2.5 THE FPS AP-120B AND DERIVATIVES

The FPS AP-120B and its derivatives—the AP 190L, the FPS-100, 164, 164/MAX, 264 and the FPS 5000—are all members of a single family of computers based on a common architecture, namely that of the FPS AP-120B. These computers have been renamed as follows: the original MSI version of the FPS-164 (now discontinued) is called the M140, the later VLSI version (first called the FPS-364) is called the M30, the FPS-164/MAX is called the M145, and the FPS-264 is now the M60. They are all manufactured by Floating Point Systems Inc.† in Beaverton near Portland, Oregon, USA. The company was founded in 1970 by C N Winningstad to manufacture low-cost yet high-performance floating-point units to boost the performance of minicomputers, particularly for signal processing applications. Starting in 1971, the company produced floating-point units for inclusion in other manufacturers' machines (e.g. Data General). The first machine marketed under the company's name, the AP-120B, was co-designed by George O'Leary and Alan Charlesworth, and had a peak performance of 12 Mflop/s. Deliveries began in 1976, and by 1985 approximately 4400 machines had been delivered. The FPS-100 is a cheaper version of the AP-120B, made for inclusion as a part of other computer systems. The AP-120B was designed for attachment to minicomputers, and a version with more memory, called the AP-190L, was introduced for attachment to larger mainframe computers such as the IBM 370 series.

In 1980 the concept of the AP-120B was broadened from rather specialised signal processing applications to general scientific computing by increasing the word length from 38 to 64 bits, and the addressing capability from 16 to 24 bits. The memory capacity was also greatly increased, first to 1 Mword then to 7.25 Mword. The new machine which evolved was the FPS-164 which was first delivered in 1981. By 1985 about 180 FPS-164s had been sold. Although capable of solving much larger problems than the AP-120B, the FPS-164 was no faster at arithmetic—indeed its peak performance of 11 Mflop/s was 1 Mflop/s less than that of the AP-120B. The first improvement in arithmetic speed came in 1984 with an enhancement to the architecture called the matrix accelerator (MAX) board. Each such MAX board can be regarded computationally as the equivalent of two additional FPS-164 CPUs, so that a machine with the maximum of 15 MAX boards has a theoretical peak performance of 31 FPS-164 CPUs or 341 Mflop/s. The AP-120B and the FPS-164 are both implemented in low-power (and therefore low-speed) transistor–transistor logic (TTL), and the next improvement

†Headquarters: PO Box 23489, Portland, Oregon 97223, USA. UK Office: Apex House, London Road, Bracknell, Berkshire, UK.

in performance came in 1985 with the announcement of an ECL version of the FPS-164, called the FPS-264, which had a peak performance of 38 Mflop/s. The FPS-164 was designed in 1979 with the then current MSI of between 10 and 100 gates per chip. The machine has now been re-engineered in CMOS VLSI and is marketed as the FPS M30 computer, which may be attached to microVAX and Sun workstations. The architecture is the same as the FPS-164 which we describe here.

The FPS-164/MAX is a SIMD computer because the 31 CPUS in a full configuration operate in lock-step in response to a single stream of instructions. Another development of the AP-120B architecture has, however, been towards multi-instruction stream computing (MIMD). The FPS-5000 series of computers, announced in 1983, have a control processor, up to three arithmetic coprocessors and an I/O processor attached via a common bus to a common memory and a host computer. Each arithmetic coprocessor (AC) has its own control unit, and may be executing a different subroutine from the other ACS, thus providing a MIMD capability. According to the classification given in §1.2.6 the FPS-5000 is a bus-connected, shared-memory MIMD computer. The control processor of the FPS-5000 is either an AP-120B or FPS-100 computer. The XP-32 arithmetic coprocessor is of a new design due to Pincus and Kallio, but follows the same general pattern as the AP-120B.

All the above computers are called array processors because they are designed to process arrays of numbers efficiently. Architecturally, however, they are all pipelined computers with a small number of pipelined arithmetic units working from a common memory and registers. In this respect their architecture is CRAY-like. It is important to realise that although the above computers are called array processors, they are not arrays of processors like the ICL distributed array processor DAP (see Chapter 3). This ambiguity in the meaning of the expression 'array processor' has led us to avoid its use in this book. However, the term is commonly used for the computers described in this section and by the manufacturers. Some, however, prefer to interpret the initials AP to mean *attached processor*, since most require a host computer.

We start by describing in detail the father of the family, namely the AP-120B (§2.5.1 to §2.5.6), then follow with separate sections on the special features introduced in the FPS-164 and 264 (§2.5.7), the FPS 164/MAX (§2.5.8) and the FPS-5000 (§2.5.10). The use of multiple FPS-164s attached to a host, to form a loosely coupled array of processors (/CAP) is considered in §2.5.9.

2.5.1 FPS AP-120B

Apart from company documentation, the principal references describing the AP-120B are by Wittmayer (1978), Harte (1979) and Charlesworth (1981).

FIGURE 2.36 An overall view of an FPS AP-120B installation. The AP-120B occupies only 29 inches of rack space, and is attached to a PDP 11/34 with two disc units, a tape reader and output printer. The control teletype or VDU is not shown. (Photograph courtesy of D Head and Floating Point Systems, S A Ltd.)

The FPS AP-120B occupies 29 in of rack space on a standard EIA 19 in wide rack. Figure 2.36 shows a small installation comprising a PDP 11/34 host in the lower portion of the rack below the AP-120B, two disc units, a magnetic tape reader and output printer. The control teletype or VDU is not shown. The machine weighs about 160 pounds and consumes less than 1.3 kW (compared with about 115 kW for the CRAY X-MP). Cooling is by forced air driven by push–pull fans that are part of the AP-120B chassis. The machine can therefore be carried by one man, and plugged into a standard 13-amp domestic power socket. In contrast to the CRAY X-MP and CYBER 205, no auxiliary coolant plant or motor–generator sets are required to drive the machine.

Figure 2.37 shows a rear view of an AP-120B with a circuit board partially withdrawn. The cooling fans which can be seen at the top, blow air over the

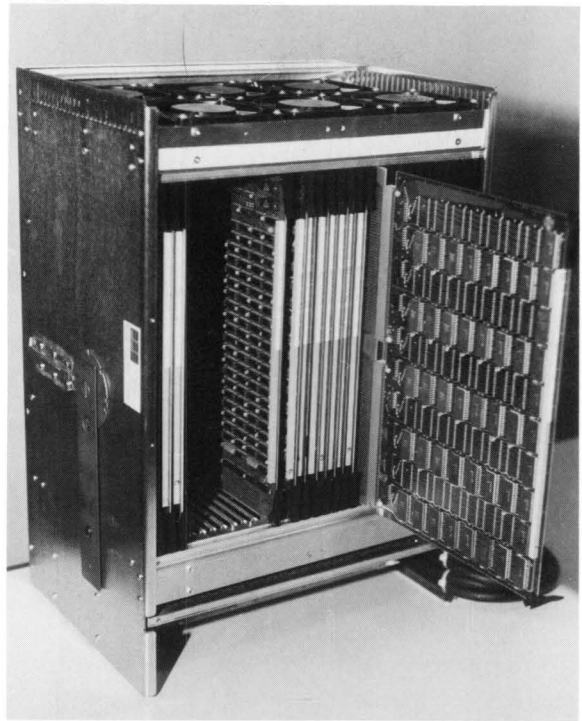

FIGURE 2.37 Rear view of the FPS AP-120B showing the vertically mounted 10 in × 15 in circuit boards and fan cooling. (Photograph courtesy of D Head and Floating Point Systems, S A Ltd.)

vertically mounted 15 in × 10 in circuit boards. The chassis has capacity for 28 etched-circuit boards which are chosen to suit the particular requirements for memory and input/output. Two of the circuit boards are shown in greater detail in figure 2.38. The circuit boards are six-layered, comprising three signal layers and three power supply layers for ± 5 V and $+12$ V. The power supplies, which are not shown in figure 2.37, are mounted on a separate power panel that occupies the rear part of the 29 in rack space behind the circuit boards.

2.5.2 Architecture

The overall architecture of the AP-120B is shown in figure 2.39. It is based on multiple special-purpose memories feeding two floating-point pipelined arithmetic units via multiple data paths. The machine is driven synchronously from a single clock with a period of 167 ns. This means that the state of the machine after a sequence of operations is always known and reproducible.

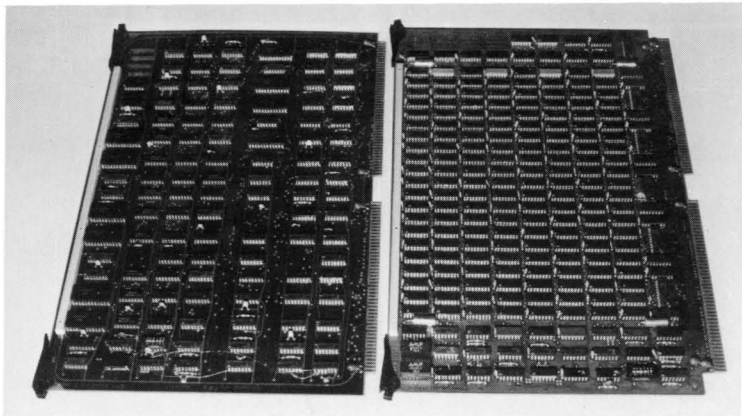

FIGURE 2.38 Two circuit boards from the FPS AP-120B. Left: the control buffer logic board from the control unit which decodes instructions; right: a board from the program memory which stores instructions. (Photograph courtesy of D Head and Floating Point Systems, S A Ltd.)

The operation of the machine can therefore be exactly simulated, clock period by clock period, and the machine does not suffer from the delicate timing uncertainties that had plagued some earlier computers which had separate clocks driving several independent units. Multiple data paths are provided between the memories and the pipelines, in order to minimise the delays and contentions that can occur if a single data path is shared between many units.

Starting at the top of figure 2.39 the memories are: a *program memory* of up to 4K 64-bit words for storing the program (cycle time 50 ns); a *scratch-pad* (S-pad) memory of 16 16-bit registers for storing addresses and indices; a *table memory* (167 ns cycle time, either read only or read/write memory) of up to 64K 38-bit words for storing frequently used constants, such as the sine and cosine tables for use in calculating a Fourier transform; two sets (*data pad X* and *data pad Y*) of 32 38-bit registers for storing temporary floating-point results; and a *main data memory* for 38-bit words (plus three parity bits), directly addressable to 64 Kwords but, with an additional 4-bit page address, expandable up to 1 Mword. Separate 38-bit data paths are provided to each of the two inputs to the floating-point adder and multiplier. These four independent paths may be fed from the main data memory, the data pads or from table memory. Three further 38-bit data paths feed results from the two pipelines back to their own inputs, or to the data pads or main data memory. These multiple paths allow an operand to be read from each data pad and a result written to each data pad during one machine cycle.

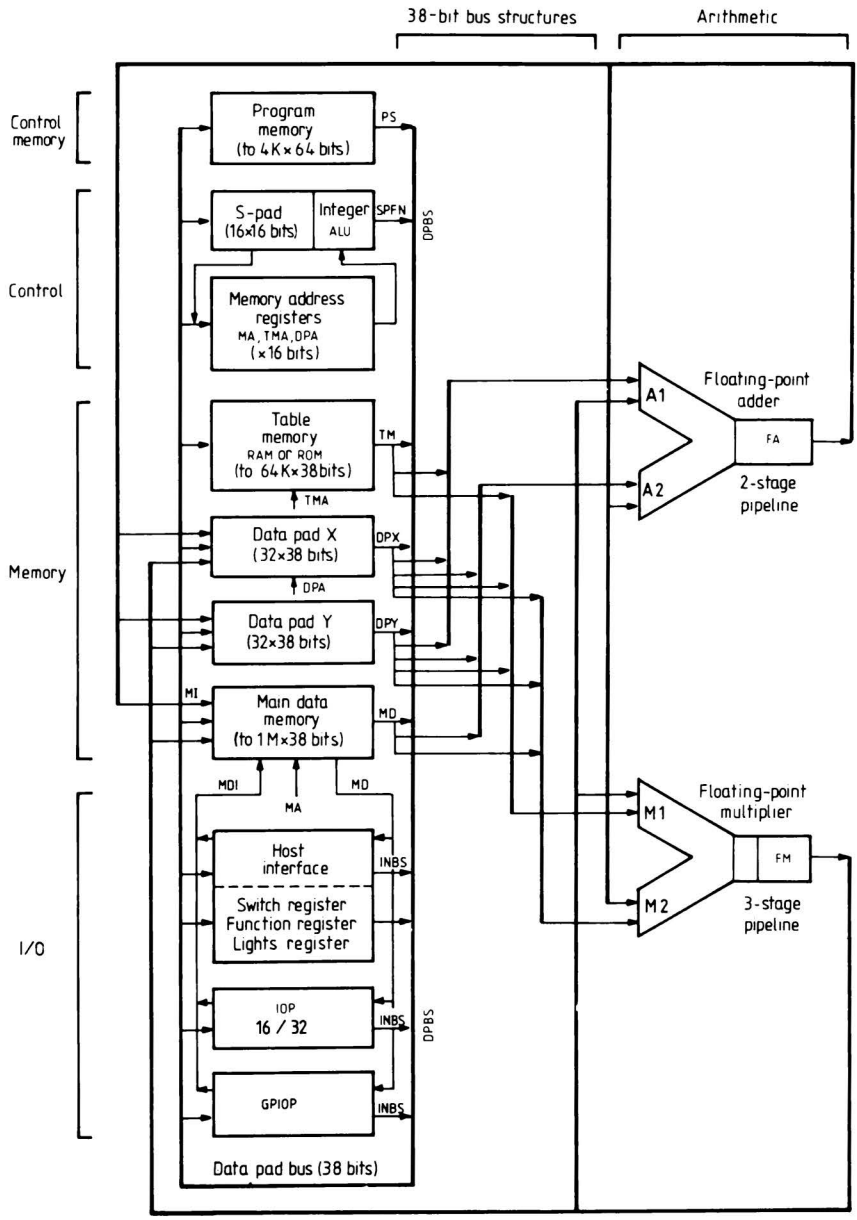

FIGURE 2.39 Overall architecture of the FPS AP-120B, showing the multiple memories, arithmetic pipelines and data paths. (Diagram courtesy of Floating Point Systems Inc.)

The address within both data pads is given by the contents of the data pad address register (DPA). Relative addressing (-4 to $+3$) with respect to this address is available separately for each data pad within the instruction in the data pad index fields XR, YR, XW, YW (see §2.4.4).

The main data memory is available in 8K modules (or 32K modules depending on the chip type) which are each organised as a pair of independent memory banks, one bank for the odd addresses and the other bank for the even addresses. The standard memory has an access/cycle time of 500 ns and the optional fast memory has a cycle time of 333 ns. Successive references to the same memory bank (e.g. all even addresses less than 8K) must be separated by at least three clock periods with standard memory or two clock periods with fast memory. Two successive references to different memory banks (e.g. two neighbouring addresses which are from odd and even banks, or two even addresses separated by 8K and therefore in different modules) may however be made on successive clock periods. Alternating references to the odd and even memory banks, as would occur when accessing sequential elements of a long vector, can occur at one reference per clock period for fast memory, giving an access to the same bank every 333 ns (matching the capability of the memory chip), and an effective minimum cycle time between requests to memory as a whole of 167 ns. For standard memory this rate must be halved, giving an effective memory cycle time for such optimal sequential access of 333 ns. If repeatedly accessing the same bank, a cycle time of 500 ns (three clock periods) applies. The memory is therefore described by the manufacturer as having an *interleaved* 'cycle' time of 167 ns for fast memory or 333 ns for standard memory, even though the memory chips have a physical cycle time of 333 and 500 ns respectively. However, it should be remembered, when making comparisons with other machines, that we have previously quoted the cycle time of the memory chips as a measure of the quality of the memory (e.g. 38 ns main memory of the CRAY X-MP, although this is organised into banks so as to give an interleaved 'cycle' time of 9.5 ns). If a memory reference to a part of the memory that is busy occurs, the machine stops execution until the memory becomes quiet.

Instructions on the AP-120B are 64 bits wide, and each instruction controls the operation of all units in the machine. Thus there is, in this sense, only one instruction in the instruction set (see §2.5.4) with fields which control each of the 10 functions, although some fields overlap and thus exclude certain combinations of functions. This arrangement of control is referred to as 'horizontal microcode'. Instructions are processed at the maximum rate of one per clock period, i.e. 6 million instructions per second, but since each instruction controls many operations this is equivalent to a higher rate on a conventional machine whose instructions only control one unit.

The S pad contains an independent 16-bit integer arithmetic and logical unit (ALU) for computing addresses, loop counts and indices in its set of 16 16-bit registers. The operations provided are: integer addition, subtraction; logical AND, OR, and equivalent; and shifts and bit reversal for use in Fourier transformation (see §5.5). All these operations take one clock period. There is no provision for multiplication. These integer operations are performed in parallel with floating-point calculations in the pipelines.

Both the addition and multiplication floating-point pipelines may take a new pair of operands every clock period and deliver a result every clock period. Thus the maximum rate of generation of results is 6 Mflop/s per pipeline, or 12 Mflop/s if both pipelines can be kept supplied with data. The performance on actual problems is more likely to be in the range 4–8 Mflop/s (see §2.5.6). The length of the multiplication pipeline is three clock periods (500 ns) and of the addition pipeline is two clock periods (333 ns). The addition pipeline also performs the logical operations of AND, OR and equivalence, absolute value, scale and number conversions between sign–magnitude and two's complement formats. The operation of the addition and multiplication pipelines is clarified by figures 2.40 and 2.41 which show the possible source and destinations of operands, the two input registers (A1, A2 or M1, M2), the partial operations performed at each stage of the pipelines, and the buffer registers for holding the intermediate partial results. The AP-120B does not provide a hardware divider, and floating-point division is accomplished in software by evaluating an approximating polynomial.

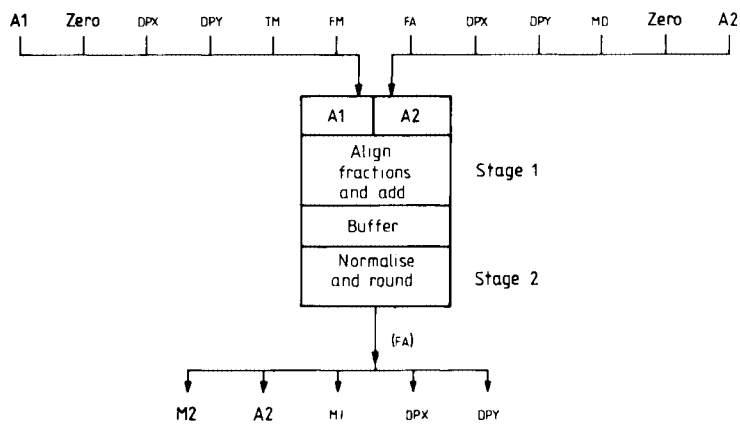

FIGURE 2.40 The two-stage addition pipeline of the FPS AP-120B. Note the possible sources of operands and destinations for results, using the notation of figure 2.39. Buffers are provided between each stage to store intermediate results. (Diagram courtesy of Floating Point Systems Inc.)

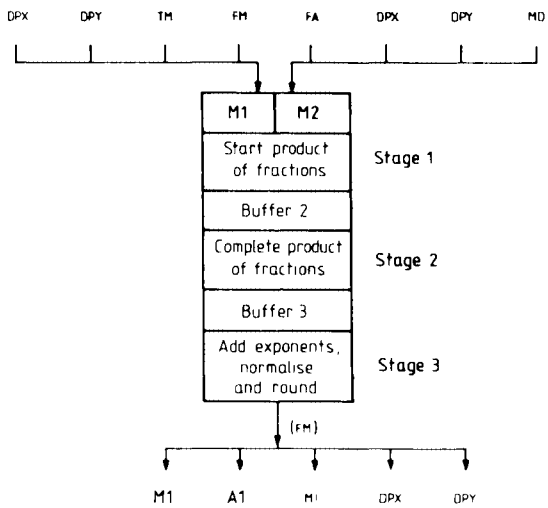

FIGURE 2.41 The three-stage multiplication pipeline of the FPS AP-120B. Notation as used in figure 2.39. (Diagram courtesy of Floating Point Systems Inc.)

Floating-point arithmetic is performed in a 38-bit format (10-bit radix-2 exponent and a 28-bit two's complement mantissa). This representation has a dynamic range $10^{\pm 153}$ and a precision of eight decimal places. It provides significantly more precision than IBM 32-bit radix-16 format ($10^{\pm 78}$ and six decimal places). Extra guard digits are also kept during the arithmetic operation in order to minimise loss of precision. The conversion of numbers between the format used by the host computer and the internal 38-bit format of the AP-120B takes place as the numbers are transferred between the machines.

Input and output to the AP-120B is performed by an I/O port (IOP) or a general programmable I/O port (GPIOP). The IOP provides either 16- or 38-bit direct memory access to the main data memory of the AP-120B, by 'stealing' memory cycles as required. Data transfer rates of 1.5 Mword/s into the AP and 1.3 Mword/s out of the AP are obtained. The 16-bit wide port is used for analog-to-digital input, display outputs and standard peripherals. The 38-bit wide port also contains a full adder and can be wired for a variety of data format conversions. As well as normal I/O this port can be used to link one AP-120B to another. The IOP occupies one circuit board of the AP-120B, and can accommodate up to 256 external devices. The GPIOP is a programmable I/O channel which provides up to 3 Mword/s continuous transfer with 'in-flight' format conversion including fixed-point to floating-point. It contains two 18 Mips microprocessors and occupies three circuit

boards. It is used typically to interface with discs, real-time displays, video cameras and other computers.

The architecture of the AP-120B with a 64K standard main data memory can be expressed in the ASN notation of §1.2.4 as follows:

$$C(\text{AP-120B}) = I1[\{Fp_{38}(+), Fp_{38}(*)\}_{\overline{7\times 38}}$$
$$\{P1, M1 - a, M2, M3, P2\}]_h$$

$$I1 = \{I_{64}^{167} - M4\}$$

$$P1(\text{S-pad}) = B_{16} - M_{16*16}$$

$$M1(\text{main data}) = 8\{2M_{4K*38}^{500}(\text{MOS})\}$$

$$M2(\text{table}) = M_{64K*38}$$

$$M3(\text{X, Y data pads}) = 2M_{32*38}$$

$$M4(\text{program}) = M_{4K*64}^{50}(\text{bipolar})$$

$$P2(\text{IOP}) = 256D - IO_{16} - a$$

2.5.3 Technology

The AP-120B is designed for reliability and therefore uses only well proven components and technologies, under conditions well clear of any operating limits. As a result mean time between failure (MTBF) of the hardware is typically several months to a year. The logic of the computer is made from low-power Schottky bipolar TTL (transistor–transitor logic) chips with a level of integration varying from a few gates per chip to a few hundred gates per chip. Typical gate delays in this logic technology are 3–5 ns. Various registers in the computer are also made in this logic technology. These are the S-pad and data-pad registers, and the subroutine return stack. The 50 ns program source memory and the 167 ns table memory both use 1K Schottky bipolar memory chips, whereas the slower and larger main data memory uses either 4K or 16K MOS memory chips.

It is interesting to compare the CRAY X-MP with the AP-120B from the point of view of technology, speed and power consumption, as they represent opposite extremes. The CRAY X-MP uses high-speed and high-power bipolar ECL technology with sub-nanosecond gate delays and a clock period of 9.5 ns, leading to the need for a large freon cooling system to dissipate a total of about 115 kW. The AP-120B on the other hand uses mostly low-power technology and consequently has a much longer clock period of 167 ns. However this permits the use of air cooling and limits the total power consumption to about 1.3 kW.

2.5.4 Instruction set

There are no vector instructions *per se* on the FPS AP-120B. Instead a 64-bit instruction is issued every clock period that has fields which control the operation of all units in the computer during that clock period. As an example, if the field FM (see below) controlling the multiplication pipeline is activated (i.e. bit 51 of the instruction is one), then all data in the multiplication pipeline are advanced to the next stage. The pipeline must therefore be activated three times to 'push' one pair of arguments completely through the three-stage pipeline and thereby complete one multiplication operation. A further activation of the pipeline is necessary for every subsequent element of a vector operation. This would most likely be set within a loop. The format of the single universal instruction is shown in figure 2.42. The notation for the data sources and destinations corresponds with figure 2.39. For a complete description of the instruction the reader is referred to the AP-120B Processor Handbook (FPS 1976a). In order to indicate the operation of the instruction, we give below examples of the uses of the data fields:

(1) S-pad group (control of 16-bit integer ALU and 16-bit registers)

SOP specifies dyadic S-pad operation, e.g. ADD, SUB, MOV, AND, OR, EQUIV; operands are SPS and SPD registers; result goes to SPD.

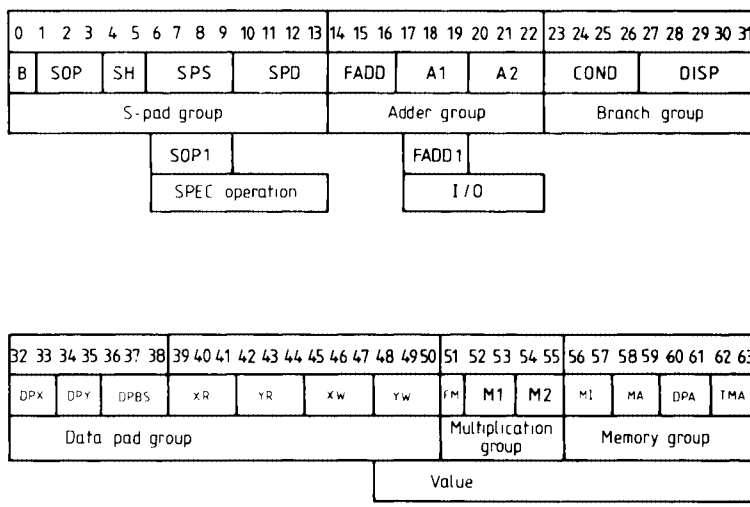

FIGURE 2.42 The data fields in the 64-bit instruction of the FPS AP-120B. This single instruction controls the operation of all units in the computer at every clock period. (Diagram courtesy of Floating Point Systems Inc.)

SPS number (4 bits) of 16-bit source register.
B if 1, reverses bits of source register before performing operation.
SPD number of 16-bit destination register.
SH specifies single left or right shift, or double right shift after operation.
SOP1 specifies monadic operations on data in SPD register when SOP = 0, e.g. increment (+1), decrement (−1) or complement SPD register.
SPEC control, conditional and branch instructions.

(2) Floating-point adder group

FADD specifies dyadic operations, e.g. FADD, FSUB, AND, OR, EQUIV using data in A1 and A2. Intermediate results are moved one segment down the pipeline to the next buffer register.
A1 source of data to be loaded into first adder input register, e.g. FM (multiplier output), DPX, TM.
A2 source of data to be loaded into second adder input.
FADD1 specifies monadic operations on data in A2 when FADD = 0, e.g. convert A2 to integer, sign–magnitude or two's complement; take absolute value.

(3) I/O group (controls I/O and transfers to and from buses), e.g.

DPBS → SPD data-pad bus contents to S-pad destination.
DPBS → TMA data-pad bus contents to table memory address register.
SPFN → PNLBS output of S-pad to panel bus.
INTA interrupt acknowledge; device address put in DPBS.

(4) Branch group

COND condition for branch, e.g. always, on flag, on arithmetic error, return jump from subroutine, on FA or SPFN =, ≠, ⩾, > 0.0.
DISP if branch true next address is current address + DISP − 16, relative jump of −16 to +15.

(5) Data-pad group (controls transfers to and from data pads X and Y)

DPA current data-pad address.
DPX load data pad X from DPBS, FA or FM.

DPY load data pad Y from DPBS, FA or FM.
DPBS selects DPX, DPY, MD, SPFN or TM to be sent to data-pad bus.
XR data-pad register with address DPA + XR − 4 is sent to DPX.
YR data-pad register with address DPA + YR − 4 is sent to DPY.
XW DPX is sent to data-pad register with address DPA + XW − 4.
YW DPY is sent to data-pad register with address DPA + YW − 4.

(6) Floating-point multiplier group

FM multiply or no operation. Intermediate results are moved one stage down the pipeline to the next buffer register.
M1 loaded from FM, DPX, DPY or TM.
M2 loaded from FA, DPX, DPY or MD.

(7) Memory group (controls transfers to and from main data and table memories)

MI load memory input register from FA, FM or DPBS.
MA increment or decrement memory-address register by one, or read from SPFN, and initiate data memory cycle.
DPA increment or decrement data-pad address by one or set address from SPFN.
TMA as DPA but for table memory.

2.5.5 Software

Software for the AP-120B, except for device drivers, is written in FORTRAN, so that it may be compiled to run on a variety of host computers. It may be subdivided into the following categories:

(1) operating system;
(2) program development software;
(3) application libraries.

The operating system consists of an executive APEX and a set of diagnostic routines APTEST. The executive controls transfers of data between the host and the AP-120B, transfers AP programs from the host to the AP program source (PS) memory and initiates the execution of programs in the AP. The operation of APEX is illustrated in figure 2.43. Most user programs will be FORTRAN programs that call either upon AP-120B maths library programs

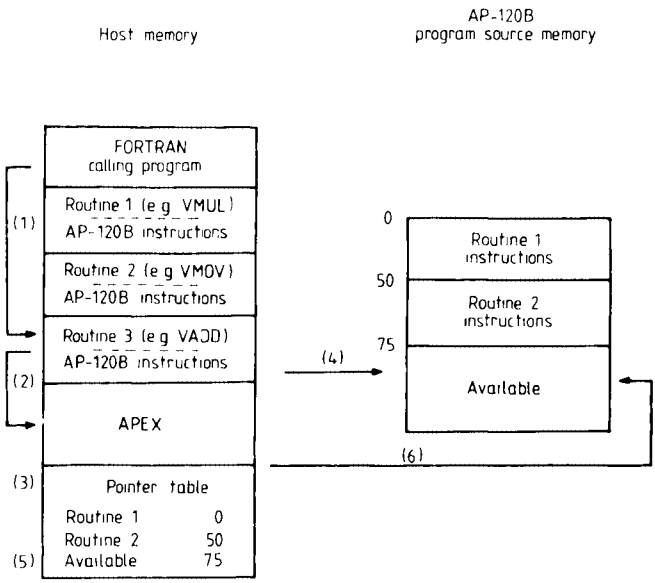

FIGURE 2.43 Diagram showing the action of the APEX executive during program execution (courtesy of Floating Point Systems Inc). Both AP-120B instructions and data are transferred between the host computer and the AP-120B under the control of APEX.

or user-supplied subroutines of AP-120B instructions in order to manipulate arrays in the AP-120B. APEX itself is a subroutine that is linked as part of the compiled FORTRAN program and runs on the host. It contains a table recording the location and contents of all AP routines that have already been loaded in the AP program source memory. The AP-120B instructions for each AP routine are stored as part of each subroutine in the host memory. Assuming that subroutines 1 and 2 have already been called, the following sequence of events takes place as the FORTRAN program executes in the host computer:

(1) FORTRAN program calls on AP using routine 3 (VADD);
(2) routine 3 calls APEX;
(3) PS memory table searched: routine 3 not in PS memory;
(4) APEX transfers AP-120B instructions from host to PS memory;
(5) PS memory table updated;
(6) APEX initiates execution of routine 3 in the AP-120B.

Data is transferred from the host to the AP-120B by calling the subroutine APPUT and results are transferred back with the subroutine APGET. Both these routines also call upon APEX to control the transfer. Once the execution of a program on the AP has begun, APEX returns control to the FORTRAN

calling program on the host, which may then proceed with other calculations that do not use the AP. If a call to another AP routine is met before the first has finished, APEX will wait for the first call to be completed.

The program development software comprises:

(1) *Mathematics library*—over 250 FORTRAN callable subroutines for the manipulation of arrays. The subroutines are written in assembler code and carefully optimised. Examples from the standard library are:

VADD, VMUL	element-by-element vector addition and multiplication;
SVE, DOTPR	sum of vector elements and dot product;
MMUL, MATINV	matrix multiplication and inverse;
CFFT, ACORT	complex fast Fourier transform and autocorrelation.

In addition there exists the advanced maths library which contains routines for function generation, binary search, tridiagonalisation, diagonalisation, solution of real and complex sparse systems of equations, and the solution of ordinary differential equations by Runge–Kutta integration.

(2) *APAL*—the assembler for AP-120B code which assembles programs on the host for subsequent execution on the AP.

(3) *APLOAD*—links separate APAL object modules into a single module for execution on the AP.

(4) *APSIM, APDBUG*—simulates an AP program and allows debugging on the host or AP respectively.

(5) *VFC*—vector function chainer. Consolidates multiple calls to maths library routines into a single call and thereby reduces calling overhead.

(6) AP FORTRAN—a compiler running on the host that accepts FORTRAN IV and produces code for execution on the AP.

The above software allows programs to be prepared in a variety of ways. In the order of decreasing speed in execution and increasing ease of preparation, these methods are likely to be: APAL assembler, vector function chainer, maths library routines and AP FORTRAN.

Application libraries are available for:

(1) *SIGLIB*—signal processing library with routines for histograms, hanning windows, correlations, transfer and coherence functions etc.

(2) *IMP*—image processing library with routines for 2D fast Fourier transform and convolution, image filtering etc.

(3) *AMLIB*—advanced maths library with routines for function generation, Runge–Kutta integration, sparse matrix solution and matrix eigenvalues.

2.5.6 Performance

The AP-120B does not include a real-time clock and it is therefore impossible to time the execution of programs accurately. Attempts to time programs by using the clock on the host computer are usually imprecise and variable because of the effect of the host operating system. This is particularly the case if a time-sharing system is in use. In estimating the performance of the AP-120B we are therefore forced to rely on the timing formula given in the AP-120B maths library documents. The document (FPS 1976b) which we use gives timing formulae that may be related directly to formula (1.4a) defining r_∞ and $n_{1/2}$. The minor differences from later documents (FPS 1979a, b) are unimportant. Because of the synchronous nature of these machines theoretical timings should be reliable; however the absence of a clock makes the optimisation of large programs very difficult. The detailed timing of large programs soon becomes tedious and error prone. An alternative is to simulate the execution of the AP program on the host computer using the program APSIM (see §2.5.5). This program produces the theoretical program timing but again may be impractical for the timing of large programs because it runs about 1000 times slower than the program would execute on the AP-120B itself.

Using the maths library documents (FPS 1976b) we give the timing formulae for a selection of simple vector operations, and derive from them estimates for $n_{1/2}$ and r_∞. We quote the timing formulae for the standard memory (500 ns chip cycle time) and give the improved values of r_∞ for the fast memory (333 ns chip cycle time) in parentheses. Where there is a small timing variation because of the choice of odd or even memory locations for the vectors, we have taken the minimum timing. None of these minor timing alternatives substantially change the character of the machine, and they can largely be ignored.

(1) Vector move CALL VMOV (A, I, C, K, N)

$$C_{mK} \leftarrow A_{mI}, \qquad m = 0, 1, \ldots, N-1$$

Memory-to-memory move of vector **A** to **C**. K and I are the memory increments between successive elements of **A** and **C** respectively. The time for N operations is:

$$t = \tfrac{2}{3}(N+1)\ \mu\text{s},$$

hence

$$r_\infty = 1.5(3.0)\ \text{Mop/s}, \qquad n_{1/2} = 1.$$

We note that this operation is memory bound and the transfer rate doubles for the fast memory. However $n_{1/2}$ is unaffected by the memory type.

(2) Vector addition CALL VADD (A, I, B, J, C, K, N)

$$C_{mK} \leftarrow B_{mJ} + A_{mI}, \qquad m = 0, 1, \ldots, N-1.$$

The time for N operations is:

$$t = N + 1 \ \mu s,$$

therefore

$$r_\infty = 1(2) \text{ Mflop/s}, \qquad n_{1/2} = 1.$$

(3) Vector multiplication CALL VMUL (A, I, B, J, C, K, N)

$$C_{mK} \leftarrow B_{mJ} * A_{mI}, \qquad m = 0, 1, \ldots, N-1.$$

The time for N operations is:

$$t = N + 2 \ \mu s,$$

therefore

$$r_\infty = 1(2) \text{ Mflop/s}, \qquad n_{1/2} = 2.$$

(4) Vector division CALL VDIV (A, I, B, J, C, K, N)

$$C_{mK} = B_{mJ} / A_{mI}, \qquad m = 0, 1, \ldots, N-1.$$

The time for N operations is:

$$t = 1.83(N + 3) \ \mu s,$$

therefore

$$r_\infty = 0.55(0.55) \text{ Mflop/s}, \qquad n_{1/2} = 3,$$

and we see that, because the calculation is dominated by arithmetic, the faster memory does not increase the performance.

(5) Vector exponential CALL VEXP (A, I, C, K, N)

$$C_{mK} \leftarrow \exp(A_{mI}), \qquad m = 0, 1, \ldots, N-1.$$

The time for N exponentials is:

$$t = 4.87(N + 0.3) \ \mu s,$$

therefore

$$r_\infty = 0.2 \text{ Mflop/s}, \qquad n_{1/2} = 0.3.$$

(6) Dot product CALL DOTPR (A, I, B, J, C, N)

$$C \leftarrow \sum_{m=0}^{N-1} A_{mI} * B_{mJ}.$$

The time for $2N$ operations is:

$$t = \tfrac{2}{3}(N + 2)\ \mu s,$$

therefore

$$r_\infty = 3(6)\ \text{Mflop/s}, \qquad n_{1/2} = 2.$$

The above selection of timings and performances, which may be considered typical for simple memory-to-memory vector operations, shows that only a small fraction of the potential performance of 12 Mflop/s can be realised for such operations. This is because the memory bandwidth is insufficient to support a memory-to-memory processing rate of this magnitude. A single vector operation has two input vector operands and one output result vector. Therefore a memory-to-memory processing rate of 12 Mflop/s requires a memory bandwidth of 36 Mword/s. The standard memory on the AP-120B has a bandwidth varying between 2 and 3 Mword/s depending on whether references are all to the same memory bank or alternate between different memory banks. The fast memory has similarly a bandwidth varying between 3 and 6 Mword/s. Thus it is evident that the memory bandwidth is only about one-tenth of that required to sustain both the addition and multiplication pipelines working with data from main memory.

The most likely constraint on the realisation of fast processing rates on the AP-120B is therefore the low memory bandwidth. In order that memory bandwidth does not become a bottleneck, it is necessary for the *computational intensity* (see p106), f, to be at least two floating-point operations per memory reference (flop/ref) with fast memory, or at least 4 flop/ref with standard memory. These conditions are realised for more complicated algorithms and large enough problem sizes, n: e.g. matrix multiplication, $f = 2n/3$ flop/ref; fast Fourier transform (FFT), $f = 1.25 \log_2 n$ flop/ref.

The FFT algorithm is worthy of closer examination because it is the main algorithm around which machines like the AP-120B were designed. Throughout the algorithm operations are of the 'butterfly' type (see equation (5.87)):

$$c = a + wb, \tag{2.17a}$$

$$d = a - wb, \tag{2.17b}$$

where a and b are complex elements from main memory, and c and d are complex results to be returned to main memory. The complex constant w may reside in a register. Equations (2.17) require one complex multiplication ($s = w*b$) and two complex additions ($a + s$, $a - s$) for four memory references to complex numbers. This is the equivalent of 10 real arithmetic operations for eight real memory references, or $f = 1.25$ flop/ref. The above

considerations are for the radix-2 transform and show that this algorithm does not have a high enough computational intensity to keep the arithmetic pipes busy. By combining two levels of the FFT together, we obtain the radix-4 algorithm and increase the computational intensity to 2.5 flop/ref, which is a figure satisfying the conditions given above for the fast memory. We find that the maths library subroutine CFFT does use the radix-4 algorithm and gives a performance of 8 Mflop/s.

The values of half-performance length found above are in the range $n_{1/2} = 1-3$, showing that the AP-120B, although it has many parallel features, actually behaves very similarly to a serial computer. In this respect the computer is similar to the CRAY-1 ($n_{1/2} \sim 10$) and quite different from the CYBER 205 ($n_{1/2} \sim 100$) or ICL DAP ($n_{1/2} \sim 1000$). The selection of the best algorithm is normally determined by the value of $n_{1/2}$ (see Chapter 5) and we would expect algorithms optimised to perform well on a serial computer also to perform well on the AP-120B. However, as has been emphasised above, the performance of a program may be more dependent on the management of memory references than on the questions of vector length that are addressed by the value of $n_{1/2}$.

2.5.7 FPS-164 (renamed M140 and M30) and 264 (renamed M60)

As can be seen from figure 2.44 the FPS-164 is substantially larger than the AP-120B, being about 5.5 ft high and occupying about 2.5 ft × 7 ft of floor space, principally because of the need to accommodate a much larger memory. The same cabinet is also used for the FPS-164/MAX and FPS-264. The principal improvements introduced in the FPS-164 (compared with the AP-120B) are:

(a) 64-bit floating-point arithmetic compared with 38-bit;
(b) 32-bit integer arithmetic compared with 16-bit;
(c) 24-bit addressing to 16 Mword compared to 16-bit addressing to 64 Kword only;
(d) 64-bit X- and Y-pad data registers compared with 32-bit;
(e) 64 32-bit S-pad address registers compared with 16 16-bit;
(f) 1024 64-bit instruction cache replacing program memory;
(g) 256 32-bit subroutine return address register stack;
(h) main memory expandable from 0.25 to 7.25 Mword with memory protection;
(i) table memory of 32 Kwords RAM;
(j) a clock for timing programs—sadly lacking on the AP-120B.

The increase in arithmetic precision and addressing range generally lift the

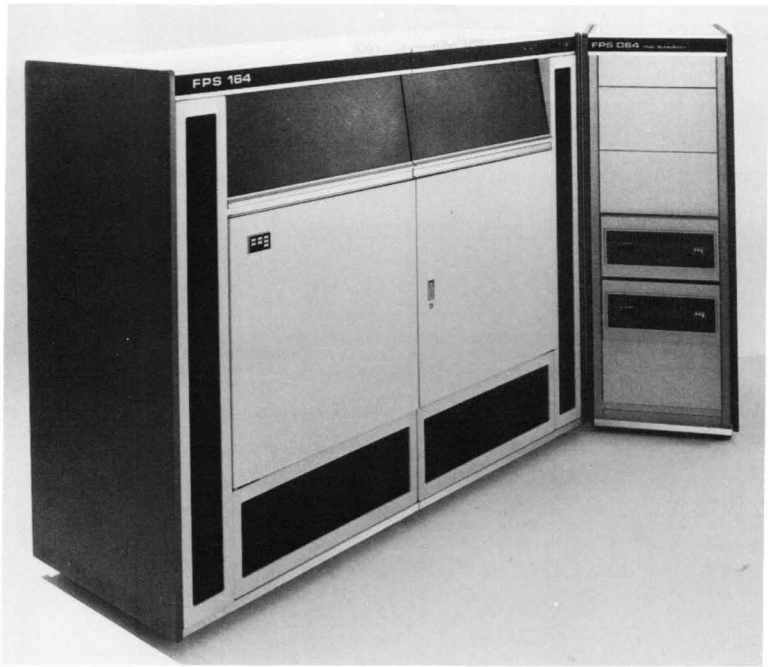

FIGURE 2.44 Overall view of the FPS-164.

specification of the computer up from that of a minicomputer to that of current large mainframe computers. The way these features fit into the overall architecture can be seen in figure 2.45. In contrast to the AP-120B, both instructions and data share the same main memory, and instructions are automatically read into the instruction cache as needed. This cache memory therefore replaces the separate program memory of the AP-120B. Subroutine referencing is made more efficient on the FPS-164 by the inclusion of the subroutine stack which provides storage for 256 32-bit subroutine return addresses. The table memory, also called auxiliary memory, has become primarily a random access memory for use as temporary register storage for intermediate results. The first 8K of this memory is however reserved for read-only constants of which about 5K are assigned, the next 16K or 32K (depending on the option purchased) may be used as random access memory by user programs. The main memory is organised into modules, each with even and odd memory banks, as in the AP-120B. The original FPS-164 used 16K-bit dynamic NMOS chips and had 12 memory modules occupying 24 memory boards (each board a bank), giving a capacity of 1.5 Mword. Subsequently, the use of 64K-bit memory chips has enabled the memory

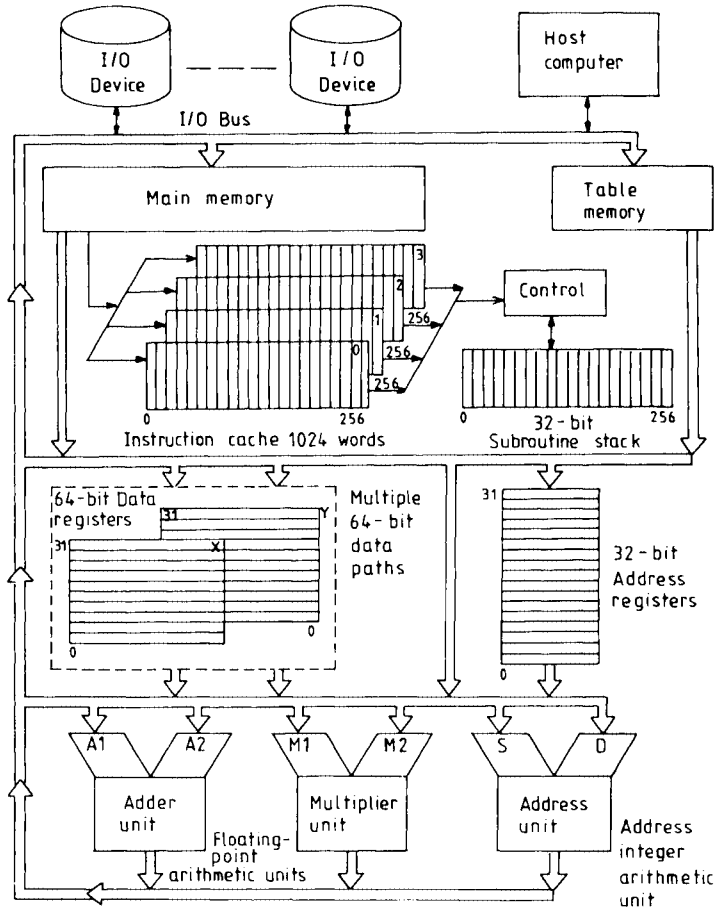

FIGURE 2.45 Overall architecture of the FPS-164.

capacity to be increased to 7.25 Mword. Main memory acts as a three-stage pipeline, and successive requests to the same bank may occur every other clock period. Table memory acts as a two-stage pipeline, and successive requests may occur on successive clock cycles. A new feature on main memory is memory mapping and protection. If this is available, the contents of the MDBASE register are added to the requested address on line MA (see figure 2.39) to form a physical address. If this exceeds the contents of the MDLIMIT register no memory access takes place: a read produces zeros and a write is ignored.

The FPS-164 is said to have a nominal clock period of 167 ns like the AP-120B. However, the actual clock period used is slightly longer at 182 ns, which leads to an asymptotic pipeline rate of 11 Mflop/s. The machine is

also provided with a programmable real-time clock and a CPU timer that increments every clock period of the computer. This enables accurate timings to be made within user programs.

The theoretical arithmetic performance of the FPS-164 is 11/12 of the theoretical figures given for the AP-120B. With the availability of the CPU timer, accurate benchmarking is possible and we quote results obtained by Thompson (private communication). The result of the $(r_\infty, n_{1/2})$ benchmark for memory-to-memory arithmetic operations is given in table 2.8. The results for straight FORTRAN code (top number of pair), and for the use of the optimised mathematics library routines (bottom numbers) are compared. As expected from the theoretical timings for the AP-120B, a performance of about 1 Mflop/s is to be expected for dyadic operations. Except for the dyadic add operation, there is nothing to recommend the use of the library routines, because the same performance can be obtained using FORTRAN, and the overhead of the operation is less using FORTRAN as can be seen from the smaller value of $n_{1/2}$. A dyadic operation is the worst case for performance because there is no opportunity to store intermediate results in fast access registers, and the number of floating-point operations per memory reference, f, is 1/3 only. The last two cases of the triad and three-ops raise f to 2/3

TABLE 2.8 Benchmark results of $(r_\infty, n_{1/2})$ for memory-to-memory operations on the FPS-164. Upper case variables are vectors, lower case are scalars. (Data courtesy of Bill Thompson, TUCC.) (Upper numbers FORTRAN, lower numbers library routine or higher optimisation level, OPTC.)

Operation: statement 10 program (1.5)	r_∞ (Mflop/s)	$n_{1/2}$
A = B + C	0.88	5
CALL VADD	1.06	16
A = B*C	1.07	5
CALL VMUL	1.04	17
A = B/C	0.30	7
A = b*(C − D)		
OPTC = 1	0.8	—
OPTC = 3	3.4	—
A = B + C*(D − E)		
OPTC = 1	1.0	—
OPTC = 3	3.2	—

and 3/4 respectively. With the compiler optimisation level (OPTC) equal to one, which performs local optimisation only, the dyadic performance of about 1 Mflop/s is obtained. However, if OPTC = 3, software pipelining is employed to overlap operations and the performance is raised three-fold.

Performance on more substantial benchmarks, the Livermore loops and LINPACK, are given in table 2.9. Broadly speaking, a performance on actual problems between 1 and 5 Mflop/s is typical, depending on the care with which the problem is programmed.

The FPS-264 is an ECL logic implementation of the FPS-164 architecture, which allows the clock period to be reduced from 182 ns to 53 ns, a ratio of 3.4. Other improvements to the instruction cache and main data memory lead to a claimed performance ratio of between four and five times the FPS-164. Logic uses air-cooled custom Fairchild 100K ECL chips (compared with Schottky TTL on the FPS-164), and the main memory uses 64K-bit static NMOS chips (compared with dynamic NMOS on the FPS-164). The latter allows the memory to be packaged with 0.5 Mword per memory board, divided into two banks. A maximum main data memory of 4.5 Mword divided into 18 banks was available on the first machines. The instruction cache is divided into two interleaved banks of 4 Kword, giving a total of 8 Kword, compared with 1 Kword on the FPS-164. The FPS-264 has the same external appearance as the FPS-164, being packaged in the same cabinet (see figure 2.44). The reported performance of the FPS-264 on the Livermore loops and the LINPACK benchmark is given in table 2.9, and generally supports the assertion that the FPS-264 can be expected to perform about four times faster than the FPS-164.

2.5.8 FPS-164/MAX (renamed M145)

A novel enhancement to the FPS-164 was announced in 1984, the FPS-164/MAX which stands for matrix algebra accelerator (Charlesworth and Gustafson 1986). This machine has a standard FPS-164 as a master, and may add up to 15 MAX boards, each of which is equivalent to two 164-CPUs with the addition of four vector registers of 2048 elements in each CPU. In total there is therefore the equivalent of 31 164-CPUs or 341 Mflop/s.

The architecture of the MAX board is shown in figure 2.46. The board contains two CPUs, each with an eight-stage 64-bit floating-point multiplier which feeds its results into an eight-stage floating-point adder. One input to every multiplier in the machine (and there are 31 multipliers in a 15-board machine) is broadcast from main data memory, whilst the other input comes from the local scalar or vector registers on the MAX board. Similarly, the second input to the adder comes from the local registers. In the broadcast operation the same number is sent simultaneously from main data memory

TABLE 2.9 Performance of a selection from the Livermore and LINPACK benchmarks on the FPS-164, 264 and FPS-164/MAX. Figures are Mflop/s for 64-bit arithmetic.

Problem	FPS-164	FPS-264	FPS-164/MAX[f]
Theoretical peak	—	—	33(1)
performance	—	—	99(4)
	11	38	341(15)
$(r_\infty, n_{1/2})$ FORTRAN §1.3.3	(1.07, 5)	—	—
Livermore 3 inner product	3.0[e]	—	—
Livermore 6 tridiagonal	1.1[e]	—	—
Livermore 14 particle pusher	1.5[e]	—	—
LINPACK[a] FORTRAN[b] $n = 100$	1.4	4.7	—
Assembler[c] inner loop	—	—	6(1)[g]
$n = 100$	2.9	10	20(15)[g]
Matrix–vector[d]	—	—	15(1)[h]
best assembler			26(4)[h]
$n = 300$	8.7	33	—

Notes

a Solution of linear equation using DGEFA and DGESL for matrices of order 100 (Dongarra *et al* 1979).
b All FORTRAN code (Dongarra 1985).
c BLAS routines optimised in assembler (Dongarra 1985).
d FORTRAN matrix–vector method of Dongarra and Eisenstat (1984). Matrix order 300. Best reported assembler (Dongarra 1985).
e Gustafson 1985.
f Number of MAX boards used in parentheses.
g FPS 1985b.
h Dongarra 1986.

to all the multipliers. The clock period of the MAX board is the same as the FPS-164 main CPU, namely 182 ns, hence each board has a peak performance of 22 Mflop/s. The logic of the FPS-164/MAX is implemented in CMOS VLSI,

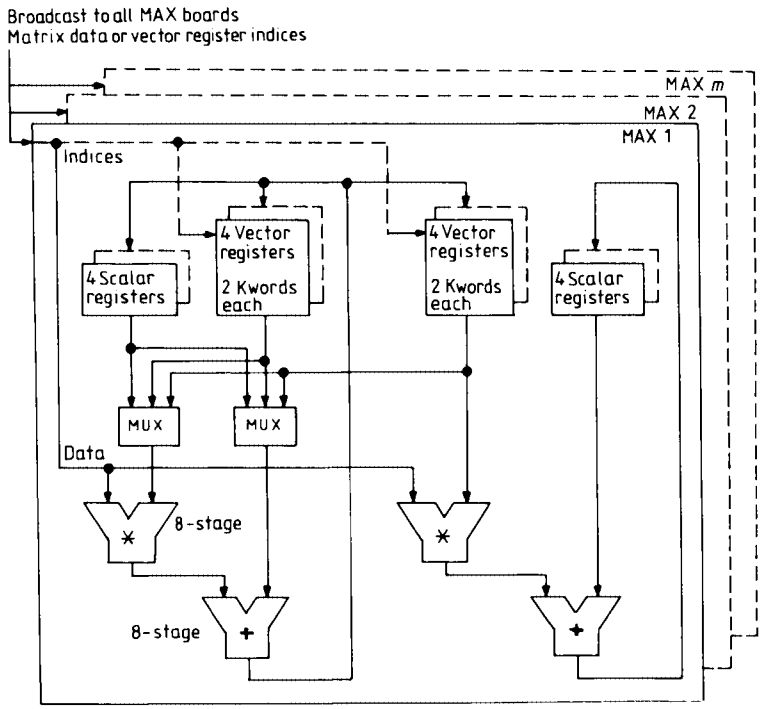

FIGURE 2.46 Architecture of a matrix accelerator (MAX) board.

and the arithmetic pipelines of the MAX board (shown in figure 2.47) use the pipelined WEITEK arithmetic chips.

MAX boards occupy memory board positions of the FPS-164, and it is possible to upgrade an existing FPS-164 to an FPS-164/MAX. The MAX boards look to the host computer to be the top 1 Mword of the 16 Mword of its address space, leaving a maximum addressable normal memory of 15 Mword. The FPS-164/MAX uses the same memory board as the FPS-264, with 0.5 Mword per board of static NMOS chips. A full FPS-164/MAX has 29 memory board slots of which 14 are used to hold the 7 Mword of physical main data memory, and 15 slots are used for the 15 MAX boards. The availability of 256 K-bit static NMOS chips will allow 1 Mword per memory board and a physical memory size of 15 Mword, to match the full addressing capability.

The idea of the MAX board is to speed up the arithmetic in a nest of two or three DO loops, such as one finds in many matrix operations, in particular in the code for matrix multiply. In this example the 31 164-CPUs of a full system would be used to simultaneously calculate the 31 inner products that are required to produce 31 elements in a column of the product matrix. The FPS architecture is already optimised for the efficient calculation of inner

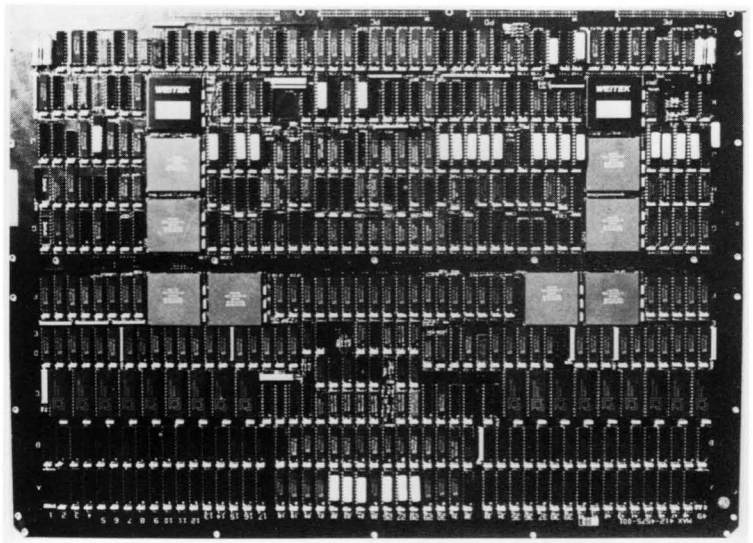

FIGURE 2.47 A MAX board.

products, and the only problem is to ensure that the required data is available to the pipelines. One of the vector registers of each of the 31 CPUs receives one of the 31 rows of the first matrix, whilst the elements of the column of the second matrix are broadcast, one-by-one, to all CPUs as the 31 inner products are accumulated, as is illustrated in figure 2.48. This produces 31 elements in the corresponding column of the result matrix. All other columns for the same 31 rows can be computed without altering the contents of the vector registers, and it is only when this saving in data movement is possible that the FPS 164/MAX can approach its maximum performance. Fortunately many important problems in linear algebra (solving equations, eigenvalues etc) can be formulated to satisfy this condition. At this stage in the calculation of the matrix product, 31 rows of the product matrix have been computed. The vector registers must now be refilled in order to compute the next 31 rows, until the product matrix is completed.

The kernel of the above algorithm is the multiplication of the (31 × 2048) matrix **A** by the (2048 × 2048) matrix **B**, to give the (31 × 2048) product matrix **C**. We assume **C** is initially cleared and transferred to main data memory after the execution of the following code

```
DO 1   J = 1, 2048
DO 1   K = 1, 2048
DO 1   I = 1, 31
1   C(I,J) = C(I,J) + A(I,K)*B(B,J)
```
(2.18)

where I is the CPU-number.

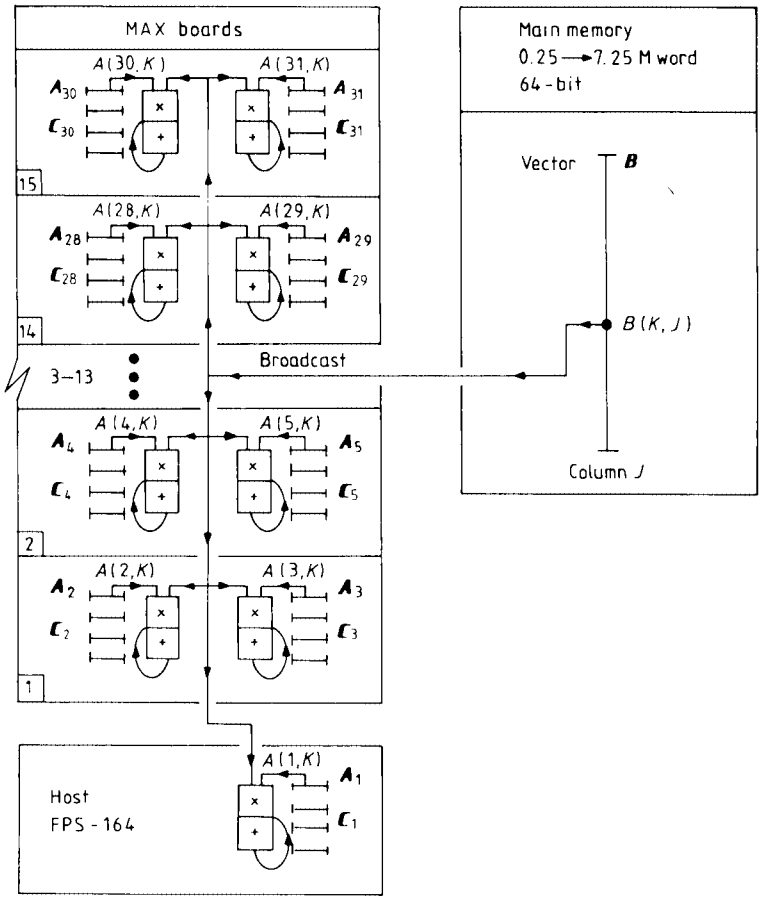

FIGURE 2.48 The simultaneous calculation of 31 inner products, $(A_i.B; i = 1, 31)$ using 15 MAX boards. The elements of B are broadcast one-by-one to all boards, and each board accumulates two inner products $(S_i = S_i + A_{ik}B_{k,j}; k = 1, 2048)$.

The DO-I loop is implemented by the broadcast of the scalar quantity B(K,J) to all 31 CPUS, multiplying it by the A(I,K) which is an element taken from the local vector A_i, and accumulating the inner product in C(I,J) which is taken from the local vector C_i. In these operations all the 31 CPUS work in lockstep (i.e. in unison) but on their own individual data (I is the CPU-number), that is to say the control is SIMD. The DO-K loop accumulates the inner product, and the DO-J loop moves from column to column. All

the three DO loops in (2.18) can be executed without transferring data between main data memory and the MAX board registers, and this is the essential requirement for efficient MAX board utilisation. This is expressed by saying that there must be re-use of data in both space and time if the peak speed is to be reached. Re-use in space refers to the broadcast of a single quantity B(K,J) simultaneously to the 31 CPUs in the DO-I loop, and re-use in time refers to the fact that C(I,J) and A(I,K) are continually re-used from local memory in the DO-K and DO-J loops.

The purpose of the re-use facility is to limit the need for transfers between the main data memory and the MAX boards, and to perform the maximum amount of arithmetic between such transfer so as to dilute the penalty of loading the registers. We have expressed this before by the computational intensity, f, which is the number of floating-point operations per memory reference. In the above matrix multiply example we have two floating-point operations per execution of statement 1, and the references are the read of **A** and **B**, and the store of **C** (there is hardware provision for the initial clearing of **C**). Thus

$$f = (2 \times 31 \times 2048 \times 2048)/(2110 \times 2048) = 60. \tag{2.19}$$

This is to be compared with the hardware parameter, $f_{1/2}$, which is half the ratio of asymptotic arithmetic performance to memory bandwidth in the relevant case of memory transfer overlap which occurs on the FPS-164 (see §1.3.6 and equation (1.20)). The memory bandwidth to the MAX boards, r_∞^{m}, is one word per clock period, and the maximum arithmetic rate, r_∞^{a}, is 62 arithmetic operations per clock period, hence

$$f_{1/2} = 31. \tag{2.20}$$

The expected average performance can be estimated from (1.22b) as

$$\bar{r}_\infty = r_\infty^{\mathrm{a}}\ knee\ (0.5f/f_{1/2}) \qquad x = f/f_{1/2} \tag{2.21a}$$

whence

$$x = 1.9 \qquad \bar{r}_\infty = 0.97 r_\infty^{\mathrm{a}}. \tag{2.21b}$$

Thus we find that when the conditions for re-use in space and time are satisfied, a performance within 3% of the maximum peak performance is possible.

The MAX boards can only execute a limited number of instructions of the type that are given in table 2.10. The operation of the MAX boards and their registers are memory-mapped onto the top Mword of the addressable 16 Mword, as shown in figure 2.49. That is to say that the boards are operated simply by writing and reading to appropriate parts of the upper Mword of

TABLE 2.10 The instructions that may be executed by a MAX board, and the peak performance in Mflop/s for 1 and 15 boards. The same performance applies to both real and complex arithmetic. Full vector operations use J(I) = I.

		MAX boards	
Name	FORTRAN program line	1	15
Dot product	S = S + A(I)*B(J(I))	22	341
Complex dot product	S = S + A(I)*B(I)	22	341
VSMA	A(J(I)) = S*B(J(I)) + C(I)	11	167
VMSA	A(J(I)) = B(J(I))*C(I) + S	11	167
Vector mult	A(J(I)) = B(I)*C(J(I))	5.5	83
Vector add	A(J(I)) = B(I) + C(J(I))	5.5	83

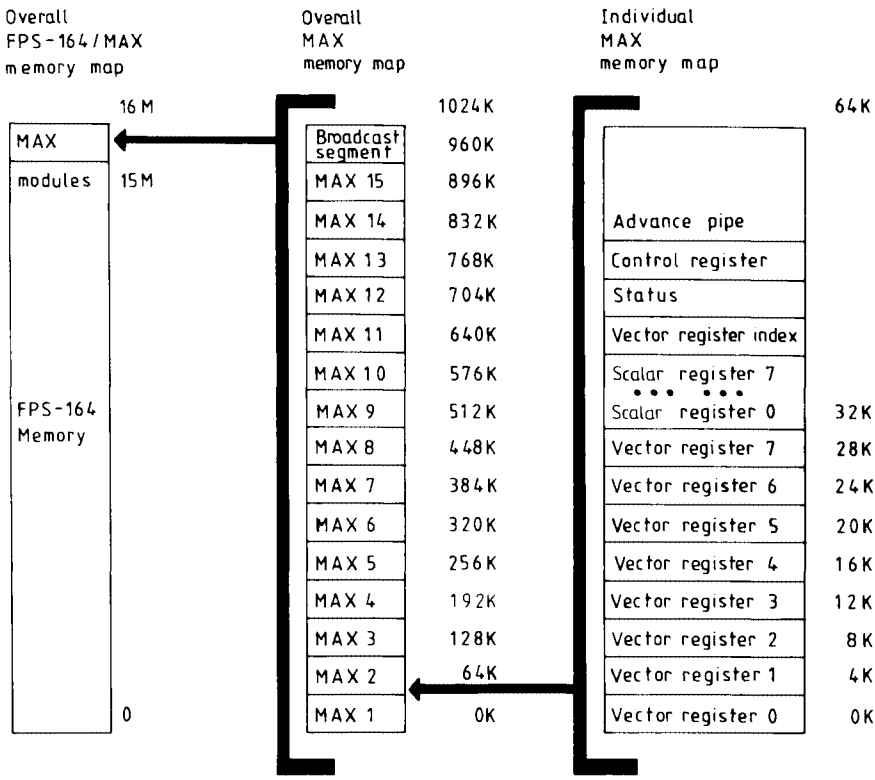

FIGURE 2.49 The memory mapping of MAX boards onto 16 Mword address space of an FPS-164.

the FPS-164 address space. This is divided into 16 individual MAX memory maps of 64 Kword each. The first 15 of these maps operate the 15 MAX boards individually, and the last is the broadcast segment which operates all the boards in unison. The first 32 Kword of the memory map addresses is the vector register storage which allows for eight registers of 4K elements each. The first MAX implementation, however, limits the vector length to 2K elements. Next are the eight scalar registers and the vector index registers. The MAX board is operated by placing appropriate words in the 'advance pipe' section.

The nature of the software that is available for driving the FPS-164/MAX can be seen from the following FORTRAN code that implements the matrix multiply discussed above

```
      CALL SYS$AVAILMAX(NUMMAX)
      MAXVEC = 8*NUMMAX + 4
      NUMVEC = MAXVEC

      DO 10 I = 1, N, MAXVEC
        NUMVEC = MIN(NUMVEC, N - I + 1)                    (2.22)
      IF(NUMVEC .LE. 0) GOTO 10
        CALL PLOADD(A(I,1),N, 1, NUMVEC, ITMA, 1, IERR)
        DO 20 J = 1, N
          CALL PDOT(B(1,J),1, N, C(I,J), 1, NUMVEC, ITMA, 1, 0, IERR)
20      CONTINUE
10    CONTINUE
```

The maximum performance is achieved by using as many vector registers as possible. The first statement in the above code obtains in NUMMAX the number of MAX boards that are mounted on the system. Since each contains eight vector registers, and there are four vector registers on the host FPS-164, NUMVEC is the number of vector registers. The DO-I loop loads NUMVEC rows of the matrix **A** into the available vector registers in preparation for accumulating NUMVEC inner products. The CALL PDOT forms the inner product with the Jth row, and this is repeated for all the rows by the DO-J loop.

2.5.9 IBM /CAP and Cornell systems

Both IBM and Cornell University are developing replicated MIMD computing

systems, based on linking together multiple FPS processors. The IBM loosely coupled array of processors (/CAP) is the brainchild of Enrico Clementi and is installed at the IBM Kingston laboratory. In 1985 a similar configuration was installed at the IBM Scientific Center, Rome, to be the first computational heart of the newly set up 'European Center for Scientific and Engineering Computing' (ECSEC).

A simplified drawing of the IBM /CAP computer system is shown in figure 2.50. Ten FPS-164 computers, each with 4 Mbytes of main data memory are connected by 2–3 Mbyte/s channels to IBM host computers (Berney 1984). Seven are connected to an IBM 4381 for computational work and three to an IBM 4341 for program development, although all ten can be switched to the IBM 4381, making a system with a theoretical peak performance of 110 Mflop/s. The actual performance on quantum chemical problems for the ten-FPS-164 configuration is reported to be about the same as a CRAY-1S, or about 60 Mflop/s (Clementi *et al* 1984). Possible enhancements to the initial configuration involve the addition of two MAX boards to the ten FPS-164s, which gives a peak performance of 550 Mflop/s. If the maximum number of 15 boards were added to each FPS-164 the peak performance would be raised to 3.4 Gflop/s.

The above computer system is described as a loosely coupled array because in the initial configuration there was no direct connection between the computing elements (the FPS-164s), and because the connection to the host is by slow channels. Consequently, only problems which exhibit a very large grain of parallelism can be effectively computed. That is to say that a very

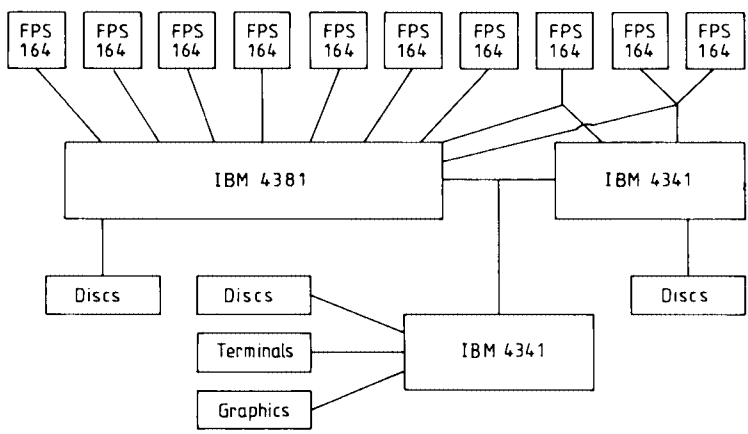

FIGURE 2.50 A simplified block diagram of the IBM experimental loosely coupled array of processors (/CAP) at Kingston, with ten FPS-164 computers connected via IBM host computers.

large amount of work must be performed on data within the FPS-164 before the results are transferred over the slow channels to the host, or via the host to other FPS-164s. A fast 22 Mbyte/s FPS bus (called FPSBUS, directly connecting the FPS-164s, has subsequently been developed by FPS which substantially reduces the overhead of transferring data between the FPS-164s.

A similar project to the above has been developed for some years at Cornell University's Theory Center under the direction of Professor Kenneth Wilson. Initially, this comprised eight FPS-100 processors connected by a custom 24 Mbyte/s bus. The work is now part of the Cornell Advanced Scientific Computing Center which is sponsored by NSF, IBM and FPS. It is envisaged that up to 4000 MAX boards could be interconnected to give a peak performance rate of about 40 Gflop/s.

In order to quantify the synchronisation and communication delays on the /CAP, measurements have been made of the performance parameters $(\hat{r}_\infty, \hat{s}_{1/2}, f_{1/2})$ and these are discussed in §1.3.6, part (iv). The benchmark has been conducted using either the channels or the FPS BUS for communication, and gives rise to the following total timing equations (Hockney 1987d)

$$t_{\text{channel}} = 2500 + 4777p + (1.87m/p) + t_a(p)s \quad \mu\text{s} \tag{2.23a}$$

$$t_{\text{BUS}} = 18926 + 8181p + 0.191m + t_a(p)s \quad \mu\text{s} \tag{2.23b}$$

where m is the number of I/O words and s the number of floating-point operations in the work segment (see §1.3.6, part (iv)). The first two terms in equations (2.23) are a fit to the synchronisation time, the third term is the time spent on communication, and the last term is the time spent on calculation. The fact that the communication term is inversely proportional to the number of processors, p, in the case of the channels, shows that the channels, although slow, are working in parallel. On the other hand, we see that the communication time does not depend on p in the case of the FPSBUS showing that the bus, although much faster, is working serially.

In order to compare the use of the channels with the use of the FPSBUS we equate the two timing formulae (2.23a) and (2.23b) and obtain the equation for the equal performance line (EPL):

$$m = p\frac{(16426 + 3404p)}{(1.87 - 0.191p)}. \tag{2.24}$$

This relationship is plotted on the (p, m) phase plane in figure 2.51. Given a number of processors p and the number of I/O words m, a point is specified on this plane. Its location in the plane determines whether bus or channel communication should be used. There is an infinity in the relationship (2.24) at $p = 9.78$ (broken line) showing that the channels will always be faster if

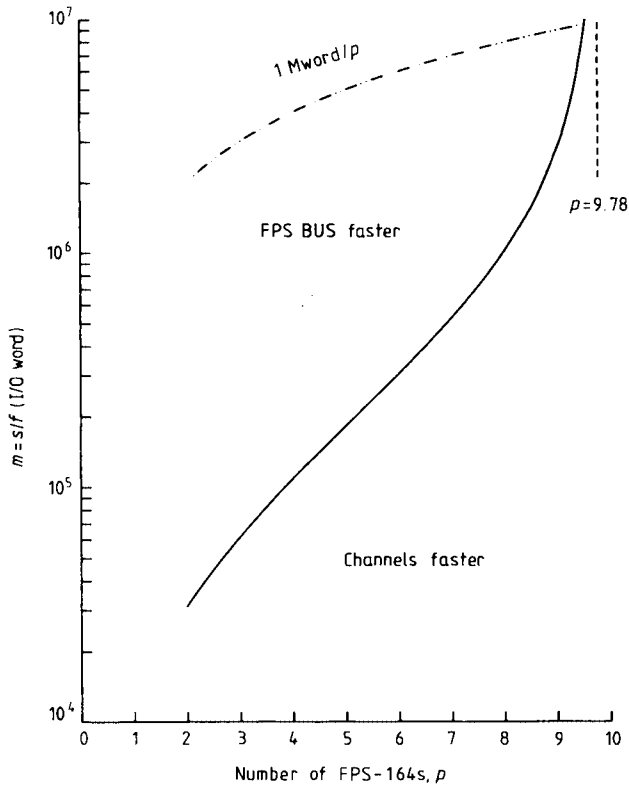

FIGURE 2.51 Phase diagram comparing the use of the FPS BUS with the use of the channels. For any number of processors chosen, p, either the FPS BUS or the channels is faster depending on the number of I/O words m. For more than about 10 processors the channels are always faster.

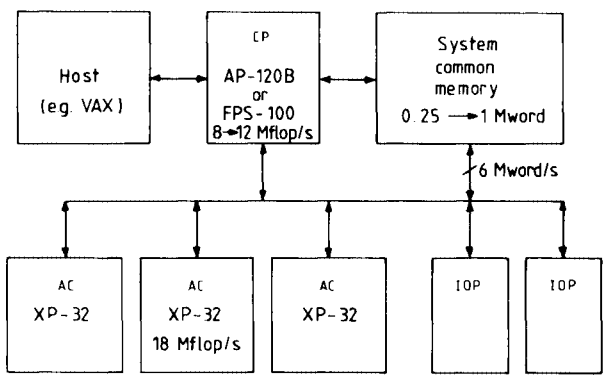

FIGURE 2.52 Overall architecture of the FPS-5000 series of computers.

there are ten processors or more. This is because the channels have a smaller start-up and synchronisation overhead than the bus, and a faster asymptotic rate if there are ten or more working in parallel. That is to say, more than ten 2–3 Mbyte/s channels working in parallel are faster than a 22 Mbyte/s bus working serially. We also show in figure 2.51 the line corresponding to 1 Mword per processor, which is a typical main memory size for an FPS-164 installation. Values of m above this line are inaccessible in such an installation, because problems requiring such a magnitude of I/O would not fit into the memory of the installation. However, the memory size can be increased to 28 Mword/FPS-164 (using 1 Mword memory boards) corresponding to a line off the top of the diagram.

2.5.10 FPS-5000 series

The FPS-5000 series of computers is interesting because it forms one of the few commercially available MIMD computer systems. It is the latest offering of so-called 'array processors' from FPS, and is a multicomputer development of AP-120B type of architecture. In our classification of MIMD computers (§1.2.6 and figure 1.9), it falls in the class of bus-connected, shared-memory, switched MIMD systems.

The overall architecture of an FPS-5000 series computer system is shown in figure 2.52 (FPS 1984a). The system is connected to its host computer by the *control processor* (CP) which is either an FPS AP-120B with a 167 ns clock or the slower FPS-100 with a 250 ns clock. Both CPs have the same architecture but differ in technology and packaging. The CP may perform useful arithmetic itself at peak rates, respectively, of 12 and 8 Mflop/s. More importantly, however, it controls the rest of the system which comprises up to three *arithmetic coprocessors* (ACs) and a number of *I/O processors* (IOPs) which share a common bus connection to a *system common memory* (SCM) of between 0.25 and 1 Mword. The ACs are FPS XP-32 computers with a peak performance of 18 Mflop/s. The largest system announced in 1983 used an FPS-100 CP (8 Mflop/s) and three XP-32 ACs, giving a theoretical peak performance of 62 Mflop/s. Continuity with previous systems is maintained because all software prepared for the AP-120B will run on the control processor, and the computing power can be enhanced in stages by adding arithmetic coprocessors as required.

The progression of technology in the 1970s and early 1980s is exemplified by the progression from the AP-120B (1975), through the FPS-100 (1978) to the XP-32 (1983). The basic processor of the AP-120B used Schottky TTL chips operating at 8 MHz and occupied 20 boards. Of these, three boards were required for the multiplier and three for the adder. The FPS-100 was able to compress the identical architecture onto ten boards by using low-

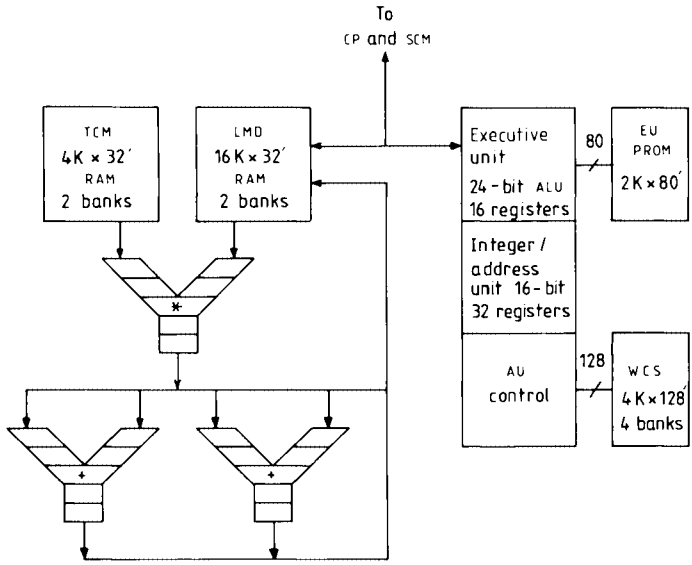

FIGURE 2.53 Internal architecture of the FPS XP-32 arithmetic coprocessor.

power Schottky TTL chips, at the cost of a slower clock (4 MHz). The multiplier and adder were both compressed to one board each. The XP-32, on the other hand, fits the whole processor onto a single board and, furthermore, includes a second floating-point adder. This is achieved by using fast Schottky VLSI chips with a 6 MHz clock. The multiplier is now reduced to a single chip (from the three boards in the AP-120B), namely the 32-bit WEITEK WTL-1032. Similarly the floating-point adders each use the WEITEK WTL-1033 floating-point ALU chip. The rest of the logic uses the Advanced Micro Devices 29500 series of VLSI circuits, and INMOS IMS-1040 static RAMS. The use of standard WEITEK floating-point chips has meant a change to the floating-point number representation. The FPS AP-120B and FPS-100 use a 38-bit format as described in §2.5.2. This has a dynamic number range from 10^{-155} to 10^{+153}, and a precision of 28 bits. The WEITEK chips, however, have adopted the IEEE 754 32-bit floating-point standard (IEEE 1983), which has a much reduced dynamic range of 10^{-38} to 10^{+38}, but a more precise mantissa equivalent to 33 bits.

The architecture of the XP-32 coprocessor (figure 2.53) is similar in general concept to that of the AP-120B but differs in detail (FPS 1984c). A five-stage floating-point multiplier pipeline and two five-stage floating-point adder

pipelines are connected by multiple data paths to a *local main data* (LMD) and a *table coefficient memory* (TCM). The LMD stores 16K 32-bit words arranged in two banks, and the TCM has 4K 32-bit words also arranged in two banks. Overall control of the XP-32 is exercised by the *executive unit* (EU) which can operate simultaneously with the *arithmetic unit* (AU), thereby providing for the parallel execution of I/O and address calculation with floating-point arithmetic. The EU performs all communication of programs and data between the AC and the CP and SCM. The AU performs arithmetic only on data in the local TCM and LMD memories. Microcode programs for the EU reside in EU PROM, which contains 2K 80-bit microcode instructions. Similarly, microcode programs for the AU reside in a *writable control store* (WCS) of 4K 128-bit microcode instructions, arranged in four banks.

There is no direct connection between the arithmetic coprocessors. The coprocessors can only take data from and return results to the SCM, so that any communication between the ACS is by shared data in the SCM. The SCM acts as the main data memory of the control processor. In the case of the AP-120B control processor, it operates as described in §2.5.2 for the fast memory (333 ns access) with a 167 ns clock period. In the case of the FPS-100 control processor, the memory works on the slower 250 ns clock period. The arithmetic coprocessors may also have direct memory access (DMA) to the SCM by taking turns with the CP with the available memory cycles, according to a priority scheme. In the case of access by the ACS, the SCM may either read or write one word per clock period (but not both at the same time), giving a total SCM memory bandwidth of either 6 Mword/s (24 Mbyte/s) or 4 Mword/s (16 Mbyte/s). However, the memory is organised such that any individual XP-32 coprocessor may only use half this bandwidth, thereby allowing two ACS on an FPS-5000 system before the memory bus restricts its performance. This is achieved by limiting memory requests from any particular AC to every other memory cycle.

The FPS-5000 may be programmed entirely by calls to library programs in a subset of FORTRAN 77 called CPFORTRAN. In most cases company documentation gives timing formulae from which values of r_∞ and $n_{1/2}$ can be derived (FPS 1984a, b, 1985a). We give below a selection indicating the facilities provided.

(*i*) *Host interface routines*

The following routines run on the host computer, load programs and data into the FPS-5000, and start the CPFORTRAN program running on the CP.

CPOPEN Open a CPFORTRAN program file.

CPLOAD Load a CPFORTRAN program file from host to CP.

CPRUN Start CPFORTRAN program running on CP.

EXPUT Start data transfer from host to FPS-5000.

EXGET Start data transfer from FPS-5000 to host.

APWAIT Wait for data transfer and CP program to stop.

APWD Wait for data transfer to stop.

APWR Wait for CP program to stop.

(*ii*) *Synchronisation of the* ACS *by the* CP

The following routines run on the CP, and control the ACS.

XPSEL Select the XP-32 for subsequent XPWAIT.

XPRUN Start program running in selected XP-32.

XPWAIT Wait for selected XP-32 to finish.

XPSTAT Obtain status of XP-32.

(*iii*) *Data transfer to and from* SCM

The following routines run on the XP-32s, and transfer data prior to and after calculation.

XPDMAR Transfer data between SCM and LMD.

XTMDMA Transfer data between SCM and TCM.

$r_\infty = 2$ Mop/s

XPISNC Wait for transfer (or arithmetic) to finish.

(*iv*) *Arithmetic within the XP-32*

The following XPMLIB routines run on the XP-32 and perform arithmetic on data in the LMD of the XP-32.

ZVMUL(IA, IB, IC, N) Element-by-element vector dyadic multiply of $A*B$ to C, N elements ($r_\infty = 4$ Mflop/s, $n_{1/2} = 33$).

ZVDIV(IA, IB, IC, N) Element-by-element vector divide ($r_\infty = 0.5$ Mflop/s, $n_{1/2} = 9$).

ZVSASM(IA, IB, ID, IC, N) One-vector triad, vector scalar add scalar multiply: $C = (A + b)*d$ ($r_\infty = 12$ Mflop/s, $n_{1/2} = 56$).

ZVASM(IA, IB, ID, IC, N) CDC 205-type two-vector triad, vector add scalar multiply: $C = (A + B)*d$ ($r_\infty = 8$ Mflop/s, $n_{1/2} = 37$).

ZVAM(IA, IB, ID, IC, N) All-vector triad, vector add multiply: $C = (A + B)*D$ ($r_\infty = 6$ Mflop/s, $n_{1/2} = 28$).

The above performance figures are for dyadic and triadic operations within a single XP-32 processor and they do not take into account the time taken to synchronise the multiple ACS of an FPS-5000 (i.e. the MIMD performance), or the time to transfer data from the SCM to LMD prior to performing the calculations. These have been separately measured by Curington and Hockney (1986) and interpreted in terms of the parameters $n_{1/2}$, $s_{1/2}$ and $f_{1/2}$ (§1.3.6). The results are given in tables 2.11 and 2.12.

The FPS-5320A computer which was used for the measurements comprises a control processor and either one or two XP-32 arithmetic coprocessors.

TABLE 2.11 Measurements of $(r_\infty, n_{1/2}, s_{1/2})$ on a multiprocessor FPS-5320A with a control processor (CP), and one or two XP-32 coprocessors, operating on data in their respective local memories.

Operation	Configuration	r_∞ (Mflop/s)	$n_{1/2}$† or $s_{1/2}$
Dyad	CP only	1.5	14†
$A_i = B_i*C_i$	One XP-32	4.0	470
VMUL or ZVMUL	Two XP-32	8.0	1320
	CP + two XP-32	9.2	1545
Triad	CP only	3.9	40†
$A_i = (B_i + s)*c$	One XP-32	12.0	1490
VSASM or ZVSASM	Two XP-32	24.0	4200
	CP + two XP-32	27.7	4820

TABLE 2.12 Values of peak performance, $\hat{r}_\infty$, and $f_{1/2}$ for a single FPS XP-32 arithmetic coprocessor when performing triadic ZVSASM operations on data originating in system common memory.

Case	$\hat{r}_\infty$ (Mflop/s)	$f_{1/2}$
Sequential I/O	12.5	4.2
Overlapped I/O	12.6	2.2

The measurements with the CP alone in table 2.11 involve no synchronisation and hence are of $n_{1/2}$, whereas those using multiprocessors include the synchronisation time and are therefore of $s_{1/2}$. The values of the latter indicate the minimum amount of arithmetic that is worth dividing amongst the XP-32 coprocessors. Both the CP and ACS work on a 6 MHz clock, and have arithmetic pipelines with a peak performance of 6 Mflop/s for dyadic operations which use only one pipeline, or 12 Mflop/s for triadic operations which use two pipelines. The peak performance for the maximum configuration of the control processor and two XP-32s is therefore 18 Mflop/s for dyads and 36 Mflop/s for triads. These peak rates are achieved in the case of the XP-32 executing triads. However, in the other cases inadequate memory bandwidth prevents the peak rates being realised, although the XP-32 suffers much less than the CP in this respect.

The measurements in table 2.11 are for operations performed on data residing in the local memories (LMD) of the XP-32s, which it is assumed has already been loaded. However, in actual use the data for a problem will be stored in SCM, and will have to be transferred to the LMD before calculation can take place. The overall rate of computation will critically depend, therefore, on the amount of arithmetic performed in the XP-32 per data transfer between SCM and LMD (i.e. the variable f defined in §1.3.6) and can be characterised by the parameter $f_{1/2}$. The FPS-5000 is an ideal vehicle for studying this dependence because f can easily be varied and the value required to achieve half the peak performance can be obtained. This is the performance parameter $f_{1/2}$, and is given in table 2.12. The observed peak rate of 12 Mflop/s is as expected for the ZVSASM operation. The test proceeds by transferring a vector of n data from SCM to LMD using XPDMAR, and then performing f ZVSASM operations upon the vector. Two cases are considered, first when the I/O transfers between SCM and LMD take place sequentially with the arithmetic operation, and secondly when the I/O takes place simultaneously with the arithmetic (i.e. is overlapped with it). We find that the effect of overlapping the I/O is to halve the value of $f_{1/2}$.