FLOATING POINT SYSTEMS, INC.

Processor Handbook

860-7259-003

by FPS Technical Publications Staff

Processor Handbook

860-7259-003

Publication No. FPS-860-7259-003
February 1979

NOTICE

CONTENTS

ILLUSTRATIONS

TABLES

PREFACE

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

NOTE

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

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

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

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

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

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

CHAPTER 1

GENERAL INFORMATION

1.1 INTRODUCTION

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

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

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

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

Specifically:

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

In a typical application, such as a fast fourier transform (FFT), the above features allow nearly the entire computation to be overlapped with data memory access time.

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

1.2 SYSTEM OVERVIEW

A general block diagram of AP arithmetic paths appears in Figure 1-1.

Connection is made to the host in a manner that permits data transfers to occur under control of either the host computer or the AP. For most host computers, this means that the AP is interfaced to both the programmed I/O and DMA channels.

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

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

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

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

The floating adder (FA) consists of two input registers (A1 and A2) and a two-stage pipeline which performs the operations and convergently rounds the normalized result.

The floating multiplier (FM) consists of input registers (M1 and M2) and a three-stage pipeline which performs the multiply operation. Products are normalized and convergently rounded 38-bit numbers.

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

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

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

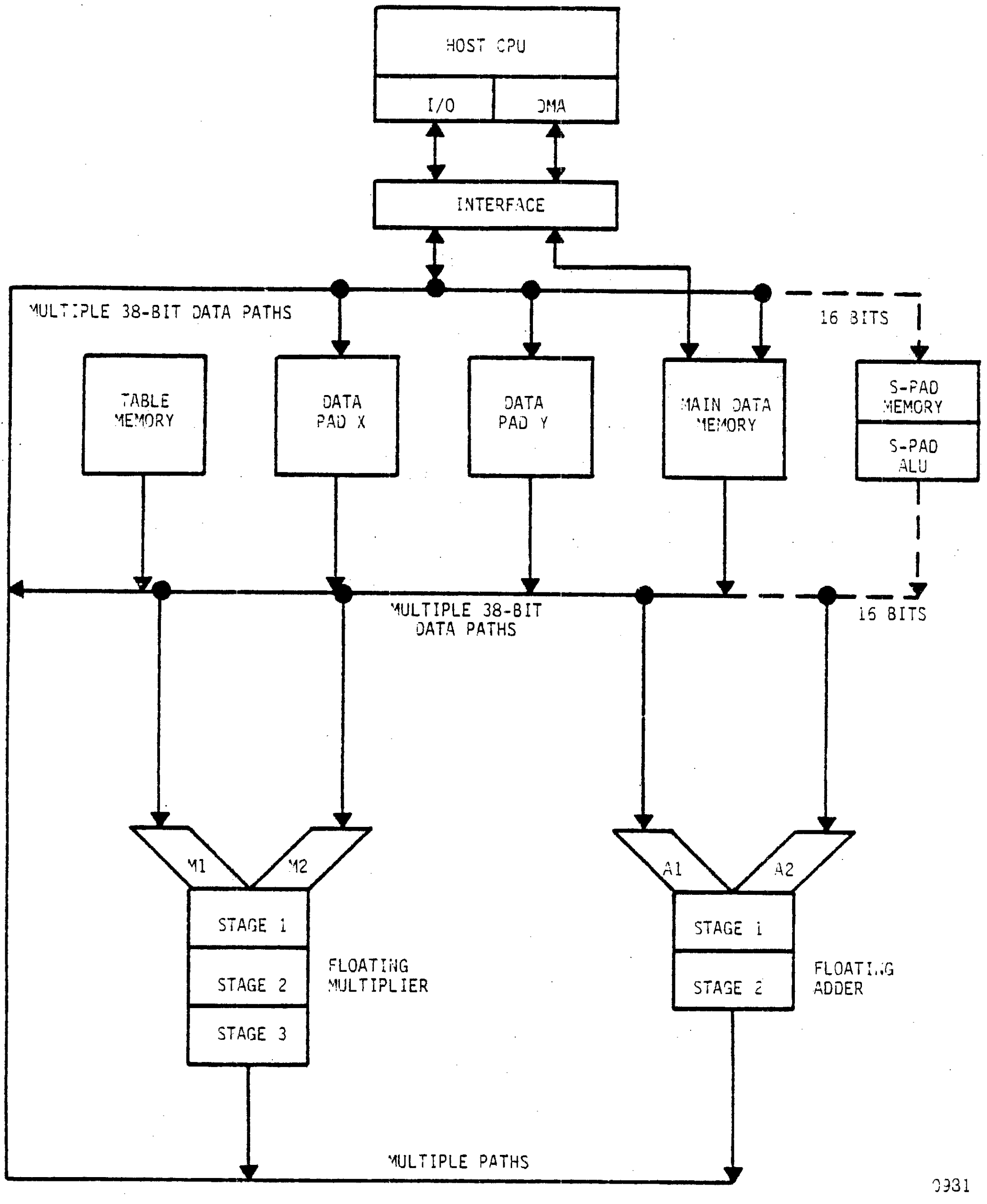


Figure 1-1 General AP Block Diagram

1.3 EXAMPLE AP APPLICATION

A simple FFT processing sequence goes as follows:

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

1. The AP requests host DMA cycles to transfer the array from host memory to internal data memory. Data is converted from host floating-point format to internal AP floating-point format on-the-fly.

2. The FFT algorithm is performed with data remaining in internal AP format. This yields the benefit of 38-bit precision and convergent rounding during the critical phases of processing.

3. The frequency domain array is transferred back to host memory by requesting host DMA cycles. Data is converted from internal format to host format on-the-fly.

4. The AP proceeds to another process or stops executing, depending on previously established conditions. An interrupt to the host can be issued.

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

The AP data memory has DMA capability. That is to say, MD cycles can be stolen from the AP microprocessor by the interface. This capability allows host computer DMA to AP DMA data transfers to occur, thereby minimizing both host CPU and AP overhead.

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

1.4 PHYSICAL DESCRIPTION

The following sections describe the AP hardware.

1.4.1 GENERAL

The AP is available in rack configuration. Mounting is as a standard 19-inch EIA rack-mounted unit requiring 24-1/2 inches of vertical space. The unit is equipped with rack slides permitting easy access to the etched and/or wire-wrapped circuitry with the chassis mounted on the forward portion of the unit. The power panel is mounted at the rear. One and three-quarter inches of space should be available above and below the 24-1/2 inches of the processor. This is for proper intake and exhaust of air through the processor. The control panel (refer to section 1.4.4) and/or blank panels may be used for proper spacing if the customer's equipment mounted above and below the processor does not have the proper free-air space built into it. Intake air should be between 10 and 40 degress centigrade.

1.4.2 FORWARD UNIT

The forward unit contains all AP circuitry except the power supply. There is provision for up to 31 15-by-10-inch etched-circuit boards (ECB). The ECBs plug into a mother board. The ECBs are arranged in a vertical plane (chimney style) with push/pull fans to assure adequate upwards air circulation even in the event of a fan failure. The I/O cable exits at the bottom rear (the exact configuration is computer dependent). This unit is called the processor.

1.4.3 REAR UNIT

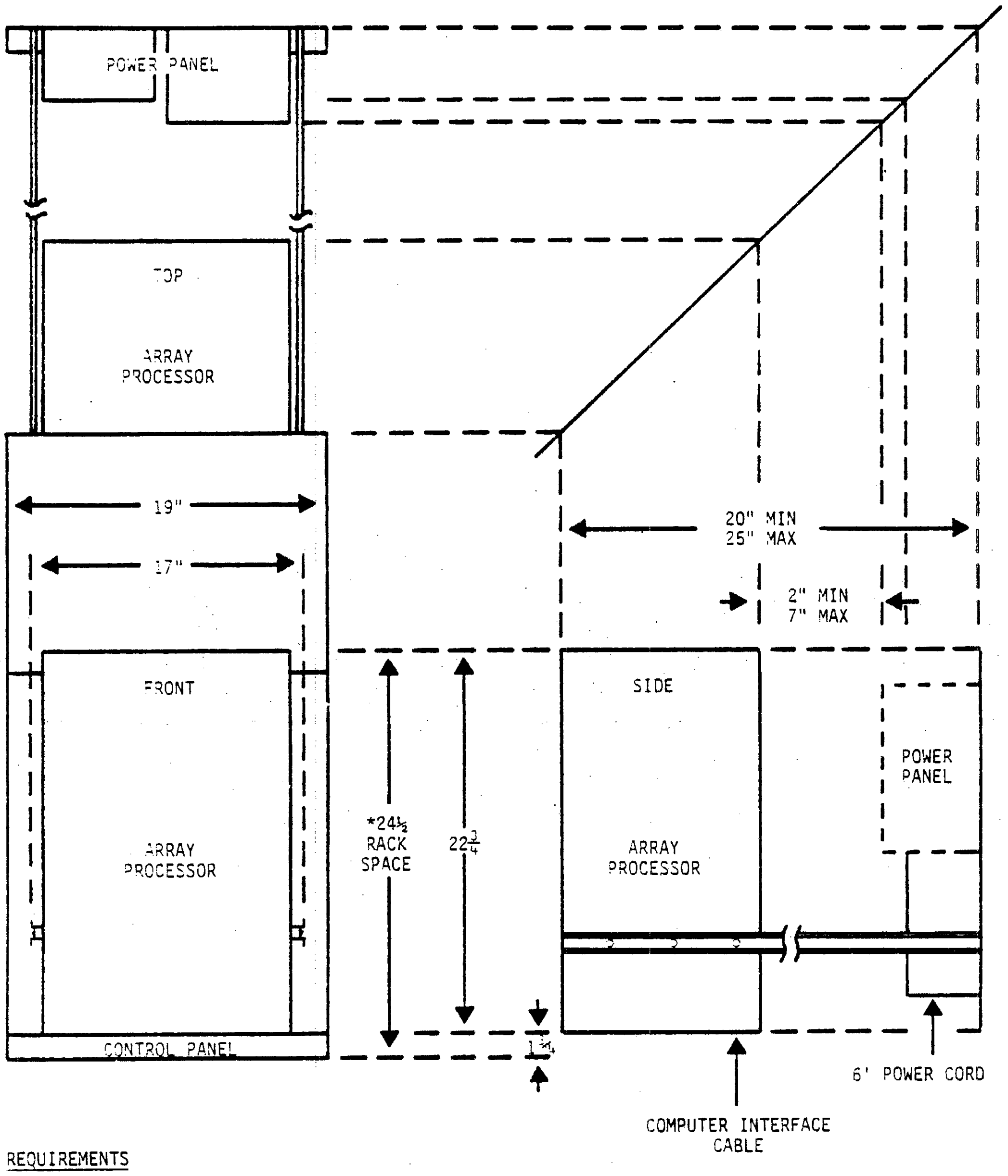
The power supply consists of three assemblies. The first is the main +5 volt supply and is capable of 100 amperes output. The other smaller supplies are -5 and +12 volts. The power supplies have forced convection cooling. All supplies are rear-mounted, along with the line box (containing line filters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

1.4.4 POWER, CONTROLS, AND INDICATORS

The AP is expected to be normally powered up and down with the rest of the system. The AP switch and indicators are on a control panel. There is a single power cord (US standard three-wire with ground) which must be connected to 105 to 125 volts, 50 to 60 hertz. The service should be rated for 20 amperes (10 amperes in the case of the higher ranges) in order to provide a low-impedance source (power required is approximately 1200 volt-amps). The control panel may be mounted above or below either the processor or the power panel. Availability of line power is indicated by a neon LINE VOLTAGE indicator. If the ON/OFF switch is on, then power supplies should come on. There are two operation indicators: one shows array processor action and the other shows DMA transfers. The three individual power supplies have separate indicators (electroluminescent diodes). There are no external adjustments. The internal adjustments are the three power supply setting potentiometers on the power panel.

1.4.5 SERIAL NUMBERS

The processor has a serial number tag on its starboard side near the top and forward ending in A. The power panel tag, ending in B, is located inside and near the top. The control panel has its tag ending in C, also inside.



REQUIREMENTS

1) ENVIRONMENT: Ø - 4Ø°C @ Ø - 9Ø% RELATIVE HUMIDITY.
   (DERATE 1°C PER 25ØØ FT. (762 M) ABOVE SEA LEVEL, 5°C FOR 5Ø HERTZ OPERATION.)

2) POWER CONSUMPTION ≈ 1ØØØ W; SERVICE:
   A. 1Ø5 - 125 VOLTS, 5Ø - 6Ø HERTZ @ 2Ø AMPS. (VOLTAGE OPTION "A" HAS A WHITE WIRE IN THE FAN POWER CABLE.)
   B. 188 - 223 VOLTS, 5Ø - 6Ø HERTZ @ 1Ø AMPS. (VOLTAGE OPTION "B" HAS A BLUE WIRE IN THE FAN POWER CABLE.)
   C. 21Ø - 25Ø VOLTS, 5Ø - 6Ø HERTZ @ 1Ø AMPS. (VOLTAGE OPTION "C" HAS A RED WIRE IN THE FAN POWER CABLE.)
   D. LOW IMPEDANCE SERVICE ADVISED.

3) SPACE:
   *HEIGHT: WITH CONTROL PANEL AT THE FRONT; 24½" (62.23 CM).
            WITH CONTROL PANEL AT THE REAR; 22¾" (57.79 CM).
   WIDTH: 29" (48.26 CM).
   DEPTH: 2Ø - 25" (5Ø.3Ø - 63.5Ø CM).

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

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

0982

Figure 1-2 AP Physical Configuration

1.5 SOFTWARE

Four software packages can be supplied with the AP which assist the user toward the solution of the particular processing task.

1.5.1 APEX (AP EXECUTIVE)

APEX is a mechanism for communicating with the AP via a series of FORTRAN or machine language subroutine calls. The executive driver routine interprets the particular user call and directs the AP to perform the specified action. For example, in FORTRAN, to load an array A containing N real data points into the AP and perform a real fast fourier transform upon that data:

```
        .
        .
        .
      IA=0

      CALL APPUT (A,IA,N,2)

      CALL RFFT (IA,N,1)
        .
        .
        .
```

Both the standard applications subroutines described below and user-developed AP programs may be called from the host computer using APEX.

1.5.2 APMATH (AP MATH LIBRARY)

There are 239 subroutines written in AP assembly language. They are callable from the host computer FORTRAN or machine language using APEX. They are listed in Table 1-3.

1.5.3 PROGRAM DEVELOPMENT PACKAGE

Six FORTRAN IV programs, which are compiled on the host computer during installation, aid user program development.

These are:

APAL	AP assembly language. Cross-assembler which provides a two-pass assembly of symbolic coding into an object module. APAL generates detailed error diagnostics.
APLOAD	AP loader. Links and relocates separate APAL and AP-FORTRAN object modules together. It produces a load module and a host FORTRAN subroutine which transfers the load module to the AP.
APDBUG	AP debugger. Interactive debugging program. The user may selectively set breakpoints, examine and change memory, and register contents and run program segments.
APSIM	AP simulator. Called by APDBUG, APSIM provides a programmed simulation of the various hardware elements of the AP. All timing characteristics of the AP are emulated and the floating-point arithmetic is simulated (including rounding) to the least significant bit. APSIM is a convenient tool in bringing up new AP programs off-line without interferring with production runs.
VFC	Vector Function Chainer. A translator to convert VFC syntax to AP assembly language (APAL). It consolidates multiple CALLS to the AP from the host computer into one CALL whenever possible.

AP-FORTRAN	Array processor FORTRAN. A compiler which allows FORTRAN subprograms to execute on the AP. The compiler produces object modules which are used as input to the AP loader (APLOAD).

1.5.4 APTEST (AP TEST PROGRAMS)

APTEST is a collection of interactive diagnostic tests and verify programs which aid in isolation of hardware faults.

These are:

APTEST	AP tester. Exercises the panel, DMA interface, and various internal registers and memories. Tests main data memory with simple patterns and then with random numbers. Board-level diagnostic indicators are provided.
APPATH	AP path tester. Tests the various internal data paths and gives board level diagnostics.
APARTH	AP arithmetic test. Tests the floating-point adder, multiplier, and s-pad arithmetic unit with pseudorandom number and operation sequences.
FIFFT	Forward/inverse FFT test. Verifies the correct operation of the AP as a complete unit by doing forward/inverse FFT transforms on both spikes and random number sequences.

Table 1-1 Floating-Point Arithmetic Times

OPERATION	TRAVEL TIME	PIPELINE INTERVAL
Add/Subtract	Ø.333 us	Ø.167 us
Multiply	Ø.5ØØ us	Ø.167 us
Multiply-Add	Ø.833 us	Ø.167 us
Complex Add/Subtract	Ø.5ØØ us	Ø.333 us
Complex Multiply	1.333 us	Ø.667 us
Complex Multiply-Add	1.667 us	Ø.667 us

0983

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

Table 1-2 Basic Scalar Functions

OPERATION	TYPICAL EXECUTION TIME/LOOP (us)		PROGRAM SIZE (AP PS WORDS)	
	167 ns	333 ns	167 ns	333 ns
Divide	3.8	3.8	28	28
Square Root	3.8	3.8	28	28
Exponential	4.2	4.2	28	28
Natural Logarithm	4.0	4.0	37	37
Base 1Ø Logarithm	4.7	4.7	37	37
Sine	4.9	4.9	35	35
Cosine	5.4	5.4	35	35
Arctangent	8.7	8.7	74	74
Arctangent of (Y/X)	13.8	13.8	74	74

0984

These functions take arguments from data pad and return full-word accuracy results to data pad. Full-precision polynomial coefficients for these functions are contained on the standard 512 words of table memory.

Table 1-3 Summary of AP FORTRAN Callable Routines

Name	Operation	Typical Execution Time/Loop (us) 167	333	Program Size (AP PS words) 167	333
DATA TRANSFER AND CONTROL OPERATIONS (APEX)					
APPUT	PUT DATA INTO THE AP	#.#	#.#	0	0
APGET	GET DATA FROM THE AP	#.#	#.#	0	0
APCLR	INITIALIZE THE AP	#.#	#.#	0	0
APWD	WAIT FOR AP DATA TRANSFER	#.#	#.#	0	0
APWR	WAIT FOR AP PROGRAM EXECUTION	#.#	#.#	0	0
APWAIT	WAIT FOR AP	#.#	#.#	0	0
APGSP	READ AN AP S-PAD REGISTER	#.#	#.#	0	0
APCHK	CHECK AP PROGRAM ERROR CONDITION	#.#	#.#	0	0
APSTAT	GET AP HARDWARE STATUS	#.#	#.#	0	0
BASIC VECTOR ARITHMETIC					
VCLR	VECTOR CLEAR	0.2	0.3	16	4
VMOV	VECTOR MOVE	0.5	0.8	16	6
VSWAP	VECTOR SWAP	1.2	1.5	21	12
VFILL	VECTOR FILL	0.3	0.3	5	5
VRAMP	VECTOR RAMP	0.3	0.3	12	12
VNEG	VECTOR NEGATE	0.5	0.8	18	7
VADD	VECTOR ADD	0.8	1.3	20	8
VSUB	VECTOR SUBTRACT	0.8	1.3	20	8
VMUL	VECTOR MULTIPLY	0.8	1.3	20	11
VDIV	VECTOR DIVIDE	1.7	1.7	75	75
VSADD	VECTOR SCALAR ADD	0.5	0.8	19	8
VSMUL	VECTOR SCALAR MULTIPLY	0.5	0.8	20	9
VTSADD	VECTOR TABLE SCALAR ADD	0.5	0.8	8	8
VTSMUL	VECTOR TABLE SCALAR MULTIPLY	0.5	0.8	8	8
VSQ	VECTOR SQUARE	0.5	0.8	9	9
VSSQ	VECTOR SIGNED SQUARE	0.5	0.8	21	9
VABS	VECTOR ABSOLUTE VALUE	0.5	0.8	17	7
VSQRT	VECTOR SQUARE ROOT	1.8	1.8	79	79
VLOG	VECTOR LOGARITHM (BASE 10)	2.7	2.7	54	58
VLN	VECTOR NATURAL LOGARITHM	2.7	2.7	42	42

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us) 167	333	Program Size (AP PS words) 167	333
VALOG	VECTOR ANTILOGARITHM (BASE 10)	2.3	2.3	58	58
VEXP	VECTOR EXPONENTIAL	2.3	2.3	55	55
VSIN	VECTOR SINE	1.3	1.3	34	34
VCOS	VECTOR COSINE	1.3	1.3	34	34
VATAN	VECTOR ARCTANGENT	9.7	9.8	87	87
VATN2	VECTOR ARCTANGENT OF Y/X	14.2	14.2	88	88
VRAND	VECTOR RANDOM NUMBERS	1.2	1.2	16	16
VMSA	VECTOR MULTIPLY AND SCALAR ADD	0.8	1.3	23	14
VSMA	VECTOR SCALAR MULTIPLY AND ADD	0.8	1.3	21	14
VSMSB	VECTOR SCALAR MULTIPLY AND SUBTRACT	0.8	1.3	21	14
VMA	VECTOR MULTIPLY AND ADD	1.2	1.8	23	15
VMSB	VECTOR MULTIPLY AND SUBTRACT	1.2	1.8	23	15
VAM	VECTOR ADD AND MULTIPLY	1.2	1.8	23	14
VSBM	VECTOR SUBTRACT AND MULTIPLY	1.2	1.8	23	14
VSMSA	VECTOR SCALAR MULTIPLY AND SCALAR ADD	0.5	0.8	23	15
VMMA	VECTOR MULTIPLY, MULTIPLY, AND ADD	1.5	2.3	27	19
VMMSB	VECTOR MULTIPLY MULTIPLY AND SUBTRACT	1.5	2.3	27	19
VAAM	VECTOR ADD, ADD, AND MULTIPLY	1.5	2.3	13	20
VSBSBM	VECTOR SUBTRACT SUBTRACT AND MULTIPLY	1.5	2.3	13	20
VAND	VECTOR LOGICAL AND	0.8	1.3	20	8
VEQV	VECTOR LOGICAL EQUIVALENCE	0.8	1.3	20	8
VOR	VECTOR LOGICAL OR	0.8	1.3	20	8
VFRAC	VECTOR TRUNCATE TO FRACTION	0.7	0.8	13	13
VINT	VECTOR TRUNCATE TO INTEGER	0.5	0.8	9	9
VINDEX	VECTOR INDEX	0.8	1.3	28	26
	VECTOR-TO-SCALAR OPERATIONS				
SVE	SUM OF VECTOR ELEMENTS	0.3	0.3	7	7
SVEMG	SUM OF VECTOR ELEMENT MAGNITUDES	0.3	0.3	10	10
SVESQ	SUM OF VECTOR ELEMENT SQUARES	0.3	0.3	10	10
SVS	SUM OF VECTOR SIGNED SQUARES	0.3	0.3	11	11
DOTPR	DOT PRODUCT	0.5	0.8	21	9
MAXV	MAXIMUM ELEMENT IN VECTOR	0.3	0.3	19	19
MINV	MINIMUM ELEMENT IN VECTOR	0.3	0.3	19	19
MAXMGV	MAXIMUM MAGNITUDE ELEMENT IN VECTOR	0.3	0.3	19	19
MINMGV	MINIMUM MAGNITUDE ELEMENT IN VECTOR	0.3	0.3	19	19

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)		Program Size (AP PS words)	
		167	333	167	333
MEANV	MEAN VALUE OF VECTOR ELEMENTS	0.3	0.3	49	49
MEAMGV	MEAN OF VECTOR ELEMENT MAGNITUDES	0.3	0.3	52	52
MEASQV	MEAN OF VECTOR ELEMENT SQUARES	0.3	0.3	52	52
RMSQV	ROOT-MEAN-SQUARE OF VECTOR ELEMENTS	0.3	0.3	81	81
	VECTOR COMPARISON OPERATIONS				
VMAX	VECTOR MAXIMUM	0.8	1.3	22	13
VMIN	VECTOR MINIMUM	0.8	1.3	22	13
VMAXMG	VECTOR MAXIMUM MAGNITUDE	0.8	1.3	14	14
VMINMG	VECTOR MINIMUM MAGNITUDE	0.8	1.3	14	14
VCLIP	VECTOR CLIP	0.5	0.8	16	16
VICLIP	VECTOR INVERTED CLIP	0.7	0.8	19	19
VLIM	VECTOR LIMIT	0.5	0.8	14	14
LVGT	LOGICAL VECTOR GREATER THAN	0.8	1.3	23	13
LVGE	LOGICAL VECTOR GREATER THAN OR EQUAL	0.8	1.3	23	13
LVEQ	LOGICAL VECTOR EQUAL	0.8	1.3	23	13
LVNE	LOGICAL VECTOR NOT EQUAL	0.8	1.3	23	13
LVNOT	LOGICAL VECTOR NOT	0.5	0.8	21	12
VLMERG	VECTOR LOGICAL MERGE	0.8	1.5	23	16
	COMPLEX VECTOR ARITHMETIC				
CVMOV	COMPLEX VECTOR MOVE	0.8	1.3	9	9
CVFILL	COMPLEX VECTOR FILL	0.5	0.7	8	8
CVCOMB	COMPLEX VECTOR COMBINE	1.1	1.7	10	10
CVREAL	FORM COMPLEX VECTOR OF REALS	0.8	1.2	9	9
VREAL	EXTRACT REALS OF COMPLEX VECTOR	0.5	0.8	17	7
VIMAG	EXTRACT IMAGINARIES OF COMPLEX VECTOR	0.5	0.8	18	8
CVNEG	COMPLEX VECTOR NEGATE	0.8	1.3	11	11
CVCONJ	COMPLEX VECTOR CONJUGATE	0.7	1.3	10	12
CVADD	COMPLEX VECTOR ADD	1.0	2.0	13	12
CVSUB	COMPLEX VECTOR SUBTRACT	1.0	2.0	13	12

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us) 167	333	Program Size (AP PS words) 167	333
CVMUL	COMPLEX VECTOR MULTIPLY	1.0	2.0	25	26
CVSMUL	COMPLEX VECTOR SCALAR MULTIPLY	0.8	1.3	12	12
CVRCIP	COMPLEX VECTOR RECIPROCAL	5.2	5.2	50	50
CRVADD	COMPLEX AND REAL VECTOR ADD	1.3	1.8	14	14
CRVSUB	COMPLEX AND REAL VECTOR SUBTRACT	1.3	1.8	14	14
CRVMUL	COMPLEX AND REAL VECTOR MULTIPLY	1.3	1.8	14	14
CRVDIV	COMPLEX AND REAL VECTOR DIVIDE	3.3	3.3	92	92
CVMA	COMPLEX VECTOR MULTIPLY AND ADD	1.3	2.7	29	30
CVMAGS	COMPLEX VECTOR MAGNITUDE SQUARED	0.7	1.2	13	18
SCJMA	SELF-CONJUGATE MULTIPLY AND ADD	0.8	1.5	14	15
POLAR	RECTANGULAR TO POLAR CONVERSION	19.5	19.5	120	120
RECT	POLAR TO RECTANGULAR CONVERSION	2.3	2.3	49	49
CVEXP	COMPLEX VECTOR EXPONENTIAL	2.0	2.0	43	43
CVMEXP	VECTOR MULTIPLY COMPLEX EXPONENTIAL	2.3	2.3	48	48
CDOTPR	COMPLEX DOT PRODUCT	0.7	1.3	15	16

DATA FORMATTING OPERATIONS

Name	Operation	167	333	167	333
VFLT	VECTOR INTEGER FLOAT	0.5	0.8	13	11
VFIX	VECTOR INTEGER FIX	0.7	0.8	18	7
VSMAFX	VECTOR SCALAR MULTIPLY, ADD, AND FIX	0.7	0.8	14	13
VSCALE	VECTOR SCALE (POWER 2) AND FIX	0.7	0.8	12	12
VSCSCL	VECTOR SCAN, SCALE (POWER 2) AND FIX	1.5	1.7	19	19
VSHFX	VECTOR SHIFT AND FIX	0.7	0.8	9	9
VUP8	VECTOR 8-BIT BYTE UNPACK	0.5	0.5	71	71
VUPS8	VECTOR 8-BIT SIGNED BYTE UNPACK	0.9	0.9	107	107
VPK8	VECTOR 8-BIT BYTE PACK	0.9	0.9	65	65
VUP16	VECTOR 16-BIT BYTE UNPACK	0.8	0.8	61	61
VUPS16	VECTOR 16-BIT SIGNED BYTE UNPACK	1.3	1.3	58	58
VPK16	VECTOR 16-BIT BYTE PACK	0.8	0.8	46	46
VFLT32	VECTOR 32-BIT INTEGER FLOAT	1.7	1.7	65	65
VFIX32	VECTOR 32-BIT INTEGER FIX	1.2	1.2	33	33
VSEFLT	VECTOR SIGN EXTEND AND FLOAT	0.8	0.8	15	15

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us) 167	333	Program Size (AP PS words) 167	333
	MATRIX OPERATIONS				
MTRANS	MATRIX TRANSPOSE	0.5	0.9	18	22
MMUL	MATRIX MULTIPLY	0.62*	0.83	59	59
MMUL32	MATRIX MULTIPLY (DIMENSION <=32)	0.50*	0.73	27	27
MATINV	MATRIX INVERSE	1.6 *	2.1	160	160
SOLVEQ	LINEAR EQUATION SOLVER	0.7 *	0.9	216	222
MVML3	MATRIX VECTOR MULTIPLY (3X3)	2.0 *	2.2	30	30
MVML4	MATRIX VECTOR MULTIPLY (4X4)	3.3 *	3.8	39	39
CTRN3	3-DIMENSION COORDINATE TRANSFORMATION	2.3 *	2.5	37	37
FMMM	FAST MEMORY MATRIX MULTIPLY	0.43*		61	
FMMM32	FAST MEMORY MATRIX MULTIPLY (<=32)	0.41*		33	
	FFT OPERATIONS				
CFFT	COMPLEX TO COMPLEX FFT (IN PLACE)	0.28*	0.40	186	184
CFFTB	COMPLEX TO COMPLEX FFT (NOT IN PLACE)	0.20*	0.28	189	189
RFFT	REAL TO COMPLEX FFT (IN PLACE)	0.18*	0.27	253	251
RFFTB	REAL TO COMPLEX FFT (NOT IN PLACE)	0.14*	0.20	252	252
CFFTSC	COMPLEX FFT SCALE	0.8	1.3	42	42
RFFTSC	REAL FFT SCALE AND FORMAT	0.7	0.8	59	59
CFFT2D	COMPLEX TO COMPLEX 2-DIMENSIONAL FFT	0.5 *	0.5	274	274
RFFT2D	REAL TO COMPLEX 2-DIMENSIONAL FFT	0.4 *	0.4	585	585
	AUXILIARY OPERATIONS				
CONV	CONVOLUTION (CORRELATION)	0.28*	0.28	106	106
DEQ22	DIFFERENCE EQUATION, 2 POLES, 2 ZEROS	0.8	0.8	25	25
VPOLY	VECTOR POLYNOMIAL EVALUATION	1.0 *	1.2	41	41
VSUM	VECTOR SUM OF ELEMENTS INTEGRATION	0.7	0.8	13	13

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)		Program Size (AP PS words)	
		167	333	167	333
VTRAPZ	VECTOR TRAPEZOIDAL RULE INTEGRATION	0.7	0.8	16	16
VSIMPS	VECTOR SIMPSONS 1/3 RULE INTEGRATION	0.7	0.8	25	25
WIENER	WIENER LEVINSON ALGORITHM	0.50*	0.65	100	100

SIGNAL PROCESSING OPERATIONS (optional)

HIST	HISTOGRAM	1.3	1.4	71	71
HANN	HANNING WINDOW MULTIPLY	0.7	0.8	41	41
ASPEC	ACCUMULATING AUTO-SPECTRUM	0.8	1.5	21	22
CSPEC	ACCUMULATING CROSS-SPECTRUM	1.3	2.7	39	40
VAVLIN	VECTOR LINEAR AVERAGING	0.8	1.3	54	46
VAVEXP	VECTOR EXPONENTIAL AVERAGING	0.8	1.3	55	46
VDBPWR	VECTOR CONVERSION TO DB (POWER)	1.2	1.3	75	75
TRANS	TRANSFER FUNCTION	3.3	3.3	100	100
COHER	COHERENCE FUNCTION	4.0	4.5	109	114
ACORT	AUTO-CORRELATION (TIME-DOMAIN)	0.29*	0.29	121	121
ACORF	AUTO-CORRELATION (FREQUENCY-DOMAIN)	1.80*	2.70	501	489
CCORT	CROSS-CORRELATION (TIME-DOMAIN)	0.29*	0.29	121	121
CCORF	CROSS-CORRELATION (FREQUENCY-DOMAIN)	2.58*	3.93	526	510
TCONV	POSTTAPERED CONVOLUTION (CORRELATION)	0.30*	0.30	112	112

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us) 167	333	Program Size (AP PS words) 167	333
	TABLE MEMORY OPERATIONS (optional)				
MTMOV	VECTOR MOVE (MD TO TM)	0.2	0.3	6	7
TMMOV	VECTOR MOVE (TM TO MD)	0.2	0.3	5	5
MTIMOV	VECTOR MOVE WITH INCREMENT (MD TO TM)	0.5	0.5	7	7
TMIMOV	VECTOR MOVE WITH INCREMENT (TM TO MD)	0.3	0.3	15	15
TTIMOV	VECTOR MOVE WITH INCREMENT (TM TO TM)	0.5	0.5	7	7
MMTADD	VECTOR ADD (MD+MD TO TM)	0.7	0.8	20	13
MMTSUB	VECTOR SUBTRACT (MD-MD TO TM)	0.7	0.8	20	13
MMTMUL	VECTOR MULTIPLY (MD*MD TO TM)	0.7	0.8	20	13
MTMADD	VECTOR ADD (MD+TM TO MD)	0.5	0.8	20	9
MTMSUB	VECTOR SUBTRACT (MD-TM TO MD)	0.5	0.8	20	9
TMMSUB	VECTOR SUBTRACT (TM-MD TO MD)	0.5	0.8	20	9
MTMMUL	VECTOR MULTIPLY (MD*TM TO MD)	0.5	0.8	20	9
MTTADD	VECTOR ADD (MD+TM TO TM)	0.5	0.5	20	20
MTTSUB	VECTOR SUBTRACT (MD-TM TO TM)	0.5	0.5	20	20
TMTSUB	VECTOR SUBTRACT (TM-MD TO TM)	0.5	0.5	20	20
MTTMUL	VECTOR MULTIPLY (MD*TM TO TM)	0.5	0.5	20	20
TTMADD	VECTOR ADD (TM+TM TO MD)	0.5	0.5	20	20
TTMSUB	VECTOR SUBTRACT (TM-TM TO MD)	0.5	0.5	20	20
TTMMUL	VECTOR MULTIPLY (TM*TM TO MD)	0.5	0.5	20	20
TTTADD	VECTOR ADD (TM+TM TO TM)	0.7	0.7	9	9
TTTSUB	VECTOR SUBTRACT (TM-TM TO TM)	0.7	0.7	9	9
TTTMUL	VECTOR MULTIPLY (TM*TM TO TM)	0.7	0.7	10	10

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)			Program Size (AP PS words)	
		167	|	333	167	333
	APAL-CALLABLE UTILITY OPERATIONS					
DIV	SCALAR DIVIDE	3.8	@	3.8	28	28
SQRT	SCALAR SQUARE ROOT	3.8	@	3.8	28	28
LOG	SCALAR LOGARITHM (BASE 10)	4.7	@	4.7	37	37
LN	SCALAR NATURAL LOGARITHM	4.0	@	4.0	37	37
EXP	SCALAR EXPONENTIAL	4.2	@	4.2	28	28
SIN	SCALAR SINE	4.9	@	4.9	35	35
COS	SCALAR COSINE	5.4	@	5.4	35	35
ATAN	SCALAR ARCTANGENT	8.7	@	8.7	74	74
ATN2	SCALAR ARCTANGENT OF Y/X	13.8	@	13.8	74	74
SPFLT	FLOAT S-PAD INTEGER	0.8	@	0.8	5	5
SPUFLT	S-PAD UNSIGNED FLOAT	0.8	@	0.8	8	8
SPNEG	S-PAD NEGATE	0.3	@	0.3	2	2
SPADD	S-PAD ADD	0.2	@	0.2	1	1
SPSUB	S-PAD SUBTRACT	0.2	@	0.2	1	1
SPMUL	S-PAD MULTIPLY	2.3	@	2.3	14	14
SPDIV	S-PAD DIVIDE	6.2	@	6.2	43	43
SPRS	S-PAD RIGHT SHIFT	0.3	*	0.3	5	5
SPLS	S-PAD LEFT SHIFT	0.3	*	0.3	5	5
SPAND	S-PAD AND	0.2	@	0.2	1	1
SPOR	S-PAD OR	0.2	@	0.2	1	1
SPNOT	S-PAD NOT	0.2	@	0.2	1	1
SAVESP	SAVE S-PAD INTO PROGRAM MEMORY	0.8	*	0.8	18	18
SAVSPO	SAVE S-PAD 0 INTO PROGRAM MEMORY	2.0	*	2.0	11	11
SETSP	LOAD S-PADS FROM PROGRAM MEMORY	2.3	*	2.3	33	33
SET2SP	LOAD 2 S-PADS FROM PROGRAM MEMORY	5.7	@	5.7	33	33
MDCOM	MAIN DATA COMPARE AND SET S-PAD	1.8	@	2.0	11	11
ZMD	CLEAR ALL PAGES OF MAIN DATA MEMORY	0.2		0.3	29	29
RDC5	READ CONTROL BIT 5 INTERRUPT	1.5	@	1.5	9	9
SETC5	SET CONTROL BIT 5 INTERRUPT	0.2	@	0.2	1	1
DAREAD	READ DEVICE ADDRESS REGISTER	0.3	@	0.3	2	2
DAWRIT	WRITE DEVICE ADDRESS REGISTER	0.3	@	0.3	2	2
VFCL1	VECTOR FUNCTION CALLER (1 ARGUMENT)	0.8		1.0	10	10
VFCL2	VECTOR FUNCTION CALLER (2 ARGUMENT)	1.0		1.0	11	11
BITREV	COMPLEX VECTOR BIT REVERSE ORDERING	0.9		1.4	45	43
REALTR	REAL FFT UNRAVEL AND FINAL PASS	0.4		0.7	68	68
FFT2	RADIX 2 FFT FIRST PASS	1.3		2.7	16	16

Table 1-3 Summary of AP FORTRAN Callable Routines (cont.)

Name	Operation	Typical Execution Time/Loop (us)		Program Size (AP PS words)	
		167	333	167	333
FFT4	RADIX 4 FFT PASS	3.7	5.3	79	79
FFT2B	RADIX 2 FFT FIRST PASS + BIT REVERSE	1.3	2.7	25	25
FFT4B	RADIX 4 FFT FIRST PASS + BIT REVERSE	2.7	5.3	43	43
STSTAT	SET FFT MODE STATUS BITS	5.0 @	5.0	19	19
CLSTAT	CLEAR FFT MODE STATUS BITS	0.5 @	0.5	19	19
ILOG2	LOGARITHM (BASE 2)	4.0 @	4.0	19	19
ADV2	ADVANCE POINTERS AFTER RADIX 2 FFT	0.7 @	0.7	7	7
ADV4	ADVANCE POINTERS AFTER RADIX 4 FFT	0.7 @	0.7	7	7
SET24B	SETUP FOR FFT2B AND FFT4B	1.2 @	1.2	8	8
XCFFT	EXPANDED COMPLEX FFT	0.32*	0.42	187	187
XRFFT	EXPANDED REAL FFT	0.19*	0.28	256	256
XBITRE	EXPANDED BIT REVERSE	3.7	3.7	44	44
XREALT	EXPANDED REAL FFT FINAL PASS	0.4	0.7	71	71
PCFFT	PARTIAL COMPLEX FFT	1.05*	1.50	117	117
XFFT4	EXPANDED RADIX 4 FFT PASS	3.7	5.3	79	79
CTOR	COMPLEX TO REAL FFT UNSCRAMBLE	0.13*	0.13	80	80
RTOC	REAL TO COMPLEX FFT SCRAMBLE	0.19*	0.09	143	143
SSDA	SINGLE + SINGLE TO DOUBLE ADD	1.5 @	1.5	10	10
SSDM	SINGLE * SINGLE TO DOUBLE MULTIPLY	11.5 @	11.5	81	81
SDDA	SINGLE + DOUBLE TO DOUBLE ADD	4.5 @	4.5	28	28
DDDA	DOUBLE + DOUBLE TO DOUBLE ADD	7.5 @	7.5	48	48
DDDM	DOUBLE * DOUBLE TO DOUBLE MULTIPLY	18.5 @	18.5	117	117

NOTE

#.#	Timing host system dependent
*	Refer to description of routine for explanation of timing
@	Total execution time

Table 1-4 Convolution (Correlation)

ELEMENT COUNTS		TYPICAL EXECUTION TIME/LOOP (us)	
M	N	167ns	333ns
8	128	0.28	0.28
32	128	0.83	0.83
128	128	3.0	3.0
8	1024	2.3	2.3
32	1024	6.6	6.6
128	1024	24.0	24.0
1024	1024	186.2	186.2

0985

Table 1-5 Fast Fourier Transforms

POINTS	RFFT		RFFTB		CFFT		CFFTB	
	167ns	333ns	167ns	333ns	167ns	333ns	167ns	333ns
64	0.18	0.27	0.14	0.20	0.28	0.40	0.20	0.28
128	0.35	0.50	0.27	0.38	0.62	0.95	0.47	0.72
256	0.74	1.13	0.58	0.90	1.28	1.86	0.97	1.41
512	1.50	2.22	1.20	1.76	2.86	4.38	2.26	3.48
1024	3.30	5.08	2.70	4.18	5.95	8.73	4.75	6.93
2048	6.81	10.12	5.61	8.32	13.32	20.10	10.33	16.60
4096	14.95	22.96	12.56	19.37	27.44	40.33	22.66	33.16
8192	30.88	45.86	26.09	38.69	60.33	91.66	50.76	77.31
16384	67.19	102.70	57.63	88.36	124.70	183.27	105.58	154.59
32768	138.42	205.35	119.30	176.68	--	--	--	--

0986

CHAPTER 2

FUNCTIONAL DESCRIPTION

2.1 INTRODUCTION

The hardware of the AP is composed of the following three types of functional elements:

- logical and control elements

  control unit
  s-pad unit

- floating-point arithmetic elements

  floating-point adder
  floating-point multiplier

- memory elements

  data pad unit
  main data memory unit
  table memory unit

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

2.2 CONTROL UNIT

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

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

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

Branching is accomplished in two ways. A short-range branch is provided by adding the 5-bit branch displacement field to the current PSA. This gives a branch range of from -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 and sets PSA to the subroutine address. Return is via a return, which loads the PSA with the last entered return address on the SRS.

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

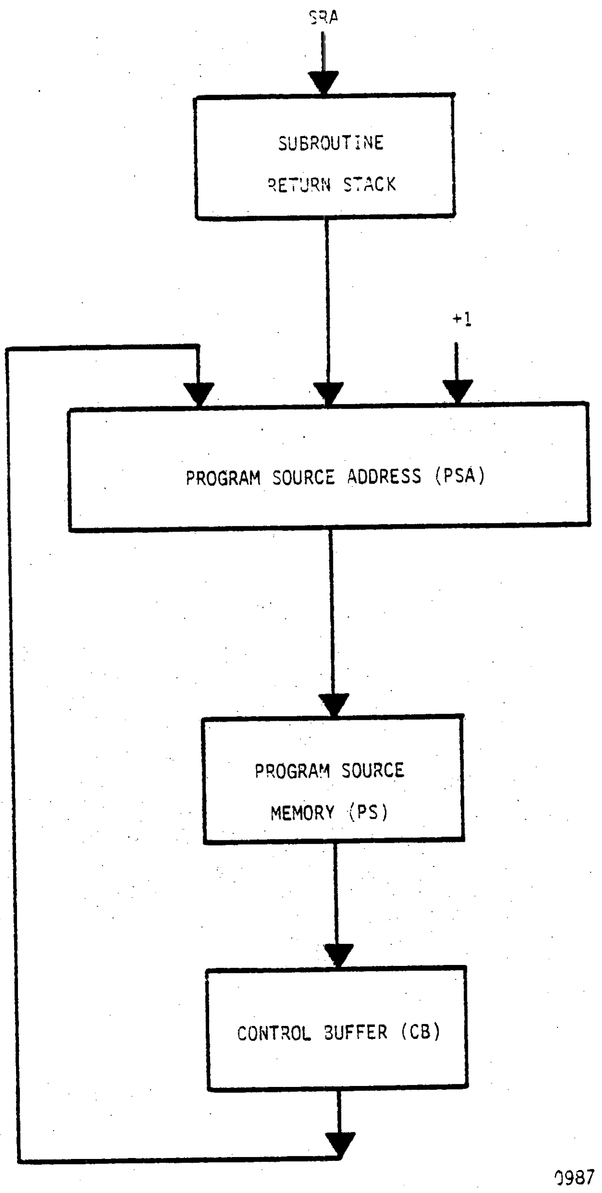


Figure 2-1 Control Unit

2.3 S-PAD UNIT

This unit, illustrated by Figure 2-2, performs the integer address indexing, loop counting, and control functions necessary to direct completion of a given algorithm. In form, it is similar to familiar minicomputers such as the PDP-11 and Nova.

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

The output of the ALU may be directed back to the specified s-pad destination register and also may be directed to any of the following address memory registers: memory address (MA), table memory address (TMA), or data pad address (DPA).

The s-pad integer ALU functions include the following:

function	effect	
move	$S \rightarrow D$	S-source register
logical complement	$S \rightarrow D$	D-destination register
clear	$0 \rightarrow D$	
increment	$S+1 \rightarrow D$	
decrement	$S-1 \rightarrow D$	
add	$D+S \rightarrow D$	
subtract	$D-S \rightarrow D$	
logical AND	D AND $S \rightarrow D$	
logical OR	D OR $S \rightarrow D$	
logical equivalence	D EQV $S \rightarrow D$	

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

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

The s-pad ALU also sets three condition bits in the AP status register depending upon the output of the ALU/shifter:

N: set if result <0; cleared otherwise

Z: set if result $=0$; cleared otherwise

C: set if a carry occurred; cleared otherwise

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

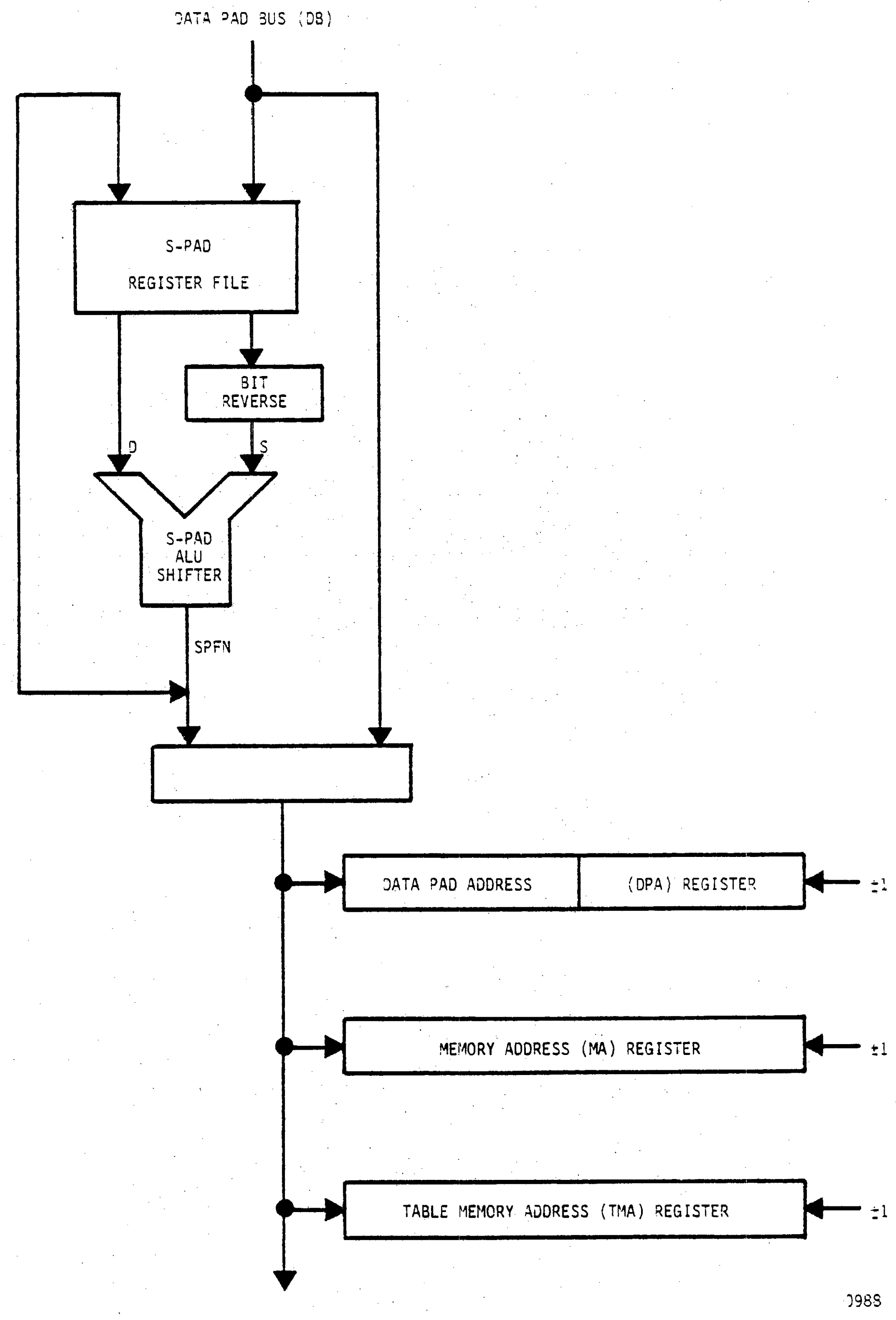


Figure 2-2 S-Pad Unit

2.4 FLOATING-POINT ADDER UNIT

The floating-point adder, shown in Figure 2-3, performs addition or subtraction operations on the contents of the adder input registers (A1 and A2). The operation is performed in two stages, each of which takes one machine cycle.

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

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

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

A1 may be loaded from data pad (DP), from the output of the floating multiplier (FM), or from table memory (TM). A2 may be loaded from data pad (DP), from the output of the floating adder (FA), or from main data memory (MD).

The output of the floating adder (FA) may be directed to the multiplier (M2), to the adder (A2), to data pad (DP), or to memory input (MI).

The operations performed by the floating adder are:

- A1+A2
- A1-A2
- A2-A1
- A1 EQV A2
- A1 AND A2
- A1 OR A2
- convert A2 from signed magnitude to 2's complement format
- convert A2 from 2's complement to signed magnitude format
- scale A2
- absolute value of A2
- 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 is forced to the signed maximum value.
FU	Set to one if exponent underflow occurred. The result is forced to zero.

The overflow and underflow bits remain set until cleared by the program. These bits may be tested by the instruction after the floating adder result is completed (i.e., three cycles after the floating adder operation is initiated).

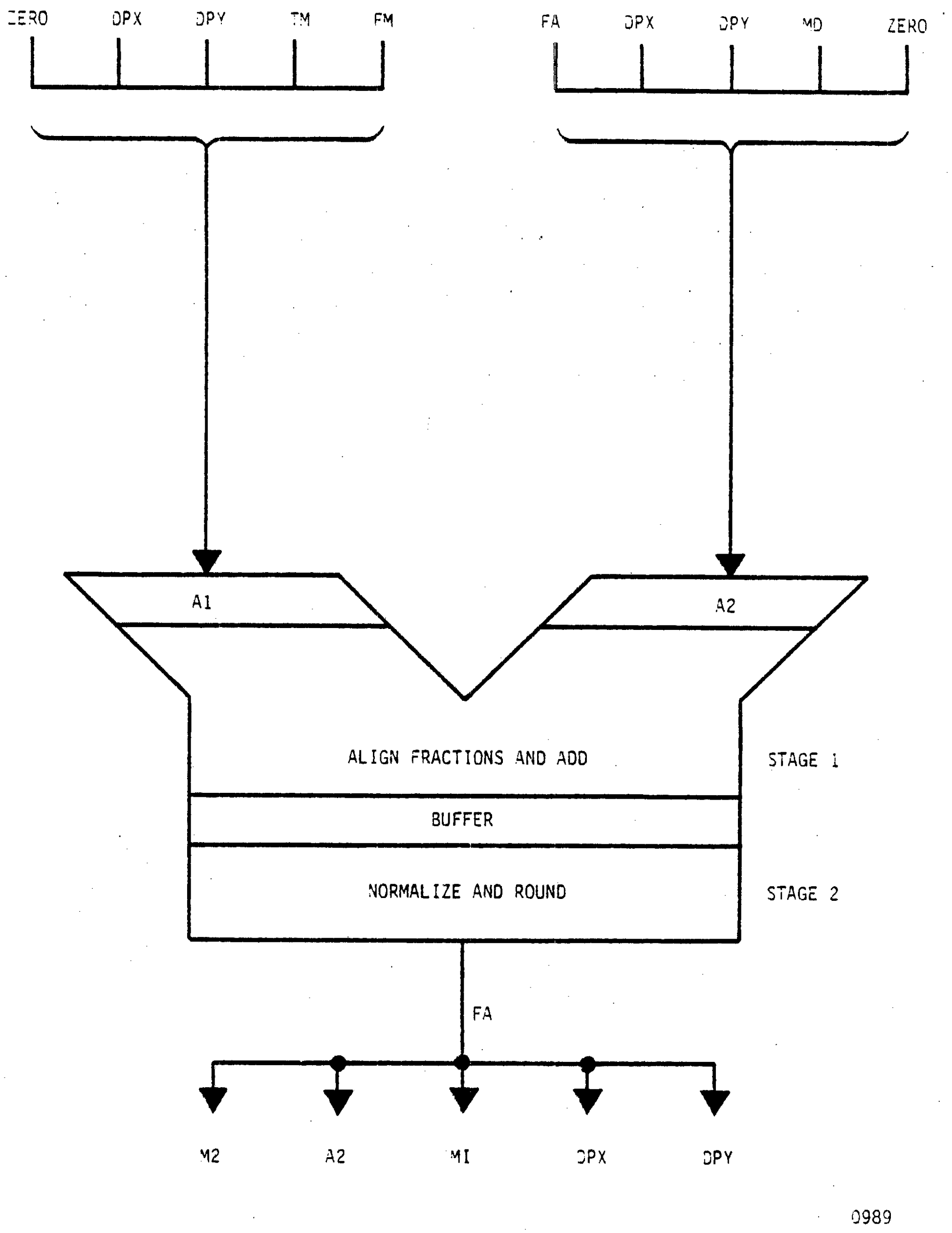


Figure 2-3 Floating-Point Adder Unit

2.5 FLOATING-POINT MULTIPLIER UNIT

The floating multiplier, as illustrated in Figure 2-4, forms the product of the two multiplier input registers (M1 and M2). The product is formed in three stages, each of which takes one machine cycle.

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

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

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

M1 can be loaded from data pad (DPX or DPY), from the output of the floating multiplier (FM), or from table memory (TM). M2 is loaded from data pad (DPX or DPY), from the adder (A1), from the multiplier (M1), or from the main data memory (MD).

Two error bits in the AP status register are affected by the floating multiplier:

FO	Set if exponent overflow occurred. The result is forced to the signed maximum value.
FU	Set if exponent underflow occurred. The result is forced to zero.

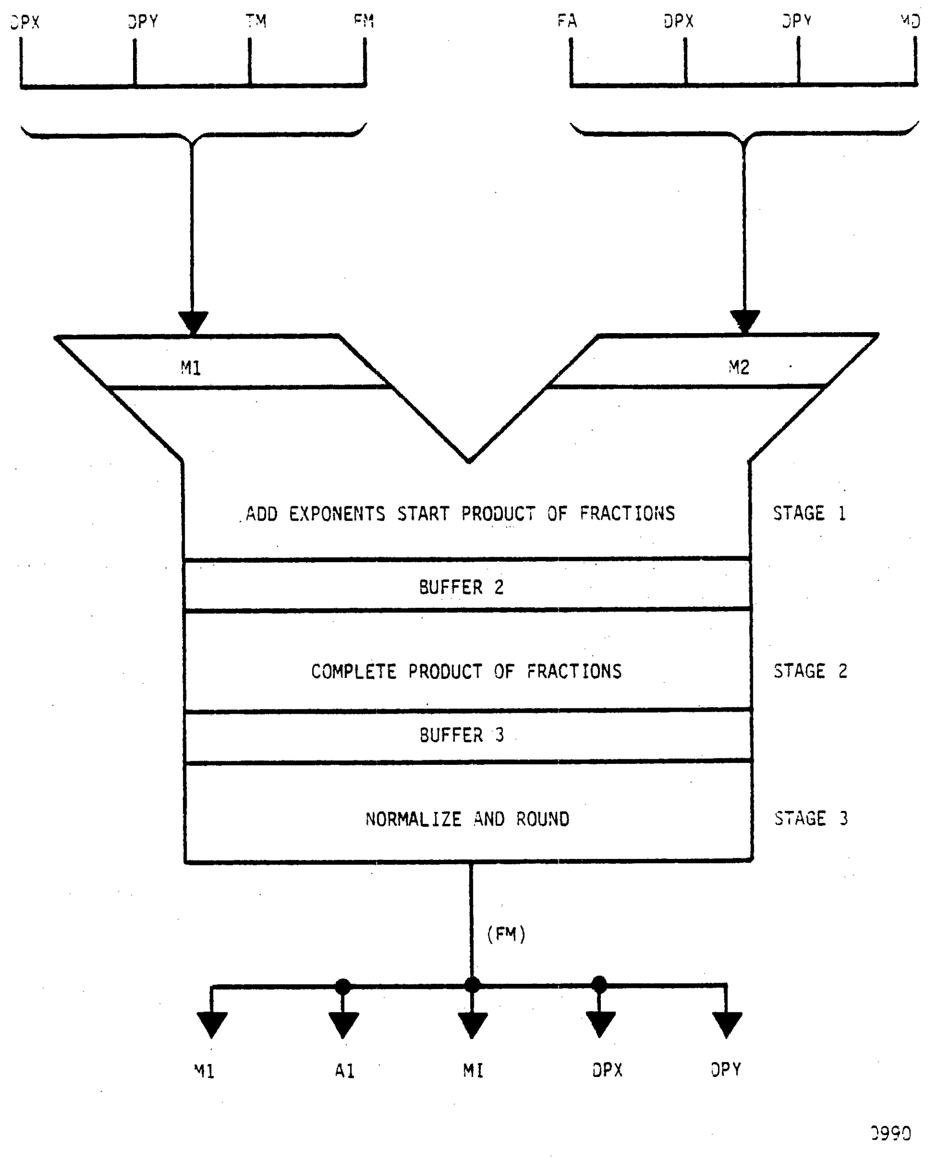


Figure 2-4 Floating Multiplier

2.6 DATA PAD UNIT

Data pad, illustrated in Figure 2-5, consists of two fast accumulator blocks (each with 32 floating-point locations) called data pad X (DPX) and data pad Y (DPY). In a single-machine cycle, the contents of one location from each data pad can be read out and used. In addition, data can also be stored into one location in each data pad in the same cycle. For example, in a single instruction (167ns), a multiply can be initiated specifying one argument from DPX and another from DPY; an adder result (FA) can be stored into a DPX location, and a data element in main data stored into a DPY location. On the very next instruction, similar multiple data pad accessing could be accomplished again.

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

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

Data from either data pad can be used by the multiplier (M1, M2), adder (A1, A2), or memory input (MI). Data can be stored into data pad from the adder (FA), multiplier (FM), s-pad function output (SPFN), command buffer value (VALUE), or from data pad (DP).

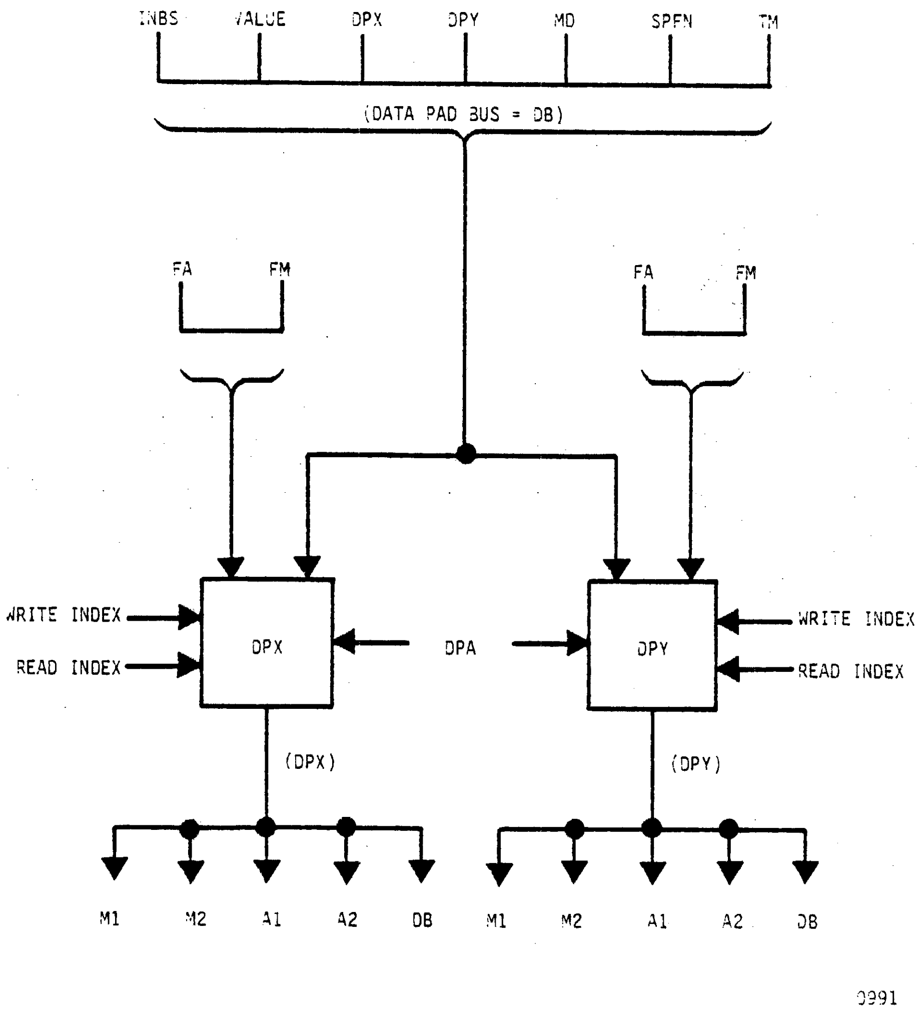


Figure 2-5 Data Pad

2.7 DATA MEMORY UNIT

The data memory unit, as illustrated in Figure 2-6, is the primary data store for the AP. It is available in 38-bit wide 8K modules which have an interleaved cycle time of 333ns (for the standard memory) and 167ns (for the fast memory).

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

In referencing memory for read or write operations, the selected operation is initiated by making a change to the memory address (MA) register. The MA register can be loaded from the s-pad (SPFN) or its contents incremented or decremented by one.

A write operation is specified by loading MI with the data to be written during the same instruction in which MA is changed. This data is then written into memory from MI during the next two AP cycles. Data can be loaded into MI from the floating adder (FA), floating multiplier (FM), data pad (DP), main data memory (MD), table memory (TM), the input bus (INBS), s-pad function (SPFN), or the command buffer value (VALUE). A memory operation can be initiated every other cycle. The intervening cycle can be used for any other AP function except another memory initiate.

When a memory read is initiated, the requested memory data is placed by the memory controller into the main data memory (MD) register three cycles after the reqest is made. Two instructions after the read request, another memory operation can be initiated. Again, the intervening cycle can be used for any non-memory function. Data in MD can be used by the floating adder (A2), floating multiplier (M2), or data pad (DP).

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

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

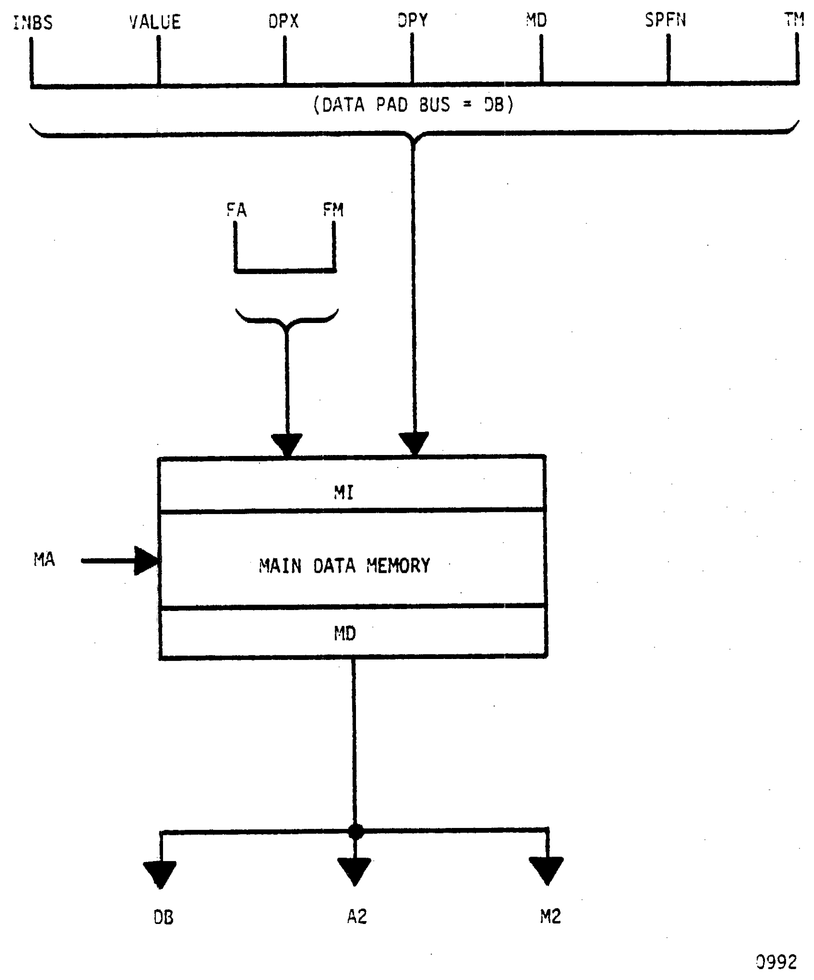


Figure 2-6 Data Memory Unit

2.8 TABLE MEMORY UNIT

The repeated use of standard constants (such as complex roots of unity and transcendental values) in signal processing routines dictates their ready availability to the programmer. A separate table memory, as illustrated in Figure 2-7, eliminates memory accessing conflicts by allowing data values (constants) to be placed in separate memory banks.

Values read from table memory are placed by the controller into the table memory buffer register. The table memory address (TMA) register serves as a pointer to the desired location.

A table memory read is initiated by changing the contents of TMA either by loading a value from the s-pad (SPFN) or by incrementing or decrementing the contents of TMA.

A new table value may be requested every machine cycle. This value is available for use two cycles later. The value can be used by the floating adder (A1), floating multiplier (M1), or data pad (DP).

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

Refer to Programmer's Reference Manual Part One (FPS 860-7319-000) for information on TMRAM.

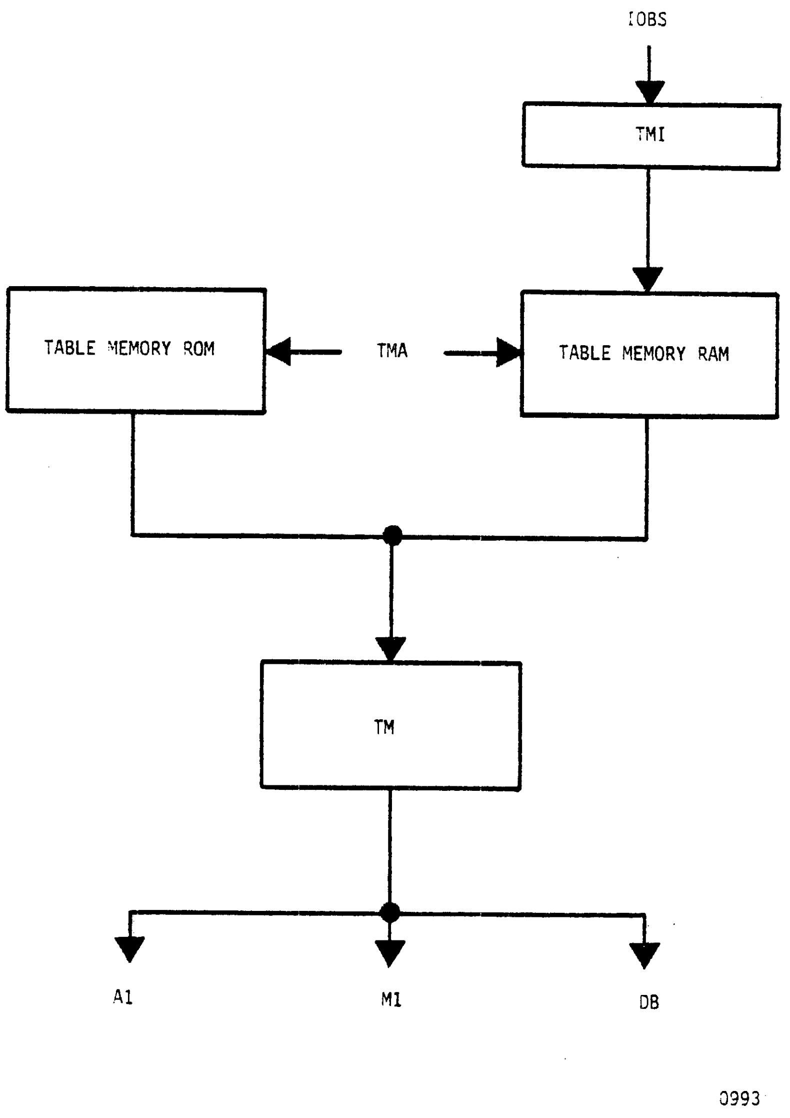
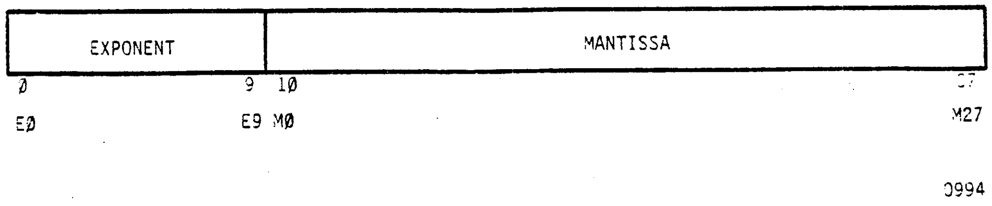


Figure 2-7 Table Memory

2.9 INTERNAL FLOATING-POINT FORMAT

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



where:

mantissa	28-bit 2's complement fraction
exponent	10-bit binary exponent, biased by 512

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

mantissa * 2 (exponent -512)

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

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

The convergent rounding hardware rounds up when the magnitude of the remainder is greater than one-half of the least significant bit of the mantissa. This serves to minimize truncation errors in long series of arithmetic calculations.

Format conversion between host format and AP format occurs in the interface and in the floating adder unit. The dynamic range of the internal format is large enough to accommodate IBM 360 format and other host formats. The extended precision of the AP internal format ensures that accuracy is maintained during critical stages of data analysis.

CHAPTER 3

PROGRAMMING CONSIDERATIONS

3.1 INTRODUCTION

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

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

3.2 FLOATING-POINT ADDER

The following sections describe the floating-point adder.

3.2.1 FLOATING ADDER OPERATIONS

Floating adder operations are initiated by the following instructions:

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-point number to fixed integer.
FSM2C	A2	Convert A2, signed magnitude to 2′s complement.
F2CSM	A2	Convert A2, 2′s complement to signed magnitude.
FSCALE	A2	Scale A2.

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

A1:	FM	floating multiplier result
	DPX	data pad X accumulator
	DPY	data pad Y accumulator
	TM	last data read from table memory
	ZERO	floating-point zero
A2:	FA	floating adder result
	DPX	
	DPY	
	MD	last data read from data memory
	ZERO	

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

```
FADD DPX, DPY            "DPX+DPY
```

To subtract a number read out of data memory from a constant in table memory:

```
FSUB TM,MD               "TM-MD
```

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

```
FSUBR TM,MD              "MD-TM
```

subtracts a constant from table memory from a number in data memory.

To negate a number from DPX:

```
FSUB ZERO, DPX           "0.0 - DPX = -DPX
```

To take the absolute value of a number from data memory:

```
FABS MD                  "ABS(MD)
```

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

```
FIX DPY                  "FIX (DPY)
```

3.2.2 ADDER PIPELINE

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

Example 1 illustrates how the adder pipeline works, where A,B,...,G,H are floating-point numbers to be added.

Example 1

TIME	CYCLE	INSTRUCTION	ADDER PIPELINE		ADDER RESULT (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

0995

The FADD without arguments in cycle 5 is used only to push the last computation into the buffer register and hence to the end of the pipeline. Thus, it is a dummy add because the results are unimportant and are never used. In Example 1, the floating-point adds are completed in one microsecond. During cycles 2 through 4, when the pipeline is full, adds are done every 167ns, the maximum rate. The completed results as they come out of the adder pipeline are referred to by the mnemonic FA. FA is dynamic in the sense that it must be used or stored elsewhere before being changed by the next floating adder instruction. The programmer, however, has complete control over the pipeline. Arguments advance only when pushed through the pipeline by floating adder instructions.

3.2.3 AN EXAMPLE

A complete computational sequence to do the vector sum $A_i=A_i+B_i$, i=0,1,2,3, is shown in Example 2. A_i is stored in data pad X locations 0-3, and B_i is stored in data pad Y location 0 through 3.

<u>Example 2</u>

```
1.  FADD DPX(0),DPY(0)                  "Do A0+B0

2.  FADD DPX(1),DPY(1)                  "Do A1+ B1

3.  FADD DPX(2),DPY(2); DPX(0)<FA       "Do A2+B2, A0+B0 is now
                                         done, save it in A0

4.  FADD DPX(3),DPY(3); DPX(1)<FA       "Do A3+B3, A1+B1 is now
                                         done, save it in A1

5.  FADD; DPX(2)<FA                     "Push Adder; save A2+B2 in A2

6.  DPX(3)<FA                           "Save A3+ B3 in A3
```

Example 3 is a chart of this computation showing the state of the adder pipeline and data pad after each instruction is executed.

Example 3

CYCLE	ADDER PINPELINE		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

0997

3.2.4 FLOATING ADDER TESTS

Table 3-1 lists the conditional branches that test the floating adder result (FA):

Table 3-1 Floating Adder Tests

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

1059

The branches test FA one instruction cycle after it is ready for use. That is, an adder result may be tested one cycle after it comes out of the adder pipeline. This is shown in Example 4.

<u>Example 4</u>

```
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)
                                                            0998
```

Compound tests may also be made. Test MD to see if it is between a lower limit contained in DPX (1) and an upper limit in DPX (2) (i.e., see if $DPX(1) \leq MD \leq DPX(2)$). This is shown in Example 5.

<u>Example 5</u>

```
1.  FSUBR DPX(2),MD              "Do MD-DPX(2)

2.  FSUB DPX(1),MD               "Do DPX(1)-MD

3.  FADD                         "Push first test result out

4.  BFGT BIG                     "Was too big

5.  BFGT SMALL                   "Was too small

6.  . . .                        "OK
                                                            0999
```

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

3.2.5 FLOATING-POINT LOGICAL OPERATIONS

Instructions FAND, FOR, and FEQV perform logical operations on floating-point numbers. Exponent alignment occurs as for a normal floating-point add. The two mantissas are then combined using the specified logical operation. The result is then normalized and rounded.

3.3 FLOATING-POINT MULTIPLIER

The following sections describe the floating-point multiplier.

3.3.1 MULTIPLY INSTRUCTION

Floating-point multiplies are initiated by the following instruction:

```
        FMUL M1,M2
```

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

M1	FM	floating multiplier result
	DPX	data pad X accumulator
	DPY	data pad Y accumulator
	TM	last data read from table memory
M2	FA	floating adder result
	DPX	
	DPY	
	MD	last data read from data memory

Thus, any of the data sources listed under M1 can be multiplied by any of the data sources in M2. For example, to multiply a number read from data memory by a constant from table memory:

```
        FMUL TM,MD      "TM * MD
```

or, to multiply a number in data pad X by another number in data pad Y:

```
        FMUL DPX,DPY    "DPX * DPY
```

3.3.2 MULTIPLIER PIPELINE

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

The instruction sequence shown in Example 6 illustrates how the multiplier pipeline works where A,B,...,G,H are floating-point numbers to be multiplied together.

Example 6

TIME	CYCLE	INSTRUCTION	MULTIPLIER PIPELINE			MULTIPLIER RESULT (FM)
			M1,M2	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

1000

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

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

3.3.3 AN EXAMPLE

A computation example to square the elements in a vector is shown in Example 7.

Example 7

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

1. FMUL DPX(0),DPSX(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

1001

Example 8 illustrates this computation showing the state of the multiplier pipeline and data pad X after each instruction is executed.

Example 8

CYCLE	MULTIPLIER PIPELINE			MULTIPLIER RESULT (FM)	DATA PAD X			
	M1,M2	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

1002

3.3.4 MULTIPLY-ADDS

The full floating-point computational power of the AP is utilized when a process involving both multiplies and adds is considered. The dot product of two eight-element vectors $A_i \bullet B_i = \Sigma A_i B_i$, $i = -4, -3, \ldots, 1, 2, 3$, where A_i is in Data Pad X and B_i is in Data Pad Y, is formed in Example 9.

Example 9

```
Fill the            1.  FMUL DPX(-4),DPY(-4)            "Do A-4B-4
Multiplier          2.  FMUL DPX(-3),DPY(-3)            "Do A-3B-3
Pipeline            3.  FMUL DPX(-2),DPY(-2)            "Do A-2B-2

                    4.  FMUL DPX(-1),DPY(-1);           "Do A-1B-1.  A-4B-4 is
                        FADD FM,ZERO                    "  now done, save it in
Fill the                                                "  adder.
Adder               5.  FMUL DPX(0),DPY(0);             "Do A0B0. A-3B-3 is now
Pipeline                FADD FM,ZERO                    "  done, save it in the
                                                        "  adder.

                    6.  FMUL DPX(1),DPY(1);             "Do A1B1.  A-2B-2 is now
                        FADD FM,FA                      "  coming out of the multiplier,
                                                        "  and A-4B-4from the adder, add
Both                                                    "  them together.
Pipelines           7.  FMUL DPX(2),DPY(2),DPY(2);      "Do A2B2.  A-1B-1 is now coming
full                    FADD FM,FA                      "  out of the multiplier, and
                                                        "  A-3B-3 from the adder, add
                                                        "  them together.
                    8.  FMUL DPX(3),DPY(3);             "Do A3B3.  A0B0 is now coming
                        FADD FM,FA                      "  out of the multiplier, and
                                                        "  (A-4B-4 + A-2B-2) from the
                                                        "  adder, add them together.
                    9.  FMUL; FADD FM,FA                "  A1B1 is coming out of the
                                                        "  multiplier, and (A-3B-3
                                                        "  +A-1B-1) from the adder,
Empty the                                               "  add them together.
Multiplier          10. FMUL; FADD FM,FA                "A2B2 is coming out of the
Pipeline                                                "  multiplier, and (A-4B-4
                                                        "  +A-2B-2+A0B0) from the
                                                        "  adder, add them together.
                    11. FADD FM,FA                      "A3B3 is coming out of the
                                                        "  multiplier, and
                                                        "  (A-3B-3+A-1B-1+A1B1)
                                                        "  from the adder, add
                                                        "  them together.
                    12. FADD; DPX(3)<FA                 "(A-4B-4+A-2B-2+A0B0+A2B2)
                                                        "  is coming out of the
Empty                                                   "  adder, save it in DPX(3).
the Adder           13. FADD DPX(3),FA                  "(A-3B-3+A-1B-1+A1B1+A3B3)
Pipeline                                                "  is coming out of the
                                                        "  adder, add it to
                                                        "  (A-4B-4+A-2B-2+A0B0+A2B2)
                                                        "  which was saved in DPX(3).
                    14. FADD                            "Push result out of Adder
                    15. DPX(3)<FA                       "The result:  (A-4B-4+
                                                        "  A-3B-3+A-2B-2+A-1B-1+
                                                        "  A0B0+A1B1+A2B2+A3B3),
                                                        "  Saved in DPX(3).
```

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

Example 10 summarizes the computation as a further aid in understanding the multiply add interaction in the sum-of-products computation of Example 9.

Example 10

CYCLE	MULTIPLIER		ADDER:		DATA PAD: 3
	M1, M2	FM	A1, A2	FA	
1.	A_{-4}, B_{-4}	---	---	---	---
2.	A_{-3}, B_{-3}	---	---	---	---
3.	A_{-2}, B_{-2}	---	---	---	---
4.	A_{-1}, B_{-1}	$A_{-4} * B_{-4}$	$A_{-4} B_{-4}$, Ø.Ø	---	---
5.	A_0, B_0	$A_{-3} * B_{-3}$	$A_{-3} B_{-3}$, Ø.Ø	---	---
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	$A_0 * A_0$	$A_0 B_0$, ES_2	ES_2	---
9.	---	$A_1 * A_1$	$A_1 B_1$, OS_2	OS_2	---
10.	---	$A_2 * A_2$	$A_2 B_2$, ES_3	ES_3	---
11.	---	$A_3 * A_3$	$A_3 B_3$, OS_3	OS_3	---
12.	---	---	---	ES_4	ES_4
13.	---	---	OS_4, ES_4	OS_4	ES_4
14.	---	---	---	---	ES_4
15.	---	---	---	OS_4+ES_4	OS_4+ES_4

NOTE

ES is n terms of the even term Sum: $A_i B_i$, i = -4, -2, Ø, 2

OS is n terms of the odd term Sum: $A_i B_i$, i = -3, -1, 1, 3

1004

3.4 DATA PAD

The following sections describe the data pad.

3.4.1 DATA PAD ADDRESSING

Data pad is a block of 64 high-speed accumulators used to store intermediate results during a computation. In any given AP instruction, the programmer has 16 of the data pad accumulators to work with; eight in data pad X and eight in data pad Y. They are addressed relative to the current value of the data pad address register which functions as a base register for data pad. For example, if DPA has a value of 24_8, locations 20_8 through 27_8 would be available for use. This is illustrated in Figure 3-1.

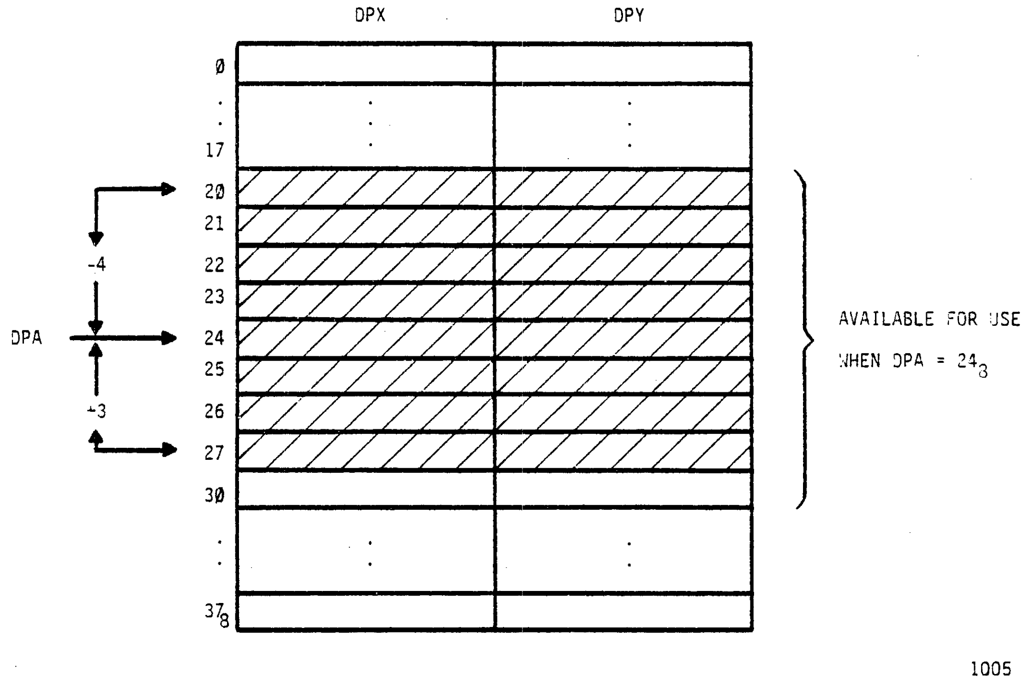


Figure 3-1 Data Pad Address

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

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

Four separate displacements are provided, one each for reading and writing DPX and DPY. Thus, four separate locations in data pad may be used in a given instruction. With DPA=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 add DPX location 27 to the last data read from data memory, multiply the last data read from table memory by the contents of DPY location 22, store the results of a previous add into DPX location 21, and store the results of a previous multiply into DPY location 25.

All 64 locations of data pad are accessed by changing the DPA pointer:

```
INCDPA        "Increments DPA by 1
DECDPA        "Decrements DPA by 1
SETDPA        "Loads DPA with the current S-PAD
              "function (SPFN, refer to section 3.7)
```

Changes in DPA take effect the next instruction after they occur (i.e., if DPA=24):

```
FADD DPX(0),DPY(0);     INCDPA    "DPA is still 24 so
                                  "DPX24 is added to
                                  "DPY24

FADD DPX(0),DPY(0);     INCDPA    "Now DPA=25, so
                                  "DPX25 is added to DPY25

FADD DPX(0),DPY(0)                "Now DPA=26, so
                                  "DPX26 is added to
                                  "DPY26
```

Thus, by successively incrementing DPA, the data pad can be used as a queue; or by properly incrementing and decrementing DPA, the data pad can be used as a stack. Data pad address is circular. That is, with successive increments of DPA the next location after 37_8 is 0; with successive decrements of DPA the next location after 0 is 37_8.

3.4.2 WRITING INTO DATA PAD

Data may be stored into DPX and DPY from FA, FM, or DB (the 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<FM
        DPY<DB
```

The following may be selected onto the data pad bus (DB):

```
        DB=ZERO    "Floating-point zero
        DB=INBS    "Input Bus
        DB=VALUE   "A 16-bit immediate value
        DB=DPX     "DPX
        DB=DPY     "DPY
        DB=MD      "Last data read from data memory
        DB=SPFN    "S-pad function (16-bit integer)
        DB=TM      "Last data read from table memory
```

Thus, if DPA=24_8, the following instruction is possible:

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

This stores the current adder result into DPX location 27 and stores the last data read from the main data memory into DPY location 22 via the data pad bus.

3.4.3 DATA PAD BUS

Data to be stored into DPX and DPY can be moved through three pathways: FM, FA, and DB. While FM and FA are fixed in meaning (output from the floating multiplier and adder, respectively), the data pad bus (DB) pathway can be connected to any one of eight possibilities depending upon the programmer's choice.

Examples:

- MD is put into both DPX and DPY:

```
DPX<DB; DPY<DB; DB=MD
```

  MD is put onto the data pad bus, and the data pad bus is stored into DPX and DPY.

- MD is put into DPX and TM into DPY:

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

  This is an error. Only one choice at a time can be made for the data pad bus. This double transfer would take two separate instructions to accomplish.

- FA is stored into DPX and MD into DPY:

```
DPX<FA; DPY<DB; DB=MD
```

  MD is put onto the data pad bus in order to get it into DPY. FA goes directly into DPX.

To simplify notation, data transfers invloving data pad bus can be written in a shorthand manner.

<u>shorthand</u>	<u>longhand</u>
DPX<MD; DPY<MD	DPX<DB; DPY<DB; DB=MD
DPX<MD; DPY<TM	DPX,DB; DB=MD; DPY,DB; DB=TM
(<u>still an error no matter how it is written</u>)	
DPX<FA; DPY<MD	DPX<FA; DPY<DB; DB=MD

In the shorthand notation, choices for the data pad bus are not explicitly indicated. Transfers are written as if there were a direct connection between the source and destination while in fact it is the data pad bus which does the connecting. Remember, however, that the programmer is still making a data pad bus choice and only one choice is allowed per instruction. Errors like the one shown above (where two data pad bus choices are attempted) are detected and flagged by the assembler.

3.5 DATA MEMORY

The following sections describe data memory.

3.5.1 MEMORY ADDRESSING

Main data memory cycles are initiated by changing the memory address register which points the memory location to be read from or written into:

INCMA	"Increment MA by 1
DECMA	"Decrement MA by 1
SETMA	"function (SPFN, refer to section 3.7)

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

3.5.2 DATA MEMORY READS

Data read from memory is available for use three instruction cycles after the read is initiated. The instruction sequence shown in Example 11 illustrates how memory data is accessed: A, B, and C are floating-point numbers in memory locations 101, 102, and 103, respectively. It is assumed that MA is set to 100 before starting.

Example 11

TIME	AP CYCLE	INSTRUCTION	MEMORY ADDRESS (MA)	MEMORY DATA RESULT (MD)
0	1.	INCMA	101	---
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

1006

Three AP cycles after a given memory location is read, data from that location is ready in the memory data register and available for use. MD may be used by the adder or the multiplier as follows:

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

It can also be placed on the data pad bus and stored in data pad or back into memory as follows:

```
          DPX(2)<MD              "store MD into DPX.
```

3.5.3 AN EXAMPLE

Example 12 loads a vector A_i, i=0,1,2 stored in memory locations 101, 102, and 103 into DPX locations 10, 11, and 12. It is assumed that MA is set to 100 and DPA is set to 10 before starting.

<u>Example 12</u>

```
1.  INCMA                  "Fetch A0 from memory

2.  --

3.  INCMA                  "Fetch A1 from memory

4.  DPX<MD; INCDPA         "Store A0 into DPX location 10
                           "  and bump DPA pointer to 11

5.  INCMA;                 "Fetch A2 from memory

6.  DPX<MD; INCDPA         "Store A1 into DPX location 11
                           "  and bump DPA pointer to 12

7.  --

8.  DPX<MD                 "Store A2 into DPX location 12
```

1007

Example 13 illustrates the transfer of Example 12 showing the state of each component after each instruction.

Example 13

CYCLE	MEMORY		DATA PAD			
	MA	MD	DPA	DPX$_{10}$	DPX$_{11}$	DPX$_{12}$
1.	101	---	10	---	---	---
2.	101	---	10	---	---	---
3.	102	---	10	---	---	---
4.	102	A_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

1008

3.5.4 DATA MEMORY WRITES

Data memory write cycles are indicated by the following:

MI<FA	"write the adder result into memory
MI<FM	"write the multiplier result into memory
MI<DB	"write data pad bus into memory

These instructions load data into the memory input buffer register from where it is written into memory. Data may be written into sequential memory locations by every other AP instruction.

3.5.5 AN EXAMPLE

Example 14 squares the elements of a vector A_i, i=0,1,2, in DPX locations 10, 11, and 12 and stores the results into data memory locations 101, 102, and 103. It is assumed that MA is set to 100 and DPA is set to 10 before starting.

<u>Example 14</u>

1.	FMUL 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 location 102
7.	FMUL	"Dummy FMUL to empty pipeline
8.	MI<FM; INCMA	"Write A_2^2 into memory location 103

1009

Example 15 illustrates the sequential data memory write computation.

<u>Example 15</u>

CYCLE	DPA	MULTIPLIER		MEMORY	
		M1,M2	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

1010

3.5.6 MEMORY INTERLEAVE

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

Table 3-2 Memory Interleave Sequence

MEMORY ADDRESS SEQUENCE (OCTAL)	MEMORY BANK SEQUENCE	MEMORY REFERENCE TIMING
1Ø1, 1Ø2, 1Ø3, 1Ø4, ...	1, Ø, 1, Ø, ...	every 2 AP cycles
166, 165, 164, 163, ...	Ø, 1, Ø, 1, ...	every 2 AP cycles
1ØØ, 1Ø2, 1Ø4, 1Ø6, ...	Ø, Ø, Ø, Ø, ...	every 3 AP cycles
233, 1Ø374, 234, 1Ø376, ...	1, 2, Ø, 2, ...	every 2 AP cycles

1011

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

3.5.7 MEMORY LOCKOUT

If memory references are made too rapidly for memory to handle, the CPU suspends program execution and spins until the memory is no longer busy. Thus, suppose the following were coded:

```
1. INCMA                        "referencing memory every cycle

2. INCMA

3. INCMA
```

The following execution is the result:

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

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

3.6 TABLE MEMORY

The following sections describe table memory.

3.6.1 TABLE MEMORY ADDRESSING

Constants stored in table memory are read by setting the table memory address (TMA) register to the address of the desired table memory location. This is done with the following instructions:

```
INCTMA        "increments TMA by 1

DECTMA        "decrements TMA by 1

SETTMA        "set TMA to the current s-pad
              "function (SPFN)
```

Each of the above initiates a fetch from the table memory location pointed at by the new contents of TMA. Two AP cycles later, the contents of the desired locations are available for use. A new location can be fetched every AP cycle. The sequence in Example 16 illustrates how table memory is accessed. K0, K1, and K2 are constants stored in table memory location 235, 236, and 237. It is assumed that TMA is set to 234 before starting.

Example 16

TIME	AP CYCLE	INSTRUCTION	TABLE MEMORY ADDRESS (TMA)	TABLE MEMORY RESULT (TM)
0	1.	INCTMA	235	---
167ns	2.	INCTMA	236	---
333ns	3.	INCTMA	237	K0
500ns	4.	---	237	K1
667ns	5.	---	237	K2

1012

Two cycles after a given table memory location is fetched, the data is ready in the table memory data register and is available for use. TM can be used by the adder or the multiplier:

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

or put on the data pad bus and stored into data pad:

```
        DPX(-1)<TM                            "store TM into DPX
```

3.6.2 AN EXAMPLE

Example 17 forms the vector sum $A_i = B_i+K_i$, $i=0,1,2$, where A_i is in DPX locations 10-12, B_i is in DPY 10-12, and K_i is a series of constants stored in table memory location 235-237. A_i is stored back into DPX. It is assumed that DPA is set to 10 and TMA is set to 234 before starting.

Example 17

```
1.  INCTMA                                  "Fetch K0

2.  INCTMA                                  "Fetch K1

3.  INCTMA; FADD TM,DPY; INCDPA             "Do K0 + B0, bump DPA to 11

4.  FADD TM,DPY; INCDPA                     "Do K1 + B1, bump DPA to 12

5.  FADD TM,DPX (0); DPX(-2)<FA             "Do K2 + B2, store A in DPX10

6.  FADD:  DPS(-1)<FA                       "Store A1 in DPX11

7.  DPX(0)<FA                               "Store A2 in DPX12
```

1013

Example 18 illustrates the computations of Example 17.

Example 18

CYCLE	TABLE MEMORY		ADDER		DATA PAD X			
	TMA	TM	A1,A2	FA	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

1014

3.6.3 A COMPLEX MULTIPLY

An example using both memories, a complex multiply from the FFT (fast fourier transform) algorithm, is shown in Example 19. The multiply is between a complex signal point held in data memory and a complex exponential value (a root of unity, e^{i0}) fetched from table memory. The computation is:

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

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

Where C is the data point and W is the complex exponential, R and I denote real and imaginary parts, respectively. C is in main data memory, and W is in table memory.

Example 19

Fetch the	1.	INCMA	"Fetch C_R from data memory
four 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 two	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

1015

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

Example 20 summarizes the complex multiply.

Example 20

CYCLE	MEMORIES		MULTIPLIER		ADDER		DATA PAD	
	TM	MD	M1,M2	FM	A1,A2	FA	0	1
1.	--	--	--	--	--	--	--	--
2.	--	--	--	--	--	--	--	--
3.	--	--	--	--	--	--	--	--
4.	W_R	C_R	W_R,C_R	--	--	--	--	--
5.	W_I	C_R	W_I,C_R	--	--	--	--	--
6.	W_I	C_I	W_I,C_I	--	--	--	--	--
7.	W_R	C_I	W_R,C_I	W_R*C_R	--	--	W_RC_R	--
8.	--	--	--	W_I*C_R	--	--	W_RC_R	W_IC_R
9.	--	--	--	W_I*C_I	W_IC_I,W_RC_R	--	W_RC_R	W_IC_R
10.	--	--	--	W_R*C_I	W_RC_I,W_IC_R	X_R	W_RC_R	W_IC_R
11.	--	--	--	--	--	X_I	X_R	W_IC_R
12.	--	--	--	--	--	--	X_R	X_I

1016

3.7 S-PAD

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

3.7.1 SINGLE OPERAND INSTRUCTIONS

Table 3-3 lists the single operand instructions. One item can be chosen from each column.

Table 3-3 Single Operand Instructions

OPERATION	SHIFT	NO LOAD	DESTINATION REGISTER
INC	---	---	dst;
DEC	R	#	
COM	L		
CLR	RR		

1017

The operation is performed upon the contents of the destination register (DST), and that result is shifted. The shifted result is stored in the destination register unless a no load (#) is specified. The shifted result is the s-pad function (SPFN), which may be stored into an address register (MA, TMA, or DPA) or placed onto the data pad bus (DB=SPFN). Some examples where SP_n refers to the contents of s-pad register "n" are illustrated in Example 21.

Example 21

```
INC 6                    "(SP6+1)→SP6

DECR 3                   "(SP3-1)/2→SP3

COM 3; DPX<SPFN          "SP3→SP3→DPX

CLR# 2; SETDPA           "0→DPA; because of # (no load)
                            SP2 remains unchanged
```

1019

3.7.2 DOUBLE OPERAND INSTRUCTIONS

Table 3-4 lists the double operand instructions. One item can be chosen from each column.

Table 3-4 Double Operand Instructions

OPERATION	SHIFT	NO LOAD	DECIMATE	SOURCE REGISTER	DESTINATION REGISTER
MOV	---	---	---	src,	dst,
ADD	R	#	&		
SUB	L				
AND	RR				
OR					
EQV					

1018

The operation is performed between the source (SRC) and destination (DST) registers. If bit reverse (X) is specified, the contents of source are bit-reversed before being used. The shift is performed on the result which is then stored into the destination register unless no load (#) is specified. The shifted result is the s-pad function (SPFN), which may be stored into TMA, MA, or DPA or placed onto the data pad bus.

Example 22

```
MOV 3,15                   "SP3→SP15

ADDL 6,10; SETMA           "((SP10) + (SP5)) * 2; SP10→MA

SUB 7,13                   "(SP13-SP7) SP13

AND#5,11; SETDPA           "(SP11 AND SP5)→DPA

OR# &6,7; SETTMA           "(SP7 OR SP6 (Bit-reversed))→TMA

MOVRR 2,2                  "(SP2)/4→SP2
```

1020

For purposes of program clarity, the assembler allows names to be given to the s-pad registers. If register PTR is a pointer to an array in data memory, and register STEP contains the increment value used to step through the array, then the following instruction word advances the array pointer by the proper increment and fetches the next array element from memory:

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

3.7.3 S-PAD TEST

The following conditional branches test the s-pad function:

```
        BR LOOP          "branch unconditionally to program
                         "location "LOOP"
        BEQ LOOP         "branch if SPFN=0
        BNE LOOP         "branch if SPFN≠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.

An example of loop counting is shown in Example 23.

Example 23

```
        DEC 2            "decrement SP2
        BNE LOOP         "branch to "LOOP if SP2 has not
                         "yet reached zero
```

Example 24 tests the contents of SP_3 to see if it is between a lower limit contained in SP_2 and an upper limit in SP_4 (i.e., if $SP_2<SP_3<SP_4$.

Example 24

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

The branches are made relative to the current program source address with a 5-bit displacement value. This means that the branch target address must be within -20_8 to +17 locations of the current instruction.

3.7.4 AN EXAMPLE

Example 25 loads data pad X with an array A, with N elements starting at main data memory location 3721_8. CTR is in s-pad register which is used as a counter.

Example 25

```
1.              CLR# CTR: SETDPA          "Set DPA to 0

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, Ai+1

5.              DPX<MD;                   "Store Ai into DPXi, advance
                  INCDPA: BNE LOOP        "DPA and test counter
```

1021

Example 26 shows the loop in Example 25 for the N=3 elements.

Example 26

INSTRUCTION NUMBER	MEMORY		DATA PAD				S-PAD "CTR" TEST	
	MA	MI	DPA	0	1	2		
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

1022

A generalization on the previous example to fetch array A from every Kth memory location is shown in Example 27. The increment is stored in s-pad register STEP, and the array pointer is stored in PTR.

Example 27

```
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, A0

4.         LDSPI CTR: DB=N                "Initialize "CTR" to N

5.  LOOP:  ADD STEP,PTR: SETMA            "Advance memory pointer.  Fetch
             BEQ DONE                     "  next element, Ai+1.  Test
                                          "  counter and jump out if
                                          "  done.

6.         DPX<MD; INCDPA                 "Store Ai into DPXi, advance DPA
           DEC CTR: BR LOOP               "  Decrement "CTR" and jump
                                          "  back to LOOP.
                                                                   1023
7.  DONE:  --
```

CHAPTER 4

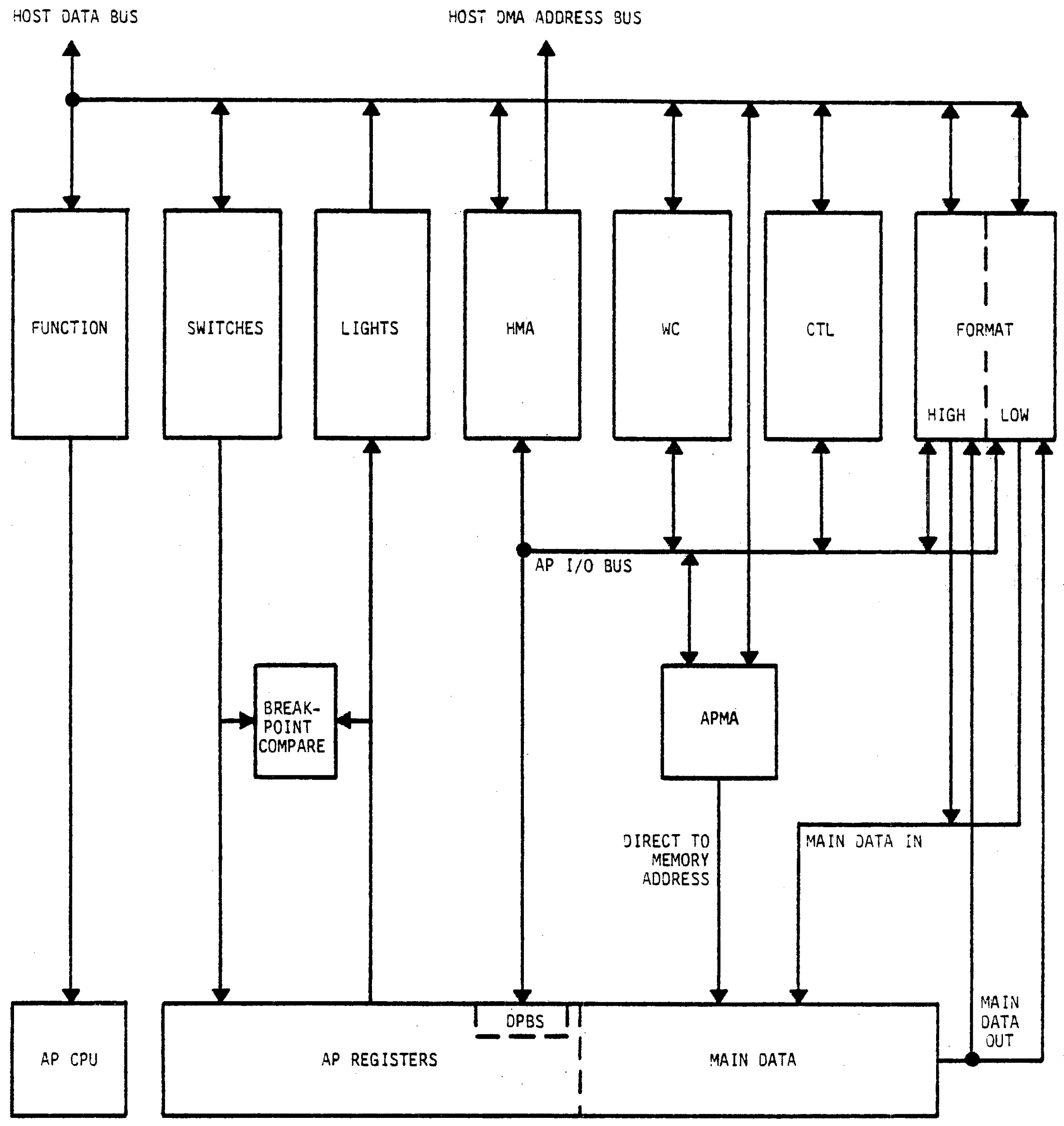
INTERFACE

4.1 INTRODUCTION

This chapter describes the interface between the host computer and the AP. The interface is composed of two basic parts: a simulated front panel and direct memory access control. The front panel allows the host computer to examine or modify the internal AP registers, as well as provides for block transfer of data from the host computer to the AP, and vice versa.

4.2 FRONT PANEL

The AP panel is used for bootstrap operations (loading and starting programs) and for debugging user software (inserting hardware breakpoints and examining and modifying AP registers and memory). The panel consists of three 16-bit registers which are under the control of the host via the host interface. The functioning of these registers closely parallels that of the switches and lights on the console of a stand-alone computer. The host can examine and/or set these registers at any time, regardless of the state of the AP. The front panel and host interface is shown in Figure 4-1.



AP DEVICE ADDRESSES

REGISTER	DA
WC	Ø
HMA	1
CTL	2
APMA	3
FORMAT	4

1025

Figure 4-1 AP Panel and Host Interface

4.2.1 SWITCH REGISTER

The switch register (SWR) is used to enter data and addresses into the AP. The SWR can be read and written by the host computer. An executing AP program can also read the switches.

4.2.2 LIGHTS REGISTER

The lights register (LITES) simulates front-panel lights and is used to display the contents of internal AP registers. This register can only be read by the host. The executing AP program can set the lights register.

4.2.3 FUNCTION REGISTER

The function register (FN) provides front-panel control operations (start, stop, continue, etc.). It can be read or written by the host. The format of the function register is shown in Figure 4-2.

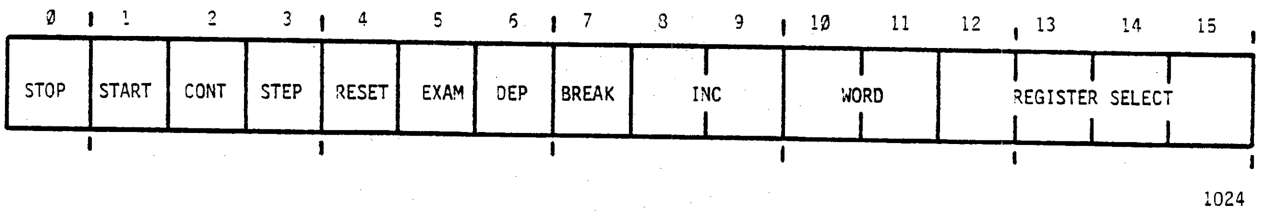


Figure 4-2 Panel Function Register Format

When the AP is running, only the STOP and RESET panel functions are valid. The other panel functions can only be exercised after the AP has halted. The panel functions are described in Table 4-1.

Table 4-1 Function Register Bits

BIT	MNEMONIC	EFFECT
Ø	STOP/HALTED	Stop AP program execution upon completion of the current instruction. When the host reads the FN register, this bit reflects the current state of the processor. This bit is set if the AP is halted. (See note.)
1	START	Start program execution at the address specified in SWR.
2	CONT	Continue program execution at the instruction pointed at by PSA (program source address).
3	STEP	Execute the instruction pointed at by PSA and then halt. Advance PSA to point to the next instruction.
4	RESET	Stop the AP immediately. Clear s-pad register Ø. Set SPFN to SP_{SPD}. Clear the AP status register. Stop the host DMA (CTL bit 15 set to Ø) and clear main data memory timing.
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.
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.
7	BREAK	Enables hardware breakpointing if PSA, MA, or TMA is specified in the register select field. The breakpoint causes the AP to halt one instruction after any instruction where the contents of the selected register was equal to the contents of the switch register. Thus, if a breakpoint is specified with PSA selected the AP halts after executing the instruction at the program location set in the switch register. PSA points to the next micro-instruction in sequence. If a breakpoint is called for on MA or TMA, the AP halts after executing the instruction following the one that referenced the trapped memory location. PSA points to the second sequential instruction after the one that caused the breakpoint. Memory breakpoints aid in debugging those elusive errors that modify memory unexpectedly.
8 & 9	INC	Increment MA, TMA, or DPA following completion of the other specified panel functions. This allows sequential memory locations to be examined or deposited into. (Refer to Table 4-2.)
1Ø & 11	WORD	Specifies which portion of a register is being examined or deposited into. (Refer to Table 4-3.)
12 - 15	REG.SELECT	Specifies which AP internal register or memory location to examine or deposit into. (Refer to Table 4-4.)

NOTE

If the current instruction performs a SPIN while waiting for I/O or memory, the STOP does not take effect until the spin condition is satisfied and the instruction completed.

1026

Table 4-2 Bits 8-9

VALUE IN BITS 8 & 9	ADDRESS REGISTER TO BE INCREMENTED
Ø	None
1	MA (Memory Address)
2	DPA (Data Pad Address)
3	TMA (Table Memory Address)

1058

Table 4-3 Bits 10-11

VALUE SET IN BITS 1Ø & 11	≤16-BIT REGISTER	38-BIT REGISTER	64-BIT REGISTER
Ø	ALL	N/A	Bits Ø-15
1	N/A	Exponent Bits ØØ-Ø9; right-justified in 16-Bit field.	Bits 16-31
2	N/A	High mantissa Bits ØØ-11; right-justified	Bits 32-47
3	N/A	Low mantissa	Bits 48-63

1027

Table 4-4 Octal Values

OCTAL VALUE SET IN BITS 12-15	REGISTER OR MEMORY SELECTED	DESCRIPTION
Ø	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 address by SPD (DEPOSIT)
6	AP STATUS	AP Internal Status Reg.
7	DA	Device Address Register
1Ø	PS_{TMA}	Program Source Memory addressed by TMA
11	IOBS	Examine I/O device output register addressed by DA
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)

1023

4.3 <u>NOTES ON THE USE OF THE FRONT PANEL AND BREAKPOINT</u>

4.3.1 WHERE DOES THE AP STOP ON A BREAKPOINT?

- With the breakpoint set on PSA, the AP stops with PSA pointing to the next instruction to be executed.

  Thus, breaking on a branch instruction and then examining PSA shows whether the branch condition is true or false.

- With the breakpoint set on TMA, the AP stops with PSA pointing to the second instruction following the one that set TMA to the break address.

- With the breakpoint set on MA, the AP stops on either the next instruction or the second instruction after the one that set MA to the break address, depending on the state of the memory lockout hardware (next instruction if memory lockout, second instruction if no memory lockout).

  Thus, the stopping point following an MA breakpoint has a one-instruction uncertainty.

4.3.2 DOES THE INSTRUCTION ON WHICH THE AP STOPS EXECUTE?

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

4.3.3 WHAT ABOUT MD TIMING AND LOCKOUT ON A BREAKPOINT IN THE MIDDLE OF AN MD MEMORY CYCLE?

- The hardware is designed so that the AP can be stopped in the middle of a memory cycle. The hardware remembers where the memory timing is when the AP stops so that the processor can continue correctly from a breakpoint that occurs during a memory cycle.

- However, the user must not examine MD nor should there be any DMA transfers going to or from MD while the AP is stopped if the user wishes to proceed from the breakpoint.

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

4.3.4 SUMMARY OF THE RULE FOR PROCEEDING FROM BREAKPOINT

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

4.3.5 WHAT ABOUT STEPPING THE AP?

The same rules for proceeding from a breakpoint apply to stepping the AP through a program. The user can examine and modify any register of memory within the constraints mentioned in section 4.3.4.

4.3.6 WHAT OTHER PITFALLS ARE THERE IN THE USE OF THE VIRTUAL FRONT PANEL?

- Note that the panel always examines SPFN, not SP_{SPD}. Thus, the user must force SPFN = SP_{SPD} to see SP_{SPD}. This can most easily be done via the panel reset function which has the side effect of also clearing SP(0).

- To examine TM, the user should first set TMA and then do a dummy panel operation (deposit TMA again, for example) in order to enter the output of table memory into the table memory buffer register. The user can then proceed to examine the addressed location using the appropriate panel functions.

- MD: setting MA from the panel initiates an MD memory read cycle. Depositing into MD from the panel initiates an MD memory write cycle.

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

- Using the increment field in the FN register: DPA and TMA always increment after the EXAM or DEP operation is complete (remember that TMA is used to address program source memory for panel operations).

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

  MA pre-increments on a DEP operation.

- The recommended procedure for starting the AP is as follows:

  1. Set the SWR to the starting address and do a deposit into PSA.

  2. Set the SWR to the desired breakpoint and do a continue to start the AP.

  This procedure has the significant advantage of placing the necessary breakpoint code into the user's program should the AP program need debugging.

  The panel START function can be used, but the user should observe the following restrictions on the first instructions executed by the AP. The first instruction should not branch, jump, or modify PSA in any way other than to advance to the next instruction. The first instruction should not use the SPEC and I/O fields. In fact, the preferred first instruction is a NOP (all zeros).

4.4 DIRECT MEMORY ACCESS

In addition to the panel function, the AP contains four 16-bit registers that are used for direct memory access (DMA) to both host and AP data memory, plus a 38-bit format conversion register that acts as a buffer between the two memories. These registers may be read and/or loaded from either the host computer or the AP.

4.4.1 HOST MEMORY ADDRESS REGISTER

The host memory register (HMA) points to consecutive locations in the memory of the host computer. It operates in either auto-increment or auto-decrement mode during DMA transfers to and from host memory. HMA is device address 1 for AP internal I/O transfers.

4.4.2 WORD COUNT REGISTER

The word count register (WC) counts the number of host memory words transferred in a DMA operation. It is preset to the desired number of words to be transferred and counts down as the transfer proceeds, stopping the DMA transfer when it reaches zero. Hardware logic prevents this register from being counted past zero. WC has AP device address 0.

4.4.3 AP DIRECT MEMORY ADDRESS REGISTER

The AP direct memory address register (APDMA) points to consecutive locations in AP main data memory during DMA transfers to and from MD. This register can operate in either auto-increment or auto-decrement mode. APDMA has AP device address 3.

4.4.4 CONTROL REGISTER

The control register (CTL) acts as a control over the DMA and interrupt functions of the host interface. This register controls the direction and mode of transfer (DMA or program control) and the type of data format and provides certain bits of status information pertaining to the transfer. CTL has AP device address 2. The format of the control register is shown in Figure 4-3. The bit descriptions are contained in Table 4-2.

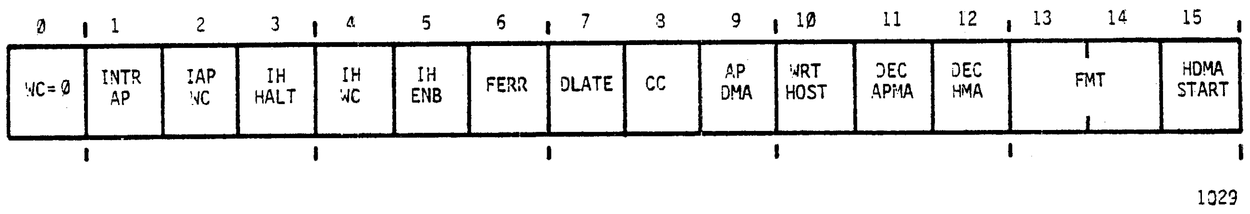


Figure 4-3 DMA Control Register Format

Table 4-5 DMA Control Register Description

BIT	MNEMONIC	EFFECT
Ø	WC = Ø	Indicates that the word count register is zero. Note that WC is decremented only during DMA transfers to/from host memory (read only bit). Should not be used to monitor DMA activity.
1	INTR AP	Sets the INTRQ (interrupt request) flag in the AP.
2	IAPWC	Sets INTRQ (interrupt request) flag in the AP when the DMA transfer is done.
3	IHALT	Enables a host interrupt when the AP halts.
4	IHWC	Enables a host interrupt when the DMA transfer is done.
5	IHENB	Interrupt Host Enable. Interrupt Host if AP attempts to set this Bit. This bit can actually be written only by the Host. (This is not supported on all host systems.)
6	FERR	Format error. Indicates that exponent underflow or overflow occurred in conversion from AP format to host floating-point format.
7	DLATE	Data late. Indicates that the AP did not empty the format buffer before the host attempted to reload it. On some hosts this bit also indicates an attempt to access non-existent host memory. In either case the DMA transfer is terminated.
8	CC	Consecutive cycle. Block DMA transfers to/from host memory occur without interruption. On typical hosts, the host CPU is locked out but other higher priority DMA devices still have access to host memory.
9	APDMA	Allows the interface to perform DMA transfers to/from AP memory. Depending on the direction of transfer, a main data memory cycle is initiated every time the host finishes reading or loading the format register, whether via DMA or program control. On the AP side, the format register is loaded from the main data bus instead of the data pad bus.
1Ø	WRTHOST	Write to host. This bit controls the direction of transfer. If set, data is read from the AP, passed through the format register, and written to the host. If clear, the direction of transfer is reversed.
11	DECAPMA	Decrement APMA. If set, APMA is decremented during DMA transfers to/from AP Main Data memory. If clear, APMA is incremented. (This capability is not present on all host systems.)
12	DECHMA	Decrement HMA. If set, HMA is decremented during DMA transfers to/from host memory. If clear, HMA is incremented.
13 & 14	FMT	Format Register Control. (See note.)
15	HDMA start/busy	Host DMA start. Initiate DMA transfers to/from host memory. When read, the state of this bit reflects the status of the host DMA activity ('1' if active, 'Ø' if inactive). Transfers continue until WC = Ø.

NOTE

The format register mode of operation is controlled entirely by bits 9, 1Ø, 13 and 14 of the control register. Thus, even the host and the AP can load and read the format register via program control I/O transfers at any time. The programmer must be sure that the type of transfer he performs is consistent with these bits of CTL for the transfer to be meaningful. (Refer to Table 4-6.)

1030

Table 4-6 Bits 13-14

VALUE IN BITS 13 & 14	FORMAT TYPE
Ø	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 FPNs. Low 16-bits of AP FPN are sent to host.
2	Conversion of "signed-magnitude mantissa with binary exponent" format to/from AP floating point format. Includes logic to handle "phantom bit" formats.
3	Conversion of IBM 32-bit format to/from AP format. IBM format can be specified to have either sign-magnitude or two's complement mantissa.

NOTE

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

1031

4.5 FORMAT CONVERSION REGISTER

This 38-bit double-buffered register is used for all transfers of floating-point numbers (FPNs) between the host and the AP. It also provides the most efficient path for transfer of microcode half-words (32 bits). It performs bi-directional format conversions under the direction of bits 9, 10, 13, and 14 of the CTL register. The programmer must be aware of the fact that the format conversion is a slave to these CTL bits. Nonsence results if transfers to and from the formatter are not consistent with these CTL bits. The host and AP can read the output of the formatter at any time without restriction; however, the input to the formatter is controlled by CTL bits 9 and 10.

Table 4-7 CTL Register Bits 9-10

CTLØ9	CTL1Ø	INPUT PATH TO FORMATTERS
Ø, 1	Ø	Host Data Bus
Ø	1	AP I/O Bus
1	1	AP Main Data Output

1032

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

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

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

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

4.6 AP INTERNAL INTERFACE TO HOST INTERFACE

The registers in the host interface are accessible to the AP via its input/output (I/O) instructions (FADD=7).

Table 4-8 AP Device Address for Host Interface Registers

I/O DEVICE	DEVICE ADDRESS
HOST INTERFACE	
DMA REGISTERS:	
WORD COUNT REGISTER (WC)	0
HOST MEMORY ADDRESS REGISTER (HMA)	1
CONTROL REGISTER (CTL)	2
AP MEMORY ADDRESS REGISTER (APMA)	3
FORMATTER (FMT)	4
WRITABLE TABLE MEMORY (TMRAM)	5
PAGE SELECT SELECT OPTION	
MEMORY ADDRESS EXTENSION (MAE)	30
APMA EXTENSION (APMAE)	31
MASK (including MODE and I/O)	32
ADDITIONAL DEVICE ADDRESSES:	
First IOP16	10-14
Second IOP16	20-24
Parity Option	33-37
First PIOP	100, 101, 110-117

1033

An IN, OUT, or SNSA instruction at DA=4 (FORMAT) generates an IODRDY response if the format register is ready to accept data from the AP (CTL bit 10=1) or if it has formatted data ready for the AP (CTL bit 10=0). If CTL bit 9 is 1, the AP cannot load the formatter via I/O instructions since the input multiplexer to the format register is set to select main data instead of the AP I/O bus. Note that the AP cannot change the state of CTL bit 5. An interrupt of the host is generated if it attempts to set this bit when the bit has already been set by the host. The AP can read the CTL at any time without interferring with the host interface. If both the host and the AP try to write CTL or access HMA, WC, or APMA at the same time, the host selection and data has priority over that of the AP.

Access to the format conversion register is controlled by CTL bits 9, 10, 13, and 14. Refer to section 4.4 for a description of the function of these bits.

4.7 AN EXAMPLE OF LOADING PROGRAMS INTO THE AP

Loading and running a program in the AP from a cold start is a five-step process which illustrates use of the front panel.

1. Using the AP front panel from the host computer, finger switch in a three-instruction bootstrap program into program memory.
2. Start the bootstrap running.
3. Set the address in the AP where the loaded program is to go.
4. Start a DMA transfer of program words from host computer memory to the AP. The bootstrap program running in the AP stores these words into program memory.
5. When the DMA transfer is done, stop the bootstrap program in the AP and then restart the AP executing the newly-loaded program.

These five steps are detailed in the remainder of Chapter 4. DMA control and front panel interrogation is done from the host computer by setting various interface registers. The actual host computer I/O instructions to accomplish this, of course, depend upon the particular host computer. For the purposes of this explanation, the indicated numbers are loaded into a designated interface register in order to accomplish the desired goals.

Step 1:

For the purpose of this example, the bootstrap program is put into program source memory locations 0, 1, and 2.

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

$0 \longrightarrow$ SWR	Put 0 into the switches.
$1003 \longrightarrow$ FN	Put 1003 into the function register (causing a deposit into TMA).

2. Put bits 0-63 of bootstrap program program word no. 1 into program memory location 0 using four deposits of SWR $\longrightarrow PS_{TMA}$.

(bits 0-15) $\longrightarrow$ SWR $1010 \longrightarrow$ FN	Put bits 0-15 into the switches. Put 1010 into the function register (causes a deposit into bits 0-15 of PS_{TMA}).
(bits 16-31) $\longrightarrow$ SWR $1030 \longrightarrow$ FN	Put bits 16-31 into bits 16-31 of PS_{TMA}.
(bits 32-47) $\longrightarrow$ SWR $1050 \longrightarrow$ FN	Put bits 32-47 into 32-47 of PS_{TMA}.
(bits 48-63) $\longrightarrow$ SWR $1370 \longrightarrow$ FN	Put bits 48-63 into bits 48-63 of PS_{TMA} and increments TMA to point to location 1.

3. Repeat the second and third bootstrap program words in no. 2 above.

It is necessary to perform these steps only once.

Step 2:

Set the address in the AP program memory where the program is to be loaded by the bootstrap into TMA. For this example, this address is 200:

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

Step 3:

Start the bootstrap program running in the AP.

Set the switches to 0 and do a start.

0 → SWR	
40000 → FN	Start the AP at location 0.

The bootstrap program (as demonstrated in step 4) spins while waiting for words to come across the DMA from the host computer.

Step 4:

Start the DMA transfer from host memory into the AP. For this example, it is assumed that the program is in host memory at location 20000. The program to be loaded is 200 AP program words (or 800 16-bit host words) long. The actual host memory location and length could be any particular value.

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

Note in particular the CTL bits. Bit 15 initiates the DMA and bit 8 requests consecutive memory cycles from the host. By not setting bits 10 or 11, the transfer is set to go to the AP, but not into main data memory. Instead, the data goes only as far as the formatter which the bootstrap reads. If bit 4 is set, the host computer is interrupted when the DMA is done.

Step 5:

Finally, the three-word bootstrap program is ready to run in the AP.

```
1. LDDA;  DB=4      "set DEVICE ADDRESS to 4
```

This instruction sets the device address register so that future I/O instructions refer to device no. 4, which is the DMA formatter (where the data from the host computer ends up).

```
2. LOOP:SPININ;     "wait for some data
        DB=INBS;    "get the data
        LPSLT       "put it into the left half of P.S.
```

The SPININ causes the processor to hang until the current I/O device address (in this case, the DMA formatter) has some new data. Then, to read that data, the DB=INBS puts the input data onto the data pad bus. The LPSLT puts what is on the data pad bus into the left half (bits 0 through 31) of the program memory location pointed at by the TMA register.

Two points should be considered:

- The formatter is 32 bits wide on the AP end; every time the interface receives 32 bits of data from the host computer, the SPIN stops waiting, and another 32 bits of data are processed. Since the program words loaded are 64 bits wide, they are halved (left, right, left, right, etc.) and stored accordingly into program memory.

- TMA is used as a pointer indicating where the bootstrap should place the program it is loading; thus, the LPSLT puts the program words into the proper place.

```
3. SPININ;          "wait for data
        DB=INBS;    "get the data
        LPSRT;      "put it into the right half
        INCTMA;     "increment pointer
        BR LOOP.    "go back for more
```

This does basically the same as no. 2 above except that this processes the right half (bits 32-63) of a 64-bit program word. The INCTMA increments the storing pointer so instruction no. 2 stores its data into the next word. The branch uses loop waiting for more program half-words.

Step 6:

Back in host, waiting for the DMA transfer is accomplished by:

- reading the CTL register
- testing for bit 15 (the LSB) equal to 1
- if so, going back to step 1

Enabling a host interrupt on DMA completion is also possible.

When DONE, the bootstrap program is stopped (which otherwise would run forever) with a panel RESET function, and the newly-loaded program is started (example starts at location 200):

```
4000  → FN     "reset the AP
200   → SWR    "new program address
1000  → FN     "set 200 into PSA
20000 → FN     "continue (from 200) (i.e., start
                          at AP location 200)
```

To set a program breakpoint, the user can set the breakpoint address into the SWR and use 20400 (continue + break on PSA) for the final panel function.

NOTE

The simplest way for the running AP program to indicate to the host computer that it is done with its task is to HALT. When this happens, bit 0 in the panel function register is set (which the host can test for) or a host interrupt can be enabled (CTL bit 3).

APPENDIX A

AP REGISTERS/DATA PATH NAMES

Table A-1 Registers and Data Paths

mnemonic	width	name
SP	16 bits	scratch pad registers (16)
SPD	4	s-pad destination address register
SPFN	16	scratch pad ALU/shifter function output
PNBLS	16	panel bus
SWR	16	panel switch register
LITES	16	panel display register
APSTATUS	16	AP status register
PS	64	program source memory
CB	64	command buffer
PSA	16	program source address register
SRS	16	subroutine return stack
SRA	16	subroutine return stack pointer
DPX	38	data pad X registers (32)
DPY	38	data pad Y registers (32)
DB	38	data pad bus
DPA	16	data pad address register
TM	38	table memory output register
TMA	16	table memory address register
MD	38	data memory output register
MI	38	data memory input register
MA	16	memory address register
A1	38	floating adder input register no. 1
A2	38	floating adder input register no. 2
FA	38	floating adder output register
M1	38	floating multiplier input register no. 1
M2	38	floating multiplier input register no. 2
FM	38	floating multiplier output register
IODEVICE		I/O device
DA	16	I/O device address
INBS	38	I/O input bus
IODRDY	1	I/O data ready flag
A	1	I/O device condition A flag
B	1	I/O device condition B flag

Subscripts indicate addressing within memory element (i.e., PS_{PSA} means the location in program source memory pointed to by the program source address register).

Superscripts indicate portions of word (i.e., $A2^E$ means the exponent portion of the A2 register).

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

Table A-2 AP Internal Status Register

Ø	1	2	3	4	5	6	7	8	9	1Ø	11	12	13	14	15
OVF	UNF	DIVZ	FZ	FN	Z	N	C	PERR	PENB	SRAO	IFFT	FFT	BIT REVERSE		

1034

bits	mnemonic	meaning
0	OVF	Set when the current adder or multiplier (FA or FM) has overflowed. Overflow occurs when an exponent value is increased above 511. The offending result is set to the signed maximum of value of $(1-2^{-27}) * 2^{511}$, which is roughly $6.7 * 10^{153}$. This bit remains set until cleared by the microprogram or host computer.
1	UNF	Set when the current adder or multiplier result (FA or FM) has underflowed. Underflow occurs when an exponent value is decreased below -512. The minimum legal magnitude which numbers can take without underflowing is roughly $3.7 * 10^{-155}$. The offending value is set to zero. This bit remains set until cleared by the microprogram or host computer.

Table A-2 Internal Status Register (cont.)

bits	mnemonic	meaning
2	DIVZ	A divide by zero has occurred. The result was set to the value of the dividend. This bit remains set until cleared by the microprogram or host computer.
3	FZ	Set when the current adder result (FA) is zero.
4	FN	Set when the current adder result (FA) is negative.
5	Z	Set when the current s-pad function (SPFN) is zero.
6	N	Set when the current s-pad function (SPFN) is negative.
7	C	S-pad carry bit. If no s-pad shift is specified, carry is the carry bit from the s-pad ALU. If a shift is specified, carry is the last bit shifted off the end of the s-pad result by the shift.
8	PERR	(Optional). Set when a main data memory parity error has occurred. Three parity bits are used, one each to check the exponent, high mantissa, and low mantissa portions of the memory word. If PENB is set, the processor halts on this error. (See Page Select/ Parity Option Manual (FPS 860-7365-000) for more information.)
9	PENB	(Optional). Enables halt on memory parity error. If set, the processor halts when a memory parity error is detected.
10	SRAO	Subroutine return stack overflow. Set if more than 16 levels of nested subroutine calls occur.

Table A-2 Internal Status Register (cont.)

bits	mnemonic	meaning
11	IFFT	Inverse FFT flag. When set in conjunction with the FFT flag, bit 12, roots of unity table references are interpreted as positive angles.
12	FFT	FFT flag. When set, table memory addresses are interpreted as negative angles referencing the roots of unity table contained in table memory.
13-15	bit reverse	15-$\mathrm{Log}_2 N$ where N is the length of a complex data array to which the s-pad address bit reverse operator is being applied.

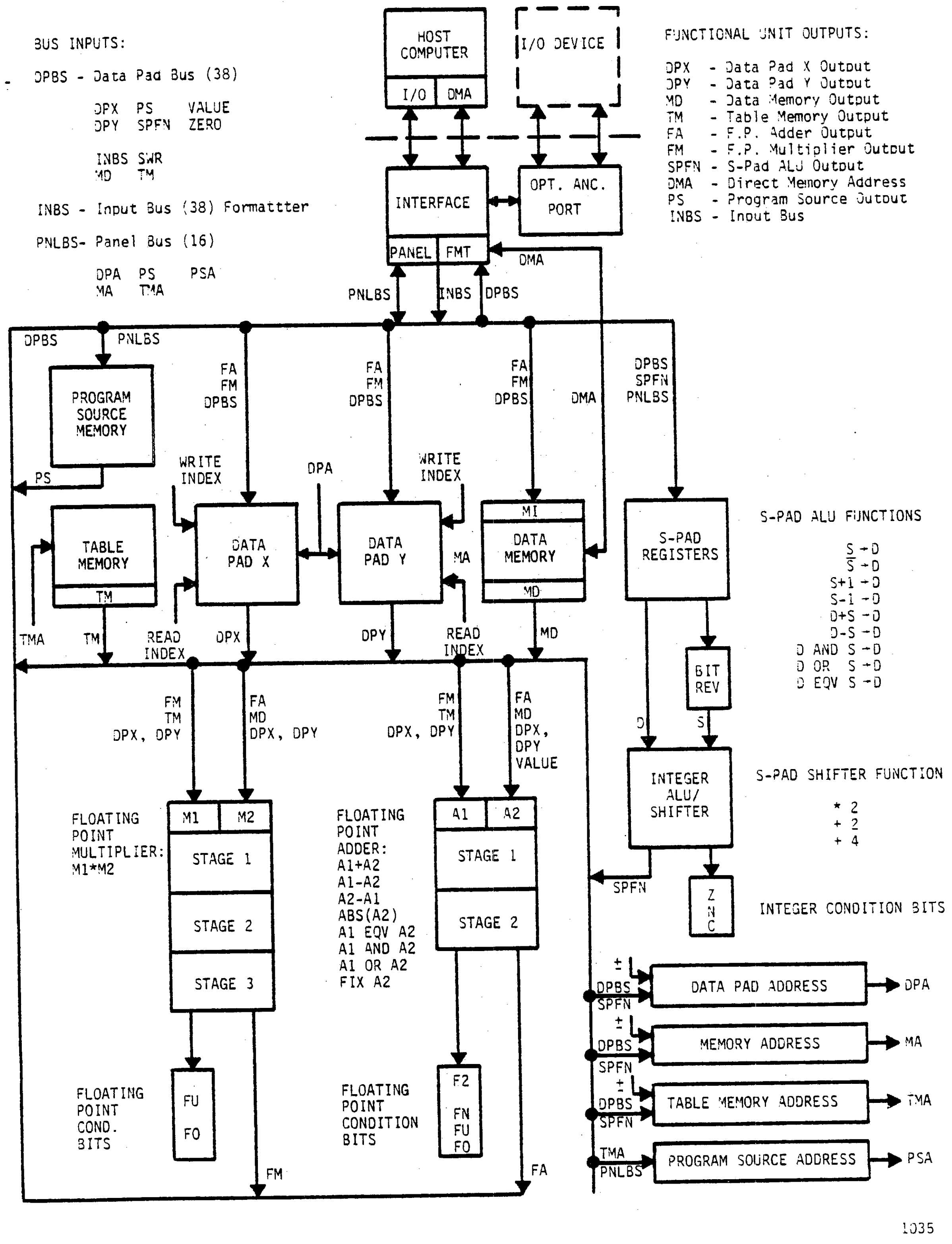


Figure A-1 AP Functional Units

Table A-3 AP Instruction Summary

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

OCTAL CODE	FIELD NAME										OCTAL CODE
	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	WRTLMN	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	#	Displace-	DB	DB	INBS	Read	Read	Write	Write	FMUL	1
2	BR	ment)	FA	FA	VALUE*	Index)	Index)	Index)	Index)		2
3	BINTRQ	(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										17

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	DECDPA	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.

0627

Table A-4 SPEC Fields

<u>SPEC FIELDS</u> 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	
Ø	STEST	BFLT	PNLLIT	JMPA*	RPSØA*	RPS1A*	RPSLA*	NOP	Ø
1	HOSTPNL	BLT	DBELIT	JSRA*	RPS2A*	RPS3A*	RPSFA*	SETEXA*	1
2	SPMDA	BNC	DBHLIT	JMP*	RPSØ*	RPS1*	RPSL*	NOP	2
3	NOP	BZC	DBLLIT	JSR*	RPS2*	RPS3*	RPSF*	SETEX*	3
4	NOP	BDBN	NOP	JMPT	RPSØT	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
1Ø	SETPSA	NOP	SWDB	NOP	WPSØA*	WPS1A*	LPSLA*	NOP	1Ø
11	PSEVEN	NOP	SWDBE	NOP	WPS2A*	WPS3A*	LPSRA*	NOP	11
12	PSODD	NOP	SWDBH	NOP	WPSØ*	WPS1*	LPSL*	NOP	12
13	PS	NOP	SWDBL	NOP	WPS2*	WPS3*	LPSR*	NOP	13
14	SETEXIT	BFLØ	NOP	NOP	WPSØT	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

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

Table A-5 I/O Fields

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

OCTAL CODE	FIELD NAME							OCTAL CODE
	IO	LDREG	RDREG	INOUT	SENSE	FLAG	CONTROL	
Ø	LDREG	NOP	RPSA	OUT	SNSA	SFLØ	HALT	Ø
1	RDREG	LDSPD	RSPD	SPNOUT	SPININ	SFL1	IORST	1
2	SPMDAV	LDMA	RMA	OUTDA	SNSADA	SFL2	INTEN	2
3	NOP	LDTMA	RTMA	SPOTDA	SPNADA	SFL3	INTA	3
4	INOUT	LDDPA	RDPA	IN	SNSB	CFLØ	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	SPNDBA	CFL3	NOP	7

1036

APPENDIX B

INSTRUCTION SUMMARY

Table B-1 AP Instruction Field Layout

0	B	S-PAD GROUP	
1	SOP		
2			
3			
4	SH		
5			
6	SPS	SOP1	SPEC OPER
7			
8			
9			
10	SPD		
11			
12			
13			
14	FADD	ADDER GROUP	
15			
16			
17	A1	FADD1	I/O
18			
19			
20	A2		
21			
22			
23	COND	BRANCH GROUP	
24			
25			
26			
27	DISP		
28			
29			
30			
31			

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

0629

Table B-2 S-pad Group

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

FIELD	OCTAL CODE	MNEMONIC	EFFECT
B	Ø	-	No-op
	1	&	Use SP_{SPS} (bit-reversed)
SOP	Ø	-	See SOP1 field
	1	-	See Special Operations Group
	2	ADD	$(SP_{SPD}) + (SP_{SPS}) \longrightarrow SPFN$
	3	SUB	$(SP_{SPD}) - (SP_{SPS}) \longrightarrow SPFN$
	4	MOV	$(SP_{SPS}) \longrightarrow SPFN$
	5	AND	(SP_{SPD}) AND $(SP_{SPS}) \longrightarrow SPFN$
	6	OR	(SP_{SPD}) OR $(SP_{SPS}) \longrightarrow SPFN$
	7	EQV	(SP_{SPD}) $\overline{\text{XOR}}$ $(SP_{SPS}) \longrightarrow SPFN$
SH (see NOTE)	Ø	-	No-op
	1	L	SPFN*2 $\longrightarrow$ SPFN (left shift)
	2	RR	SPFN÷4 $\longrightarrow$ SPFN (double right shift)
	3	R	SPFN÷2 $\longrightarrow$ SPFN (right shift)
SPS	Ø-17$_8$	Ø-17$_8$	S-Pad Source Operand Address
SPD	Ø-17$_8$	Ø-17$_8$	S-Pad Destination Address, SPFN $\longrightarrow$ SP_{SPD} unless inhibited by No Load (COND = 1)

NOTE

These are logical shifts:

Right shift	0	$\boxed{Ø\text{-}15}$	$\boxed{C}$
Left shift	$\boxed{C}$	$\boxed{Ø\text{-}15}$	Ø

1037

Table B-2 S-pad Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE)
SOP1	Ø	-	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 ØØ-11)
	3	WRTLMN	Restricts DPX, DPY & MI fields to Write Low Mantissa Only (Bits 12-27)
	4	-	-
	5	-	-
	6	-	-
	7	-	-
	1Ø	CLR	$\emptyset \longrightarrow$ SPFN
	11	INC	$(SP_{SPD}) + 1 \longrightarrow$ SPFN
	12	DEC	$(SP_{SPD}) - 1 \longrightarrow$ SPFN
	13	COM	$(\overline{SP_{SPD}}) \longrightarrow$ SPFN logical complement
	14	LDSPNL	$SP_{SPD} \longrightarrow$ SPFN, PNLBS $\longrightarrow SP_{SPD}$
	15	LDSPE	$SP_{SPD} \longrightarrow$ SPFN, $DB^{E} - 512 \longrightarrow SP_{SPD}$
	16	LDSPI	$SP_{SPD} \longrightarrow$ SPFN, $DB^{ML} \longrightarrow SP_{SPD}$
	17	LDSPT	$SP_{SPD} \longrightarrow$ SPFN, $DB^{MT} \longrightarrow SP_{SPD}$

NOTE

MH = Mantissa High = Mantissa bits ØØ-11
ML = Mantissa Low = Mantissa bits 12-27
MT = Mantissa bits for table lookups = Mantissa bits Ø2-Ø8
E = Exponent

1038

Table B-3 Special Operations Group

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

FIELD	OCTAL CODE	MNEMONIC	EFFECT
SPEC	Ø	-	See STEST Field (B-6)
	1	-	See HOSTPNL Field (B-7)
	2	SPMDA	Spin until MD available
	3	-	-
	4	-	-
	5	-	-
	6	-	-
	7	-	-
	1Ø	-	See SETPSA Field, inhibit TEST except No Load (B-8)
	11	-	See PSEVEN Field (B-9)
	12	-	See PSODD Field (B-1Ø)
	13	-	See PS Field (B-11)
	14	-	See SETEXIT Field (B-12)
	15	-	-
	16	-	-
	17	-	-

1039

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE)
STEST	Ø	BFLT	Branch if FA<Ø.Ø
	1	BLT	Branch if SPFN<Ø
	2	BNC	Branch if S-Pad carry bit = 1
	3	BZC	Branch if S-Pad carry bit = Ø
	4	BDBN	Branch if DB <Ø.Ø
	5	BDBZ	Branch if DB positive and unnormalized
	6	BIFN	Branch if Inverse FFT flag = 1
	7	BIFZ	Branch if Inverse FFT flag = Ø
	1Ø	-	-
	11	-	-
	12	-	-
	13	-	-
	14	BFLØ	Branch if Flag Ø =1
	15	BFL1	Branch if Flag 1 = 1
	16	BFL2	Branch if Flag 2 = 1
	17	BFL3	Branch if Flag 3 = 1

NOTE

If the above specified condition is true OR the condition specified in the COND field is true, a branch occurs to (PSA) + DISP-2Ø.

1040

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 1)
HOSTPNL	Ø	PNLLIT	PNLBS $\longrightarrow$ LITES
	1	DBELIT	DB^{E} $\longrightarrow$ PNLBS $\longrightarrow$ LITES
	2	DBHLIT	DB^{MH} $\longrightarrow$ PNLBS $\longrightarrow$ LITES
	3	DBLLIT	DB^{ML} $\longrightarrow$ PNLBS $\longrightarrow$ LITES
	4	-	-
	5	-	-
	6	-	-
	7	-	-
	1Ø	SWDB	(SWR) $\longrightarrow$ PNLBS $\longrightarrow$ DB
	11	SWDBE	(SWR) $\longrightarrow$ PNLBS $\longrightarrow$ DB^{E} and WRTEXP (see NOTE 2)
	12	SWDBH	(SWR) $\longrightarrow$ PNLBS $\longrightarrow$ DB^{MH} and WRTHMAN (see NOTE 2)
	13	SWDBL	(SWR) $\longrightarrow$ PNLBS $\longrightarrow$ DB^{ML} and WRTLMAN (see NOTE 2)
	14	-	-
	15	-	-
	16	-	-
	17	-	-

NOTE

1) MH = Mantissa High = Mantissa bits ØØ-11
   ML = Mantissa Low = Mantissa bits 12-27
   E = Exponent

2) Restrict DPS, DPY and MI to:

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

1041

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE)
SETPSA	Ø	JMPA	VALUE $\rightarrow$ PSA
	1	JSRA	(SRA) + 1 $\rightarrow$ SRA, (PSA) + 1 $\rightarrow$ SRS_{SRA}, VALUE $\rightarrow$ PSA
	2	JMP	VALUE + (PSA) $\rightarrow$ PSA
	3	JSR	(SRA) + 1 $\rightarrow$ SRA, (PSA) + 1 $\rightarrow$ SRS_{SRA}, VALUE + (PSA) $\rightarrow$ PSA
	4	JMPT	(TMA) $\rightarrow$ PSA
	5	JSRT	(SRA) + 1 $\rightarrow$ SRA, (PSA) + 1 $\rightarrow$ SRS_{SRA}, (TMA) $\rightarrow$ PSA
	6	JMPP	(SWR) $\rightarrow$ PNLBS $\rightarrow$ PSA
	7	JSRP	(SRA) + 1 $\rightarrow$ SRA, (PSA) + 1 $\rightarrow$ SRS_{SRA}, (SWR) $\rightarrow$ PNLBS $\rightarrow$ PSA

NOTE

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

1042

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 2)
PSEVEN	Ø	RPSØA	$(PS^{QØ}_{VALUE}) \rightarrow$ PNLBS $\rightarrow$ LITES
	1	RPS2A	$(PS^{Q2}_{VALUE}) \rightarrow$ PNLBS $\rightarrow$ LITES
	2	RPSØ	$(PS^{QØ}_{VALUE+PSA}) \rightarrow$ PNLBS $\rightarrow$ LITES
	3	RPS2	$(PS^{Q2}_{VALUE+PSA}) \rightarrow$ PNLBS $\rightarrow$ LITES
	4	RPSØT	$(PS^{QØ}_{TMA}) \rightarrow$ PNLBS $\rightarrow$ LITES
	5	RPS2T	$(PS^{Q2}_{TMA}) \rightarrow$ PNLBS $\rightarrow$ LITES
	6	-	-
	7	-	-
	1Ø	WPSØA	(SWR) $\rightarrow$ PNLBS $\rightarrow PS^{QØ}_{VALUE}$
	11	WPS2A	(SWR) $\rightarrow$ PNLBS $\rightarrow PS^{Q2}_{VALUE}$
	12	WPSØ	(SWR) $\rightarrow$ PNLBS $\rightarrow PS^{QØ}_{VALUE+PSA}$
	13	WPS2	(SWR) $\rightarrow$ PNLBS $\rightarrow PS^{Q1}_{VALUE+PSA}$
	14	WPSØT	(SWR) $\rightarrow$ PNLBS $\rightarrow PS^{QØ}_{TMA}$
	15	WPS2T	(SWR) $\rightarrow$ PNLBS $\rightarrow PS^{Q2}_{TMA}$
	16	-	-
	17	-	-

NOTE

1) This field requires 2 cycles to execute.

2) VALUE = Bits 48-63 of this instruction (CB48-CB63)
QØ = Quarter zero of Program Source Word (PSØØ-PS15)
Q2 = Quarter two of Program Source Word (PS31-PS47)

1043

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 2)
PSODD (see NOTE 1)	0	RPS1A	$(PS^{Q1}_{VALUE}) \longrightarrow PNLBS \longrightarrow LITES$
	1	RPS3A	$(PS^{Q3}_{VALUE}) \longrightarrow PNLBS \longrightarrow LITES$
	2	RPS1	$(PS^{Q1}_{VALUE+PSA}) \longrightarrow PNLBS \longrightarrow LITES$
	3	RPS3	$(PS^{Q3}_{VALUE+PSA}) \longrightarrow PNLBS \longrightarrow LITES$
	4	RPS1T	$(PS^{Q1}_{TMA}) \longrightarrow PNLBS \longrightarrow LITES$
	5	RPS3T	$(PS^{Q3}_{TMA}) \longrightarrow PNLBS \longrightarrow LITES$
	6	-	-
	7	-	-
	10	WPS1A	$(SWR) \longrightarrow PNLBS \longrightarrow PS^{Q1}_{VALUE}$
	11	WPS3A	$(SWR) \longrightarrow PNLBS \longrightarrow PS^{Q3}_{VALUE}$
	12	WPS1	$(SWR) \longrightarrow PNLBS \longrightarrow PS^{Q1}_{VALUE+PSA}$
	13	WPS3	$(SWR) \longrightarrow PNLBS \longrightarrow PS^{Q3}_{VALUE+PSA}$
	14	WPS1T	$(SWR) \longrightarrow PNLBS \longrightarrow PS^{Q1}_{TMA}$
	15	WPS3T	$(SWR) \longrightarrow PNLBS \longrightarrow PS^{Q3}_{TMA}$
	16	-	-
	17	-	-

NOTE

1) This field requires 2 cycles to execute.

2) 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)

1044

Table B-3 Special Operations Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 2)
PS (see NOTE 1)	Ø	RPSLA	$(PS^{LH}_{VALUE}) \longrightarrow DB$
	1	RPSFA	$(PS^{FP}_{VALUE}) \longrightarrow DB$
	2	RPSL	$(PS^{LH}_{VALUE+PSA}) \longrightarrow DB$
	3	RPSF	$(PS^{FP}_{VALUE+PSA}) \longrightarrow DB$
	4	RPSLT	$(PS^{LH}_{TMA}) \longrightarrow DB$
	5	RPSFT	$(PS^{FP}_{TMA}) \longrightarrow DB$
	6	RPSLP	$(PS^{LH}_{PNLBS}) \longrightarrow DB$
	7	RPSFP	$(PS^{FP}_{PNLBS}) \longrightarrow DB$
	1Ø	LPSLA	$DB \longrightarrow PS^{LH}_{VALUE}$
	11	LPSRA	$DB \longrightarrow PS^{RH}_{VALUE}$
	12	LPSL	$DB \longrightarrow PS^{LH}_{VALUE+PSA}$
	13	LPSR	$DB \longrightarrow PS^{RH}_{VALUE+PSA}$
	14	LPSLT	$DB \longrightarrow PS^{LH}_{TMA}$
	15	LPSRT	$DB \longrightarrow PS^{RH}_{TMA}$
	16	LPSLP	$DB \longrightarrow PS^{LH}_{PNLBS}$
	17	LPSRP	$DB \longrightarrow PS^{RH}_{PNLBS}$

NOTE

1) This field requires 2 cycles to execute.

2) VALUE = Bits 48-63 of this instruction (CB48-CB63)
   LH = Left half of Program Source Word (Bits ØØ-31)
   RH = Right half of Program Source Word (Bits 32-63)
   FP = Program Source bits 26-63, used for floating-point literals

1045

Table B-3 Special Operations Group (cont.)

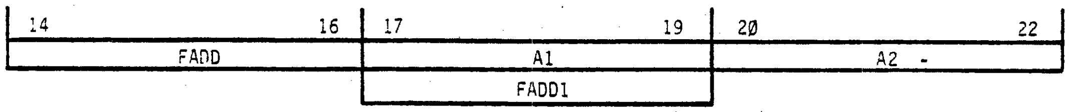
FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE)
SETEXIT	Ø	-	-
	1	SETEXA	VALUE $\longrightarrow SRS_{SRA}$
	2	-	-
	3	SETEX	VALUE + (PSA) $\longrightarrow SRS_{SRA}$
	4	-	-
	5	SETEXT	TMA $\longrightarrow SRS_{SRA}$
	6	-	-
	7	SETEXP	PSA + 1 $\longrightarrow SRS_{SRA}$

NOTE

Sets the current subroutine return address as indicated above.
SRA does not change.
VALUE = Bits 48-63 of this instruction.

1046

Table B-4 Floating Adder Group



FIELD	OCTAL CODE	MNEMONIC	EFFECT
FADD	Ø	-	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
A1	Ø	NC	$(A1) \longrightarrow A1$
	1	FM	$FM \longrightarrow A1$
	2	DPX(IDX)	$(DPX_{DPA+IDX}) \longrightarrow A1$ Where XR = IDX+4
	3	DPY(IDX)	$(DPY_{DPA+IDX}) \longrightarrow A1$ Where YR = IDX+4
	4	TM	$(TM) \longrightarrow A1$
	5	ZERO	$Ø.Ø \longrightarrow A1$
	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

1047

Table B-4 Floating Adder Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT
A2	Ø	NC	$(A2) \longrightarrow A2$
	1	FA	$FA \longrightarrow A2$
	2	DPX(IDX)	$(DPX_{DPA+IDX}) \longrightarrow A2$, where XR = IDX+4
	3	DPY(IDX)	$(DPY_{DPA+IDX}) \longrightarrow A2$, where YR = IDX+4
	4	MD	$(MD) \longrightarrow A2$
	5	ZERO	Ø.Ø $\longrightarrow A2$
	6	MDPX(IDX)	$SPFN+512 \longrightarrow A2^E$, $(DPX^M_{DPA+IDX}) \longrightarrow A2^M$
	7	EDPX(IDX)	$(DPX^E_{DPA+IDX}) \longrightarrow A2^E$, $SPFN \longrightarrow A2^M$(ØØ-Ø1), Ø $\longrightarrow A2^M$(Ø2-27)
FADD1	Ø	-	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).

1048

Table B-5 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	Ø	-	See LDREG field
	1	-	See RDREG field
	2	SPMDAV	Spin until MD available
	3	REXIT	$SRS_{(SRA)} \rightarrow PNLBS$
	4	-	See INOUT field
	5	-	See SENSE field
	6	-	See FLAG field
	7	-	See CONTROL field
LDREG	Ø	-	No-op
	1	LDSPD	$DPBS \rightarrow SPD$
	2	LDMA	$DPBS \rightarrow MA$
	3	LDTMA	$DPBS \rightarrow TMA$
	4	LDDPA	$DPBS \rightarrow DPA$
	5	LDSP	$SP_{SPD} \rightarrow SPFN$, $DPBS \rightarrow SP_{SPD}$
	6	LDAPS	$DPBS \rightarrow APSTATUS$
	7	LDDA	$DPBS \rightarrow DA$

1049

Table B-5 I/O Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT
RDREG	Ø	RPSA	(PSA) $\longrightarrow$ PNLBS
	1	RSPD	(SPD) $\longrightarrow$ PNLBS
	2	RMA	(MA) $\longrightarrow$ PNLBS
	3	RTMA	(TMA) $\longrightarrow$ PNLBS
	4	RDPA	(DPA) $\longrightarrow$ PNLBS
	5	RSPFN	SPFN $\longrightarrow$ PNLBS
	6	RAPS	(APSTATUS) $\longrightarrow$ PNLBS
	7	RDA	(DA) $\longrightarrow$ PNLBS
INOUT	Ø	OUT	DPBS $\longrightarrow$ IODEVICE_{DA}
	1	SPNOUT	SPIN if IODRDY_{DA} = Ø DPBS $\longrightarrow$ IODEVICE_{DA}
	2	OUTDA	DPBS $\longrightarrow$ IODEVICE_{DA}, SPFN $\longrightarrow$ DA
	3	SPOTDA	SPIN if IODRDY = Ø, SPFN $\longrightarrow$ DA DPBS $\longrightarrow$ IODEVICE_{DA}
	4	IN	(IODEVICE_{DA}) $\longrightarrow$ INBS
	5	SPININ	SPIN if IODRDY_{DA} = Ø (IODEVICE_{DA}) $\longrightarrow$ INBS
	6	INDA	(IODEVICE_{DA}) $\longrightarrow$ INBS, SPFN $\longrightarrow$ DA
	7	SPINDA	SPIN if IODRDY_{DA} = Ø, SPFN $\longrightarrow$ DA (IODEVICE_{DA}) $\longrightarrow$ INBS

1050

Table B-5 I/O Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE)
SENSE	Ø	SNSA	$A_{DA} \longrightarrow$ IODRDY Flag
	1	SPINA	$A_{DA} \longrightarrow$ IODRDY, SPIN if IODRDY = Ø
	2	SNSADA	$A_{DA} \longrightarrow$ IODRDY, SPFN $\longrightarrow$ DA
	3	SPNADA	$A_{DA} \longrightarrow$ IODRDY, SPIN if IODRDY = Ø, SPFN $\longrightarrow$ DA
	4	SNSB	$B_{DA} \longrightarrow$ IORDY Flag
	5	SPINB	$B_{DA} \longrightarrow$ IODRDY, SPIN if IODRDY = Ø
	6	SNSBDA	$B_{DA} \longrightarrow$ IODRDY, SPFN $\longrightarrow$ DA
	7	SPNBDA	$B_{DA} \longrightarrow$ IODRDY, SPIN if IODRDY = Ø, SPIN $\longrightarrow$ DA
FLAG	Ø	SFLØ	$1 \longrightarrow FLAG_Ø$
	1	SFL1	$1 \longrightarrow FLAG_1$
	2	SFL2	$1 \longrightarrow FLAG_2$
	3	SFL3	$1 \longrightarrow FLAG_3$
	4	CFLØ	$Ø \longrightarrow FLAG_Ø$
	5	CFL1	$Ø \longrightarrow FLAG_1$
	6	CFL2	$Ø \longrightarrow FLAG_2$
	7	CFL3	$Ø \longrightarrow FLAG_3$

NOTE

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

1051

Table B-5 I/O Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT
CONTROL	Ø	HALT	Halt
	1	IORST	I/O reset
	2	INTEN	If CTLØ5 is set see Programmer's Reference Manual Part II, page E9.
	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 Ø-27)

1052

Table B-6 Branch Group

23 — 26	27 — 31
COND	DISP

FIELD	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 2)
COND	Ø	-	No-op
	1	#	Inhibit load of SPFN $\longrightarrow$ 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 = Ø
	6	BFPE	Branch on floating-point arithmetic error (overflow, underflow, or divide by zero).
	7	RETURN (see NOTE 1)	(SRS_{SRA}) $\longrightarrow$ PSA, (SRA) - 1 $\longrightarrow$ SRA (subroutine return jump).
	1Ø	BFEQ	Branch if FA = Ø.Ø
	11	BFNE	Branch if FA ≠ Ø.Ø
	12	BFGE	Branch if FA ≥ Ø.Ø
	13	BFGT	Branch if FA > Ø.Ø
	14	BEQ	Branch if SPFN = Ø
	15	BNE	Branch if SPFN ≠ Ø
	16	BGE	Branch if SPFN ≥ Ø
	17	BGT	Branch if SPFN > Ø
DISP (see NOTE 3)	Ø to 37		If branch condition is true, (PSA) + DISP - 20 $\longrightarrow$ PSA

NOTE

1) "RETURNS" may not be made in two successive instructions.
2) FA and SPFN are tested as to their state for the previous instruction.
3) Thus the effective Branch Range is -2Ø to +17 relative to the current instruction.

1053

Table B-7 Data Pad 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 (see NOTE 1)	Ø	-	No-op
	1	DPX(1DX)<DB	DPBS $\longrightarrow$ $DPX_{DPA+1DX}$, Where XW = 1DX+4
	2	DPX(1DX)<FA	FA $\longrightarrow$ $DPX_{DPA+1DX}$, Where XW = 1DX+4
	3	DPX(1DX)<FM	FM $\longrightarrow$ $DPX_{DPX+1DX}$, Where XW = 1DX+4
DPY (see NOTE 1)	Ø	-	No-op
	1	DPY(1DX)<DB	DPBS $\longrightarrow$ $DPY_{DPA+1DX}$, Where YW = 1DX+4
	2	DPY(1DX)<FA	FA $\longrightarrow$ $DPY_{DPA+1DX}$, Where YW = 1DX+4
	3	DPY(1DX)<FM	FM $\longrightarrow$ $DPY_{DPA+1DX}$, Where YW = 1DX+4
DPBS	Ø	DB=ZERO	Ø.Ø $\longrightarrow$ DB
	1	DB=INBS	INBS $\longrightarrow$ DB
	2	DB=VALUE	VALUE $\longrightarrow$ DB^{E}, VALUE $\longrightarrow$ DB^{ML}, sign extended into DB_{MH}
	3	DB=DPX(1DX)	$(DPX_{DPA+1DX})$ $\longrightarrow$ DB , Where XR = 1DX+4
	4	DB=DPY(1DX)	$(DPY_{DPA+1DX})$ $\longrightarrow$ DB , Where YR = 1DX+4
	5	DB=MD	(MD) $\longrightarrow$ DB
	6	DB=SPFN	SPFN + 512 $\longrightarrow$ DB^{E}, SPFN $\longrightarrow$ DB^{ML}, sign extended into DB_{MH}
	7	DB=TM	(TM) $\longrightarrow$ DB

NOTE

1) All bits written unless WRTEXP, WRTHMAN or WRTLMAN set. See SOP1 and HOSTPNL field.
2) DPBS forced to Ø if HOSTPNL field = 1Ø to 13
ML = Mantissa Low (Mantissa Bits 12-27)
MH = Mantissa High (Mantissa Bits ØØ-11)
E = Exponent
VALUE is a 16-bit 2's complement number, contained in bits 48-63 of the instruction word.

1054

Table B-7 Data Pad Group (cont.)

FIELD	OCTAL CODE	MNEMONIC	EFFECT
XR	Ø to 7		DPX Read EFA is (DPA) + XR - 4
YR	Ø to 7		DPY Read EFA is (DPA) + YR - 4
XW	Ø to 7		DPX Write EFA is (DPA) + XW - 4
YW	Ø to 7		DPY Write EFA is (DPA) + YW - 4, YW=XW if VALUE is used in another field.

1055

Table B-8 Floating Multiplier Group

51	52 53	54 55
FM	M1	M2

FIELD	OCTAL CODE	MNEMONIC	EFFECT
FM	Ø	-	No-op
	1	FMUL	Multiply: (M1)*(M2)
M1	Ø	FM	FM $\longrightarrow$ M1
	1	DPX(IDX)	$(DPX_{DPA+IDX}) \longrightarrow$ M1, Where XR=IDX+4
	2	DPY(IDX)	$(DPY_{DPA+IDX}) \longrightarrow$ M1, Where YR=IDX+4
	3	TM	(TM) $\longrightarrow$ M1
M2	Ø	FA	FA $\longrightarrow$ M2
	1	DPX(IDX)	$(DPX_{DPA+IDX}) \longrightarrow$ M2, Where XR=IDX+4
	2	DPY(IDX)	$(DPY_{DPA+IDX}) \longrightarrow$ M2, Where YR=IDX+4
	3	MD	(MD) $\longrightarrow$ 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.

1056

Table B-9 Memory Group

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

FIELD (see NOTE 1)	OCTAL CODE	MNEMONIC	EFFECT (see NOTE 3)
MI	Ø	-	No-op
	1	MI<FA	FA → MI, Write MI into Data Memory (see NOTE 2)
	2	MI<FM	FM → MI, Write MI into Data Memory (see NOTE 2)
	3	MI<DB	DB → MI, Write MI into Data Memory (see NOTE 2)
MA	Ø	-	No-op
	1	INCMA	(MA)+1 → MA, initiate a Data Memory cycle
	2	DECMA	(MA)-1 → MA, initiate a Data Memory cycle
	3	SETMA	SPFN → MA, initiate a Data Memory cycle
DPA	Ø	-	No-op
	1	INCDPA	(DPA)+1 → DPA
	2	DECDPA	(DPA)-1 → DPA
	3	SETDPA	SPFN → DPA
TMA	Ø	-	No-op
	1	INCTMA	(TMA)+1 → TMA, initiate a read from Table Memory
	2	DECTMA	(TMA)+1 → TMA, initiate a read from Table Memory
	3	SETTMA	SPFN → TMA, initiate a read from Table Memory

NOTE

1) These fields are not in effect if a value is used by another field. Changes made in MA, TMA, or DPA do not affect the values of these registers used by other fields during the current instruction.
2) All bits written unless WRTEXP, WRTHMAN or WRTLMAN is set. See SOP1 and HOSTPNL fields.
3) DB is used in place of SPFN if LDREG field is used.

1057

Notice to the Reader

- Help us improve the quality and usefulness of this manual.

- Your comments and answers to the following READERS COMMENT form would be appreciated.

To mail: fold the form in three parts so that Floating Point Systems' mailing address is visible; seal.

Thank you

READERS COMMENT FORM

Document Title ______________________________

Your comments and answers will help us improve the quality and usefulness of our publications. If your answers require qualification or additional explanation, please comment in the space provided below.

How did you use this manual?

( ) AS AN INTRODUCTION TO THE SUBJECT
( ) AS AN AID FOR ADVANCED TRAINING
( ) TO LEARN OF OPERATING PROCEDURES
( ) TO INSTRUCT A CLASS
( ) AS A STUDENT IN A CLASS
( ) AS A REFERENCE MANUAL
( ) OTHER ______________________

Did you find this material . . .

	YES	NO
● USEFUL?	( )	( )
● COMPLETE?	( )	( )
● ACCURATE?	( )	( )
● WELL ORGANIZED?	( )	( )
● WELL ILLUSTRATED?	( )	( )
● WELL INDEXED?	( )	( )
● EASY TO READ?	( )	( )
● EASY TO UNDERSTAND?	( )	( )

Please indicate below whether your comment pertains to an addition, deletion, change or error; and, where applicable, please refer to specific page numbers.

Page	Description of error or deficiency

From:

Name ______________________ Title ______________________
Firm ______________________ Department ______________________
Address ______________________ City, State ______________________
Telephone ______________________ Date ______________________

First Class
Permit No. A-737
Portland,
Oregon

BUSINESS REPLY

No postage stamp necessary if mailed in the United States

Postage will be paid by:

FLOATING POINT SYSTEMS, INC.

P.O. Box 23489

Portland, Oregon 97223

Attention: Technical Publications

FLOATING POINT SYSTEMS, INC.

CALL TOLL FREE 800-547-1445
P.O. Box 23489, Portland, OR 97223
(503) 641-3151, TLX: 360470 FLOATPOINT PTL