one instruction cycle.

The Floating Adder (FA) consists of two input registers (A1 and A2) and a two-stage pipe-line which performs the operations, and convergently rounds the normalized result.

The Floating Multiplier (FM) consists of input registers (M1 & M2) and a three-stage pipe-line which performs the multiply operation. Products are normalized and convergently rounded 38-bit numbers.

The S-PAD consists of 16 integer registers and an integer arithmetic unit which is used to form operand addresses and to perform integer arithmetic.

Section 2 contains a more detailed description of each of the functional elements, and Section 3 describes programming considerations.

Section 4 describes in detail the host computer interface, which Floating Point Systems supplies. A number of off-the-shelf interfaces are available.

1.3 EXAMPLE AP-120B APPLICATION

A simple FFT processing sequence would go as follows:

Initial conditions are that the FFT program is resident in Program Source Memory internal to the AP-120B, the array to be transformed is resident in host memory, and the host CPU has initiated the AP-120B processor with an I/O instruction.

1. The AP-120B requests host DMA cycles to transfer the array from host memory to internal data memory. Data is converted from host floating-point format to internal AP-120B floating-point format "on the fly".
2. The FFT algorithm is performed, with data remaining in internal AP-120B format. This yields the benefit of 38-bit precision and convergent rounding during the critical phases of processing.
3. The frequency domain array is transferred back to host memory by requesting host DMA cycles. Data is converted from internal format to host format "on the fly."
4. The AP-120B proceeds to another process or stops executing, depending on previously established conditions. An interrupt to the host can be issued.

The AP-120B is most efficiently used when a sequence of operations is performed on one or more sets of data which reside in internal data memory. This reduces data-transfer overhead, and retains maximum numerical precision. For example, a reasonable sequence would be to transfer a trace and a filter, FFT both, array multiply, inverse FFT, and transfer the result back to host memory.

The AP-120B Data Memory has DMA capability. That is to say that MD cycles can be stolen from the AP-120B microprocessor by the interface. This capability allows Host Computer DMA to AP-120B DMA data transfers to occur, thereby minimizing both host CPU and AP-120B overhead.

The AP-120B has been designed with enough flexibility built-in so that its power can be harnessed in a variety of ways. Subsequent sections describe its use in detail.

1.4 PHYSICAL DESCRIPTION

1.4.1 General. The AP-120B is available in rack configuration. Mounting is as a standard 19-inch EIA rack-mounted unit, requiring 22-3/4 vertical inches of space. The unit is equipped with rack slides, permitting easy access to the etched and/or wire-wrapped circuitry, chassis mounted on the forward portion of the unit. The Power Panel is mounted at the rear. 1-3/4" of space should be available above and below the 22-3/4" of the Processor. This is for proper intake and exhaust of air thru the Processor. The Control Panel (see 1.4.4), and/or blank panels, may be used for proper spacing, if the customer's equipment mounted above and below the Processor do not have the proper free-air space built into them. Intake air should be between 10°C and 40°C.

1.4.2 Forward Unit. The forward unit contains all AP-120B circuitry except the power supply. There is provision for up to 28 15- by 10-inch etched-circuit boards (ECB). The ECBs plug into a mother board. The ECBs are arranged in a vertical plane (chimney style) with push-pull fans to assure adequate upwards air circulation even in the event of a fan failure. The I/O cable exits at the bottom rear (the exact configuration is computer dependent). This unit is called the Processor.

1.4.3 Rear Unit. The Power Supply consists of three assemblies. The first is the main +5-volt supply, and is capable of 100-amperes output. The two other smaller supplies are -5, and +12 volts. The power supplies have forced convection cooling. All supplies are rear mounted, along with the line box (containing line filters and contactor, on the Power Panel).

1.4.4 Power, Controls and Indicators. The AP-120B is expected to be normally powered up and down with the rest of a system. The AP-120B switch and indicators are on a Control Panel. There is a single power cord (U.S. standard 3-wire with ground) which must be connected to 105 to 125 volts, 50 to 60 hertz). The service should be rated for 20 amperes (or 10 amperes in the case of the higher ranges) in order to provide a low impedance source (power required is approximately 1200 volt-amps). The Control Panel may be mounted above or below either the Processor or the Power Panel. Availability of line power is indicated by a neon "Line Voltage" indicator. If the "On Off" switch is On, then the power supplies should come on. There are two operation indicators; one shows Array Processor action, and the other shows DMA transfers. The three individual power supplies have separate indicators (electroluminescent diodes). There are no external adjustments. The internal adjustments are the three power-supply setting potentiometers on the Power Panel.

1.4.5 Serial Numbers. The Processor has a serial-number tag on its starboard side, near the top, forward, ending in "A". The Power Panel tag, ending in "B", is located inside near the top. The Control Panel has its tag, ending in "C", also "inside".

REQUIREMENTS:

1. ENVIRONMENT:

0 - 40°C @ 0 - 90% RELATIVE HUMIDITY.
(DERATE 1°C PER 2500 FT. (762 M) ABOVE
SEA LEVEL, 5°C FOR 50 HERTZ OPERATION.)

2. POWER CONSUMPTION ≈ 1000 W; SERVICE:

- A. 105 - 125 VOLTS, 50 - 60 HERTZ @ 20 AMPS.
- B. 199 - 223 VOLTS, 50 - 60 HERTZ @ 10 AMPS.
- C. 210 - 230 VOLTS, 50 - 60 HERTZ @ 10 AMPS.

NOTE:

VOLTAGE OPTION "A" HAS A WHITE WIRE IN THE
FAN POWER CABLE.

VOLTAGE OPTION "B" HAS A BLUE WIRE IN THE
FAN POWER CABLE.

VOLTAGE OPTION "C" HAS A RED WIRE IN THE
FAN POWER CABLE.

D. LOW IMPEDANCE SERVICE ADVISED.

3. SPACE:

* HEIGHT:

A: WITH CONTROL PANEL AT THE FRONT;

24 1/2" (62.23 CM).

B: WITH CONTROL PANEL AT THE REAR;

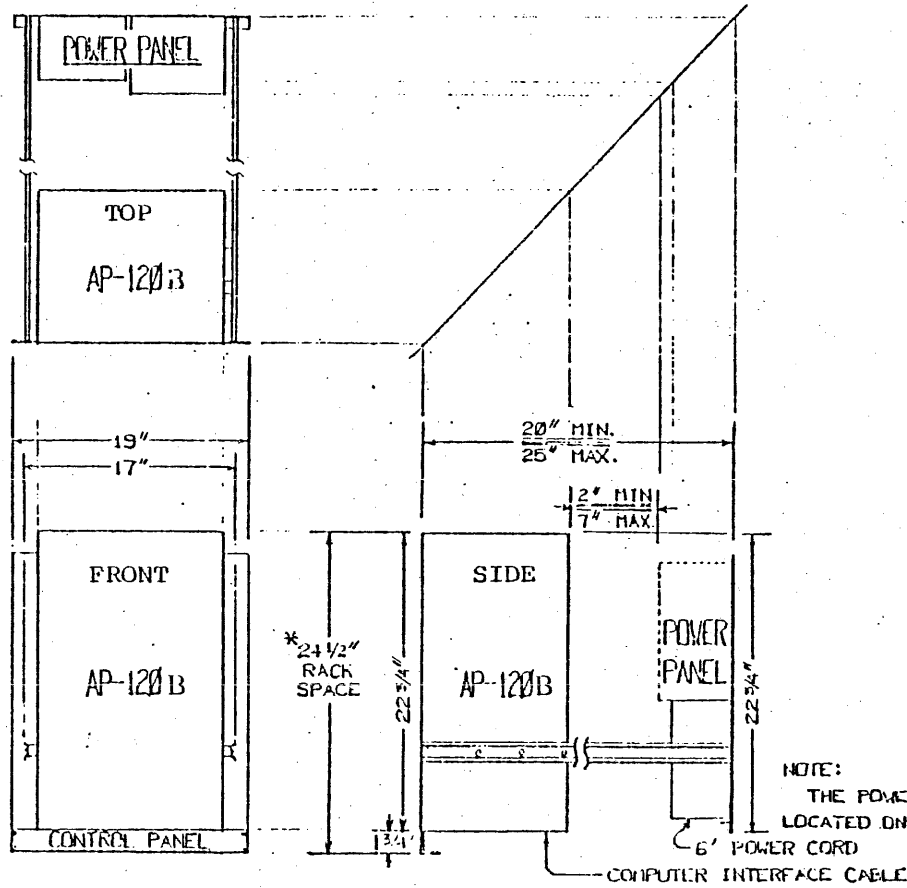
23 1/4" (59.37 CM).

WIDTH: 19" (48.26 CM)

DEPTH: 20 - 25" (50.80 - 63.50 CM).

CAUTION:

ALLOW AT LEAST 1.75" OF FREE AIR SPACE
ABOVE THE AP-120B IF USED AS SHOWN. IF THE
CONTROL PANEL IS MOVED, ALLOW 1.75" OF FREE
AIR SPACE BELOW THE AP-120B



NOTE:
THE POWER PANEL TO AP-120B POWER CABLE IS
LOCATED ON THE LOWER RIGHT SIDE (NOT SHOWN).


PROPRIETARY INFORMATION This document contains proprietary information and is supplied for identification, maintenance, engineering evaluation or inspection purposes only and shall not be duplicated or disclosed without written permission of FLOATING POINT SYSTEMS, INC. By accepting this document the recipient agrees to make every effort to prevent unauthorized use of this information	FRACTOR ± 1/32	AP-120B MOUNTING REQUIREMENTS 	
	DEC.TOL ±.005		
	ANG.TOL REV BY: <i>JKC</i> 6/22/75 REV CHK: <i>TSB</i> 8/7/75	CRW ENG	CNK APPD
	SCALE DWG.NO.		

Figure 1.2 AP-120B Physical Configuration

1.5 SOFTWARE

Four packages of software are supplied with the AP-120B which assist the user toward the solution of his particular processing task.

1.5.1 APEX (A.P. Executive). APEX is a mechanism for communicating with the AP-120B via a series of FORTRAN or machine language subroutine "calls". The executive driver routine interprets the particular user call and directs the AP-120B to perform the specified action. For example, in Fortran, to load an array A containing N real data points into the AP-120B, and perform a real Fast Fourier Transform upon that data:

```
IA=0  
:  
CALL APPUT (A,IA,N,2)  
CALL RFFT (IA,N,1)  
:  
:
```

Both the Standard Applications Subroutines described below and user developed AP-120B programs may be called from the host computer using APEX.

1.5.2 APMATH (A.P. Math Library). These are 70 sub-routines written in AP-120B assembly language. They are callable from the host computer Fortran or machine language using APEX. They are listed in Table 1-2.

1.5.3 Program Development Package. Four Fortran IV programs which are compiled on the host computer during installation aid user program development.

These are:

1. APAL A.P. Assembly Language. A cross-assembler which provides a two pass assembly of symbolic coding into an object module. APAL generates detailed error diagnostics.
2. APLINK A.P. Linker. Links and relocates separate APAL object modules together into a single execution module.
3. APDEBUG A.P. Debugger. An interactive debugging program. The user may selectively set breakpoints, examine and change memory and register contents, and run program segments.
4. APSIM A.P. Simulator. Called by APDEBUG, APSIM provides a programmed simulation of the

APSIM (con't) various hardware elements of the AP-120B. All timing characteristics of the AP-120B are emulated, and the floating point arithmetic is simulated (including rounding) to the least significant bit. APSIM is a convenient tool in bringing up new AP-120B programs off line without interfering with production runs.

1.5.4 APTEST. (A. P. Test programs). APTEST is a collection of interactive diagnostic test and verify programs which aid in isolation of hardware faults. These are:

1. APTEST A. P. Tester. Exercises the Panel, DMA interface, and various internal registers and memories. Tests Main Data Memory with simple patterns and then with random numbers. Board level diagnostic indicators are provided.
2. APPATH A. P. Path Tester. Tests the various internal data paths and gives board level diagnostics.
3. APARTH A. P. Arithmetic Test. Tests the floating point adder, multiplier, and S-Pad arithmetic unit with pseudo-random number and operation sequences.
4. FIFFT Forward/Inverse FFT Test. Verifies the correct operation of the AP-120B as a complete unit by doing forward/inverse FFT transforms on both spikes and random number sequences.

TABLE 1.1
 FLOATING POINT ARITHMETIC TIMES

	<u>Travel Time</u>	<u>Pipeline Interval</u>
Add/Subtract	0.333 us	0.167 us
Multiply	0.500 us	0.167 us
Multiply-Add	0.833 us	0.167 us
Complex Add/Subtract	0.500 us	0.333 us
Complex Multiply	1.333 us	0.667 us
Complex Multiply-Add	1.667 us	0.667 us

Travel time is the total time required to get from the data source to the destination, including the full transport through the arithmetic units. Pipeline Interval is the time between successively available resultants. The former is important when the successive arguments of a computation are dependent upon previous calculations. The latter is indicative of the maximum throughput rate available for successively independent calculators.

TABLE 1.2
BASIC SCALAR FUNCTIONS

	<u>Timing</u>	<u>Program Size</u> (AP-120B Program Words)
Divide	3.83 us	28
Square Root	3.83 us	28
Exponential	4.17 us	27
Natural Logarithm	4.00 us	37
Base 10 Logarithm	4.67 us	37
Sine	4.42 us	31
Cosine	4.75 us	31
Arctangent	8.67 us	46
Arctangent of (Y/X)	13.83 us	46

These functions take arguments from Data Pad and return full-word accuracy results to Data Pad. Full-precision polynomial coefficients for these functions are contained on the standard 512 words of Table Memory.

TABLE 1.3

SUMMARY OF AP-120B FORTRAN CALLABLE ROUTINES

Local Vector Operations

Operation	Name	Timing (us per point)	Size (AP-120B Prog. Words)
Vector Clear	VCLR	.4	4
Vector Move	VMOV	.8	6
Vector Negate	VNEG	.8	7
Vector Add	VADD	1.2	8
Vector Subtract	VSUB	1.2	8
Vector Multiply	VMUL	1.2	11
Vector Divide	VDIV	1.8	44
Vector-Scalar Add	VSADD	.8	8
Vector-Scalar Multiply	VSMUL	.8	9
Vector-Signed Square	VSSQ	.8	10
Vector Absolute Value	VABS	.8	7
Vector Square Root	VSQRT	1.8	43
Vector Logarithm (Base 10)	VLOG	5.6	52
Vector Natural Logarithm	VLN	4.9	52
Vector Exponential	VEXP	5.1	42
Vector Sine	VSIN	5.1	46
Vector Cosine	VCOS	5.6	46
Vector Arctangent	VATN	9.6	89
Vector Arctangent of (Y/X)	VATN2	15.0	90
Sum of Vector Elements	SVE	.4	7
Sum of Vector Squares	SVS	.4	11
Dot Product of two Vectors	DOTPR	.8	9
Vector Float	VFLT	.8	11
Vector Scan and Scale (Fix)	VSCSCL	1.5	19

Vector Maximum/Minimum Operation

Maximum Element in a Vector	MAXV	.2	19
Minimum Element in a Vector	MINV	.2	19
Maximum Magnitude Element in a Vector	MAXMGV	.2	19
Minimum Magnitude Element in a Vector	MINMGV	.2	19
Maximum and Minimum of a Vector	MAXMIN	.8	20
Maximum and Minimum Magnitude of a Vector	MXMNMG	.8	28
Vector Maximum (of two vectors)	VMAX	1.2	13
Vector Minimum (of two vectors)	VMIN	1.2	13
Vector Maximum Magnitude of two vectors	VMAXMG	1.2	14
Vector Minimum Magnitude of two vectors	VMINMG	1.2	14

Vector Filter Operations

Vector Polynomial evaluate	VPOLY		40
Difference Equations	DIFEQ	2.1	27
4 pole filter (difference equation)	RECUR4	.8	15

Complex Vector Operations

Complex Vector Multiply	CVMUL	2.0	26
Complex Vector Reciprocal	CVRCIP	5.0	51
Complex Vector Magnitude (Square)	CVMAGS	2.0	18
Rectangular to Polar Conversion	POLAR	19.4	118
Polar to Rectangular Conversion	RECT	10.7	45

Matrix Operations

Matrix Transpose	MTRANS	.8	17
Matrix Multiply	MMUL	*	58
Matrix Multiply (Dimension ≤ 32)	MMUL32	*	27
Matrix Inverse	MATINV	*	130
Matrix Vector Multiply (3 X 3)	MVML3	2.5/vector	30
Matrix Vector Multiply (4 X 4)	MVML4	4.6/vector	39

Fast Fourier Transform Operations

Complex FFT	CFFT	*	187
Real FFT	RFFT	*	235
Scrambled to True Order FFT Passes	STFFT	*	139
Bit-reverse Order an Array	BITREV	1.75/complex pt	140
Real Transform Unravel Pass	REACTR	*	42

Signal Processing Operations

Convolution (or correlation)	CONV	*	102
Wiener-Levinson Algorithm	WIENER	*	68
Bandpass Filter	BNDPS		287
Power Spectrum	PWRSPC		268
Complex Cepstrum	ICEPST		289
Inverse Complex Cepstrum	ICEPST		289
Schaffer's Phase Unwrapping	SHPH	1.3	17

EXAMPLE TIMINGS FOR STANDARD MEMORY (333ns in milliseconds)

Fast Fourier Transform

Points	Real Data			Complex Data		
	Bit-Reverse	Real FFT	Total	Bit-Reverse	Complex FFT	Total
256	0.23	0.90	1.13	0.45	1.41	1.86
512	0.45	1.76	2.22	0.90	3.48	4.38
1,024	0.90	4.18	5.08	1.80	6.93	8.73
2,048	1.80	8.32	10.12	3.59	16.60	20.19
4,096	3.59	19.37	22.96	7.17	33.16	40.33
8,192	7.17	38.69	45.86	14.34	77.31	91.66
16,384	14.34	88.36	102.70	28.68	154.59	183.27
32,768	28.68	176.68	205.35	57.35	353.29	410.64
65,536	57.6	417.5	475.1			

Convolution or correlation

Operator Length	Result Length	Time
32	100	0.70
32	1,000	6.7
100	1,000	19.0
1,000	1,000	181.4

Wiener-Levenson Algorithm

Size	Spike Case	General Case
50	1.9	4.1
100	7.0	15.5
200	26.6	60.4

Matrix Operations

Dimension	Transpose	Add	Multiply	Inverse
10 X 10	0.10	0.13	0.63	2
20 X 20	0.37	0.51	4.2	8
30 X 30	0.81	1.1	13.2	55
50 X 50	2.2	3.2	68.3	230
100 X 100	8.6	12.6	540.8	1,840

(INTENTIONALLY BLANK)

AP-120B
CONFIGURATION GUIDE

Configuration: Rack Mount
Common: floating _____ or _____ Connected to Chassis

Input Power: 105/125 V _____ or _____ 188/228 V
or _____ 210/250 V (20/11/10 A Service)
50/60 Hz _____ or _____ 50/400 Hz

Data Memory: 38-bit words, MOS. 333 ns interleaved cycle time. 8K word increments: _____ K
56K maximum without expansion chassis. 1 million words maximum with expansion chassis.

Table Memory: 512 words of 38-bit bipolar ROM standard for transcendental coefficients. Order optional N-words for each 4N-point real or complex FFT in 512-word increments.
_____ words (65K maximum).

Program Source Memory: 256 words of 64-bit bipolar RAM standard. Order in 256-word increments.
_____ words (4K maximum).

Host Computer Type: _____ Company & Model

Host Operating System: _____

Purchaser: _____ (name)
_____ (company)
_____ (address)

_____ (phone)
_____ (P. O. Number)

(INTENTIONALLY BLANK)

SECTION 2 FUNCTIONAL DESCRIPTION

The hardware of the AP-120B is composed of three types of functional elements.

1. Logical and control elements
 - a. Control unit
 - b. S-Pad unit
2. Floating-Point arithmetic elements
 - a. Floating-point adder
 - b. Floating-point multiplier
3. Memory elements
 - a. Data Pad unit
 - b. Main data memory unit
 - c. Table memory unit

Each of these functional units is independent and thus can independently perform the programmed operations for which it was designed in parallel with the other functional units.

2.1 CONTROL UNIT

The Control Unit, as illustrated by Figure 2.1, consists of:

- a. Program Source Memory (PS).
- b. Program Source Address (PSA) Register.
- c. Control Buffer (CB) with decoding logic.
- d. Subroutine Return Stack (SRS).

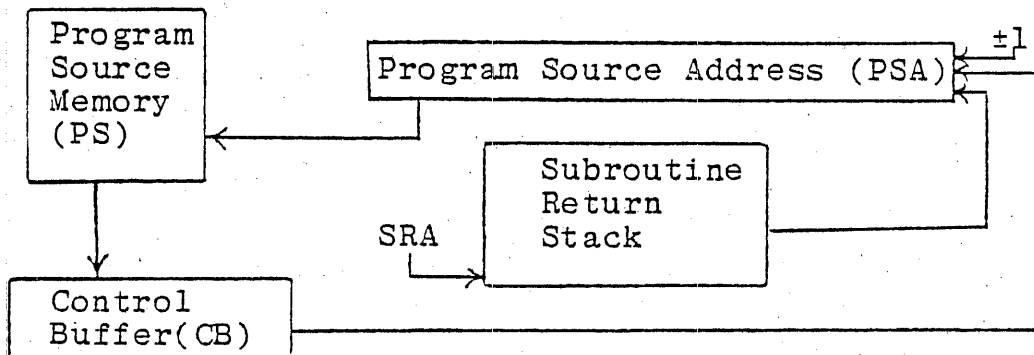
The operation of the AP-120B is controlled by the execution of 64-bit instruction words which reside in Program Source (PS) Memory. The program word for the next instruction to be performed is selected by the address in the Program Source Address (PSA) register. At the initiation of the next machine cycle, this program word is transferred to the Control Buffer (CB) where it is decoded and executed. The PSA is incremented by one unless a branch in the current instruction causes the PSA to move to another location in Program Source (PS) memory. Access to Program Source memory and instruction decoding are overlapped so that the AP-120B can operate at a 6 MHz rate (167 ns).

Branching is accomplished in two manners. A short-range branch is provided by adding the 5-bit branch displacement field to the current PSA. This gives a branch range of from -20_8 to $+17_8$. A long-range jump to any location in PS is accomplished by loading the desired target address into PSA.

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

SRA (Subroutine Return Address) is the Subroutine Return Stack pointer, which is automatically incremented or decremented as subroutines are called and returns are made from the subroutine.

Figure 2.1 Control Unit



2.2 S-PAD UNIT

This unit, illustrated by Figure 2.2, performs the integer address indexing, loop counting and control functions necessary to direct completion of a given algorithm. In form, it is similar to familiar mini-computers such as the PDP-11 or Nova.

The S-Pad contains sixteen 16-bit directly-addressable registers. The contents of these registers pass through a special integer ALU associated with this unit.

The output of the ALU may be directed back to the specified S-Pad destination register, and also to any of the following address memory registers: Memory Address (MA), Table Memory Address (TMA), or Data Pad Address (DPA).

The S-PAD integer ALU functions include:

Function	Effect	
a. Move	$S \rightarrow D$	S-Source register
b. Logical complement	$\bar{S} \rightarrow D$	D-Destination register
c. Clear	$0 \rightarrow D$	
d. Increment	$S+1 \rightarrow D$	
e. Decrement	$S-1 \rightarrow D$	
f. Add	$D+S \rightarrow D$	
g. Subtract	$D-S \rightarrow D$	
n. Logical AND	$D \text{ AND } S \rightarrow D$	
i. Logical OR	$D \text{ OR } S \rightarrow D$	
j. Logical Equivalence	$D \text{ EQV } S \rightarrow D$	

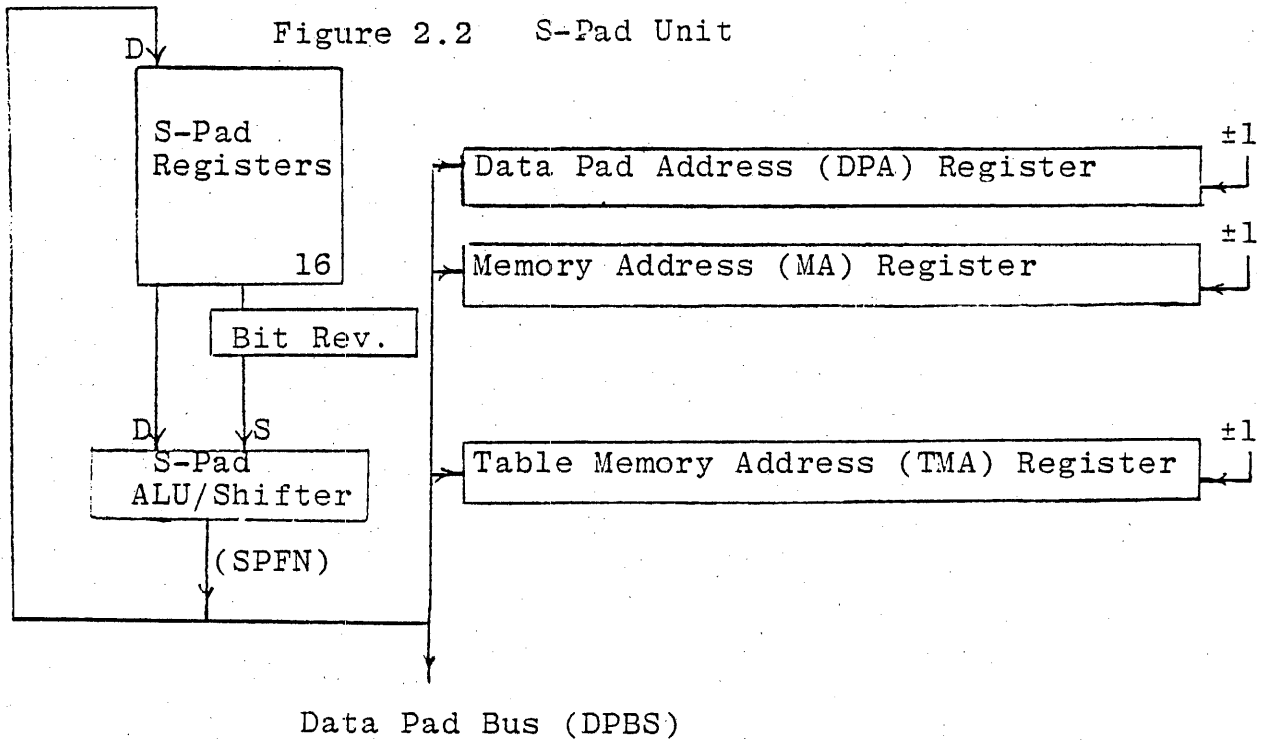
The output of the S-PAD ALU (called S-PAD FUNCTION or SPFN), may be used unmodified, shifted left once, shifted right once, or shifted right twice.

A hardware bit-reverse function included in the S-Pad accomplishes the bit swapping necessary to access data in scrambled order after an FFT.

The S-PAD ALU also sets three condition bits in the AP-120B Status Register depending upon the output of the ALU/shifter :

N: set if result < 0 ; cleared otherwise
 Z: set if result $= 0$; cleared otherwise
 C: set if a carry occurred; cleared otherwise

These bits may be tested by the next AP instruction, and a branch made depending upon whether the specified condition was true.



2.3 FLOATING POINT ADDER UNIT

The Floating Point Adder, shown in Figure 2.3, does addition (or subtraction) operations on the contents of the Adder input registers (A1 and A2). The operation is performed in two stages, each of which takes one machine cycle.

In the first stage, the exponents of the two numbers are compared and the fractions are aligned by shifting the fraction of the smaller number right. The fractions are then added (or subtracted). In the second stage the resulting fraction is normalized and convergently rounded.

Since the two stages are independent of each other, a new pair of numbers may be entered into A1 and A2 every AP cycle (167 ns). The result is available for use two cycles later (333 ns).

In effect, the Floating Adder (FA) is a pipeline, where new inputs may be entered into the pipeline stream every cycle. Initiation of an add operation loads the two numbers to be added into the A1 and A2 input registers. The previous Adder input is pushed down the pipeline to the Adder Buffer register. One cycle later the completed result (called FA) from the Buffer is available for storage or use by another unit. Thus a new add may be started every 167 ns, and the result is ready 333 ns later.

A1 may be loaded from Data Pad (DP), from the output of the Floating Multiplier (FM), or from Table Memory (TM). A2 may be loaded from Data Pad (DP), from the output of the Floating Adder (FA), or from Data Memory (MD).

The output of the Floating Adder (FA) may be directed to the Multiplier (M2), to the Adder (A2), to Data Pad (DP), or to Memory Input (MI).

The operations performed by the Floating Adder are:

- a. $A1+A2$
- b. $A1-A2$
- c. $A2-A1$
- d. $A1 \text{ EQV } A2$
- e. $A1 \text{ AND } A2$
- f. $A1 \text{ OR } A2$
- g. Convert A2 from signed magnitude to 2's complement format.
- h. Convert A2 from 2's complement to signed magnitude format.
- i. Scale A2
- j. Absolute value of A2.
- k. Fix A2.

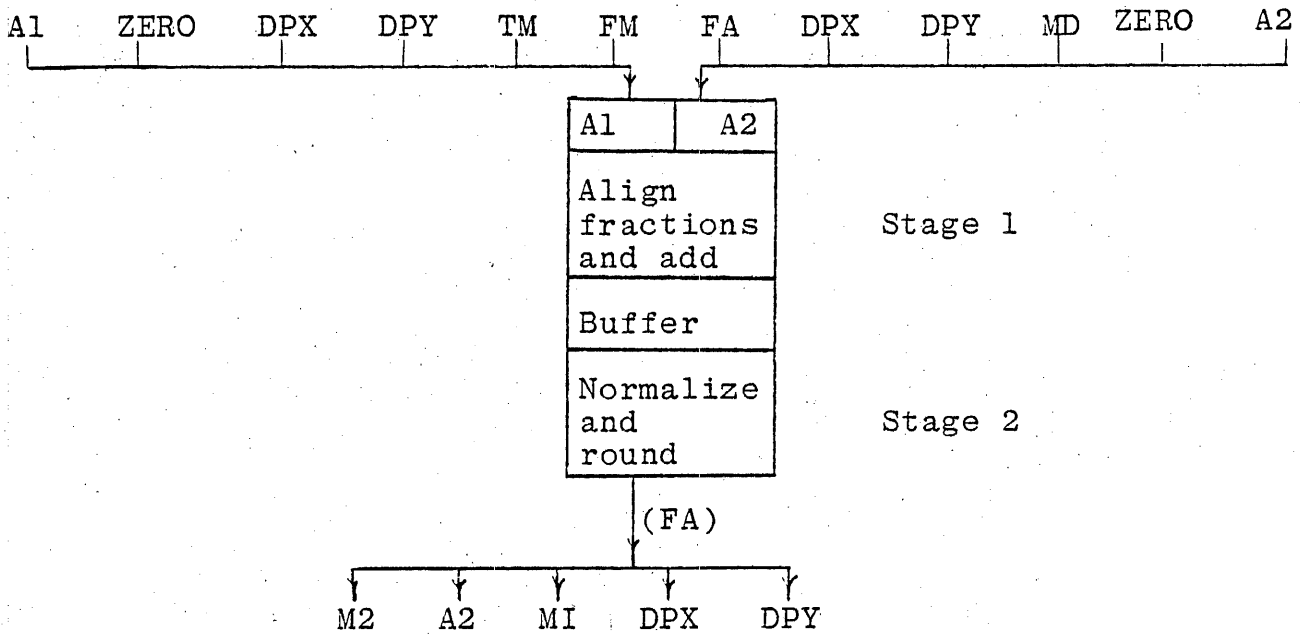
Four condition bits in the AP Status Register are set or cleared by the Floating Adder depending upon the current result:

- FZ - set to one if result is zero, else cleared to zero.
- FN - set to one if result is negative, else cleared to zero.
- FO - set to one if exponent overflow occurred. The result was forced to the signed maximum value.
- FU - set to one if exponent underflow occurred. The result was forced to zero.

The overflow and underflow bits remain set until cleared by the program.

These bits may be tested by the instruction after the Floating Adder result is completed; i.e., three cycles after the Floating Adder operation was initiated.

Figure 2.3 Floating-Point Adder Unit



2.4 FLOATING POINT MULTIPLIER UNIT

The Floating Multiplier, Figure 2.4, forms the product of the two multiplier input registers (M1 and M2). The product is formed in three stages, each of which takes one machine cycle.

In the first stage, the 56-bit product of the two 28-bit fractions are partially completed. The second stage completes the product of the fractions. In the third and final stage the exponents are added, and the mantissa product is normalized and convergently rounded.

The Floating Multiplier, like the Floating Adder, is organized as a pipeline. Initiation of a multiply loads the two numbers to be multiplied into the M1 and M2 input registers. The two previous multiplier inputs are pushed down the pipeline to Buffer 2 and Buffer 3 respectively. One cycle later, the result from Buffer 3 is available for storage or use by another unit.

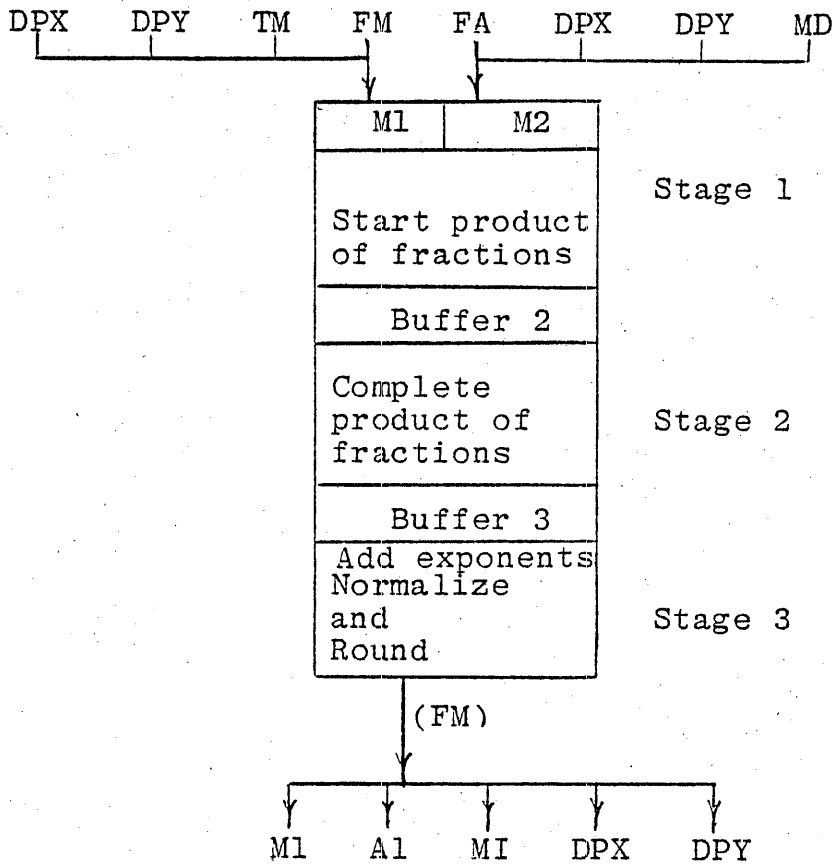
Thus a new product may be started every 167 ns, and the result is ready 500 ns later.

M1 may be loaded from Data Pad (DP), the output of the Floating Multiplier (FM) or from Table Memory (TM). M2 is loaded from Data Pad (DP), the Adder (A1), the Multiplier (M1), or to main Data Memory (MD).

Two error bits in the AP Status Register are affected by the Floating Multiplier:

- FO - set if exponent overflow occurred. The result was forced to the signed maximum value.
- FU - set if exponent underflow occurred. The result was forced to zero.

Figure 2.4 Floating Multiplier



2.5 DATA PAD UNIT

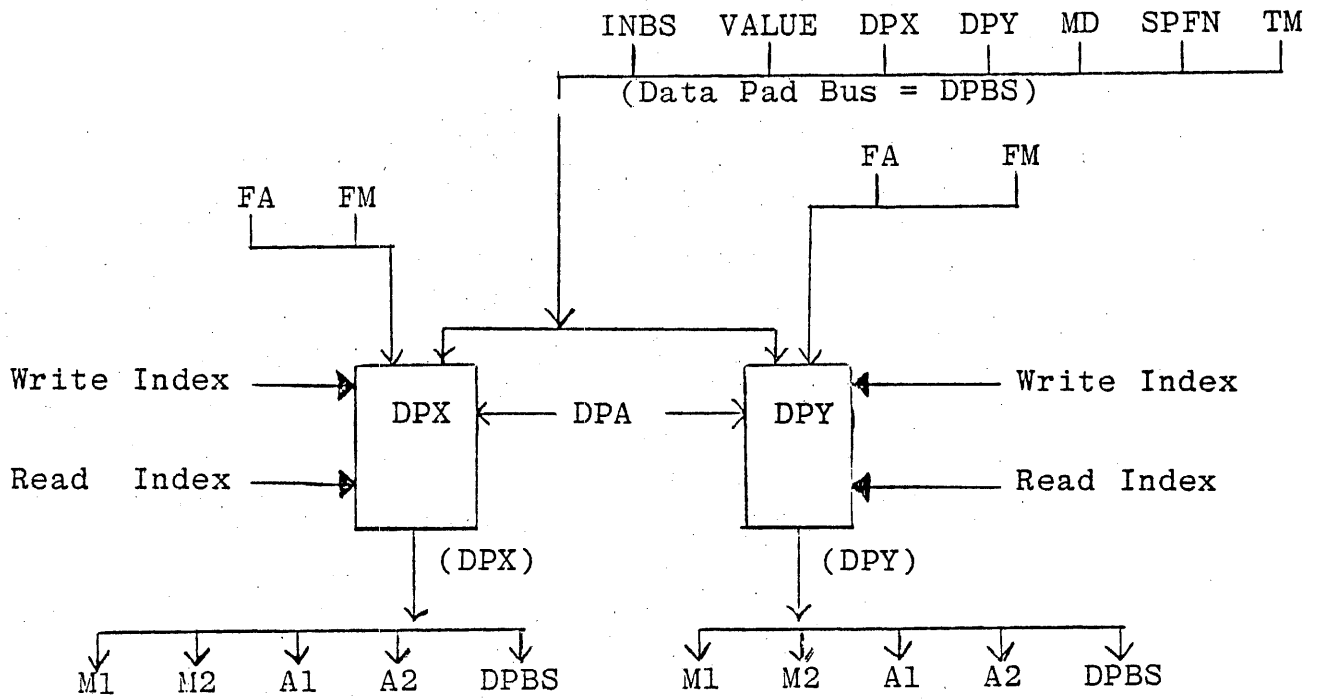
Data Pad, illustrated by Figure 2.5, consists of two fast accumulator blocks, each with 32 floating-point locations, called Data Pad X (DPX) and Data Pad Y (DPY). In a single machine cycle the contents of one location from each Data Pad may be read out and used. In addition, data may also be stored into one location in each Data Pad in the same cycle. That is, for example, in a single instruction (167 ns) a multiply may be initiated specifying one argument from DPX and another from DPY; an Adder result (FA) may be stored into a DPX location, and a data element in Main Data stored into a DPY location. On the very next instruction similar multiple Data Pad accessing could be accomplished again.

The two memories are addressed via a combination of the Data Pad Address (DPA) register and four index field values contained in a given instruction word. DPA may be thought of as a base address register or stack pointer. It may be loaded from the S-Pad (SPFN) or its contents may be incremented or decremented by one.

For a given read or write operation, say reading from Data Pad X, an index value contained in the instruction is added to the current contents of DPA to give the effective address for that particular operation. The four index fields (one each for read DPX, read DPY, write DPX, and write DPY) are each 3 bits wide, and have a range from -4 to +3 relative to DPA.

Data from either Data Pad may be used by the Multiplier (M1, M2), Adder (A1, A2), or Memory Input (MI). Data may be stored into Data Pad from the Adder (FA), Multiplier (FM), S-Pad Function output (SPFN), the Command Buffer Value (VALUE), or from Data Pad (DP).

Figure 2.5 Data Pad



2.6 DATA MEMORY UNIT

The Data Memory unit, illustrated in Figure 2.6, is the primary data store for the AP-120B. It is available in 38-bit wide 8K modules which have an interleaved cycle time of 333 ns.

The memory unit contains a Memory Data (MD) buffer and a Memory Input (MI) buffer. Data read from memory is placed by the controller into MD, while data is written into memory from the MI. The Memory Address (MA) register points to the desired memory location.

In referencing memory for read or write operations, the selected operation is initiated by making a change to the Memory Address (MA) register. The MA register may be loaded from the S-Pad (SPFN) or its contents incremented or decremented by one.

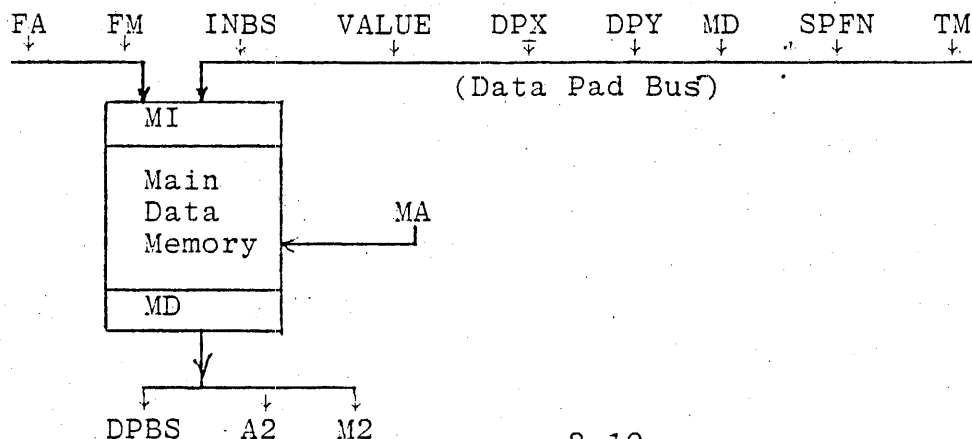
A write operation is specified by loading MI with the data to be written during the same instruction in which MA is changed. This data is then written into memory from MI during the next two AP cycles. Data may be loaded into MI from the Floating Adder (FA), Floating Multiplier (FM), Data Pad (DP), Memory (MD), Table Memory (TM), the Input Bus (INBS), S-Pad Function (SPFN), or the Command Buffer Value (VALUE). A memory operation may be initiated every other cycle. The intervening cycle may be used for any other AP-120B function except another memory initiate.

When a memory read is initiated, the requested memory data is placed by the memory controller into the Memory Data (MD) register 3 cycles after the request was made. Two instructions after the read request, another memory operation may be initiated. Again, the intervening cycle may be used for any non-memory functions. Data in MD may be used by the Floating Adder (A2), Floating Multiplier (M2), or Data Pad (DP).

To optimize the operation of the AP-120B it is necessary for the programmer to "look ahead" and initiate memory reads prior to the actual time that arguments from data memory are to be used in a calculation.

The system provides a "memory lock-out" which serves to insure that erroneous reads and writes of memory do not occur. If a memory initiate occurs while memory is "busy," further program execution is halted until the previous memory cycle is completed.

Figure 2.6 Data Memory Unit



2.7 TABLE MEMORY UNIT

The repeated use of standard constants (such as complex roots of unity and transcendental values) in signal processing routines dictates their ready availability to the programmer. A separate Table Memory (TM), shown in Figure 2.7, eliminates memory accessing conflicts by allowing data values and table values (constants) to be placed in separate memory banks.

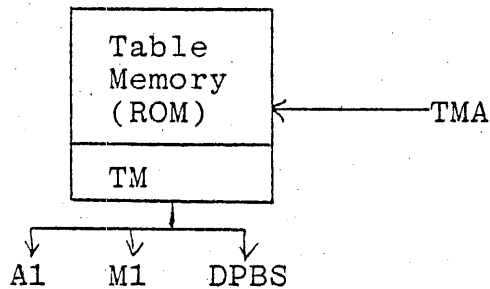
Values read from Table Memory are placed by the controller into the Table Memory (TM) buffer register. The Table Memory Address (TMA) register serves as a pointer to the desired location.

A Table Memory read is initiated by changing the contents of TMA, either by loading a value from the S-PAD (SPFN), or by incrementing or decrementing the contents of TMA.

A new table value may be requested every machine cycle. This value is available for use two cycles later. The value may be used by the Floating Adder (A1), Floating Multiplier (M1), or Data Pad (DP).

In FFT mode (i.e., when a FFT is being computed), the address in TMA is interpreted by the hardware to be an angle which points to the appropriate root of unity for a particular step in the algorithm. This allows the full table of roots of unity to be compressed into a single quadrant of cosines.

Figure 2.7 Table Memory



2.8 INTERNAL FLOATING POINT FORMAT

Floating-point data internal to the AP-120B is represented as follows:

Exponent	Mantissa
0	37
E0	M27

Where:

Mantissa 28-bit two's complement fraction
 Exponent 10-bit binary exponent, biased by 512

The value of a floating-point number in this format is defined as:

$$\text{Mantissa} * 2^{(\text{Exponent} - 512)}$$

The dynamic range of this format is from $0.5 * 2^{-512}$ to $(1-2^{-28}) * 2^{511}$; or, from $3.7 * 10^{-155}$ to $6.7 * 10^{153}$.

The 28-bit fraction, combined with the convergent rounding algorithm used in the Floating Adder and Multiplier, gives a maximum relative error of $7.5 * 10^{-9}$ per arithmetic operation. This is a precision of 8.1 decimal digits. As a comparison, unrounded IBM 360 format gives only 6.0 decimal digits of arithmetic accuracy.

The convergent rounding hardware rounds up when the magnitude of the remainder is greater than $\frac{1}{2}$ of the least significant bit of the mantissa. This serves to minimize truncation errors in long series of arithmetic calculations.

Format conversion between Host format and AP-120B format occurs in the Interface and in the Floating Adder unit. The dynamic range of the internal format is large enough to accommodate IBM 360 format and other Host formats. The extended precision of the AP-120B internal format insures that accuracy is maintained during critical stages of data analysis.

SECTION 3
PROGRAMMING- CONSIDERATIONS

This section provides an introduction to programming the AP-120B. The principal operations which control each of the six functional units are described below. A complete listing of the AP-120B instruction word fields may be found in Appendix B.

In the coding examples semicolon (;) is used to separate operations within a complete instruction word. A comma (,) separates operands. A quote mark (") is used to denote a comment. A less than (" $<$ ") is used to mean " $+$ " (replaced by) where the operation involved is a data transfer.

3.1 FLOATING POINT ADDER.

3.1.1 Floating Adder Operations. Floating Adder operations are initiated by the following instructions:

(Instruction)	(Operands)	(Operations Initiated)
FADD	A1, A2	A1+A2
FSUB	A1, A2	A1-A2
FSUBR	A1, A2	A2-A1
FAND	A1, A2	A1 AND A2
FOR	A1, A2	A1 OR A2
FEQV	A1, A2	A1 EQV A2
FABS	A2	ABS(A2)
FIX	A2	Convert A2, floating to fixed
FSM2C	A2	Convert A2, Signed Magnitude to 2's complement.
F2CSM	A2	Convert A2, 2's complement to signed magnitude.
FSCALE	A2	Scale A2

where A1 and A2 are any of the following data sources:

A1:	FM	Floating Multplier result
	DPX	Data Pad X accumulator
	DPY	Data Pad Y accumulator
	TM	Last data read from Table Memory
	ZERO	Floating-point zero
A2:	FA	Floating Adder result
	DPX	
	DPY	
	MD	Last data read from Data Memory
	ZERO	

Any data source listed under A1 may be combined with any data source listed under A2. For example, to add a number from Data Pad X to another from Data Pad Y:

FADD DPX, DPY "DPX + DPY

or to subtract a number read out of Data Memory from a constant in Table Memory:

FSUB TM, MD "TM - MD

A reverse subtract changes the order of the subtraction, i.e.

FSUBR TM, MD "MD - TM

subtracts a constant from Table Memory from a number in Data Memory.

To negate a number from DPX:

FSUB ZERO, DPX "0.0 - DPX = -DPX

To take the absolute value of a number from Data Memory:

FABS MD "ABS(MD)

To fix (convert from floating-point to integer) a number from DPY:

FIX DPY "FIX (DPY)

3.1.2 Adder Pipeline. The Floating Adder is a two-stage pipeline. A "FADD" instruction loads the designated operands into the A1 and A2 registers. The previous contents of A1 and A2 are pushed down the pipeline to the Buffer register. One AP cycle later the new contents of Buffer have been normalized and rounded, and are then available for use or storage elsewhere.

The following instruction sequence illustrates how the Adder pipeline works, where A,B...G,H are floating-point numbers to be added:

Time	Cycle	Instruction	Adder Pipeline:		Adder Result (FA)
			A1, A2	Buffer	
0	1.	FADD A,B	A,B	---	---
167ns	2.	FADD C,D	C,D	A,B	---
333ns	3.	FADD E,F	E,F	C,D	A+B
500ns	4.	FADD G,H	G,H	E,F	C+D
667ns	5.	FADD	---	G,H	E+F
833ns	6.	---	---	G,H	G+H

The "FADD" without arguments in cycle 5 is used only to push the last computation into the Buffer Register, and hence to the end of the pipeline. Thus, it is a dummy add in the sense that we don't care what its arguments are, since we will never use the results. In the above example we completed our floating-point adds in one microsecond. During cycles 2-4, while we kept the pipeline full, adds were being done every 167ns, the maximum rate.

The completed results, as they come out of the Adder pipeline, are referred to by the mnemonic "FA". FA is dynamic, in the sense that it must be used or stored elsewhere before being changed by the next floating-adder instruction. The programmer has, however, complete control over the pipeline. Arguments advance only when pushed through the pipeline by floating-adder instructions.

3.1.3 An Example. A complete computational sequence is to do the vector sum $A_i = A_i + B_i$, $i=0,1,2,3$. A_i is stored in Data Pad X locations 0-3 and B_i is stored in Data Pad Y location 0-3.

- | | | |
|----|---|---|
| 1. | FADD DPX(\emptyset), DPY(\emptyset) | "Do A_0+B_0 |
| 2. | FADD DPX(1), DPY(1) | "Do A_1+B_1 |
| 3. | FADD DPX(2), DPY(2); DPX(\emptyset)<FA | "Do A_2+B_2 , A_0+B_0 is now done, save it in A_0 |
| 4. | FADD DPX(3), DPY(3); DPX(1)<FA | "Do A_3+B_3 , A_1+B_1 is now done, save it in A_1 |
| 5. | FADD DPX(2)<FA | "Push Adder; save A_2+B_2 in A_2 |
| 6. | DPX(3)<FA | "Save A_3+B_3 in A_3 |

Below is a chart of this computation, showing the state of the Adder pipeline and Data Pad after each instruction is executed.

Cycle	Adder Pipeline		Adder Result	Data Pad X:			
	A1, A2	Buffer		0	1	2	3
1.	A_0, B_0	---	---	A_0	A_1	A_2	A_3
2.	A_1, B_1	A_0, B_0	---	A_0	A_1	A_2	A_3
3.	A_2, B_2	A_1, B_1	A_0+B_0	A_0+B_0	A_1	A_2	A_3
4.	A_3, B_3	A_2, B_2	A_1+B_1	A_0+B_0	A_1+B_1	A_2	A_3
5.	---	A_3, B_3	A_2+B_2	A_0+B_0	A_1+B_1	A_2+B_2	A_3
6.	---	A_3, B_3	A_3+B_3	A_0+B_0	A_1+B_1	A_2+B_2	A_3+B_3

3.1.4 Floating Adder Tests. The following conditional branches test the Floating Adder result (FA).

BR LOOP	"Branch unconditionally to program "location "LOOP"
BFEQ LOOP	"Branch if FA=0.0
BFNE LOOP	"Branch if FA≠0.0
BFGT LOOP	"Branch if FA>0.0
BFGT LOOP	"Branch if FA>0.0

The above branches test "FA" one instruction cycle after it is ready for use. That is, an Adder result may be tested one cycle after it has come out of the Adder pipeline. An example:

1. FSUB DPX,DPY	"Do a computation
2. FADD	"Push the result out
3. DPX>FA	"Save the result
4. BFEQ LOOP	"Test the result here (branch to " location "LOOP" if result was " zero)

Compound tests may be made also. Test MD to see if it is between a lower limit contained in DPX (1) and an upper limit in DPX (2), i.e., see if $DPX(1) \leq MD \leq DPX(2)$:

1. FSUBR DPX(2), MD	"Do MD-DPX(2)
2. FSUB DPX(1), MD	"Do DPY(1)-MD
3. FADD	"Push first test result out
4. BFGT BIG	"Was too big
5. BFGT SMALL	"Was too small
6. . . .	"OK

The branches are made relative to the current Program Source Address (PSA), with a 5-bit displacement value. This means that the conditional branch target address must be within -20_8 to $+17_8$ locations of the current instruction.

3.1.5 Floating Point Logical Operations. These instructions (FAND, FOR, FEQV) perform logical operations on floating-point numbers. Exponent alignment occurs as for a normal floating-point add. The two mantissas are then combined using the specified logical operation. The result is then normalized and rounded.

3.2 FLOATING POINT MULTIPLIER

3.2.1 Multiply Instruction. Floating-point multiplies are initiated by the following instruction:

FMUL M1, M2

which initiates a multiply between M1 and M2, where M1 and M2 are any of the following data sources:

M1	FM	Floating Multiplier result
	DPX	Data Pad X accumulator
	DPY	Data Pad Y accumulator
	TM	Last data read from Table Memory
M2	FA	Floating Adder result
	DPX	
	DPY	
	MD	Last data read from Data Memory

Thus, any of the data sources listed under M1 may be multiplied by any of the data sources in M2. For example, to multiply a number read from Data Memory by a constant from Table Memory:

FMUL TM,MD "TM * MD

or, to multiply a number in Data Pad X by another number in Data Pad Y:

FMUL DPX,DPY "DPX * DPY

3.2.2 Multiplier Pipeline. The Floating Multiplier is a three-stage pipeline. An "FMUL" instruction loads the specified operands into the M1 and M2 registers. The two previous partially completed products are pushed down the pipeline to Buffer 2 and Buffer 3 respectively. One AP cycle later the new contents of Buffer 3 have been normalized and rounded, and are then available for use or storage elsewhere.

The following instruction sequence illustrates how the Multiplier pipeline works, where A,B...G,H are floating-point numbers to be multiplier together.

Time	Cycle	Instruction	Multiplier Pipeline			Multiplier Result (FM)
			M1, M2	Buffer 2	Buffer 3	
0	1.	FMUL A,B	A,B	---	---	---
167ns	2.	FMUL C,D	C,D	A,B	---	---
333ns	3.	FMUL E,F	E,F	C,D	A,B	---
500ns	4.	FMUL G,H	G,H	E,F	C,D	A*B
667ns	5.	FMUL	---	G,H	E,F	C*D
833ns	6.	FMUL	---	---	G,H	E*F
1.0us	7.	---	---	---	G,H	G*H

The "FMUL" in cycles 5 and 6 are dummy multiplies used to push the last two computations to the end of the pipeline. In the above example we completed four floating-point multiplies in 1.0us. During cycles 3-4, while the pipeline was full, products were being done every 167ns, the maximum rate.

The completed products as they come out of the Multiplier pipeline are referred to by the mnemonic "FM." FM is dynamic, in that it must be used or stored before being changed by the next "FMUL" instruction.

3.2.3 An example. A computational example is to square the elements in a vector:

$A_i = A_i * A_i$, $i=0,1,2,3$. A_i is stored in Data Pad X.

1.	FMUL DPX(0),DPX(0)	"Do A_0^2
2.	FMUL DPX(1),DPX(1)	"Do A_1^2
3.	FMUL DPX(2),DPX(2)	"Do A_2^2
4.	FMUL DPX(3),DPX(3); DPX(0)<FM	"Do A_3^2 , save A_0^2
5.	FMUL; DPX(1)<FM	"Save A_1^2
6.	FMUL; DPX(2)<FM	"Save A_2^2
7.	DPX(3)<FM	"Save A_3^2

Below is a chart of this computation, showing the state of the Multiplier pipeline and Data Pad X after each instruction is executed.

Cycle	Multiplier Pipeline			Multiplier Result (FM)	Data Pad X			
	M1, M2	Buffer 2	Buffer 3		0	1	2	3
1.	A_0, A_0	---	---	---	A_0	A_1	A_2	A_3
2.	A_1, A_1	A_0, A_0	---	---	A_0	A_1	A_2	A_3
3.	A_2, A_2	A_1, A_1	A_0, A_0	---	A_0	A_1	A_2	A_3
4.	A_3, A_3	A_2, A_2	A_1, A_1	A_0^2	A_0^2	A_1	A_2	A_3
5.	---	A_3, A_3	A_2, A_2	A_1^2	A_0^2	A_1^2	A_2	A_3
6.	---	---	A_3, A_3	A_2^2	A_0^2	A_1^2	A_2^2	A_3
7.	---	---	A_3, A_3	A_3^2	A_0^2	A_1^2	A_2^2	A_3^2

3.2.4 Multiply-Adds. The full floating-point computational power of the AP-120B is utilized when we consider a process involving both multiplies and adds. Form the dot product of two eight-element vectors $A_i \cdot B_i = \sum A_i B_i$, $i = -4, -3, \dots, 1, 2, 3$ where A_i is in Data Pad X and B_i is in Data Pad Y:

Fill the Multiplier Pipeline	}	1.	FMUL DPX(-4),DPY(-4)	"Do $A_{-4}B_{-4}$
		2.	FMUL DPX(-3),DPY(-3)	"Do $A_{-3}B_{-3}$
		3.	FMUL DPX(-2),DPY(-2)	"Do $A_{-2}B_{-2}$
		4.	FMUL DPX(-1),DPY(-1); FADD FM, ZERO	"Do $A_{-1}B_{-1}$. $A_{-4}B_{-4}$ is " now done, save it in " adder.
Fill the Adder Pipeline	}	5.	FMUL DPX(\emptyset),DPY(\emptyset); FADD FM, ZERO	"Do A_0B_0 . $A_{-3}B_{-3}$ is now " done, save it in the " adder.
		6.	FMUL DPX(1),DPY(1); FADD FM, FA	"Do A_1B_1 . $A_{-2}B_{-2}$ is now " coming out of the mul- " tiplier, and $A_{-4}B_{-4}$ " from the adder, add " them together.
Both Pipelines full	}	7.	FMUL DPX(2),DPY(2); FADD FM, FA	"Do A_2B_2 . $A_{-1}B_{-1}$ is now " coming out of the mul- " tiplier, and $A_{-3}B_{-3}$ " from the adder, add " them together.
		8.	FMUL DPX(3),DPY(3); FADD FM, FA	"Do A_3B_3 . A_0B_0 is now " coming out of the mul- " tiplier, and ($A_{-4}B_{-4} +$ " $A_{-2}B_{-2}$) from the adder, " add them together.
		9.	FMUL; FADD FM,FA	" A_1B_1 is coming out of the " multiplier, and ($A_{-3}B_{-3}$ " $+A_{-1}B_{-1}$) from the " adder, add them to- " gether.
Empty the Multiplier Pipeline	}	10.	FMUL; FADD FM,FA	" A_2B_2 is coming out of the " multiplier, and ($A_{-4}B_{-4}$ " $+A_{-2}B_{-2}+A_0B_0$) from the " adder, add them to- " gether.
		11.	FADD FM,FA	" A_3B_3 is coming out of " the multiplier, and " ($A_{-3}B_{-3}+A_{-1}B_{-1}+A_1B_1$) " from the adder, add " them together.
Empty the Adder Pipeline	}	12.	FADD; DPX(3)<FA	"($A_{-4}B_{-4}+A_{-2}B_{-2}+A_0B_0+A_2B_2$) " is coming out of the " adder, save it in DPX(3).
		13.	FADD DPX(3),FA	"($A_{-3}B_{-3}+A_{-1}B_{-1}+A_1B_1+A_3B_3$) " is coming out of the " adder, add it to " ($A_{-4}B_{-4}+A_{-2}B_{-2}+A_0B_0+$ " A_2B_2) which was saved " in DPX(3).

14. FADD
 15. DPX(3)<FA

"Push result out of Adder
 "The result: ($A_{-4}B_{-4} +$
 " $A_{-3}B_{-3} + A_{-2}B_{-2} + A_{-1}B_{-1} +$
 " $A_0B_0 + A_1B_1 + A_2B_2 + A_3B_3$),
 " saved in DPX(3).

In accumulating the sum-of-products, the even term sum was kept in one half of the adder pipeline and the odd term sum in the other half. During cycles 5-7 when both pipelines were full, floating-point multiply-adds were being computed every 167ns. This is 12 million floating-point computations per second. A longer sum of products calculation, involving more terms, would maintain this maximum computation rate for nearly all of the computation loop. Here, in a short calculation, most of the time was spent filling and emptying pipelines. Even so, the seven adds and eight multiplies took 15 cycles (2.5us) to complete, or an overall rate of 333ns per floating-point multiply-add.

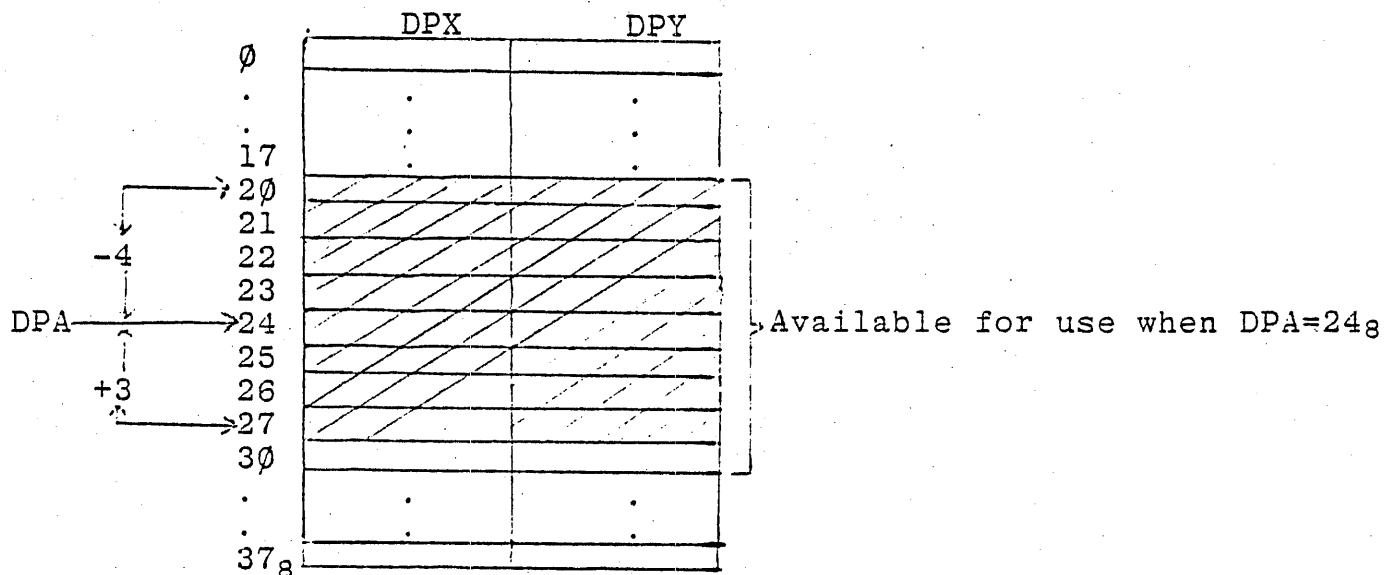
As a further aid in understanding the multiply-add interaction in the above sum-of-products computation, the chart below summarizes the computation:

Cycle	Multiplier:		Adder:		Data Pad:
	M1, M2	FM	A1, A2	FA	3
1.	A_{-4}, B_{-4}	---	---	---	---
2.	A_{-3}, B_{-3}	---	---	---	---
3.	A_{-2}, B_{-2}	---	---	---	---
4.	A_{-1}, B_{-1}	$A_{-4} * B_{-4}$	$A_{-4}B_{-4}, \emptyset.\emptyset$	---	---
5.	A_0, B_0	$A_{-3} * B_{-3}$	$A_{-3}B_{-3}, \emptyset.\emptyset$	---	---
6.	A_1, B_1	$A_{-2} * B_{-2}$	$A_{-2}B_{-2}, A_{-4}B_{-4}$	$A_{-4}B_{-4}$	---
7.	A_2, B_2	$A_{-1} * A_{-1}$	$A_{-1}B_{-1}, A_{-3}B_{-3}$	$A_{-3}B_{-3}$	---
8.	A_3, B_3	$A_0 * A_0$	A_0B_0, ES_2	ES_2	---
9.	---	$A_1 * A_1$	A_1B_1, OS_2	OS_2	---
10.	---	$A_2 * A_2$	A_2B_2, ES_3	ES_3	---
11.	---	$A_3 * A_3$	A_3B_3, OS_3	OS_3	---
12.	---	---	---	ES_4	ES_4
13.	---	---	OS_4, ES_4	OS_4	ES_4
14.	---	---	---	---	ES_4
15.	---	---	---	$OS_4 + ES_4$	$OS_4 + ES_4$

ES is n terms of the even term Sum: $A_i B_i, i = -4, -2, 0, 2$
 OS is n terms of the odd term Sum: $A_i B_i, i = -3, -1, 1, 3$

3.3 DATA PAD

3.3.1 Data Pad Addressing. Data Pad is a block of 64 high-speed accumulators used to store intermediate results during a computation. In any given AP-120B instruction the programmer has 16 of the Data Pad accumulators to work with, 8 in Data Pad X (DPX) and 8 in Data Pad Y (DPY). They are addressed relative to the current value of the Data Pad Address (DPA) register, which functions as a base register for Data Pad. For example, if DPA has a value of 24_8 , locations 20_8 through 27_8 would be available for use.



A displacement value from -4 to $+3$ may be specified when using DPX and DPY, i.e., if $DPA=24_8$:

DPX(3)	means DPX location $24+3=27$
DPY(-4)	means DPY location $24-4=20$
DPX(0)	means DPX location $24+0=24$
DPY	means DPY location $24+0=24$

Four separate displacements are provided, one each for reading and writing DPX and DPY. Thus four separate locations in Data Pad may be used in a given instruction. With $DPA=24_8$, the following instruction occurs in one cycle:

FADD DPX(3),MD; FMUL TM,DPY(-2); DPX(-3)<FA; DPY(1)<FM

(read DPX) (read DPY) (write DPX) (write DPY)

This would 1) add DPX location to the last data read from Data Memory, 2) multiply the last data read from Table Memory by the contents of DPY location 22, 3) store the results of a previous add into DPX location 21, and 4) store the results of a previous multiply into DPY location 25.

All 64 locations of Data Pad are accessed by changing the DPA pointer:

INCDPA	"Increments DPA by 1
DECDPA	"Decrements DPA by 1
SETDPA	"Loads DPA with the current S-PAD " function (SPFN, see Section 3.6)

Changes in DPA take effect for the next instruction after they occur; i.e., if we start with DPA=24:

1. FADD DPX(\emptyset), DFY(\emptyset); INCDPA "DPA is still 24, so we add
DPX₂₄ to DPY₂₄
2. FADD DPX(\emptyset), DPY(\emptyset); INCDPA "Now DPA=25, so we add
DPX₂₅ to DPY₂₅
3. FADD DPX(\emptyset), DPY(\emptyset) Now DPA=26, so we add
DPX₂₆ to DPY₂₆

Thus, by successively incrementing DPA we can use Data Pad as a queue, or by properly incrementing and decrementing DPA, we can use Data Pad as a stack. Data Pad Address is circular. That is, with successive increments of DPA the next location after 37₈ is \emptyset ; with successive decrements of DPA the next location after \emptyset is 37₈.

3.3.2 Writing into Data Pad. Data may be stored into DPX and DPY from FA, FM, or DB (Data Pad Bus).

DPX<FA	"Store adder result into DPX
DPX<FM	"Store Multiplier result into DPX
DPX<DB	"Store Data Pad Bus into DPX

and

DPY<FA	"Store into DPY
DPY<DB	
DPY<DB	

The following may be selected onto the Data Pad Bus (DB):

DB=ZERO	"Floating-point zero
DB=INBS	"Input Bus
DB=VALUE	"A 16-bit immediate value
DB=DPX	"DPX
DB=DPY	"DPY
DB=MD	"Last data read from Data Memory
DB=SPFN	"S-Pad Function (16-bit integer)
DB=TM	"Last data read from Table Memory

Thus, if DPA=24₈, we could have an instruction

DPX(3)<FA; DPY(-2)<DB; DB=MD

which would store the current Adder result into DPX location 27 and store the last data read from Main Data memory into DPY location 22 via the Data Pad Bus.

3.3.3 Data Pad Bus. Data to be stored into DPX and DPY can be moved through three pathways: FM, FA, and DB. While FM and FA are fixed in meaning (output from the Floating Multiplier and Adder respectively), the Data Pad Bus (DB) pathway can be connected to any one of eight possibilities, depending upon the programmers choice. Some example situations:

1. We want to put MD into both DPX and DPY:

DPX<DB; DPY<DB; DB=MD

We put MD onto the Data Pad Bus, and store the Data Pad Bus into DPX and DPY.

2. We want to put MD into DPX and TM into DPY:

DPX<DB; DB=MD; DPY<DB; DB=TM

This is an error! Only one choice at a time can be made for the Data Pad Bus. Alas, not all things are possible even with 64-bits of program word. This double transfer would take two separate instructions to accomplish.

3. We want to store FA into DPX and MD into DPY:

DPX<FA; DPY<DB; DB=MD

We put MD onto the Data Pad Bus in order to get it into DPY. FA goes directly into DPX.

To simplify notation, data transfers involving Data Pad Bus can be written in a shorthand manner, as illustrated by the three above examples:

Shorthand	Longhand
1. DPX<MD; DPY<MD	DPX<DB; DPY<DB; DB=MD
2. DPX<MD; DPY<TM (Still an error no matter how we write it!!)	DPX<DB; DB=MD; DPY<DB; DB=TM
3. DPX<FA; DPY<MD	DPX<FA; DPY<DB; DB=MD

In the shorthand notation, choices for the Data Pad Bus are not explicitly indicated. We write the transfers as if there was a direct connection between the source and destination, while in fact it is the Data Pad Bus which does the connecting. The programmer must remember, however, that he is still making a Data Pad Bus choice, and that only one choice is allowed per instruction. Errors like that in #2 (where we tried to make two Data Pad Bus choices) are detected and flagged by the assembler.

3.4 DATA MEMORY

3.4.1 Memory Addressing. Main Data Memory cycles are initiated by changing the Memory Address (MA) register, which points to the memory location to be read from or written into:

INCMA "Increment MA by 1
DECMA "Decrement MA by 1
SETMA "function (SPFN, see Section 3.6)

All of the above initiate a memory cycle at the address pointed at by the new contents of MA. If an "MI" (Memory Input) field is also included in the instruction, then the memory cycle is a write cycle, otherwise a read cycle is initiated. When sequential memory locations are accessed, a new memory cycle may be initiated by every other AP instruction.

3.4.2 Data Memory Reads. Data read from memory is available for use three AP instructions cycles after the "read" was initiated. The following instruction sequence illustrates how memory data is accessed: A, B, & C are floating-point numbers in memory locations 101, 102, 103 respectively. We assume that MA was set to 100 before we started.

Time	AP Cycle	Instruction	Memory Address (MA)	Memory Data Result (MD)
0	1.	INCMA	101	---
167ns	2.	---	101	---
333ns	3.	INCMA	102	---
500ns	4.	---	102	A
667ns	5.	INCMA	103	A
833ns	6.	---	103	B
1.0us	7.	---	103	B
1.17us	8.	---	103	C

Three AP cycles after a given memory location was "read" the data from that location is ready in the Memory Data (MD) register and available for use. MD may be used by the Adder or the Multiplier:

FADD DPX(3),MD; FMUL DPY(-2),MD "Do MD+DPX and MD*DPY.

or also put on the Data Pad Bus and stored in Data Pad or back into memory:

DPX(2)<MD "Store" MD into DPX.

3.4.3 An Example. Load a vector $A_i, i=0,1,2$ stored in Memory locations 101, 102, 103 into DPX locations 10, 11, 12. We will assume that MA was set to 100 and DPA was set to 10 before we started.

1.	INCMA	"Fetch A_0 from Memory
2.	---	
3.	INCMA	"Fetch A_1 from Memory
4.	DPX<MD; INCDPA	"Store A_0 into DPX location 10 " and bump DPA pointer to 11.
5.	INCMA;	"Fetch A_2 from Memory
6.	DPX<MD; INCDPA	"Store A_1 into DPX location 11 " and bump DPA pointer to 12.
7.	---	
8.	DPX<MD	"Store A_2 into DPX location 12.

Below is a chart of the above transfer, showing the state of each component after each instruction.

Cycle	Memory		Data Pad			
	MA	MD	DPA	DPX ₁₀	DPX ₁₁	DPX ₁₂
1.	101	---	10	---	---	---
2.	101	---	10	---	---	---
3.	102	---	10	---	---	---
4.	102	A_0	10	A_0	---	---
5.	103	A_0	11	A_0	---	---
6.	103	A_1	11	A_0	A_1	---
7.	103	A_1	12	A_0	A_1	---
8.	103	A_2	12	A_0	A_1	A_2

3.4.4 Data Memory Writes. Data Memory write cycles are indicated by:

MI<FA	"Write the Adder result into memory
MI<FM	"Write the Multiplier result into memory
MI<DB	"Write Data Pad Bus into memory

These instructions load data into the Memory Input (MI) buffer register, from whence it is written into memory.

Data may be written into sequential memory locations by every other AP instruction.

3.4.5 An Example. Square the elements of a vector $A_i, i=0,1,2$, in DPX locations 10, 11, 12 and store the results into Data Memory locations 101, 102, 103. We will assume that MA was set to 100 and DPA was set to 10 before we started.

1.	FMUL DPX,DPX; INCDPA	"Square A_0 , bump DPA pointer " to 11.
2.	FMUL	"Push down the multiplier " pipeline.
3.	FMUL DPX,DPX; INCDPA	"Square A_1 , bump DPA pointer " to 12.
4.	FMUL; MI<FM; INCMA	"Write A_0^2 into memory location " 101.

- | | |
|-----------------------|-------------------------------------|
| 5. FMUL DPX, DPX | "Square A_2 |
| 6. FMUL; MI<FM; INCMA | "Write A_1^2 into memory loc. 102 |
| 7. FMUL | "Dummy FMUL to empty pipeline. |
| 8. MI<FM; INCMA | "Write A_2^2 into memory loc. 103 |

Below is a chart of this computation:

Cycle	DPA	Multiplier		Memory	
		M1, M2 A ₀ , A ₀	FM	MA	MI
1.	10	A ₀ , A ₀	--	--	--
2.	11	---	--	--	--
3.	11	A ₁ , A ₁	--	--	--
4.	12	---	A ₀ ²	101	A ₀ ²
5.	12	A ₂ , A ₂	--	101	A ₀ ²
6.	12	---	A ₁ ²	102	A ₁ ²
7.	12	---	--	102	A ₁ ²
8.	12	---	A ₂ ²	103	A ₂ ²

3.4.6 Memory Interleave. Data Memory is divided into 16 banks of 4K words each using MA00-MA02 and MA15 as a memory bank select. (These are the three highest-order bits and the least-significant bit of MA.) Memory references to different banks may be made every 2 AP cycles, while references to the same bank may be made every 3 AP cycles. For some possible memory addressing sequences we have:

Memory Address Sequence (Octal)	Memory Bank Sequence	Memory Reference Timing
101, 102, 103, 104, ...	1, 0, 1, 0, ...	every 2 AP cycles
166, 165, 164, 163, ...	0, 1, 0, 1, ...	every 2 AP cycles
100, 102, 104, 106, ...	0, 0, 0, 0, ...	every 3 AP cycles
233, 10374, 234, 10376, ...	1, 2, 0, 2, ...	every 2 AP cycles

Thus references to successive sequential memory locations may be made every other AP cycle, but references to successive-odd or successive-even locations must be 3 cycles apart.

3.4.7 Memory Lockout. If memory references are made too rapidly for memory to handle, the CPU suspends program execution and "spins" until the memory is no longer busy. Thus if we coded:

1. INCMA "We are trying to refer-
2. INCMA "ence memory every cycle.
3. INCMA

We will get the following execution:

0 ns	1.	INCMA
167 ns	2.	INCMA
333 ns		"SPIN"
500 ns	3.	INCMA
667 ns		"SPIN"

The processor "waits" an extra cycle after instructions 2 and 3 because memory is still busy from the previous memory references. This arrangement is fine if there is no useful computing to do during the "spin" cycles. Otherwise, it is better to space out the "INCMA's" and to do something useful during the cycles between memory references.

3.5 TABLE MEMORY

3.5.1 Table Memory Addressing. Constants stored in Table Memory are read by setting the Table Memory Address (TMA) register to the address of the desired Table Memory location. This is done by the instructions:

```
INCTMA          "Increment TMA by 1
DECTMA          "Decrement TMA by 1
SETTMA         "Set TMA to the current S-Pad
                " function (SPFN, see Section 3.6)
```

Each of the above initiates a fetch from the Table Memory location pointed at by the new contents of TMA. Two AP cycles later the contents of the desired location are available for use. A new location may be fetched every AP cycle. The following sequence illustrates how Table Memory is accessed. K_0 , K_1 , and K_2 are constants stored in Table Memory location 235, 236, 237. We assume that TMA was set to 234 before we start.

Time	AP Cycle	Instruction	Table Memory Address (TMA)	Table Memory Result (TM)
0	1.	INCTMA	235	---
167ns	2.	INCTMA	236	---
333ns	3.	INCTMA	237	K_0
500ns	4.	---	237	K_1
667ns	5.	---	237	K_2

Two cycles after a given Table Memory location was fetched, the data is ready in the Table Memory (TM) data register, and is available for use. TM may be used by the adder or the multiplier:

```
FADD TM, DPX(2);FMUL TM,DPY(-3)      "Do TM+DPX and TM*DPY
```

or put on the Data Pad Bus and stored into Data Pad:

```
DPX(-1)<TM                          "Store TM into DPX.
```

3.5.2 An Example. Do the vector sum $A_i = B + K$, $i=0,1,2$, where A_i is in DPX locations 10-12, B_i is in DPY 10-12, and K_i is a series of constants stored in Table Memory location 235-237. A_i will be stored back into DPX. We will assume that DPA was set to 10 and TMA was set to 234 before we start.

1.	INCTMA	"Fetch K_0
2.	INCTMA	"Fetch K_1
3.	INCTMA; FADD TM, CPY; INCDPA	"Do $K_0 + B_0$, bump DPA to 11
4.	FADD TM, DPY; INCDPA	"Do $K_1 + B_1$, bump DPA to 12
5.	FADD TM, DPX (0); DPX(-2)<FA	"Do $K_2 + B_2$, store A in DPX ₁₀
6.	FADD: DPX(-1)<FA	"Store A_1 in DPX ₁₁
7.	DPX(0)<FA	"Store A_2 in DPX ₁₂

The following charts the above computation:

Cycle	Table	Memory	Adder	FA	Data Pad X			
	TMA	TM	A1, A2		DPA	10	11	12
1.	235	---	---	---	10	---	---	---
2.	236	---	---	---	10	---	---	---
3.	237	K_0	K_0, B_0	---	10	---	---	---
4.	237	K_1	K_1, B_1	---	11	---	---	---
5.	237	K_2	K_2, B_2	K_0+B_0	12	A_0	---	---
6.	237	K_2	---	K_1+B_1	12	A_0	A_1	---
7.	237	K_2	---	K_2+B_2	12	A_0	A_1	A_2

3.5.3 A Complex Multiply. An example using both memories is a complex multiply from the FFT (Fast Fourier Transform) algorithm. The multiply is between a complex signal point held in Data Memory and a complex exponential value (a root of unity, $e^{i\theta}$) fetched from Table Memory. The computation is:

$$X_R = C_R * W_R - C_I * W_I$$

$$X_I = C_R * W_I + C_I * W_R$$

where C is the data point and W is the complex exponential "R" and "I" denote real and imaginary parts respectively. C is in Main Data Memory, and W is in Table Memory.

Fetch the	1.	INCMA	"Fetch C_R from Data Memory
4 arguments	2.	INCTMA	"Fetch W_R from Table Memory
	3.	INCMA; INCTMA	"Fetch C_I fetch W_I
	4.	FMUL TM, MD	"Do $C_R * W_R$
Do the	5.	FMUL TM, MD; DECTMA	"Do $C_R * W_I$ fetch W_I
multiplies	6.	FMUL TM, MD	"Do $C_I * W_I$
	7.	FMUL TM, MD; DPX(0)<FM	"Do $C_I * W_R$, Save $C_R * W_R$, In DPX
	8.	FMUL; DPX(1)<FM	"Save $C_R * W_I$ in DPX
Do the 2	9.	FMUL; FSUBR FM, DPX(0)	"Do $X_R + C_R * W_R - C_I * W_I$
adds.	10.	FADD FM, DPX(1)	"Do $X_I = C_R * W_I + C_I * W_R$

- 11. DPX(0)<FA; FADD X_R is ready, save in DPX
- 12. DPX(1)<FA X_I is ready, save in DPX

The total elapsed time is 12 cycles or 2us. In practice, however, we can overlap all but cycles 4-7 with the preceding and following computations. The complex multiply then takes us only 667ns, when mixed in with other computations.

Below is a summary chart of the complex multiply:

Cycle	Memories		Multiplier		Adder		Data Pad	
	TM	MD	M1, M2	FM	A1, A2	FA	0	1
1.	---	---	---	---	---	---	---	---
2.	---	---	---	---	---	---	---	---
3.	---	---	---	---	---	---	---	---
4.	W _R	C	W _R , C _R	---	---	---	---	---
5.	W _I	C _R	W _I , C _R	---	---	---	---	---
6.	W _I	C _I	W _I , C _I	---	---	---	---	---
7.	W _R	C _I	W _R , C _I	W _R *C	---	---	W _R C _R	---
8.	---	I	R	W _I *C _R	---	---	W _R C _R	W _I C _R
9.	---	---	---	W _I *C _I	W _I C _I , W _R C _R	---	W _R C _R	W _I C _R
10.	---	---	---	W _R *C _I	W _R C _I , W _I C _R	X _R	W _R C _R	W _I C _R
11.	---	---	---	---	---	X _I	X _R	W _I C _R
12.	---	---	---	---	---	---	X _R	X _I

3.6 S-PAD

The S-Pad is a 16-bit wide integer unit used primarily to compute memory address pointers and to test loop counters. It is similar in capability to a minicomputer and programs like the register-to-register instructions of a Nova or PDP-11 computer. There are 16 registers in the S-Pad unit.

3.6.1 Single Operand Instructions. Choose one from each column.

Operation	Shift	No Load	Destination Register
INC	---	---	dst;
DEC	R	#	
COM	L		
CLR	RR		

The Operation is performed upon the contents of the Destination Register (dst) and that result is Shifted. The shifted result is stored into the Destination REGISTER unless a No Load (#) is specified. The shifted result is the S-Pad Function (SPFN), which may be stored into an address register (MA, TMA, or DPA), or placed onto the Data Pad Bus (DB=SPFN). Some examples, where SP_n refers to the contents of S-Pad register "n":

INC 6	"(SP ₆ +1)→SP ₆
DECR 3	"(SP ₃ -1)/2→SP ₃
COM 3; DPX<SPFN	" $\overline{SP_3}$ →SP ₃ →DPX
CLR# 2; SETDPA	"0→DPA; because of # (no load) SP ₂ remains unchanged.

3.6.2 Double Operand Instructions. Choose one from each column.

Operation	Shift	No Load	Decimate	Source Register	Destination Register
MOV	---	---	---	src,	dst,
ADD	R	#	&		
SUB	L				
AND	RR				
OR					
EQV					

The Operation is performed between the Source (src) and Destination (dst) registers. If Bit Reverse (&) is specified the contents of Source are bit-reversed before being used. The Shift is performed on the result, which is then stored into the Destination Register, unless No Load (#) is specified. The shifted result is the S-Pad Function (SPFN), which may be stored into TMA, MA, or DPA, or placed onto the Data Pad Bus. Some examples:

MOV 3,15	"SP ₃ SP ₁₅
ADDL 6,10; SETMA	"(SP ₁₀ +SP ₆)*2 SP ₁₀ →MA
SUB 7,13	"(SP ₁₃ -SP ₇) SP ₁₃
AND#5,11; SETDPA	"(SP ₁₁ AND SP ₅)→DPA
OR# &6,7; SETTMA	"(SP ₇ OR SP ₆ (Bit-reversed)) →TM
MOVRR 2,2	"(SP ₂)/4→SP ₂

For purposes of program clarity, the assembler allows names to be given to the S-Pad registers. If register "PTR" is a pointer to an array in Data Memory, and register "STEP" contains the increment value we are using to step through array, then:

```
ADD STEP,PTR; SETMA
```

will advance our array pointer by the proper increment and fetch the next array element from memory.

3.6.3 S-Pad Tests. The following conditional branches test the S-Pad Function (SPFN).

BR LOOP	"Branch unconditionally to program location "LOOP"
BEQ LOOP	"Branch if SPFN=0
BNE LOOP	"Branch if SPFN≠0
BGE LOOP	"Branch if SPFN≥0
BGT LOOP	"Branch if SPFN>0

The above branches test the S-Pad result from the immediately preceding AP instruction. Thus an S-Pad operation must be done one instruction cycle before it is desired to test the result: A loop counting example:

1. DEC 2 "Decrement SP₂
2. BNE LOOP "Branch to "LOOP if SP₂ has not yet reached "zero.

Test the contents of SP₃ to see if it is between a lower limit contained in SP₂ and an upper limit in SP₄, i.e. see if SP₂ ≤ SP₃ ≤ SP₄.

1. SUB# 3,2
2. SUB 4,3; BGT SMALL "Too small, SP₃<SP₂
3. BGT BIG "Too big, SP₃>SP₄

The branches are made relative to the current Program Source Address (PSA) with a 5-bit displacement value. This means that the branch target address must be within -20_8 to $+17$ locations of the current instruction.

3.6.4 An example. Load Data Pad X with an array "A", with N elements starting at Main Data Memory location 3721_8 . "CTR" is in S-Pad register which will be used as a counter.

1.	CLR# CTR; SETDPA	"Set DPA to \emptyset
2.	LDMA; DB=3721	"Fetch the first element
3.	LDSPI CTR; DB=N	"Initialize "CTR" to N
4. LOOP:	INCMA; DEC CTR	"Fetch next element, A_{i+1}
5.	DPX <MD; INCDPA; BNE LOOP	"Store A_i into DPX_i , advance "DPA and test counter.

Below is a chart of the above loop, for N=3 elements.

Inst. #	Memory		Data Pad			S-Pad	
	MA	MI	DPA	0	1	2	"CTR" Test
1.	---	---	0	---	---	---	---
2.	3721	---	0	---	---	---	---
3.	---	---	0	---	---	---	3
4.	3722	---	0	---	---	---	3
5.	---	A ₀	0	A ₀	---	---	2 true
4.	3723	---	1	A ₀	---	---	2
5.	---	A ₁	1	A ₀	A ₁	---	1 true
4.	---	---	2	A ₀	A ₁	---	1
5.	---	A ₂	2	A ₀	A ₁	A ₂	0 false

A generalization on the above example is to fetch array "A" from every Kth memory location. The increment is stored in S-Pad register "STEP", and the array pointer is stored in "PTR":

1.	LDSPI STEP; DB=K	"Initialize "STEP" to K
2.	CLR# CTR; SET DPA	"Set DPA to 0
3.	LDMA; DB=BASE	"Fetch the first element, A ₀
4.	LDSPI CTR; DB=N	"Initialize "CTR" to N
5. LOOP:	ADD STEP, PTR; SETMA BEQ DONE	"Advance memory pointer. Fetch " next element, A_{i+1} . Test " counter and jump out if " done.
6.	DPX <MD; INCDPA DEC CTR; BR LOOP	"Store A_i into DPX_i , advance DPA " Decrement "CTR" and jump " back to LOOP.
7. DONE:	---	

SECTION
INTERFACE

This section describes the interface between the Host computer and the AP-120B. The interface is composed of two basic parts, 1) a simulated front panel and 2) direct memory access control (DMA). The front panel allows the host computer to examine or modify the internal AP-120B registers, as well as to start and stop program execution. The DMA control provides for block transfer of data from the host computer to the AP-120B, and vice versa.

4.1 FRONT PANEL

The AP-120B "Panel" is used for bootstrap operations (loading and starting programs) and for debugging of user software (inserting hardware breakpoints and examining and modifying AP-120B registers and memory). The "Panel" consists of three 16-bit registers which are under the control of the Host via the Host interface. The functioning of these registers closely parallels that of the switches and lights on the console of a stand-alone computer. The Host can examine and/or set these registers at any time, irrespective of the state of the AP-120B.

4.1.1 Switch Register. The Switch Register (SWR) is used to enter data and addresses into the AP-120B. The SWR can be read and written by the Host computer. An executing AP-120B program can also read the switches.

4.1.2 Lights Register. The Lights Register (LITES) simulates front panel lights, and is used to display the contents of internal AP-120B registers. This register can only be read by the Host. The executing AP-120B program can set the Lights Register.

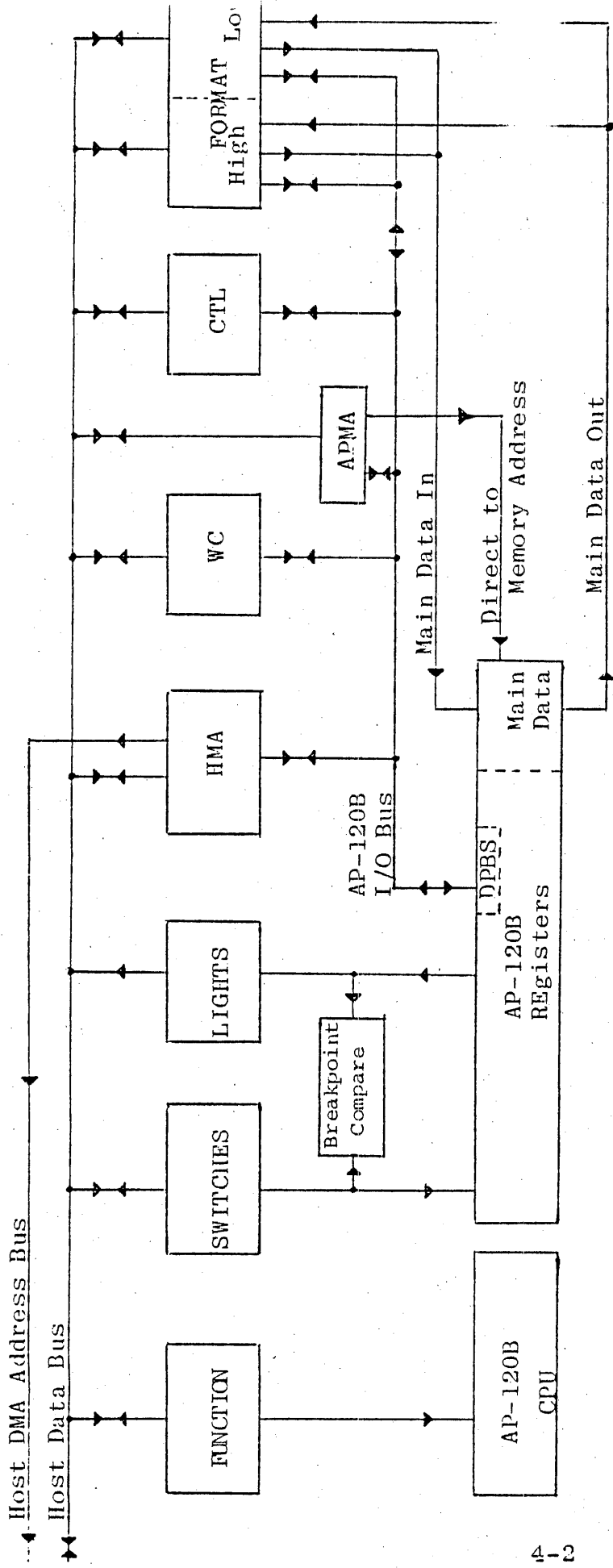
4.1.3 Function Register. The Function Register (FN) provides front-panel control operations (start, stop, continue, etc.). It can be read or written by the Host.

PANEL FUNCTION REGISTER FORMAT

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
STOP	START	CONT	STEP	RESET	EXAM	DEP	BREAK	INC		WORD		REGISTER SELECT			

When the AP-120B is running only the STOP and RESET panel functions are valid. The other panel functions can only be exercised after the AP-120B has been halted.

FIGURE 4.1 AP120-B PANEL AND HOST INTERFACE



AP-120B Device Addresses

Register	DA
WC	0
HMA	1
CTL	2
APMA	3
FORMAT	4

Function Register Bits

<u>Bit</u>	<u>Mnemonic</u>	<u>Effect</u>
Bit 0	STOP/HALTED	Stop AP-120B program execution upon completion of the current instruction. When the Host reads the FN register this bit reflects the current state of the processor, i.e., it will be a "1" if the AP-120B is halted. NOTE: If the current instruction does a "SPIN" while waiting for I/O or memory, the STOP will not take effect until the spin condition is satisfied and the instruction completed.
Bit 1	START	Start program execution at the address specified in SWR.
Bit 2	CONT	Continue program execution at the instruction pointed at by PSA (Program Source Address).
Bit 3	STEP	Execute the instruction pointed at by PSA and then halt. Advance PSA to point to the next instruction.
Bit 4	RESET	Stop the AP-120B immediately. Clear S-Pad register \emptyset , set SPFN to SP _{SPD} , clear the AP-120B status register, stop the host DMA (CTL bit 15 set to \emptyset) and clear Main Data Memory timing.
Bit 5	EXAM	Examine the register or memory selected by the Register Select field. Display the portion selected by the WORD field in the Panel Display Register.
Bit 6	DEP	Deposit the contents of the Switch Register into the register or memory selected by the Register Select field. Deposit into the portion selected by the WORD field.
Bit 7	BREAK	Enables hardware breakpointing if PSA, MA, or TMA is specified in the Register Select field. The breakpoint causes the AP-120B to halt one instruction after any instruction where the contents of the selected register was equal to the Switch Register. Thus, if a breakpoint is specified with PSA selected the AP-120B will halt after executing the instruction at the program location set in the Switch Register. PSA will be pointing at the next micro-instruction in sequence. If a breakpoint is called for on MA or TMA, the AP-120B will halt after executing the instruction following the one that referenced the trapped memory location. PSA will point to the second following sequential instruction after the one that caused the breakpoint. Memory breakpoints aid in debugging those elusive errors that modify memory unexpectedly.

<u>Bit</u>	<u>Mnemonic</u>	<u>Effect</u>
Bits 8-9	INC	Increment MA, TMA, or DPA following completion of the other specified panel functions. This allows sequential memory locations to be examined or deposited into.

<u>Value in Bits 8-9</u>	<u>Address Register to be incremented</u>
0	None
1	MA (Memory Address)
2	DPA (Data Pad Address)
3	TMA (Table Memory Address)

Bits 10-11	WORD	Specifies which portion of a register is being examined or deposited into.
------------	------	--

<u>Portion of Register Affected</u>			
<u>Value Set in Bits 10-11</u>	<u><16-Bit Register</u>	<u>38-Bit Register</u>	<u>64-Bit Register</u>
0	ALL	N/A	Word 0, Bits 0-15
1	N/A	Exponent Bits 00 to 09. Right justified in 16-bit field.	Word 1, Bits 16-31
2	N/A	High Mantissa Bits 00 to 11. Right justified	Word 2, Bits 32-47
3	N/A	Low Mantissa	Word 3, Bits 48-63

Bits 12-15	REG.SELECT	Specifies which AP-120B internal register or memory location to examine or deposit into.
------------	------------	--

Function Register Bits Cont'

<u>Octal Value set in Bits 12-15</u>	<u>Register or Memory Selected</u>	
0	PSA	Program Source Address
1	SPD	S-Pad Destination Address
2	MA	Main Data Address
3	TMA	Table Memory Address
4	DPA	Data Pad Address
5	SPFN	S-Pad Function (EXAM)
	SP _{SPD}	S-Pad addressed by SPD (DEPOSIT)
6	AP Status	AP-120B Internal Status Reg.
7	DA	Device Address Register
10	PS _{TMA}	Program Source Memory addressed by TMA
11	HD	Host Data (EXAM only) - Reads output of Format Conversion Reg.
12	CB	Control Buffer, Bits 48-63 (EXAM only)
13	DPX _{DPA-4}	Data Pad X addressed by (DPA-4)
14	DPY _{DPA-4}	Data Pad Y addressed by (DPA-4)
15	MD _{MA}	Main Data Memory addressed by MA
16	SPFN	S-Pad Function (EXAM only)
17	TM _{TMA}	Table Memory addressed by TMA (EXAM only)

4.2 NOTES ON THE USE OF THE FRONT PANEL AND BREAKPOINT

4.2.1 Where does the AP stop on a breakpoint?

- a) With the breakpoint set on PSA, the AP-120B will stop with PSA pointing to the next instruction to be executed.

Thus breaking on a branch instruction and then examining PSA will show whether the branch condition was true or false.

- b) With the breakpoint set on TMA the AP-120B will stop with PSA pointing to the second instruction following the one that set TMA to the break address.
- c) With the breakpoint set on MA the AP-120B will stop on either the next instruction or the second instruction after the one that set MA to the break address, depending on the state of the memory lockout hardware (next instruction if memory lockout, second instruction if no memory lockout).

Thus the stopping point following an MA breakpoint will have a one instruction uncertainty.

4.2.2 Does the instruction on which the AP stops execute?

Since SPFN is current, it will be set to the operation specified in the instruction that PSA is pointing to. Otherwise, the instruction that PSA is pointing to remains unexecuted and will execute correctly when the user steps or proceeds from the breakpoint.

4.2.3 What about MD timing and lockout on a breakpoint in the middle of an MD memory cycle?

- a) The hardware has been designed so that the AP can be stopped in the middle of a memory cycle. The hardware remembers where the memory timing was when the AP stopped so that the processor can continue correctly from a breakpoint that occurs during a memory cycle.
- b) However, the user must not examine MD nor should there be any DMA transfers going to or from MD while the AP is stopped if the user wishes to proceed from the breakpoint.

Thus, for example, it is possible to break in the tight-to-memory portions of the FFT and examine DATA PAD or the address registers (PSA, SPA, etc) and then proceed. But it is not possible to proceed if the user or the host interface disturbs the memory timing by reading or writing MD or TM.

4.2.4 Summary of the rules for proceeding from breakpoint.

If the breakpoint causes the AP to stop in the middle of the memory cycle (PSA pointing to first or second instruction following SETMA, INCMA, DECMA or LDMA), the user should not try to examine or modify MD.

4.2.5 What about stepping the AP?

The same rules as for proceeding from a breakpoint apply to stepping the AP through a program. The user can examine and modify any register of memory within the constraints mentioned in 4.2.4 above.

4.2.6 What other pitfalls are there in the use of the virtual front panel?

- a) Note that the panel always examines SPFN not SP_{SPD}. Thus, if the user wishes to see SP_{SPD} he must force SPFN = SP_{SPD}. This can most easily be done via the panel reset function which has the unhappy side effect of also clearing SP(\emptyset).
- b) To examine TM, the user should first set TMA and then do a dummy panel operation (deposit TMA again for example) in order to enter the output of table memory into the table memory buffer register. He can then proceed to examine the addressed location using the appropriate panel functions.

c) MD

Setting MA from the panel initiates an MD memory read cycle. Depositing into MD from the panel initiates an MD memory write cycle.

Thus, to write MD and then examine what was just written, the user must perform a deposit into MA operation (with the same address) to initiate a read cycle before examining MD.

d) Using the Increment field in the FN register.

DPA and TMA always increment after the EXAM or DEP operation is complete (remember that TMA is used to address program source memory for panel operations).

MA post-increments and initiates a new memory read cycle on an EXAM operation.

MA pre-increments on a DEP operation.

e) Starting the AP

The recommended starting procedure is as follows:

- i) Set the SWR to the starting address and do a deposit into PSA
- ii) Set the SWR to the desired breakpoint and do a continue to start the AP-120B.

This procedure has the significant advantage that it places the necessary breakpoint code into the user's program should he need to debug his AP program.

The panel START function can be used but the user should observe the following restrictions on the first instruction executed by the AP-120B:

The first instruction should not branch or jump or modify PSA in any way other than to advance to the next instruction. The first instruction should not use the SPEC or IO fields. In fact, the preferred first instruction is a NOP (all zeros).

4.3 DIRECT MEMORY ACCESS

In addition to the Panel function, the AP-120B contains four 16-bit registers that are used for Direct Memory Access (DMA) to both Host and AP-120B data memory plus a 38-bit Format Conversion Register that acts as buffer between the two memories. These registers may be read and/or loaded from either the Host computer or the AP-120B.

4.3.1 Host Memory Address Register. The Host Memory Register (HMA) points to consecutive locations in the memory of the Host Computer. It operates in either an auto-increment or auto-decrement mode during DMA transfers to and from Host memory. HMA is Device Address 1 for AP-120B internal I/O transfers.

4.3.2 Word Count Register. The Word Count Register (WC) counts the number of Host memory words transferred in a DMA operation. It is preset to the desired number of words to be transferred and counts down as the transfer proceeds, stopping the DMA transfer when it reaches zero. Hardware logic prevents this register from being counted past zero. WC has AP-120B Device Address 0.

4.3.3 AP Direct Memory Address Register. The AP-120B Memory Address Register (APDMA) points to consecutive locations in AP-120B Main Data Memory during DMA transfers to and from MD. This register can operate in either auto-increment or auto-decrement mode. APDMA has AP-120B Device Address 3.

4.3.4 Control Register. The Control Register (CTL) acts as a control over the DMA and interrupt functions of the Host Interface. This register controls the direction and mode of transfer (DMA or program control), the type of data format, and provides certain bits a status information pertaining to the transfer. CTL has AP-120B Device Address 2.

DMA CONTROL REGISTER FORMAT

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WC=0	INTR AP	IAP WC	IH HALT	IH WC	IH ENB	FERR	DLATE	CC	AP DMA	WRT HOST	DEC APMA	DEC HMA	FMT		HDMA START

All bits are read/write except as noted.

Bit 0 WC=0 Indicates that the Word Count Register is zero. Note that WC is decremented only during DMA Transfers to/from Host Memory (Read only bit). Should not be used to monitor DMA activity.

Bit 1 INTR AP Set the INTRQ (Interrupt Request) flag in the AP-120B.

Bit 2 IAPWC Set INTRQ (Interrupt Request) flag in the AP-120B when the DMA transfer is done.

Bit 3	IHALT	Enable a Host interrupt when the AP-120B halts.
Bit 4	IHWC	Enable a Host interrupt when the DMA transfer is done.
Bit 5	IHENB	Interrupt Host Enable. Interrupt Host if AP-120B <u>attempts</u> to set this Bit. This bit can actually be written only by the Host.
Bit 6	FERR	Format Error. Indicates that exponent underflow or overflow occurred in conversion from AP-120B format to Host floating-point format.
Bit 7	DLATE	Data Late. Indicates that the AP-120B did not empty the format buffer before the Host attempted to reload it. On some Hosts this bit also indicates an attempt to access non-existent Host memory. In either case the DMA transfer is terminated.
Bit 8	CC	Consecutive Cycle. Block DMA transfers to/from Host memory will occur without interruption. On typical Hosts, the Host CPU will be locked out but other higher priority DMA devices will still have access to Host memory.
Bit 9	APDMA	Allows the interface to perform DMA transfers to/from AP-120B memory. Depending on the direction of transfer, a Main Data memory cycle is initiated every time the Host finishes reading or loading the format register, whether via DMA or program control. On the AP-120B side, the format register is loaded from the Main Data Bus instead of the Data Pad Bus.
Bit 10	WRTHOST	Write to Host. This bit controls the direction of transfer. If set, data is read from the AP-120B, passed through the format register, and written to the host. If clear, the direction of transfer is reversed.
Bit 11	DECAPMA	Decrement APMA. If set, APMA is decremented during DMA transfers to/from AP-120B Main Data memory. If clear, APMA is incremented.
Bit 12	DECHMA	Decrement HMA. If set, HMA is decremented during DMA transfers to/from Host memory. If clear, HMA is incremented.

Bits 13&14 FMT

Format Register Control.

Note that the format register mode of operation is controlled entirely by bits 9, 10, and 13 and 14 of the control register. Thus even though the Host and the AP-120B can load and read the Format Register via program control I/O transfers at any time, the programmer must be sure that the type of transfer he performs is consistent with these bits of CTL for the transfer to be meaningful.

<u>Value in</u> <u>Bits 13-14</u>	<u>Format Type</u>
0	32-bit integer. No format conversion. Used to transfer integers or program half-words.
1	16-bit integer. 16-bit integers from Host are converted to unnormalized 38-bit AP-120B FPN's. Low 16-bits of AP-120B FPN are sent to Host.
2	Conversion of "Sign-magnitude mantissa with binary exponent" format to/from AP-120B Floating Point format. Includes logic to handle "Phantom bit" formats.
3	Conversion of IBM 32-bit format to/from AP-120B format. IBM format can be specified to have either sign-magnitude or 2's complement mantissa.

Note: For format types 2 and 3, the format register has the necessary logic to detect overflow and underflow on conversion from AP-120B format and to force a signed maximum quantity on overflow or floating-point zero on underflow.

Bit 15 HDMA Start /
Busy

Host DMA Start. Initiate DMA transfers to/from Host memory. When read the state of this bit reflects the status of the Host DMA activity ('1' if active, '0' if inactive). Transfers continue until WC=0.

4.4 FORMAT CONVERSION REGISTER

This 38-bit double-buffered register is used for all transfers of floating-point numbers (FPN's) between the Host and the AP-120B. It also provides the most efficient path for transfer of micro-code half-words (32-bits). It performs bi-directional format conversions under the direction of Bits 9, 10, 13 and 14 of the CTL Register. The programmer must be aware of the fact that the Format Conversion is a slave to these CTL Bits. Transfers to and from the formatter must be consistent with these bits or nonsense will result. The Host and AP-120B can read the output of the formatter at any time without restriction, however, the input to the formatter is controlled by CTL bits 9 and 10.

CTL09	CTL10	Input Path to Formatters
0,1	0	Host Data Bus
0	1	AP-120B I/O Bus
1	1	AP-120B Main Data Output

The formatter has a ready indicator that can be sampled by the AP-120B. This indicator tells the AP-120B when to load new data into the formatter (CTL10=1) and when to read data from it (CTL10=0) after the Host has finished reading or loading the last 16-bit word of an FPN.

Note that in 16-bit Host Computers, the interface expects to receive words in different order depending on CTL Bit 12 (DECHMA). If Bit 12 is clear i.e., the Host DMA interface is going through memory in forward order from low to high addresses, then the interface expects to receive the high word of an FPN followed by the low word. If bit 12 is set, the interface expects to receive the low word followed by the high word. This is done so that arrays of FPN's are always stored in forward order in Host memory.

If the Format Bits of CTL (Bits 13,14) specify a 16-bit transfer (FMT=1) then the integer is loaded and read from the low word of the formatter. And that word is considered to be the last word transferred.

There is no corresponding indicator to the Host since the AP-120B can transfer data to and from the formatter faster than most HOST processors. The DLATE bit in the CTL Register (CTL Bit 7) does indicate when an error of this type occurs, i.e., when the Host transfers data faster than the AP-120B.

4.5 AP-120B INTERNAL INTERFACE TO HOST INTERFACE

The Registers in the Host interface are accessible to the AP-120B via its Input/Output (I/O) instructions (FADD=7).

AP-120B Device Addresses for Host Interface Registers

Register	DA
WC	0
HMA	1
CTL	2
APMA	3
FORMAT	4

An IN, OUT or SNSA instruction at DA=4 (FORMAT) will generate an IODRDY Response if the FORMAT Register is ready to accept data from the AP-120B (CTL Bit 10='1') or if it has formatted data ready for the AP-120B (CTL Bit 10='0'). If CTL Bit 9 is a '1', the AP-120B cannot load the formatter via I/O instructions since the input multiplexer to the FORMAT Register will be set to select MAIN DATA instead of the AP-120B I/O Bus. Note that the AP-120B cannot change the state of CTL Bit 5. If it attempts to set this bit when the bit has been previously set by the Host, an interrupt of the Host will be generated. The AP-120B can read the CTL at any time without interfering with the HOST interface. If both the Host and the AP try to write CTL or access HMA, WC or APMA at the same time, the Host selection and data has priority over that of the AP-120B.

Access to the Format Conversion Register is controlled by CTL Bits 9, 10, 13 and 14. See Section 4.3, Format Conversion Register, for a description of the function of these bits.

4.6 AN EXAMPLE, LOADING PROGRAMS INTO THE AP-120B

Loading and running a program in the AP-120B from a "cold start" is a five step process which nicely illustrates use of the front panel:

1. Through the AP front panel from the host computer we "finger switch" in a three instruction bootstrap program into Program Memory.
2. We start the bootstrap running.
3. Set the address where we want the loaded program to go in the AP-120B.
4. We start with a DMA transfer of program words from host computer memory to the AP-120B. The bootstrap program running in the AP-120B stores these words into Program Memory.
5. When the DMA transfer is done, we stop the bootstrap program in the AP-120B; and then restart the AP-120B executing our newly loaded program.

These five steps are detailed below. DMA control and front panel interrogation is done from the host computer by setting various interface registers. The actual host computer I/O instructions to accomplish this, of course, depend upon the particular host computer. Section 5 describes the proper I/O instructions for a number of host computers. For the purposes of this explanation, we merely indicate what numbers get loaded into which interface register in order to accomplish the desired goals.

Step 1

For the purposes of this example, we are going to put the bootstrap program into Program Source Memory locations 0, 1 and 2.

1. Set TMA to 0 (TMA is the pointer used by the panel functions for examining or depositing into Program Memory):

0 → SWR	Put 0 into the switches.
1003 → FN	Put 1003 into the Function register (causing a deposit into TMA)

2. Put bits 0-63 of bootstrap program program word #1 into Program Memory location 0 using four deposits of SWR → PSTMA.

(bits 0-15) → SWR	Put bits 0-15 into the switches.
1010 → FN	Put 1010 into the function register (causes a deposit into bits 0-15 of PS _{TMA}).

(bits 16-31) → SWR 1030 → FN	Puts bits 16-31 into bits 16-31 of PS _{TMA} .
(bits 32-47) → SWR 1050 → FN	Puts bits 32-47 into 32-47 of PS _{TMA} .
(bits 48-63) → SWR 1370 → FN	Puts bits 48-63 into bits 48-63 of PS _{TMA} and increment TMA to point to location 1.

- Repeat number 2 above bootstrap program words #2 and #3.

The above sounds worse than it really is, and besides, we only have to do it once (unless we clobber the bootstrap turn off power).

Step 2

Set the address in the AP-120B Program Memory where we want our program to get loaded by the bootstrap into TMA. For this example, this address shall be 200:

200 → SWR	Put 200 in the switches.
1003 → FN	Put 1003 into the function register (cause a deposit into TMA).

Step 3

Start the bootstrap program running in the AP-120B.

- Set the switches to \emptyset and do a start.

\emptyset → SWR	Start the AP-120B at location \emptyset
40000 → FN	

The bootstrap program (as we shall see below) will sit and "spin", waiting for words to come across the DMA from the host computer.

Step 4

Start the DMA transfer going from host memory into the AP-120B. For this example, we will assume that the program we wish to load is sitting in host memory at location 20000. We will further say that the program to be loaded is 200 AP program words (or 800 16-bit host words) long. The actual host memory location and length, could, of course, be any particular value.

20000 → HMA	Set host DMA address to 20000.
800 → WC	Set word count to 800 host words (assuming a 16-bit host word width).
201 → CTL	Start the DMA going.

Note in particular, the "CTL" bits. Bit 15 starts things going and bit 8 requests consecutive memory cycles from the host. By not setting bits 10 or 11 we set the transfer to go to the AP-120B, but not into Main Data Memory. Instead, the data goes only as far as the formatter which our bootstrap will read. If we had set bit 4, the host computer would be interrupted when the DMA is done. As things are, we will keep things simple and not use the host interrupt.

Step 4.5

And now, finally, we get to our three word bootstrap program running in the AP-120B:

```
0.  LDDA;  DB=4          "Set DEVICE ADDRESS to 4
```

This instruction sets the Device Address register, so that future I/O instructions will refer to device #4, which is the DMA formatter (where the data from the host computer ends up).

```
1.  LOOP:SPININ;        "Wait for some data
      DB=INBS;           "Get the data
      LPSLT              "Put it into the left half of P.S.
```

The "SPININ" causes the processor to hang until the current I/O device address (in our case, the DMA formatter) has some new data; and then to read that data the "DB=INBS" puts the input data onto the Data Pad Bus. The "LPSLT" puts what is on the Data Pad Bus into the left half (bits 0-31) of the Program Memory location pointed at by the TMA register.

Two points here: 1) The formatter is 32-bits wide on the AP-120B end. Every time the interface has gotten 32-bits of data from the host computer the "SPIN" stops waiting and we have another 32-bits of data. Since the program words we are loading are 64-bits wide we get them in halves (left, right, left, right, etc.) and store them accordingly into Program Memory. 2) We used TMA as a pointer to where our bootstrap would put the program it is loading, so the "LPSLT" puts the program words into the proper place.

```
2.  SPININ;            "Wait for data
      DB=INBS;         "Get the data
      LPSRT;           "Put it into the right half
      INCTMA;          "Increment our pointer
      BR LOOP          "Go back for more
```

This does basically the same as #1 above, except that here we have the right half (bits 32-63) of a 64-bit program word. The "INCTMA" increments our "storing" pointer so instruction #1 will store its data into the next word. The branch keeps using loop, insatiably waiting for more program half-words.

Step 5

Back in the host, we wait for the DMA transfer to be done by:

1. Read the CTL register
2. Test for bit 15 (the LSB) equal to 1
3. If so, go back to 1.

We could, of course, have enabled a host interrupt on DMA completion.

When DONE, we stop the bootstrap program (which otherwise would run forever) with a panel RESET function; and then start our newly loaded program, (our example starts at location 200):

40000→ FN	RESET the AP-120B
200 → SWR	Our new program address
1000 → FN	Set 200 into PSA
20000→ FN	Continue (from 200), i.e., start at A.P. location 200

Had we wished to set a program breakpoint, we would have set the breakpoint address into the SWR and used 20400 (continue + break on PSA) for our final panel function.

Postscript

The simplest way for the AP-120B program that we have now set running to indicate to the host computer that it is done with its task is to HALT. When this happens bit 0 in the Panel Function Register will come on, which the host can test for; or a host interrupt can be enabled (CTL bit 3).

SECTION 5

PARTICULAR HOST INTERFACES

This section describes the particulars of several host computer / AP-120B interfaces. In general, the interface consists of eight registers which are accessible to either the host computer or the AP-120B. Three of these registers comprise the AP-120B "front panel", while the other five control the DMA (Direct Memroy Access) connection between the two processors.

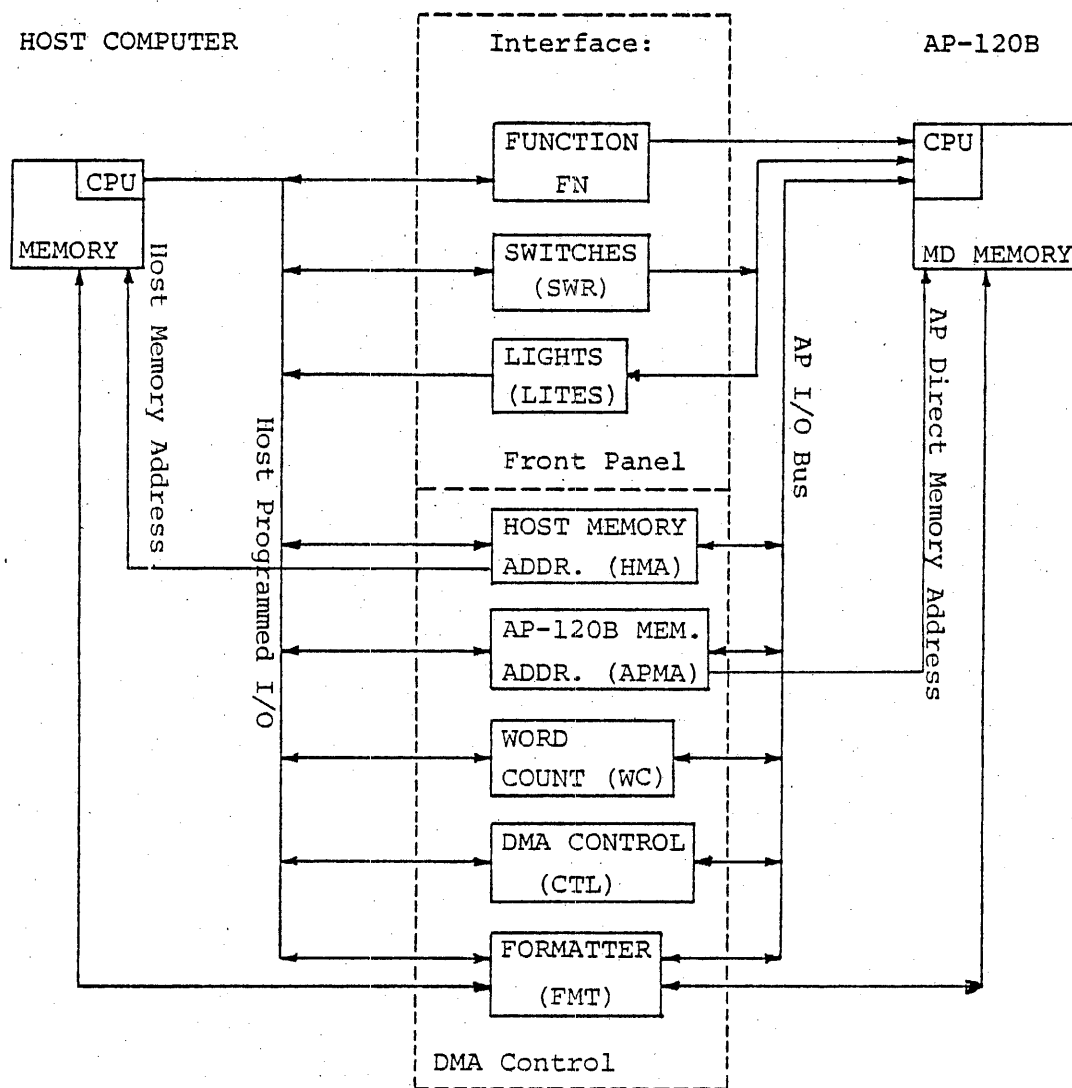


Figure 5.1 General AP-120B/Host Computer Interfaces

The host computer controls the AP-120B by loading and reading these eight registers. Thus, the AP-120B may be likened to a "smart" disc or tape unit, responding to programmed I/O instructions from the host to initiate desired tasks.

The exact effects and significance of these eight registers are described in detail in Section 4 of this manual, which explores the AP-120B side of the interface. This section details how particular host computers access the interface registers, and hence control the AP-120B.

5.2 DEC PDP-11 INTERFACE

3.2.1 Interface Register Address:

Unibus Address	Interface Register
AP+100	Word Count (WC)
AP+102	Host Memory Address (HMA)
AP+104	DMA Control (CTL)
AP+106	AP Memory Address (APMA)
AP+110	Panel Switches (SWR)
AP+112	Panel Function (FN)
AP+114	Panel Lites (LITES)
AP+116	Reset (Same as Panel Reset Function)
AP	Formatter

The base address "AP" is strappable between 174000 and 177600.

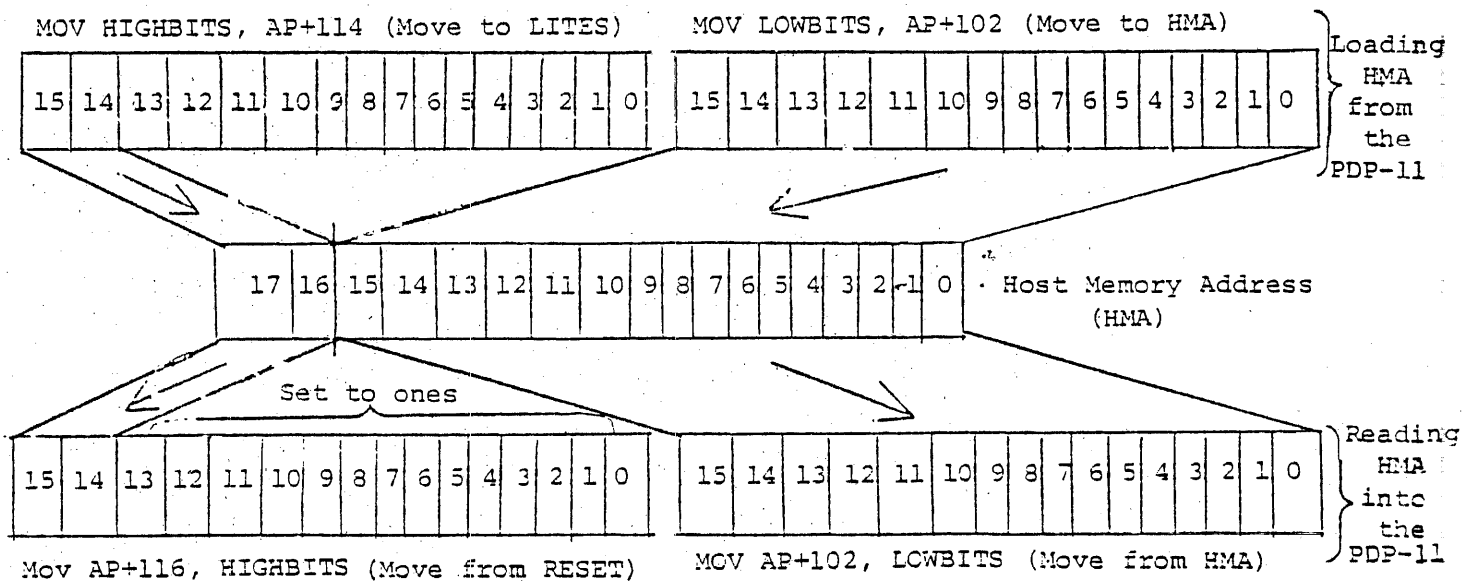
5.2.2 Comments. HMA is an 18-bit, even byte address that increments or decrements by 2. WC counts 16-bit PDP-11 word transfers.

Word format (CTL bits 13-14) type 2 refers to PDP-11 Fortran format (8-bit biased binary exponent, sign-magnitude mantissa with "hidden" MSB). Format type 3 refers to IBM 360 32-bit format.

In consecutive cycle mode (CTL bit 08 set) the PDP-11 DMA interface will steal a hardware selectable number (1 to 15) of consecutive memory cycles and then release the Bus.

This is done so that other time-critical DMA devices can get access to the bus.

5.2.3 Reading/Writing of Host Memory Address. The two high order bits of the Host Memory Address (HMA) register bits are set via an output to the Panel Lights (LITES) address, and read with an input from the Reset Address:



NOTE: The move to the LITES address (AP+114) does not affect the LITES register. Similarly, a move from the RESET address (AP+116) does not cause a Reset of the AP-120B.

An example; we wish to set HMA to 532706:

```
MOV #132706, AP+102 ; SET 12706 into HMA
MOV #100000, AP+114 ; ADD 400000 into HMA
```

and to read out of HMA this value we have set:

```
MOV AP+102, LOWBITS ; would get 12706
MOV AP+116, HIGHBITS ; would get 137777
```

5.2.4 Program Access of the Format Register. The AP-120B Formatter is 32-bits wide, and hence is accessed by the PDP-11 into 16-bit halves. The Formatter may be read/written at any even address between "AP" and AP+76. If DMA control (CTL) register bit 12 is clear (host address increments), successive high, low, high, low, ... halves of the Formatter are accessed. If CTL bit 12 is set (host address decrements) the sequence is low, high, low, high... In both cases, high refers to the first 16 bits of the floating-point number of 32 bit integer, and low refers to the last 16 bits.

When 16-bit integers are involved (CTL bits 13-14 set to "01") only 16-bit entities are involved on the PDP-11 end. The access then is simply by words: word #1, word #2, word #3...

5.2.5 Examples.

- 1) Stop the AP-120B and examine PSA (Program Address);

```
MOV #100000, AP+112 ; (STOP the AP-120B) to FN
MOV #2000, AP+112 ; (EXAM PSA) to FN
MOV AP+114, APSA ; Get the answer (from the LITES)
```

- 2) Start the AP-120B at program location 20, setting a breakpoint at PSA=200:

```
MOV #20, AP+110 ; Put 20 into SWR
MOV #1000, AP+112 ; (SWR → PSA) to FN
MOV #200, AP+110 ; 200 to SWR
MOV #20400, AP+112 ; (CONTINUE + BREAK on PSA) to FN
```

- 3) Initiate a DMA transfer of 100 PDP-11 format floating-point numbers from location 20000 in the PDP-11 to location 2000 in the AP-120B Main Data Memory:

```
MOV #200, AP+100 ; Set WC to 200 PDP-11 words
MOV #20000, AP+102 ; Set HMA to 20000
MOV #2000, AP+104 ; Set APMA to 2000
MOV #305, AP+106 ; Start the DMA
```

5.3 DATA GENERAL NOVA/ECLIPSE INTERFACE

5.3.1 I/O Instruction Assignments:

I/O Instruction	Interface Register
DOA/DIA AP1	Word Count (WC)
DOB/DIB AP1	Host Memory Address (HMA)
DOB/DIB AP1	DMA Control (CTL)
DOA/DIA AP2	AP-120B Memory Address (APMA)
DOB/DIB AP2	Formatter high word (FMT00-15)
DOC/DIC AP2	Formatter low word (FMT16-31)
DOA/DIA AP0	Panel Switches (SWR)
DOB/DIB AP0	Panel Function (FN)
/DIC AP0	Panel Lights (LITES)

Other I/O Instructions	Effect
NIOC AP0	Clear AP Interrupt Request
NIOS AP0	Reset (Same as RESET panel function)
MSKO	Set interrupt mask (bits normal)
SKPBZ AP0	Skip if AP halted
SKPDN AP0	Skip if interrupt request pending

Responds with Device Code "AP0" to INTA from the Nova.

Device codes AP0, AP1, and AP2 must lie within a single octade of device codes.

- 5.3.2 Comments. HMA increments and decrements and WC decrements, for each 16-bit Nova word and transfer.

Word Format (CTL bits 13-14) type 3 is Nova single-precision floating-point format (32-bit IBM sign-magnitude mantissa, 7-bit hex exponent). Format type 2 is not used.

5.3.3 Examples:

- 1) Stop the AP-120B and examine PSA (Program Address):

```
LDA 0, =100000
DOB 0, AP0      ; (STOP the AP-120B) to FN
LDA 0, =200
DOB 0, AP0      ; (EXAM PSA) to FN
DIC 1, AP0      ; get the answer (in LITES), into AC1
```

- 2) Start the AP-120B at program location 20, setting a breakpoint at PSA=200:

```
LOA 0, =20
DOA 0, AP0      ; Put 20 into SWR
LDA 0, =1000
DOB 0, AP0      ; (SWR + PSA) to FN
LDA 0, =200
DOA 0, AP0      ; Put 200 into SWR
LDA 0, =20400
DOB 0, AP0      ; (CONTINUE + BREAK on PSA) to FN
```

- 3) Initiate a DMA transfer of 100 Nova floating-point numbers from location 20000 in the Nova to location 2000 in AP-120B Main Data Memory:

```
LDA 0, =200
DOA 0, AP1 ; Set WC to 200 Nova words
LDA 0, =20000
DOB 0, AP1 ; Set HMA to 20000
LDA 0, =2000
DOA 0, AP2 ; Set APMA to 2000
LDA 0, =306
DOB 0, AP1 ; Start the DMA
```


5.4 RAYTHEON R7041RDS-500 INTERFACE

The AP-120B RDS 500 interface acts as a controller for the data transfers between the AP-120B interface registers and the RDS 500 data busses, as described in Section 4 of the AP-120B's Processor Handbook. The interface consists of three units: the Direct I/O controller, the DMA interface, and the data transfer receiver.

The DIO control recognizes the following DOT/DIN commands:

<u>DOT/DIN Function Assignment Field</u>	<u>Access Register</u>
1	Panel Switch Register (SWR)
2	Panel Function Register (FN)
3	Panel Lights Register (LITES) (Read Only)
4	DMA Word Count Register (WC)
5	Host Memory Address Register (HMA)
6	Control Register (CTL)
7	AP Memory Address Register (APMA)
8, A,C,E	Format High
9, B,D,F	Format Low
DIN \emptyset	Read Simple Status Word
DOT \emptyset	Reset

Note that the Panel Display Register (PDR or LITES) can only be read by the host. Attempting to load this register results in a no-op in the DIO control.

Executing a DIN instruction with the function assignment field equal to zero reads a status word on to the DIN bus, true status is indicated by a one in the appropriate position. The status bit assignment is as follows:

<u>DIN Bit</u>	<u>Assigned Status</u>
\emptyset	AP running
1	AP DMA request present
15	Error (Data late or Format Conversion Error)
2-14	Unused

Bit 15 is the result of an inclusive OR of control register (see Section 4.2) bits 6 (format error) and 7 (data late). Bits 2-14 are not used and are always zero when the status word is read.

Dot 0 is used as a reset command. Executing this instruction clears DMA timing and pending requests.

The device address that the AP-120B DIO control will respond to is selectable by a hexadecimal DIP switch mounted on the interface card. Any address from 0 - 15 present on DAD lines 8 - 11 can be selected.

DMA Operation is a slave to the state of the control register. Loading the control register with bit 15 set (HDMA start) initiates DMA transfers with mode and direction determined by CTRL bits 8 - 14. Bit 8 (consecutive cycle) selects the burst mode. When set, DMA transfers occur in blocks of up to 16 consecutive cycles, after which the DMA request line is released for one cycle to allow other devices access to the DMA channels. The actual number of cycles, (executed before the request line is freed) is selectable by a hexadecimal switch located on the interface card. When the consecutive cycle bit is clear, the request line is released for one cycle after executing one transfer. DMA transfers continue until the word count register reads zero. The word count is loaded with the actual number of 16-bit words to be transferred. If loaded with zero, however, one 16-bit transfer will occur.

Jumpers for the memory request (MRQ1 - MRQ8), memory acknowledge (MAK1 - MAK8), memory write (MTW1 - MTW8), and interrupt request (IRPT 00 - IRPT 15) lines are located on the AP connector panel. Selection of the desired priority is made by placing jumper plugs in the appropriate positions. The connector panel is clearly labeled and is visible from the rear of the unit.

The interface is equipped with a transceiver capable of accepting data in four different formats (outlined in Section 4.2 Direct Memory Access). Since the control register bits 13 and 14 specify the format to be used when loading the formatter under both DMA control and DIO functions 8 and 9, data transfers must be consistent with the state of these bits or nonsense will be output from the formatter. Data transfers that do not use the formatter will not be affected by the state of the control register (DIO functions 0 to 7).

Caution must be exercised when inputting floating-point numbers (FPN). In IBM 32-bit format, the high part of the formatter (DIO function = 8) refers to the sign, exponent, and high mantissa of the FPN, while the low part (DIO function = 9) refers to the low mantissa (see figure below). As an example, a DOT with a function assignment field of 8 would load the sign, exponent and high mantissa of the formatter. On the other hand, when in Raytheon real two-word floating-point number format,

high format refers to the mantissa least significant bits and the exponent, while the low word of the format is taken to mean the sign and mantissa MSB's. In this case then, a DOT with function assignment of 8 would load the format mantissa LSB's and exponent.

FMT HIGH

Mantissa LSBs	Exponent
Mantissa MSBs	

Raytheon Real
Two-Word

FMT LOW

FMT HIGH

Exponent	Mantissa MSBs
Mantissa LSBs	

IBM 32 bit floating
point

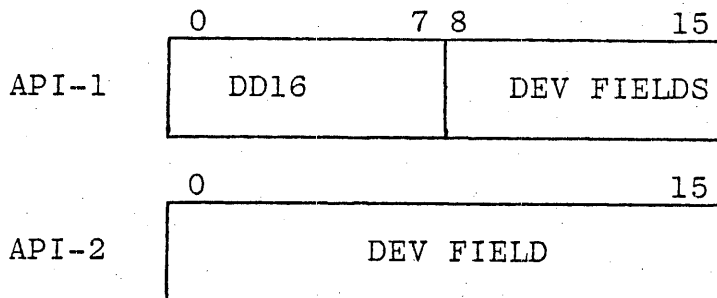
FMT LOW

32-bit integer format is a straightforward load of the high significant bits into FMT HIGH and low significant bits into FMT LOW. 16-bit integers are loaded into the FMT LOW.

5.5 TEXAS INSTRUMENTS TI980 INTERFACE

The AP-120B to TI980 interface acts as a controller for the data transfers between the AP-120B interface registers and the TI980 as described in Section 4 of the AP-120B Processor Handbook. The interface consists of three units: the Auxiliary Processor Port Controller (APP Controller), the Direct Memory Access Channel Interface (DMAC Interface), and the Data Transceiver.

5.1.1 APP Controller. The TI980 Auxiliary Processor Initiate Instruction (API Instruction) format is:

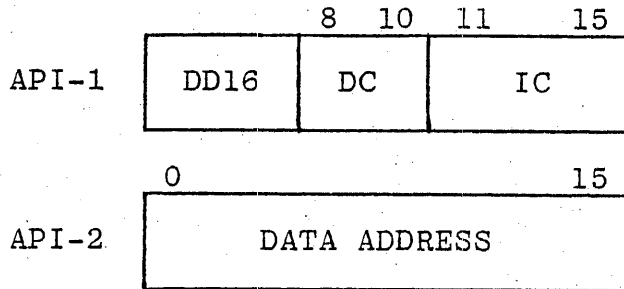


The first eight-bits of the API-1 word are decoded by the TI980 as the first word of an API Instruction. This causes the TI980 to send this instruction word, and the next sequential word in the TI980's central memory to the AP-120B through the APP.

After API-1 has been sent through the APP its other eight bits will be looked upon as having two separate fields. Bits 8,9, and 10 are labeled DC in the diagram below and used as a device code. Bits 11 through 15 are labeled IC and used as the AP-120B APP Controller's instruction code. The APP Controller will compare the device code contained in API-1 to the device code it has been wired to accept. The IC field will be taken as an instruction for execution by the APP Controller only if the DC field matches the AP-120B's APP Controller device code.

API-2 is labeled DATA ADDRESS in the diagram below and is used for this purpose if the DC field in API-1 matches the device code of the AP-120B's APP Controller. DATA ADDRESS is program relative when the TI980 is operating in User Mode and absolute when the TI980 is operating in Operator Mode.

The AP-120B's APP Controller may be wired to accept any device code of 0 through 7.



DC-APP Controller Device Code. The AP-120B's APP Controller can be strapped to user-specified codes of 0 through 7.

IC-AAP Controller Instruction Code.

5.5.2 APP Controller Instructions. Each of the AP-120B's Controller Instructions is described below. A summary is given after the description of the last instruction. IC numbers are given in radix 16 notation.

Load Switch Register

LSWR (IC=1)
DATA ADDRESS

The AP-120B's Switch Register (see AP-120B Processor Handbook 4.1.1) is loaded from the TI980 central memory at location DATA ADDRESS.

Load Function Register

LFN (IC=2)
DATA ADDRESS

The AP-120B's Function Register (see AP-120B Processor Handbook 4.1.3) is loaded from the TI980 central memory at location DATA ADDRESS.

Load Word Count Register

LWC (IC=4)
DATA ADDRESS

The AP-120B's Word Count Register (see AP-120B Processor Handbook 4.2.2) is loaded from the TI980 central memory at location DATA ADDRESS.

Load Host Memory Address Register

LHMA (IC=5)
DATA ADDRESS

The AP-120B's Host Memory Address Register (see AP-120B Processor Handbook 4.2.1) is loaded from the TI980 central memory at location DATA ADDRESS. When using this instruction TI980 central memory location DATA ADDRESS should contain the absolute address in the TI980's central memory from which DMA transfers are to be made.

Load Control Register

LCTL (IC=6)
DATA ADDRESS

The AP-120B's Control Register (see AP-120B Processor Handbook 4.2.4) is loaded from the TI980 central memory at location DATA ADDRESS.

Load AP Direct Memory Address Register

LAPMA (IC=7)
DATA ADDRESS

The AP-120B's Direct Memory Address Register (see AP-120B Processor Handbook 4.2.3) is loaded from the TI980 central memory at location DATA ADDRESS.

Load Format Conversion Register High

LFMTH (IC=8)
DATA ADDRESS

The most significant 16-bit TI980 computer word of the AP-120B's Format Conversion Register (see AP-120B Processor Handbook 4.3) is loaded from the TI980 central memory at location DATA ADDRESS. This path is intended for diagnostic use only, it is the programmer's responsibility to see that APP Controller and DMAC Interface data transfers do not conflict.

Load Format Conversion Register Low

LFMTL (IC=9)
DATA ADDRESS

The least significant 16-bit TI980 computer word of the AP-120B's Format Conversion Register (see AP-120B Processor Handbook 4.3) is loaded from the TI980 central memory at location DATA ADDRESS. This path is intended for diagnostic use only, it is the programmer's responsibility to see that APP Controller and DMAC Interface data transfers do not conflict.

Load Host Memory Address Register Biased

LHMAD (IC=D)
DATA ADDRESS

The AP-120B's Host Memory Address Register (see AP-120B Processor Handbook 4.2.1) is loaded from the TI980 central memory at location DATA ADDRESS. When using this instruction TI980 central memory location DATA ADDRESS should contain the program relative address in the TI980's central memory from which DMA transfers are to be made.

Reset

RESET (IC=10)
DATA ADDRESS

The AP-120B's DMA Timing and Memory Timing are Reset. The AP-120B's S-Pad Register 0 and Status Register are cleared. The RUN BIT in the AP-120B's Function Register is Reset. Ones are loaded into the TI980 central memory location DATA ADDRESS. This instruction performs the function of an AP-120B machine reset.

Read Switch Register

RSWR (IC=11)
DATA ADDRESS

The contents of the AP-120B's Switch Register (see AP-120B Processor Handbook 4.1.1) is stored into the TI980 central memory at location DATA ADDRESS.

Read Function Register

RFN (IC=12)
DATA ADDRESS

The contents of the AP-120B's Function Register (see AP-120B Processor Handbook 4.1.3) is stored into the TI980 central memory at location DATA ADDRESS.

Read Panel Display Register (LITES)

RPDR (IC=13)
DATA ADDRESS

The contents of the AP-120B's Panel Display Register (see AP-120B Processor Handbook 4.1.2) is stored into the TI980 central memory at location DATA ADDRESS. There is no instruction for loading the Panel Display Register from the TI980's central memory.

Read Word Count Register

RWC (IC=14)
DATA ADDRESS

The contents of the AP-120B's Word Count Register (see AP-120B Processor Handbook 4.2.2) is stored into the TI980 central memory at location DATA ADDRESS.

Read Host Memory Address Register

RHMA (IC=15)
DATA ADDRESS

The contents of the AP-120B's Host Memory Address Register (see AP-120B Processor Handbook 4.2.1) is stored into the TI980 central memory at location DATA ADDRESS.

Read Control Register

RCTL (IC=16)
DATA ADDRESS

The contents of the AP-120B's Control Register (see AP-120B Processor Handbook 4.2.1) is stored into the TI980 central memory at location DATA ADDRESS.

Read AP Direct Memory Address Register

RAPMA (IC=17)
DATA ADDRESS

The contents of the AP-120B's Direct Memory Address Register (see AP-120B Processor Handbook 4.2.3) is loaded from the TI980 central memory at location DATA ADDRESS.

Read Format Conversion Register High

RFMTH (IC=18)
DATA ADDRESS

The contents of the most significant 16-bit TI980 computer word of the AP-120B's Format Conversion Register (see AP-120B Processor Handbook 4.3) is stored into the TI980 central memory at location DATA ADDRESS. This path is intended for diagnostic use only, it is the programmer's responsibility to see that APP Controller and DMAC Interface data transfers do not conflict.

Read Format Conversion Register Low

RFMTL
DATA ADDRESS

The contents of the least significant 16-bit TI980 computer word of the AP-120B's Format Conversion Register (see AP-120B Processor Handbook 4.3) is stored into the TI980 central memory at location DATA ADDRESS. This path is intended for diagnostic use only, it is the programmer's responsibility to see the APP Controller and DMAC Interface data transfers do not conflict.

5.5.3 Summary of APP Instruction

Instruction Mnemonic	Instruction Code (IC ₁₆)	Instruction Function	Notes
LSWR	1	(DA) → SWR	
LFN	2	(DA) → FN	
LWC	4	(DA) → WC	
LHMA	5	(DA) → HMA	Absolute Memory Address
LCTL	6	(DA) → CTL	
LAPMA	7	(DA) → APMA	
LFMTH	8	(DA) → FMTH	Diagnostic Path
LFMTL	9	(DA) → FMTL	Diagnostic Path
LHMAB	D	(DA) → HMA	Program Relative Address
RESET	10	FFFF → DA	AP-120B Machine Reset
RSWR	11	(SWR) → DA	
RFN	12	(FN) → DA	
RPDR	13	(PDR) → DA	
RWD	14	(WC) → DA	
RHMA	15	(HMA) → DA	
RCTL	16	(CTL) → DA	
RAPMA	17	(APMA) → DA	
RFMTH	18	(FMTH) → DA	Diagnostic Path
RFMTL	19	(FMTL) → DA	Diagnostic Path

NOTES:

- (1) DA A location in TI980's central memory whose address is the 2nd word of an API Instruction.
- (2) () The contents of
- (3) - Stored into

5.5.4 DMA Interface. DMA transfers to and from the AP-120B are controlled by the contents of the four DMA registers in the AP-120B interface (see the Array Processor Handbook, Section 4.2). The initialization procedure for a DMA transfer calls for loading these registers via Auxiliary Processor Initiate Instructions.

DMA operation is a slave to the state of the Control Register (CTL). Mode and direction of the transfer is determined by Control bits 8-14. Bit 8 selects the consecutive cycle mode. When set, DMA transfers occur on consecutive host memory cycles, locking out the host CPU for the length of the transfer. When clear, the access request is dropped for one cycle after each word is transferred to allow the CPU access to memory. DMA transfers occur until the Word Count (WC) Register reaches zero. The WC Register is loaded with the number of 16-bit words to be passed. Setting bit-15 of the CTL Register (HDMA start) starts the transfer. Because of this, the CTL register is always loaded last.

Jumpers for access request (ARDEV,0-7), access granted (AG,0-7), interrupt request (INTDEV,0-7), and interrupt acknowledge (IRECDG, 0-7), are located on the AP-120B DMA buffer cards. Selection of the desired priority is made by placing jumper plugs in the appropriate positions.

AP-120B generated interrupts are controlled by CTL register bits 3-5. When the interrupt condition arises an interrupt request is generated by the interface. A status word is stored via a normal DMA transfer after the interface receives interrupt recognition from the host. The Status Word Format is shown below. The conditions are true when the respective bits are set to one.

- bit 0 (MSB) - Word Count equals 0. Indicates DMA transfer completed.
- bit 14 - DMA transfer error
- bit 15 (LSB)- AP-120B is halted. Micro-code execution is terminated
- bits 1-13 - always zero

The address where the status word will be stored in the host memory is selectable by strap options on the AP-120B interface card.

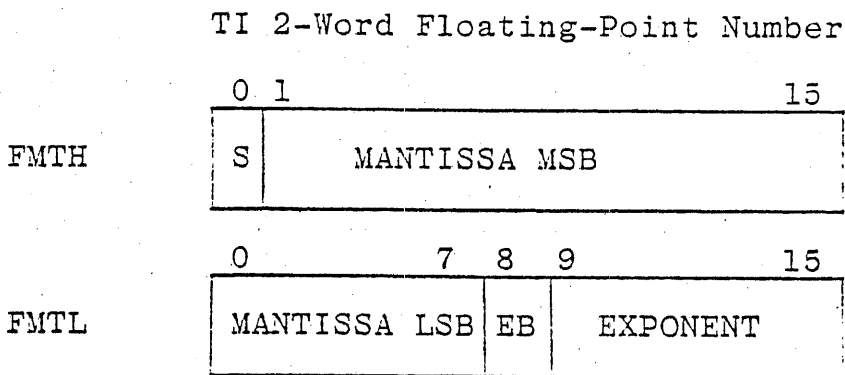
5.5.5 Data Transceiver. The Data Transceiver is capable of accepting data in four different formats. Format selection as accomplished by bit 13 and 14 of the AP-120B's Control Register (See AP-120B Processor Handbook 4.2.4). When loading the Format Conversion Register of the Data Transceiver from the DMAC or APP data transfers must be consistent with the state of bits 13 and 14 of the AP-120B's Control Register or NONSENSE will be output from the Data Transceiver.

In 32-Bit Integer Format (CRT Bits 13 and 15=0) the high parts of the Format Conversion Register (APP Controller Instruction is IC=8/18) refers to the TI980 data word of high significance. The low part of the Format Conversion Register (APP Controller Instruction IC=9/19) refers to the TI980 data word of low significance.

In 16-Bit Integer Format (CRT Bits 13 and 15=1) TI980 data words are loaded into or read from the low part of the Forward Conversion Register (APP Controller Instruction is IC=9/19).

In TI 2-Word Format (CTR Bits 13 and 15=2) the high part of the Format Conversion Register (APP Controller Instruction is IC=8/18) refers to the sign, and the high mantissa bits of the floating-point number. The low part of the Format Conversion Register (APP Controller Instruction is IC=9/19) refers to the low mantissa bits, the exponent bias bit, and the exponent of the floating-point number.

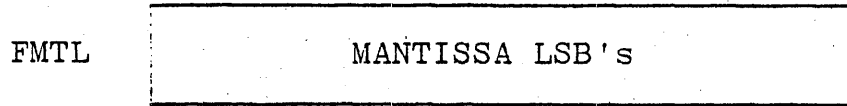
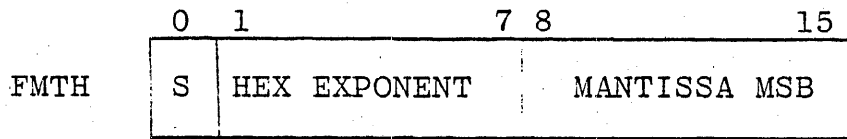
Example:



In IBM 32-Bit Format (CTR Bits 13 and 15=3) the high part of the Format Conversion Register (APP Controller Instruction is IC=8/18) refers to the sign, exponent in biased hexadecimal, and high mantissa bits of the floating-point number. The low part of the Format Conversion Register (APP Controller Instruction is IC=9/19) refers to the low mantissa bits of the floating-point number.

Example:

IBM 32-Bit Floating-Point Number



5.6 VARIAN 620 INTERFACE

The AP-120B to Varian 620 interface acts as a controller for data transfers between the AP-120B interface registers (Section 4 of the AP-120B Processor Handbook) and the Varian 620 "E" Bus. The standard interface is capable of transferring data under program control and also under control of the Buffer Interface Controller (BIC).

5.6.1 Input/Output Commands. Program Control access to the AP-120B required 9 device addresses. These addresses can be altered via hardware straps in the interface. They must however, all be included within two octades (not necessarily contiguous) of device addresses. The standard device addresses are listed in the table below.

Device Address	Register Accessed
X7	Switch Register (SR)
X6	Functional Register (FN)
X5	Panel Display Register (LITES) Read only
X4	Word Count
X3	Host Memory Address (HMA)
X2	Control (CTL)
X1	AP-120B Memory Address (APMA)
X0	Format Register Low (FMTL)
Y7	Format Register High (FMTH)

Where X and Y indicate the first octal digit of the Device Address.

Note that the HMA Register is not functional when using the BIC. The BIC Initial Register provides the V620 memory address for BIC transfers.

5.6.2 SEC Commands. Two SENSE instructions are provided to allow the V620 program to test the state of the AP-120B.

Function Field	Result
0	Branch if AP-120B running
1	Branch if AP-120B to BIC interface is active

These sense instructions use the same device address as does the AP-120B Switch Register (SR).

Thus, the following instruction:

LOOP, SEN, 0100+SR, LOOP

will wait for the AP-120B to complete a BIC transfer.

5.6.3 EXC Commands. Five external control commands are provided for initialization and control of the interrupts and the BIC. The Switch Register (SR) device address is used for these EXC instructions.

Function Field	Control Function
0	Reset AP-120B. Clears interface and memory timing. Stops AP-120B processor.
1	Enable the AP-120B BIC interface in the Word Count Stop Mode. In this mode the interface will stop the BIC transfer when the AP-120B WC Register reaches zero.
2	Enable the AP-120B BIC interface. Following this command the BIC transfer will terminate in the normal fashion, i.e., when the initial and final registers are equal.
3	Unconditional stop of the BIC transfer.
4	Clear the AP-120B interrupt. Used to clear the interrupting condition after the software has responded to an AP-120B interrupt.

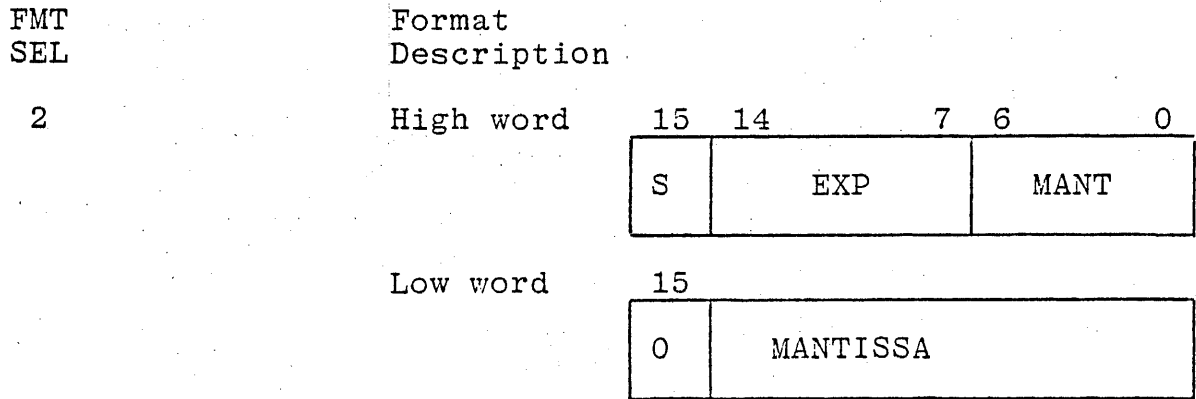
5.6.4 BIC Operation. To use the BIC with the AP-120B, the program must first initialize the BIC in the usual way by loading its initial and final registers and by issuing the activate BIC EXC instruction. The final register can be set to 0 if the transfer is to take place in the Word Count Stop Mode. The program then selects the interface mode with an

EXC ,100 + SR ,Word Count Stop

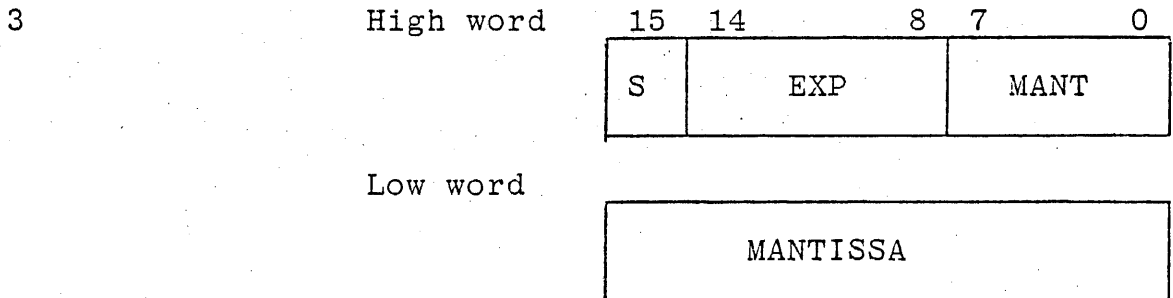
or an EXC ,200 + SR ,Normal BIC Stop

Finally, the program starts the AP-120B interface transferring by loading the CTL register with the appropriate command for the type of transfer desired. Note that the consecutive cycle bit in the Control Register has no effect on the transfer. All transfers proceed at BIC speed.

5.6.5 Floating Point Formats. The AP-120B to V620 interface can transfer floating-point arguments to and from the V620 in two floating-point formats. The format desired is specified by the format select field of the CTL Register.



This is the V620 format, the exponent is biased, and the first word is one's complemented if the mantissa is negative.



This is the IBM-360 short form format. The exponent is biased and is a power of 16. The mantissa is in sign-magnitude form.

5.6.6 Interrupts. A jumper is provided in the interface to allow strapping any interrupt level from 0 to 7.

5.6.7 Physical. The interface consists of a signal DM135 type card which mounts in the V620 mainframe or expansion chassis, and which is connected to the AP-120B via a cable of up to 10 feet in length.

APPENDICES

APPENDIX A: AP-120B REGISTERS/DATA PATH NAMES

<u>Mnemonic</u>	<u>Width</u>	<u>Name</u>
SP	16 bits	Scratch Pad Registers (16)
SPD	4	S-Pad Destination Address Register
SPFN	16	Scratch Pad ALU/shifter function output
PNBLS	16	Panel Bus
SWR	16	Panel Switch Register
LITES	16	Panel Display Register
APSTATUS	16	AP Status Register
PS	64	Program Source Memory
CB	64	Command Buffer
PSA	16	Program Source Address Register
SRS	16	Subroutine Return Stack
SRA	16	Subroutine Return Stack Pointer
DPX	38	Data Pad X Registers (32)
DPY	38	Data Pad Y Registers (32)
DPBS	38	Data Pad Bus
DPA	16	Data Pad Address Register
TM	38	Table Memory Output Register
TMA	16	Table Memory Address Register
MD	38	Data Memory Output Register
MI	38	Data Memory Input Register
MA	16	Memory Address Register
A1	38	Floating Adder Input Register #1
A2	38	Floating Adder Input Register #2
FA	38	Floating Adder Output Register
M1	38	Floating Multiplier Input Register #1
M2	38	Floating Multiplier Input Register #2
FM	38	Floating Multiplier Output Register
IODEVICE		I/O Device
DA	16	I/O Device Address
INBS	38	I/O Input Bus
IODRDY	1	I/O Data Ready Flag
A	1	I/O Device Condition "A" Flag
B	1	I/O Device Condition "B" Flag

Subscripts indicate addressing within memory element, i.e. PS_{PSA} means the location in Program Source Memory pointed to by the Program Source Address Register.

Superscripts indicate portions of word, i.e. A2^E means the exponent portion of the A2 Register.

Parenthesis around a symbol indicates "the contents of" a register, i.e. (A1) means the contents of the A1 Register.

AP-120B INTERNAL STATUS REGISTER

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OVF	UNF	DIVZ	FZ	FN	Z	N	C	PERR	PENB	SRAO	IFFT	FFT	Bit Reverse		

<u>Bits</u>	<u>Mnemonic</u>	<u>Meaning</u>
0	OVF	Set when the current adder or multiplier (FA or FM) has overflowed. Overflow occurs when an exponent value is increased above 511. The offending result is set to the signed maximum of value of $(1-2^{-27}) * 2^{511}$, which is roughly $6.7 * 10^{153}$. This bit remains on until cleared by the microprogram or host computer.
1	UNF	Set when the current adder or multiplier result (FA or FM) has underflowed. Underflow occurs when an exponent value is decreased below -512. The minimum legal magnitude which numbers can take without underflowing is roughly $3.7 * 10^{-155}$. The offending value is set to zero. This bit remains on until cleared by the microprogram or host computer.
2	DIVZ	A divide by zero has occurred. The result was set to the value of the dividend. This bit remains on until cleared by the microprogram or host computer.
3	FZ	Set when the current adder result (FA) is zero.
4	FN	Set when the current adder result (FA) is negative.
5	Z	Set when the current S-pad function (SPFN) is zero.
6	N	Set when the current S-pad function (SPFN) is negative.
7	C	S-Pad carry bit. If no S-Pad shift is specified, carry is the carry bit from the S-Pad ALU. If a shift is specified, carry is the last bit shifted off the end of the S-Pad result by the shift.

- 8 PERR (Optional). Set when a Main Data Memory parity error has occurred. Three parity bits are used, one each to check the exponent, high mantissa, and low mantissa portions of the memory word. If "PENB" is set, the processor will halt on this error.
- 9 PENB (Optional). Enables halt on memory parity error. If set, the processor will halt when a memory parity error is detected.
- 10 SRAO Subroutine return stack overflow. Set if more than 16 levels of nested subroutine calls have occurred.
- 11 IFFT Inverse FFT flag. When set in conjunction with the FFT flag, bit 12, causes roots of unity table references to be interpreted as positive angles.
- 12 FFT FFT Flag. When set causes Table Memory addresses to be interpreted as negative angles referencing the roots of unity table contained in Table Memory.
- 13-15 Bit Reverse 15- $\log_2 N$ Where N is the length of a complex data array to which the S-Pad address bit-reverse operator is being applied.

AP-120B FUNCTIONAL UNITS.

BUS INPUTS:

DPBS - Data Pad Bus (38)

DPX PS VALUE
DPY SPFN ZERO

INBS SWR
MD TM

INBS - Input Bus (38) Formatter

PNLBS - Panel Bus (16)

DPA PS PSA
MA TMA

FUNCTIONAL UNIT OUTPUTS:

DPX - Data Pad X Output

DPY - Data Pad Y Output

MD - Data Memory Output

TM - Table Memory Output

FA - F.P. Adder Output

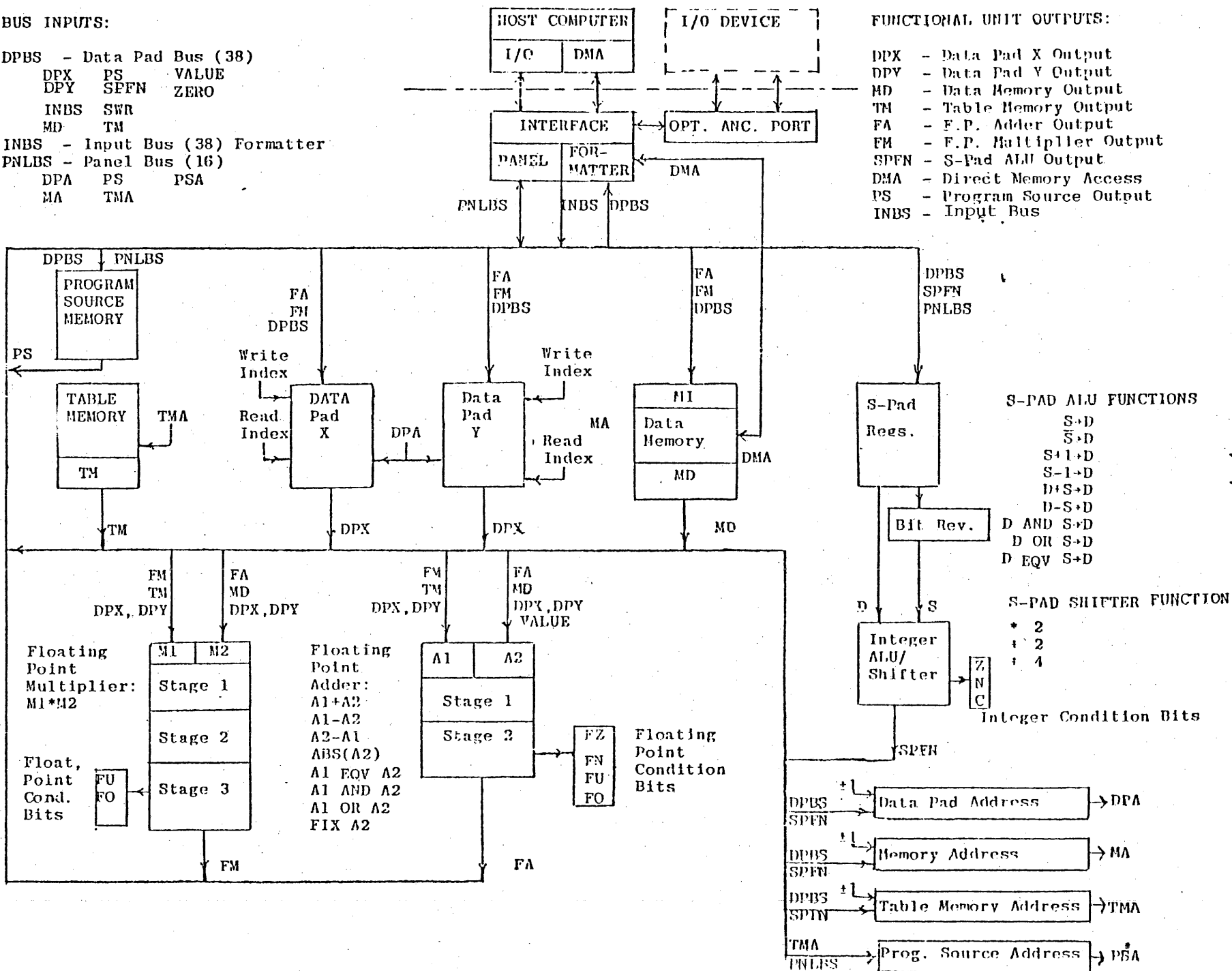
FM - F.P. Multiplier Output

SPFN - S-Pad ALU Output

DMA - Direct Memory Access

PS - Program Source Output

INBS - Input Bus



S-PAD ALU FUNCTIONS

- S+D
- S-D
- S+1+D
- S-1+D
- D+S+D
- D-S+D
- D AND S+D
- D OR S+D
- D EQV S+D

S-PAD SHIFTER FUNCTION

- * 2
 - + 2
 - + 4
- Integer Condition Bits

AP-120B Instruction Summary

Unconditional Fields

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

Octal Code	Field Name										Octal Code
	B	SOP	SOP1	SH	SPS	SPD	FADD	FADD1	A1	A2	
0	NOP	SOP1	NOP	NOP	(S-Pad	(S-Pad	FADD1	NOP	NC	NC	0
1	&	SPEC	WRTEXP	L	Source	Dest.	FSUBR	FIX	FM	FA	1
2		ADD	WRTHMN	RR	Reg.)	Reg.)	FSUB	FIXT	DPX	DPX	2
3		SUB	WRTHMN	R			FADD	FSCLT	DPY	DPY	3
4		MOV	NOP		(0-17)	(0-17)	FEQV	FSM2C	TM	MD	4
5		AND	NOP				FAND	F2CSM	ZERO	ZERO	5
6		OR	NOP				FOR	FSCALE	ZERO	MDPX	6
7		EQV	NOP				IO	FABS	ZERO	EDPX	7
10			CLR								10
11			INC								11
12			DEC								12
13			COM								13
14			LDSPNL								14
15			LDSPE								15
16			LDSPI								16
17			LDSPT								17

Octal Code	Field Name										Octal Code
	COND	DISP	DPX	DPY	DPBS	XR	YR	XW	YW	FM	
0	NOP	(Branch	NOP	NOP	ZERO	(DPX	(DPY	(DPX	(DPY	NOP	0
1	#	Displa-	DB	DB	INBS	Read	Read	Write	Write	FMUL	1
2	BR	cement)	FA	FA	VALUE*	Index)	Index)	Index)	Index)		2
3	BIWTRQ	(0-37)	FM	FM	DPX						3
4	BION				DPY	(0-7)	(0-7)	(0-7)	(0-7)		4
5	BIOZ				MD						5
6	BFPE				SPFN						6
7	RETURN				TM						7
10	BFEQ										10
11	BFNE										11
12	BFGE										12
13	BFGT										13
14	BEQ										14
15	BNE										15
16	BGE										16
17	BGT										16

Octal Code	Field Name						Octal Code
	M1	M2	MI	MA	DPA	TMA	
0	FM	FA	NOP	NOP	NOP	NOP	0
1	DPX	DPX	FA	INCMA	INCDPA	INCTMA	1
2	DPY	DPY	FM	DECMA	DECDDPA	DECTMA	2
3	TM	MD	DB	SETMA	SETDPA	SETTMA	3

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

SPEC Fields

One of the SPEC Fields may be used per instruction word. The S-PAD Fields (D, SOP, SOP1, SH, SPS, and SPD) are then disabled for this instruction.

Octal Code	Field Name								Octal Code
	SPEC	STEST	HOSTPNL	SETPSA	PSEVEN	PSODD	PS	SETEXIT	
0	STEST	BFLT	PNLLIT	JMPA*	RPSOA*	RPS1A*	RPSLA*	NOP	0
1	HOSTPNL	BLT	DBELIT	JSRA*	RPS2A*	RPS3A*	RPSFA*	SETEXA*	1
2	SPMDA	BNC	DBHLIT	JMP*	RPS0*	RPS1*	RPSL*	NOP	2
3	NOP	BZC	DBLLIT	JSR*	RPS2*	RPS3*	RPSF*	SETEX*	3
4	NOP	BDBN	NOP	JMPT	RPS0T	RPS1T	RPSLT	NOP	4
5	NOP	BDBZ	NOP	JSRT	RPS2T	RPS3T	RPSFT	SETEXT	5
6	NOP	BIFN	NOP	JMPP	NOP	NOP	RPSLP	NOP	6
7	NOP	BIFZ	NOP	JSRP	NOP	NOP	RPSFP	SETEXP	7
10	SETPSA	NOP	SWDB	NOP	WPS0A*	WPS1A*	LPSLA*	NOP	10
11	PSEVEN	NOP	SWDBE	NOP	WPS2A*	WPS3A*	LPSRA*	NOP	11
12	PSODD	NOP	SWDBH	NOP	WPS0*	WPS1*	LPSL*	NOP	12
13	PS	NOP	SWDBL	NOP	WPS2*	WPS3*	LPSR*	NOP	13
14	SETEXIT	BFL0	NOP	NOP	WPS0T	WPS1T	LPSLT	NOP	14
15	NOP	BFL1	NOP	NOP	WPS2T	WPS3T	LPSRT	NOP	15
16	NOP	BFL2	NOP	NOP	NOP	NOP	LPSLP	NOP	16
17	NOP	BFL3	NOP	NOP	NOP	NOP	LPSRP	NOP	17

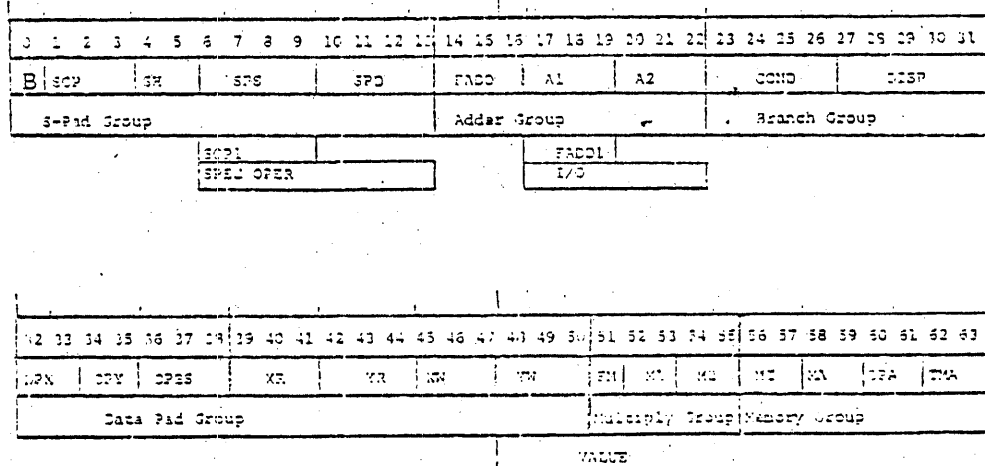
I/O Fields

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

Octal Code	Field Names							Octal Code
	IO	LDREG	RDREG	INOUT	SENSE	FLAG	CONTROL	
0	LDREG	NOP	RPSA	OUT	SNSA	SFL0	HALT	0
1	RDREG	LDSPD	RSPD	SPNOUT	SPININ	SFL1	IORST	1
2	SPMDAV	LDMA	RTMA	OUTDA	SNSADA	SFL2	INTEN	2
3	NOP	LDTMA	RTMA	SPOTDA	SPNADA	SFL3	INTA	3
4	INOUT	LDDPA	RDPA	IN	SNSB	CFL0	REFR	4
5	SENSE	LDSP	RSPFN	SPININ	SPINB	CFL1	WRTEX	5
6	FLAG	LDAPS	RAPS	OUTDA	SNSBDA	CFL2	WRTMAN	6
7	CONTROL	LDDA	RDA	SPINDA	SPNBDA	CFL3	NOP	7

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

AP-1208 Instruction Field Layout



AP-120B Instruction Field Layout

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
B	SOP		SH	SPS		SPD		FADD		A1	A2		COND			DISP															
S-Pad Group										Adder Group										Branch Group											

SOP1	
SPEC OPER	

FADD1	
I/O	

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
DPX	DPY	DPBS		XR		YR	XW	YW	FM	M1	M2	MI	MA	DPA	TMA																
Data Pad Group										Multiply Group					Memory Group																
VALUE																															

INTENTIONALLY BLANK

S-PAD GROUP

0	1	3	4	5	6	9	10	13
B	SOP		SH		SPS		SPD	
					SOP1			

Field	Octal Code	Mnemonic	Effect
B	0	-	No-Op
	1	&	Use SP _{SPS} (bit-reversed)
SOP	0	-	See SOP1 field
	1	-	See Special Operations Group
	2	ADD	(SP _{SPD}) + (SP _{SPS}) → SPFN
	3	SUB	(SP _{SPD}) - (SP _{SPS}) → SPFN
	4	MOV	(SP _{SPS}) → SPFN
	5	AND	(SP _{SPD}) AND (SP _{SPS}) → SPFN
	6	OR	(SP _{SPD}) OR (SP _{SPS}) → SPFN
	7	EQV	(SP _{SPD}) XOR (SP _{SPS}) → SPFN
*SH	0	-	No-Op
	1	L	SPFN*2 → SPFN (left shift)
	2	RR	SPFN ÷ 4 → SPFN (double right shift)
	3	R	SPFN ÷ 2 → SPFN (right shift)
SPS	0-17 ₈	0-17 ₈	S-Pad Source Operand Address
SPD	0-17 ₈	0-17 ₈	S-Pad Destination Address, SPFN → ^{SP} SPD unless inhibited by No Load (COND=1)

*Note: These are logical shifts:

Right shift 0 → 0-15 → C
 Left shift C ← 0-15 ← 0

Field	Octal Code	Mnemonic	Effect
SOP1	0	-	No-Op
	1	WRTEXP	Restricts DPX, DPY & MI fields to Write Exponent Only
	2	WRTHMN	Restricts DPX, DPY & MI fields to Write High Mantissa Only (Bits 00-11)
	3	WRTL MN	Restricts DPX, DPY & MI fields to Write Low Mantissa Only (Bits 12-27)
	4	-	-
	5	-	-
	6	-	-
	7	-	-
	10	CLR	$\emptyset \rightarrow \text{SPFN}$
	11	INC	$(\text{SP}_{\text{SPD}}) + 1 \rightarrow \text{SPFN}$
	12	DEC	$(\text{SP}_{\text{SPD}}) - 1 \rightarrow \text{SPFN}$
	13	COM	$\overline{(\text{SP}_{\text{SPD}})} \rightarrow \text{SPFN}$
	14	LDSPNL	$\text{SP}_{\text{SPD}} \rightarrow \text{SPFN}, \text{PNLBS} \rightarrow \text{SP}_{\text{SPD}}$
	15	LDSPE	$\text{SP}_{\text{SPD}} \rightarrow \text{SPFN}, \text{DPBS}^{\text{E}} - 512 \rightarrow \text{SP}_{\text{SPD}}$
	16	LDSPI	$-\text{SP}_{\text{SPD}} \rightarrow \text{SPFN}, \text{DPBS}^{\text{ML}} \rightarrow \text{SP}_{\text{SPD}}$
	17	LDSPT	$-\text{SP}_{\text{SPD}} \rightarrow \text{SPFN}, \text{DPBS}^{\text{MT}} \rightarrow \text{SP}_{\text{SPD}}$

MH=Mantissa High=Mantissa bits 00-11
 ML=Mantissa Low=Mantissa bits 12-27
 MT=Mantissa bits for table lookups=Mantissa bits 02-08
 E=Exponent

SPECIAL OPERATIONS GROUP

1	3	6	9	10	13
0	0	1	SPEC	STEST	HOSTPNL
				SETPSA	
				PSEVEN	
				PSODD	
				PS	
				SETEXIT	

Field	Octal Code	Mnemonic	Effect
SPEC	0	-	See STEST Field (B-6)
	1	-	See HOSTPNL Field (B-7)
	2	SPMDA	Spin until MD available
	3	-	-
	4	-	-
	5	-	-
	6	-	-
	7	-	-
	10	-	See SETPSA Field, inhibit TEST except No Load (B-8)
	11	-	See PSEVEN Field (B-9)
	12	-	See PSODD Field (B-10)
	13	-	See PS Field (B-11)
	14	-	See SETEXIT Field (E-12)
	15	-	-
	16	-	-
	17	-	-

Field	Octal Code	Mnemonic	Effect
STEST	0	BFLT	Branch if FA < 0.0
	1	BLT	Branch if SPFN < 0
	2	BNC	Branch if S-Pad carry bit=1
	3	BZC	Branch if S-Pad carry bit=0
	4	BDBN	Branch if DPBS < 0.0
	5	BDBZ	Branch if DPBS positive and unnormalized
	6	BIFN	Branch if Inverse FFT flag=1
	7	BIFZ	Branch if Inverse FFT flag=0
	10	--	--
	11	--	--
	12	--	--
	13	--	--
	14	BFL0	Branch if Flag 0=1
	15	BFL1	Branch if Flag 1=1
	16	BFL2	Branch if Flag 2=1
	17	BFL3	Branch if Flag 3=1

If the above specified condition is true OR the condition specified in the COND field is true, a branch occurs to (PSA)+DISP-20

Field	Octal Code	Mnemonic	Effect
HOSTPNL	0	PNLLIT	PNLBS → LITES
	1	DBELIT	DPBS ^E → PNLBS → LITES
	2	DBHLIT	DPBS ^{MH} → PNLBS → LITES
	3	DBLLIT	DPBS ^{ML} → PNLBS → LITES
	4	-	-
	5	-	-
	6	-	-
	7	-	-
	10	SWDB	(SWR) → PNLBS → DPBS
	11	SWDBE	(SWR) → PNLBS → DPBS ^E and WRTEXP*
	12	SWDBH	(SWR) → PNLBS → DPBS ^{MH} and WRTHMAN*
	13	SWDBL	(SWR) → PNLBS → DPBS ^{ML} and WRTLMAN*
	14	-	-
	15	-	-
	16	-	-
	17	-	-

*Restrict DPS, DPY and MI to:

WRTEXP: Write Exponent only
 WRTHMAN: Write High Mantissa
 only (bits 00-11)
 WRTLMAN: Write Low Mantissa
 only (bits 12-27)

MH=Mantissa High=Mantissa bits 00-11
 ML=Mantissa Low =Mantissa bits 12-27
 E=Exponent

Field	Octal Code	Mnemonic	Effect
SETPSA	0	JMPA	VALUE→PSA
	1	JSRA	(SRA)+1→SRA, (PSA)+1→SRS _{SRA} , VALUE→PSA
	2	JMP	VALUE+(PSA)→PSA
	3	JSR	(SRA)+1→SRA, (PSA)+1→SRS _{SRA} , VALUE +(PSA)→PSA
	4	JMPT	(TMA)→PSA
	5	JSRT	(SRA)+1→SRA, (PSA)+1→SRS _{SRA} , (TMA)→PSA
	6	JMPP	(SWR)→PNLBS→PSA
	7	JSRP	(SRA)+1→SRA, (PSA)+1→SRS _{SRA} , (SWR)→PNLBS→PSA

VALUE=Bits 48-63 of this instruction (CB48-CB63)

Field	Octal Code	Mnemonic	Effect
PSEVEN	0	RPS0A	(PS ^{Q0} _{VALUE}) →PNLBS→ LITES
	1	RPS2A	(PS ^{Q2} _{VALUE}) →PNLBS→ LITES
	2	RPS0	(PS ^{Q0} _{VALUE+PSA}) →PNLBS→ LITES
	3	RPS2	(PS ^{Q2} _{VALUE+PSA}) →PNLBS→ LITES
	4	RPS0T	(PS ^{Q0} _{TMA}) →PNLBS→ LITES
	5	RPS2T	(PS ^{Q2} _{TMA}) →PNLBS→ LITES
	6	-	-
	7	-	-
	10	WPS0A	(SWR) →PNLBS→PS ^{Q0} _{VALUE}
	11	WPS2A	(SWR) →PNLBS→PS ^{Q2} _{VALUE}
	12	WPS0	(SWR) →PNLBS→PS ^{Q0} _{VALUE+PSA}
	13	WPS2	(SWR) →PNLBS→PS ^{Q2} _{VALUE+PSA}
	14	WPS0T	(SWR) →PNLBS→PS ^{Q0} _{TMA}
	15	WPS2T	(SWR) →PNLBS→PS ^{Q2} _{TMA}
	16	-	-
	17	-	-

This field requires 2 cycles to execute

VALUE = Bits 48-63 of this instruction (CB48-CB63)

Q0 = Quarter zero of Program Source Word (PS00-PS15)

Q2 = Quarter two of Program Source Word (PS31-PS47)

Field	Octal Code	Mnemonic	Effect
PSODD	0	RPS1A	(PS ^{Q1} _{VALUE}) → PNLBS → LITES
	1	RPS3A	(PS ^{Q3} _{VALUE}) → PNLBS → LITES
	2	RPS1	(PS ^{Q1} _{VALUE+PSA}) → PNLBS → LITES
	3	RPS3	(PS ^{Q3} _{VALUE+PSA}) → PNLBS → LITES
	4	RPS1T	(PS ^{Q1} _{TMA}) → PNLBS → LITES
	5	RPS3T	(PS ^{Q3} _{TMA}) → PNLBS → LITES
	6	-	-
	7	-	-
	10	WPS1A	(SWR) → PNLBS → PS ^{Q1} _{VALUE}
	11	WPS3A	(SWR) → PNLBS → PS ^{Q3} _{VALUE}
	12	WPS1	(SWR) → PNLBS → PS ^{Q1} _{VALUE+PSA}
	13	WPS3	(SWR) → PNLBS → PS ^{Q3} _{VALUE+PSA}
	14	WPS1T	(SWR) → PNLBS → PS ^{Q1} _{TMA}
	15	WPS3T	(SWR) → PNLBS → PS ^{Q3} _{TMA}
	16	-	-
	17	-	-

This field requires 2 cycles to execute.

VALUE=Bits 48-63 of this instruction (CB48-CB63)
Q1=Quarter one of Program Source Word (PS16-PS31)
Q3=Quarter three of Program Source Word (PS48-PS63)

Field	Octal Code	Mnemonic	Effect
PS	0	RPSLA	(PS ^{LH} VALUE) → DPBS
	1	RPSFA	(PS ^{FP} VALUE) → DPBS
	2	RPSL	(PS ^{LH} VALUE+PSA) → DPBS
	3	RPSF	(PS ^{FP} VALUE+PSA) → DPBS
	4	RPSLT	(PS ^{LH} TMA) → DPBS
	5	RPSFT	(PS ^{FP} TMA) → DPBS
	6	RPSLP	(PS ^{LH} PNLBS) → DPBS
	7	RPSFP	(PS ^{FP} PNLBS) → DPBS
	10	LPSLA	DPBS → PS ^{LH} VALUE
	11	LPSRA	DPBS → PS ^{RH} VALUE
	12	LPSL	DPBS → PS ^{LH} VALUE+PSA
	13	LPSR	DPBS → PS ^{RH} VALUE+BA
	14	LPSLT	DPBS → PS ^{LH} TMA
	15	LPSRT	DPBS → PS ^{RH} TMA
	16	LPSLP	DPBS → PS ^{LH} PNLBS
	17	LPSRP	DPBS → PS ^{RH} PNLBS

This field requires 2 cycles to execute.

VALUE=Bits 48-63 of this instruction (CB48-CB63)

LH=Left half of Program Source Word (Bits 00-31)

RH=Right half of Program Source Word (Bits 32-63)

FP=Program Source bits 26-63, used for floating-point literals

Field	Octal Code	Mnemonic	Effect
SETEXIT	0	-	-
	1	SETEXA	VALUE→SRS _{SRA}
	2	-	-
	3	SETEX	VALUE+(PSA)→SRS _{SRA}
	4	-	-
	5	SETEXT	TMA→SRS _{SRA}
	6	-	-
	7	SETEXP	PSA+1 →SRS _{SRA}

Sets the current subroutine return address as indicated above.

SRA does not change.

VALUE=Bits 48-63 of this instruction.

FLOATING ADDER GROUP

14	16	17	19	20	22
FADD		A1		A2	
FADD1					

Field	Octal Code	Mnemonic	Effect
FADD	0	-	See FADD1 field
	1	FSUBR	Subtract: (A2) - (A1)
	2	FSUB	Subtract: (A1) - (A2)
	3	FADD	Add: (A1) + (A2)
	4	FEQV	Logical Equivalence: (A1) $\overline{\text{XOR}}$ (A2)
	5	FAND	Logical and: (A1) AND (A2)
	6	FOR	Logical or: (A1) OR (A2)
	7	-	See I/O Group
<hr/>			
A1	0	NC	(A1) → A1
	1	FM	FM → A1
	2	DPX(1DX)	(DPX _{DPA+1DX}) → A1 Where XR=1DX+4
	3	DPY(1DX)	(DPY _{DPA+1DX}) → A1 Where YR=1DX+4
	4	TM	(TM) → A1
	5	ZERO	0.0 → A1
	6	-	-
	7	-	-

Note: All floating adder op-codes:

1. Align exponents
2. Perform the specified arithmetic, logical, or shift operation
3. Normalize
4. Convergently round

Field	Octal Code	Mnemonic	Effect
A2	0	NC	(A2)→A2
	1	FA	FA→A2
	2	DPX (1DX)	(DPX _{DPA+1DX})→A2, Where XR=1DX+4
	3	DPY (1DX)	(DPY _{DPA+1DX})→A2, Where YR=1DX+4
	4	MD	(MD)→A2
	5	ZERO	0.0→A2
	6	MDPX (1DX)	SPFN+512→A2 ^E , (DPX ^M _{DPA+1DX})→A2 ^M
	7	EDPX (1DX)	(DPX ^E _{DPA+1DX})→A2 ^E , SPFN→A2 ^M (00-01), 0→A2 ^M (02-27)

FADD1	0	-	No-Op
	1	FIX	Convert (A2) to an integer
	2	FIXT	Convert (A2) to an integer (result truncated)
	3	FSCLT	Shift (A2) right and increment A2 ^E until A2 ^E =(SPFN+511) (result truncated).
	4	FSM2C	Convert (A2), from signed Magnitude to 2's complement.
	5	F2CSM	Convert (A2) from 2's complement to signed magnitude.
	6	FSCALE	Shift (A2) right and increment A2 ^E until A2 ^E =SPFN+511.
	7	FABS	Take the absolute value of (A2).

I/O GROUP

14	16	17	19	20	22
1	1	1	I/O	LDREG	
				RDREG	
				INOUT	
				SENSE	
				FLAG	
				CONTROL	

Field	Octal Code	Mnemonic	Effect
I/O	0	-	See LDREG field
	1	-	See RDREG field
	2	SPMDAV	Spin until MD available
	3	-	-
	4	-	See INOUT field
	5	-	See SENSE field
	6	-	See FLAG field
	7	-	See CONTROL field
LDREG	0	-	No-Op
	1	LDSPD	DPBS→SPD
	2	LDMA	DPBS→MA
	3	LDTMA	DPBS→TMA
	4	LDDPA	DPBS→DPA
	5	EDSP	SP _{SPD} →SPFN, DPBS→SP _{SPD}
	6	LDAPS	DPBS→APSTATUS ₄
	7	LDDA	DPBS→DA

Field	Octal Code	Mnemonic	Effect
RDREG	0	RPSA	(PSA)→PNLBS
	1	RSPD	(SPD)→PNLBS
	2	RMA	(MA)→PNLBS
	3	RTMA	(TMA)→PNLBS
	4	RDPA	(DPA)→PNLBS
	5	RSPFN	SPFN→PNLBS
	6	RAPS	(APSTATUS)→PNLBS
	7	RDA	(DA)→PNLBS
INOUT	0	OUT	DPBS→IODEVICE _{DA}
	1	SPNOUT	SPIN if IODRDY _{DA} =0 DPBS→IODEVICE _{DA}
	2	OUTDA	DPBS→IODEVICE _{DA} , SPFN→DA
	3	SPOTDA	SPIN if IODRDY _{DA} =0, SPFN→DA DPBS→IODEVICE _{DA}
	4	IN	(IODEVICE _{DA})→INBS
	5	SPININ	SPIN if IODRDY _{DA} =0 (IODEVICE _{DA})→INBS
	6	INDA	(IODEVICE _{DA})→INBS, SPFN→DA
	7	SPINDA	SPIN if IODRDY _{DA} =0, SPFN→DA (IODEVICE _{DA})→INBS

Field	Octal Code	Mnemonic	Effect
SENSE	0	SNSA	A _{DA} →IODRDY Flag
	1	SPINA	A _{DA} →IODRDY, SPIN if IODRDY=0
	2	SNSADA	A _{DA} →IODRDY, SPFN→DA
	3	SPNADA	A _{DA} →IODRDY, SPIN if IODRDY=0, SPFN→DA
	4	SNSB	B _{DA} →IODRDY Flag
	5	SPINB	B _{DA} →IODRDY, SPIN if IODRDY=0
	6	SNSBDA	B _{DA} →IODRDY, SPFN→DA
	7	SPNBDA	B _{DA} →IODRDY, SPIN if IODRDY=0, SPFN→DA

A and B are I/O device dependent conditions, either 1 or 0

FLAG	0	SFL0	1→FLAG ₀
	1	SFL1	1→FLAG ₁
	2	SFL2	1→FLAG ₂
	3	SFL3	1→FLAG ₃
	4	CFL0	0→FLAG ₀
	5	CFL1	0→FLAG ₁
	6	CFL2	0→FLAG ₂
	7	CFL3	0→FLAG ₃

Field	Octal Code	Mnemonic	Effect
CONTROL	0	HALT	Halt
	1	IORST	I/O reset
	2	---	---
	3	INTA	Interrupt acknowledge. Device Address of interrupting device put onto DPBS.
	4	REFR	Memory refresh sync
	5	WRTEX	Restricts DPX, DPY & MI to Write exponent only
	6	WRTMAN	Restricts DPX, DPY & MI to Write Mantissa Only (Bits 0-27)

BRANCH GROUP

23	26	27	31
COND	DISP		

Field	Octal Code	Mnemonic	Effect
COND	0	-	No-Op
	1	#	Inhibit load of SPFN→SP _{SPD}
	2	BR	Branch always
	3	BINTRQ	Branch if INTRQ (Interrupt Request) flag=1
	4	BION	Branch if IODRDY _{DA} flag=1
	5	BIOZ	Branch if IODRDY _{DA} flag=0
	6	BFPE	Branch on floating-point arithmetic error (overflow, underflow, or divide by zero)
	7	RETURN	(SRS _{SRA})→PSA, (SRA)-1→SRA (Sub-routine return jump).

NOTE: "RETURNS" may not be made in two successive instructions.

10	BFEQ	Branch if FA=0.0
11	BFNE	Branch if FA≠0.0
12	BFGGE	Branch if FA>=0.0
13	BFGT	Branch if FA>0.0
14	BEQ	Branch if SPFN=0
15	BNE	Branch if SPFN≠0
16	BGE	Branch if SPFN>=0
17	BGT	Branch if SPFN>0

Note: FA and SPFN are tested as to their state for the previous instruction.

DISP 0 to 37 If branch condition is true, (PSA) +DISP-20→PSA

Thus the effective Branch Range is -20 to +17 relative to the current instruction.

DATA PAL GROUP

32	33	34	35	36	38	39	41	42	44	45	47	48	50
DPX	DPY	DPBS		XR			YR		XW		YW		

Field	Octal Code	Mnemonic	Effect
DPX	0	-	No-Op
	1	DPX(1DX) < DB	DPBS → *DPX _{DPA+1DX} , Where XW=1DX+4
	2	DPX(1DX) < FA	FA → *DPX _{DPA+1DX} , Where XW=1DX+4
	3	DPX(1DX) < FM	FM → *DPX _{DPX+1DX} , Where XW=1DX+4
DPY	0	-	No-Op
	1	DPY(1DX) < DB	DPBS → *DPY _{DPA+1DX} Where YW=1DX+4
	2	DPY(1DX) < FA	FA → *DPY _{DPA+1DX} Where YW=1DX+4
	3	DPY(1DX) < FM	FM → *DPY _{DPA+1DX} Where YW=1DX+4

*All bits written unless WRTEXP, WRTHMAN or WRTLMAN set. See SOP1 and HOSTPNL field.

DPBS	0	DB=ZERO	0.0 → DPBS
	1	DB=INBS	INBS → DPBS
	2	DB=VALUE	VALUE → DPBS ^E , VALUE → DPBS ^{ML} , sign extended into DPBS ^{MH}
	3	DB=DPX(1DX)	(DPX _{DPA + 1DX}) → DPBS, Where XR=1DX+4
	4	DB=DPY(1DX)	(DPY _{DPA + 1DX}) → DPBS, Where YR=1DX+4
	5	DB=MD	(MD) DPBS
	6	DB=SPFN	SPFN + 512 → DPBS ^E , SPFN → DPBS ^{ML} , sign extended into DPBS ^{MH}
	7	DB=TM	(TM) → DPBS

DPBS forced to 0 if HOSTPNL field=10 to 13

ML=Mantissa Low (Mantissa Bits 12-27)

MH=Mantissa High (Mantissa Bits 00-11)

E=Exponent

VALUE is a 16-bit 2's complement number, contained in bits 48-63 of the instruction word.

Field	Octal Code	Mnemonic	Effect
XR	0 to 7		DPX Read EFA is (DPA)+XR-4
YR	0 to 7		DPY Read EFA is (DPA)+YR-4
XW	0 to 7		DPX Write EFA is (DPA)+XW-4
YW	0 to 7		DPY Write EFA is (DPA)+YW-4, YW=XW if VALUE is used in another field

FLOATING MULTIPLIER GROUP

51	52	53	54	55
FM	M1		M2	

Field	Octal Code	Mnemonic	Effect
FM	0	-	No-Op
	1	FMUL	Multiply: (M1)*(M2)
M1	0	FM	FM→M1
	1	DPX (1DX)	(DPX _{DPA+ 1DX})→M1, Where XR=1DX+4
	2	DPY (1DX)	(DPY _{DPA+ 1DX})→M1, Where YR=1DX+4
	3	TM	(TM)→M1
M2	0	FA	FA→M2
	1	DPX (1DX)	(DPX _{DPA+ 1DX})→M2, Where XR=1DX+4
	2	DPY (1DX)	(DPY _{DPA+ 1DX})→M2, Where YR=1DX+4
	3	MD	(MD)→M2

Note: These fields are not in effect if VALUE is used in another field.

Arguments that are unnormalized by more than one position will produce incorrect results.

MEMORY GROUP

56 57	58 59	60 61	62 63
MI	MA	DPA	TMA

Field	Octal Code	Mnemonic	Effect
MI	0	-	No-Op
	1	MI<FA	FA→MI, write MI into Data Memory**
	2	MI<FM	FM→MI, write MI into Data Memory**
	3	MI<DB	DPBS→MI, write MI into Data Memory**

**All bits written unless WRTEXP, WRTHMAN or WRTLMAN is set. See SOP1 and HOSTPNL fields.

MA	0	-	No-Op
	1	INCMA	(MA)+1→MA, initiate a Data Memory cycle
	2	DECMA	(MA)-1→MA, initiate a Data Memory cycle
	3	SETMA	*SPFN→MA, initiate a Data Memory cycle

*DPBS is used in place of SPFN if LDREG field is used.

DPA	0	-	No-Op
	1	INCDPA	(DPA)+1→DPA
	2	DECDPA	(DPA)-1→DPA
	3	SETDPA	*SPFN→DPA

*DPBS is used in place of SPFN if LDREG field is used.

Note: These fields are not in effect if a value is used by another field. Changes made in MA, TMA, or DPA do not affect the values of these registers used by other fields during the current instruction.

Field	Octal Code	Mnemonic	Effect
TMA	0	-	No-Op
	1	INCTMA	(TMA)+1→TMA, initiate a read from Table Memory
	2	DECTMA	(TMA)+1→TMA, initiate a read from Table Memory
	3	SETTMA	*SPFN→TMA, initiate a read from Table Memory

*DPBS is used in place of SPFN if LDREG field is used.

Note: These fields are not in effect if a VALUE is used by another field. Changes made in MA, TMA, or DPA do not affect the values of these registers used by other fields during the current instruction.

AP-120B Instruction Field Layout

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
B	SOP			SH	SPS			SPD			FADD		A1	A2		COND			DISP												
S-Pad Group													Adder Group										Branch Group								

SOP1				FADD1					
SPEC OPER						I/O			

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
DPX	DPY		DPBS		XR		YR		XW		YW		FM	M1	M2	MI	MA	DPA		TMA											
Data Pad Group															Multiply Group					Memory Group											
VALUE																															

FLOATING POINT SYSTEMS, INC.

P.O. BOX 23489 PORTLAND, OR 97223 11000 S.W. 11TH STREET, BEAVERTON, OR 97005 (503) 641-3151 TLX: 360470 FLOATPOINT PT