Array processor responds in real time

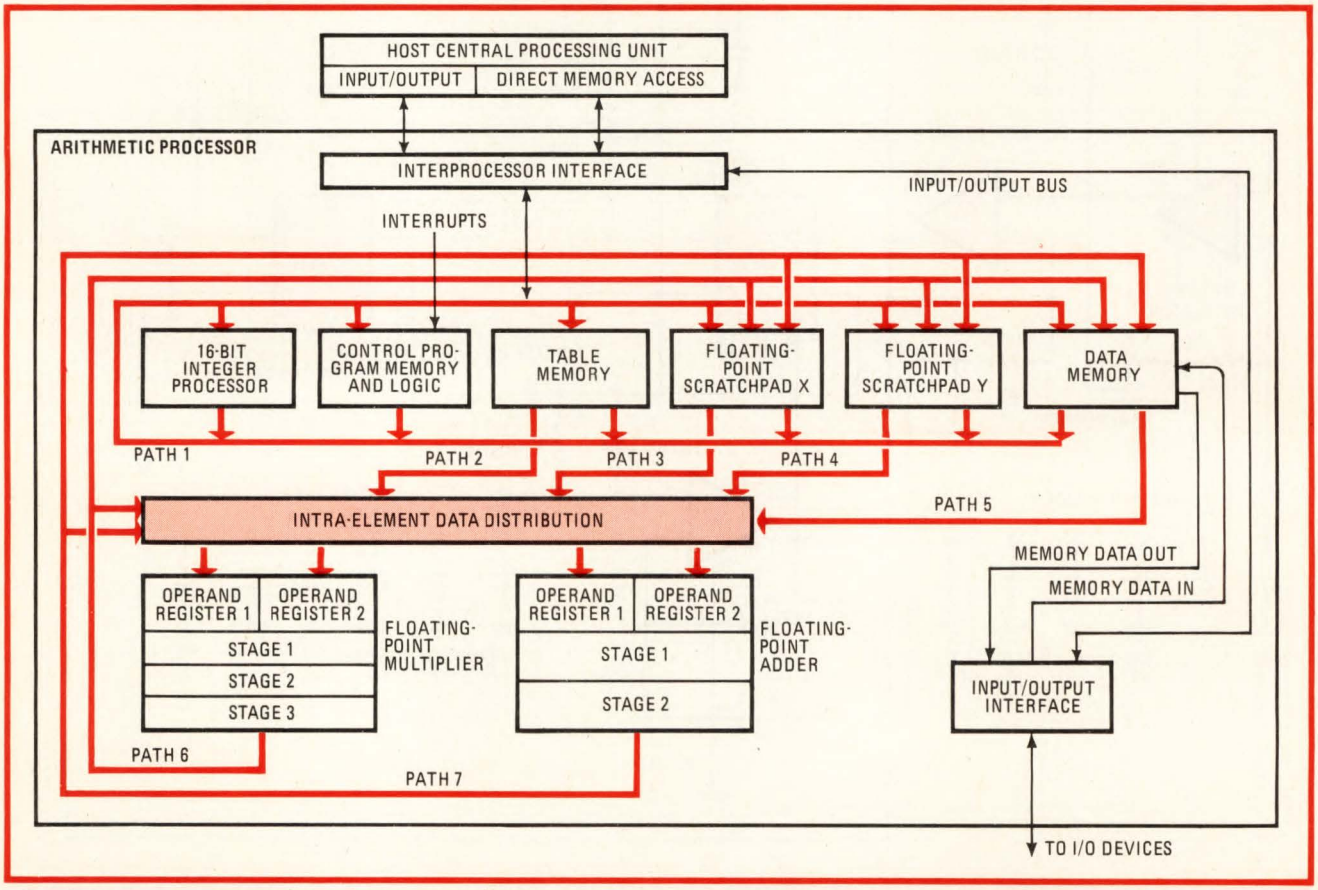
Peripheral machine has its own multitasking supervisor and performs up to 8 million floating-point operations per second

by James Strelchun, *Floating Point Systems Inc., Beaverton, Ore.*

☐ Specialized number-crunching machines able to perform rapid high-precision arithmetic over a large dynamic range are becoming very popular for beefing up the throughput of computer systems used in scientific applications. Often called array processors because of their optimization for performing repetitive calculations on an array of data, these peripheral arithmetic processors are not to be confused with machines having an internal array architecture, such as vector processors (see "Array processors, vector processors").

Floating Point Systems Inc. has added a new member to its line of array processors that incorporates up-to-date semiconductor technology and a priority interrupt scheme supported by the machine's own multitasking operating system, thus adding new real-time operating capabilities. These design changes make the FPS-100, which is intended for original-equipment manufacturers and systems integrators [*Electronics*, April 12, 1979, p. 209], easier to attach to the host computer system. The use of today's low-power Schottky TTL in medium- and large-scale integrated circuits reduces the new peripheral processor's chip count, physical size, and power consumption.

Applications such as flight simulation, radar signal analysis, X-ray tomography, image analysis, speech synthesis, and nuclear reactor monitoring require large



1. Parallel procedures. A separate floating-point multiplier and adder allow the FPS-100 to perform arithmetic operations in parallel. Seven 38-bit data paths distribute data among these arithmetic units and the parallel memories; a 16-bit integer processor handles control functions.

Array processors, vector processors

Array processors are computers dedicated by their design to performing repetitive arithmetical calculations on large arrays of data with high precision, wide dynamic range, and high throughput. Usually most input/output operations and file management chores are left to the host computer, in order to free the peripheral array processor to concentrate on its calculations.

As they become more popular, however, a semantic distinction must be made between array processors and other specialized processors with similar sounding names. An array processor consists of a single computer that operates on one piece of data at a time.

Vector processors are also specially designed for performing arithmetic on arrays of data, but they operate on an entire row or column of the array—the so-called vector—at once. Among new computer architectures, there is something called a distributed array processor. It consists of multiple arithmetic and logic units, each of which is associated with its own block of memory and operates on a separate piece of data simultaneously with all the others [*Electronics*, April 27, 1978, p. 69].

The well-known vector processors—the Illiac IV, Control Data's Star, or the Cray Research I—sell for several million dollars, much more than the minicomputer array processors available from several companies, including Floating Point Systems, Data General, and CSP. **-A. Durniak**

amounts of scientific computations such as fast Fourier transforms, convolutions, and vector and matrix arithmetic. Many of these applications require the calculations to be performed rapidly enough to provide almost instantaneous response, for so-called real-time operation. Programmable array processors attached to standard commercial minicomputers can provide designers or OEM suppliers of such systems with an inexpensive alternative to a large, specialized scientific processor.

After considerations of throughput, precision, and dynamic range, the interfacing flexibility of the array processor is of concern to the system designer. He may have chosen one of a variety of host computers based on other application requirements; of course the array processor must be compatible with that host. If system design is to be completed quickly, an array processor that is relatively easy to program is very desirable. This requires an easily understood high-level programming language, or at the very least assembly language; the availability of a library of standard mathematical software routines is also a big help. And the systems integrator must of course concern himself with questions of physical size, power consumption, reliability, serviceability, and cost. Tradeoffs between cost and the desired features must be carefully weighed in the design of a marketable peripheral processor.

The FPS-100 is easier to program and interface than the current models—the AP-120B, which is also intended to be attached to minicomputers by end users, and the AP-190L, designed for use with larger mainframe computers. The new arithmetic unit is also roughly half the size and uses half the power of the other models. It is as much as one third slower than the other models, however, even though it can perform some 8 million floating-point operations per second—some 50 to 200 times more than standard minicomputers.

System hardware overview

Based on the same synchronous, multiple-bus hardware architecture as the older models, the FPS-100 is divided into two sections: the arithmetic hardware and the interprocessor interface. Its 38-bit floating point numeric format, compatible with the other models, is maintained throughout the hardware with the exception of the 16-bit integer control processor. This floating-point format provides extended precision by devoting 28 bits to the mantissa and 10 bits to the exponent. (Binary coding rather than hexadecimal is used.) The result is precision of 8 decimal digits and a dynamic range of $10^{\pm 153}$. In contrast, 6 decimal digits and a range of $10^{\pm 38}$ are provided by a typical 32-bit minicomputer format.

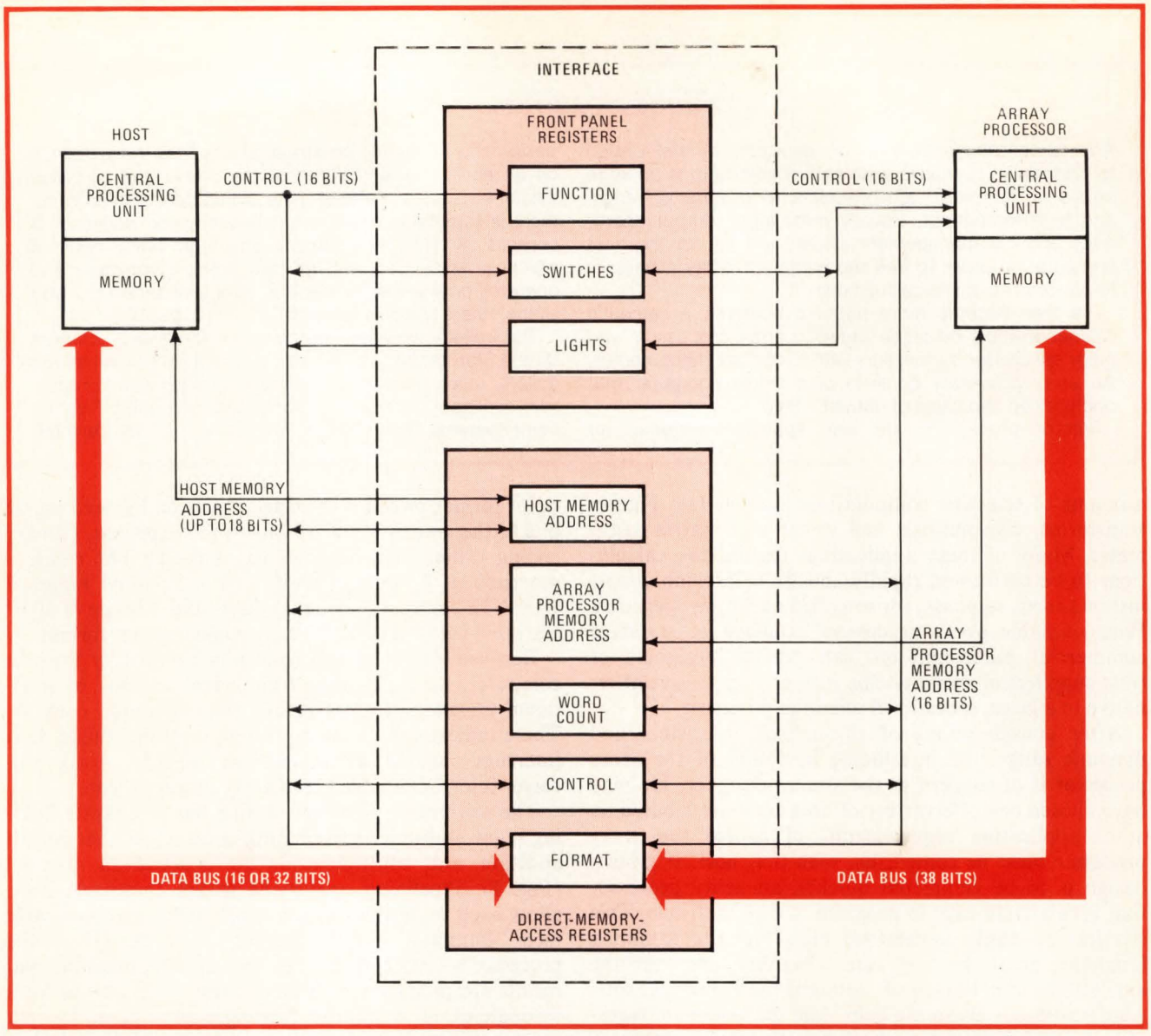
Because overall system control is handled by the host computer, the FPS-100's front panel consists of functional status indicators rather than operator controls. These indicators include power on, real-time mode, host interrupt enabled, direct memory transfer, array processor interrupt enabled, and array processor run.

The arithmetic processor section has a separate floating-point multiplier and floating-point adder that permit addition and multiplication to proceed in parallel (Fig. 1). Pipelining operations within these two units allow each to produce a new result every machine cycle (see "Pumping a full pipeline," p. 123). Given the processor's 4-MHz clock rate, this means computational results are produced as often as twice every 250 ns for a throughput of 8 million floating-point operations per second.

To keep up with these fast arithmetic units, parallel memories are used. One or 4 kilowords of memory are available for storing the 64-bit control program instructions; between 8 and 64 kilowords of memory are available for data storage. Numerical constants are stored in a separate table memory that consists of 2.5 or 4.5 kilowords of read-only memory or, optionally, 4 or 8 kilowords of random-access memory. Two banks of 32 38-bit floating-point registers are used as scratchpad memory for intermediate results. The FPS-100's synchronous design allows all of its memory elements to be accessed in a single 250-nanosecond clock cycle.

Seven data paths, each 38 bits wide, connect the various memories to the arithmetic units to avoid the delays which would result from all the data flow sharing a single bus. To simplify Fig. 1, the connections among these paths are labeled intra-element data distribution.

Overhead functions, including instruction decoding, address calculations, and program indexing for overall system synchronization, are performed by the arithmetic and logic unit of the separate 16-bit integer controller. This control unit has its own set of 16 16-bit general-purpose registers as well as a subroutine-return stack of



2. Host control. Two sets of registers provide the interface between the host computer and the arithmetic processor. One set performs functions analogous to the switches and lights on an operator's front panel and the second controls direct-memory-access data transfer.

16 12-bit registers. This frees the 38-bit floating-point hardware to concentrate on the programmed scientific computation, increasing throughput substantially.

To control the operation of so many parallel hardware elements, the 64-bit program instruction is divided into six groups of command fields. Just as a wide microcode instruction in a general-purpose computer will control many functions at once, so the program instruction of the FPS-100 simultaneously governs the operation of different parts of the hardware.

The first group within the program instruction is 14 bits long and directs the operation of the 16-bit integer controller. A 9-bit adder group controls the floating-point adder while a 9-bit branch group directs conditional branching. The next 19 bits in the program instruction, called the accumulator group, direct the flow of intermediate results between the floating-point arithmetic units and the registers. The floating-point multiplier is controlled by the next 5-bit group, and the final 8-bit group controls memory addressing.

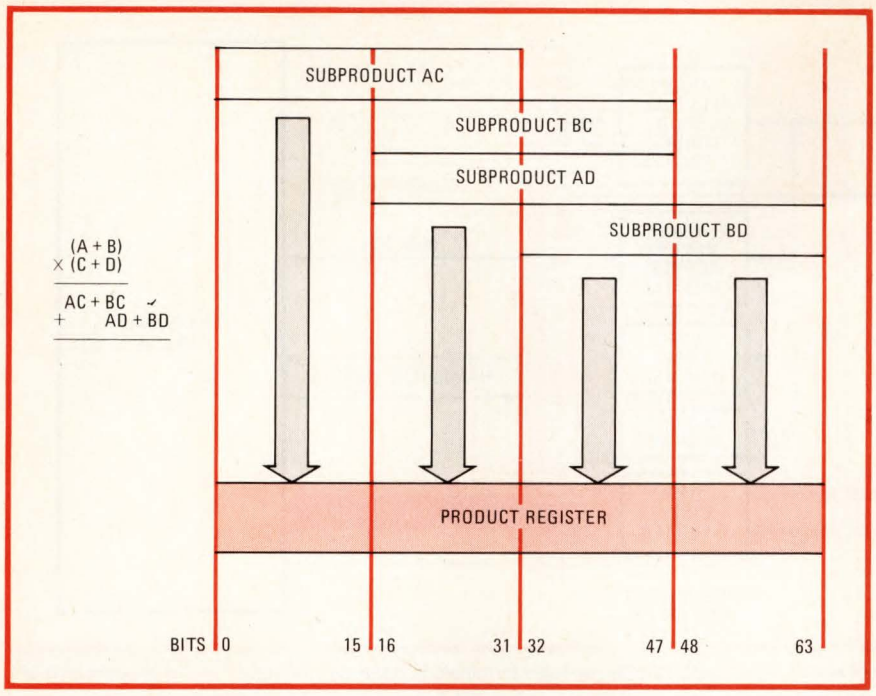
This 64-bit instruction word thereby allows up to 10 operations to proceed simultaneously—or some 40 million operations per second. This assures that in actual operation the arithmetic processor achieves most of its potential throughput of 8 million floating-point operations per second.

Communicating

The interprocessor interface provides communications between this parallel floating-point arithmetic hardware, the host computer, and additional input/output devices. Interfaces are available for operation of the arithmetic processor with a variety of hosts. Currently complete hardware and software support exists for operation with the Digital Equipment Corp. RSX-11M or RT-11 operating systems or the Data General RDOS operating system. Additional interfaces are planned.

The host computer and the arithmetic processor communicate through two sets of registers: one for the programmed I/O commands used for control and the other for data transfer (Fig. 2).

Although, as has been mentioned, the FPS-100 does



3. Dividing multiplication. Because the MPY-16 multiplier chip handles only 16-bit operands, four are needed to process the 28-bit mantissa used in the arithmetic processor. Each mantissa is divided in two and the subproducts added, with the result going into a single 64-bit register.

not have an operator's front panel, the control registers can be thought of as a simulated front panel controlled by the host. The first register is analogous to the front panel switches in that it is used to enter control or parameter data into the arithmetic processor. The second register is like panel lights; it is used by the host to examine the contents of the arithmetic processor's internal registers. The host writes front panel commands such as start, stop, reset, and continue into the third or "function" register.

Data transfer is accomplished using the direct-memory-access (DMA) technique on a cycle-stealing basis. A set of DMA registers accommodate data transfers in either direction between host and array processor, with either machine controlling the transfer.

Separate registers are provided for host memory addresses and arithmetic-processor memory addresses. The word count register keeps track of the number of data words transferred, while the direction of data transfer is governed by the control register. The actual data transfer is accomplished through the format register, a DMA register that converts data from the floating-point format of the host into the 38-bit floating-point format of the arithmetic processor on the fly. Control of this double-buffered 38-bit register is handled by 4 bits in the control register. Since there are many floating-point formats in use, the exact details of the format register are determined by the host chosen.

Although most input/output operations are handled by the host, a direct interface between the FPS-100 and the outside world is sometimes required. This is handled by disk-drive subsystems available separately or the I/O processors, which will be discussed later.

Implementation tradeoffs

The choice of a semiconductor family for implementing the arithmetic processor is based on a four-way tradeoff between speed, power consumption, reliability, and package size requirements. To reduce the amount of power needed, low-power Schottky TTL is used predominately in the design except for the MOS memory and logic areas where speed is an overriding concern. Although this allows a 250-ns machine cycle time, it is still as much as one third slower than the previous models, which use standard Schottky logic.

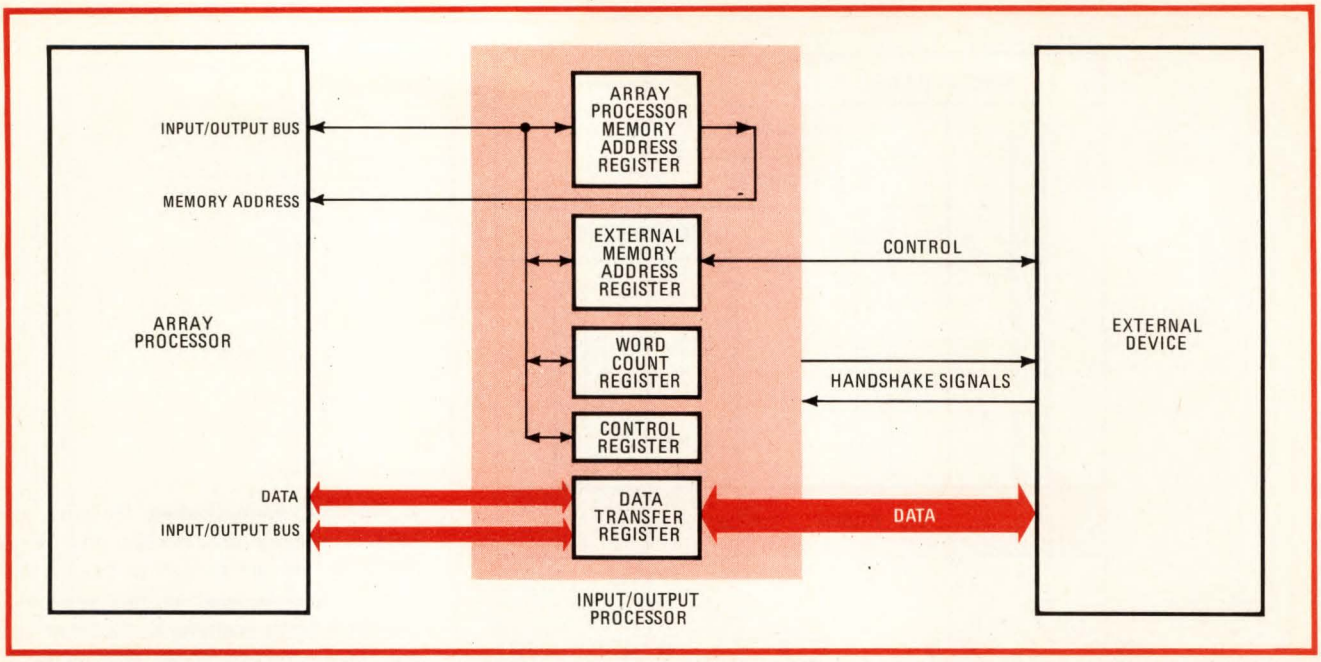
For some functions, such as the 38-bit floating-point multiplier, use of standard medium-scale integrated circuits results in a large chip count. This chip count combined with the substantial number of interconnections involved would increase packaging space. Instead, the MPY-16, a large-scale integrated circuit recently introduced by TRW, is used for the mantissa hardware of the multiplier, reducing package count from 112 to 24.

The MPY-16 multiplier chip is capable of taking a 16-bit multiplicand and a 16-bit multiplier and producing a 32-bit product. Since the mantissa of the FPS-100 contains 28 bits, a single MPY-16 is not sufficient. Each mantissa must, therefore, be divided into a high-order and a low-order set of bits. Since the multiplication of these divided mantissas results in four subproducts (Fig. 3), four MPY-16s are required for the operations to proceed in parallel. The four 32-bit subproducts are in turn each divided into the 16 high-order and 16 low-order bits and added as shown in the figure. The result is deposited in a 64-bit register; convergent rounding returns the product to a 28-bit format.

The four MPY-16s are actually capable of handling a 32-bit-by-32-bit multiplication. Since only 28-bit-by-28-bit multiplication is required, the 4 high-order bits on the inputs to two of the units are simply held at zero.

The total hardware required to arrive at the 64-bit product is now the 2-by-2 array of MPY-16s plus twenty adders for a total of 24 chips. Before the MPY-16, two 7-by-7 arrays of 4-bit-by-2-bit multipliers plus fourteen 4-bit adders were required, a total of 112 chips.

Another example of use of state-of-the-art LSI occurs



4. Intelligent I/O. Interfacing with fixed-protocol devices such as cathode-ray-tube terminals is done by the input/output processor (IOP). One set of registers provides for address conversion and control of the peripheral device, while the second handles data transfer.

in main data memory. As mentioned, the 38-bit main data memory may consist of 8, 16, 32, or 64 kilowords. It is desirable to fit the entire memory on one printed-circuit card to minimize packaging space. Using 16-K memory chips it is possible to put the 32-kiloword main memory—some 152 kilobytes—onto a single 10-by-15-inch board.

For the complete 64 kilowords, however, 128 packages would be required—more than a little unwieldy for a single card of reasonable size. To supply 64 kilowords without resorting to two boards, a dual-16-K memory package is used, supplying 32-K bits per package without increasing package size or pin count.

Packaging

The FPS-100 comes in a 19-inch-wide rack mountable chassis 10.5 inches high and 24.4 inches deep. Fifteen circuit-board slots accommodate the entire machine, including the floating-point arithmetic unit, the integer controller, all memories, the real-time hardware, the host-computer interface, and one I/O processor. Additional I/O processors and programmable I/O processors, if needed, are housed in an I/O expansion chassis. The power supply housed in the main chassis supplies 5 and 12 volts to power both chassis.

Cooling is provided by a push-pull system using separate input and output fans. In the event of a fan failure, either fan alone can provide adequate cooling air. As a further precaution, a thermal sensor triggers power disconnection should there be an excessive heat rise.

Serviceability is enhanced by a hinged power supply that swings out to reveal the printed backplane interconnecting the circuit boards. Printed boards can be pulled out of their zero-insertion-force connectors without removing the arithmetic processor from the rack.

To facilitate the system designer's task, ways must be found to minimize programming time and complexity while retaining the high throughput potential of the arithmetic processor. At the same time, software flexibility is desirable to allow the system designer to balance programming complexity against coding efficiency and throughput for a given application.

Programming methods

The easiest programming method on the FPS-100 is the Fortran call approach. A program is written for the host consisting of a series of Fortran call statements from the host to the arithmetic processor. Each call then initiates a standard manufacturer-written subroutine already stored in the arithmetic processor.

A Fortran cross compiler for the FPS-100 allows the system designer the convenience of generating his custom programs in that higher-level language. The compiler, which resides in the host, automatically converts the program into parallel instructions in the arithmetic processor's own language. This eliminates the need for parallel assembly-language programming of the subdivided 64-bit instruction word and provides a satisfactory level of coding efficiency for most applications.

Where the ultimate in throughput is required, careful parallel coding in assembly language allows maximum use of the machine's capabilities. Software pipelining can often be implemented more efficiently by a human programmer than by the automatic Fortran compiler.

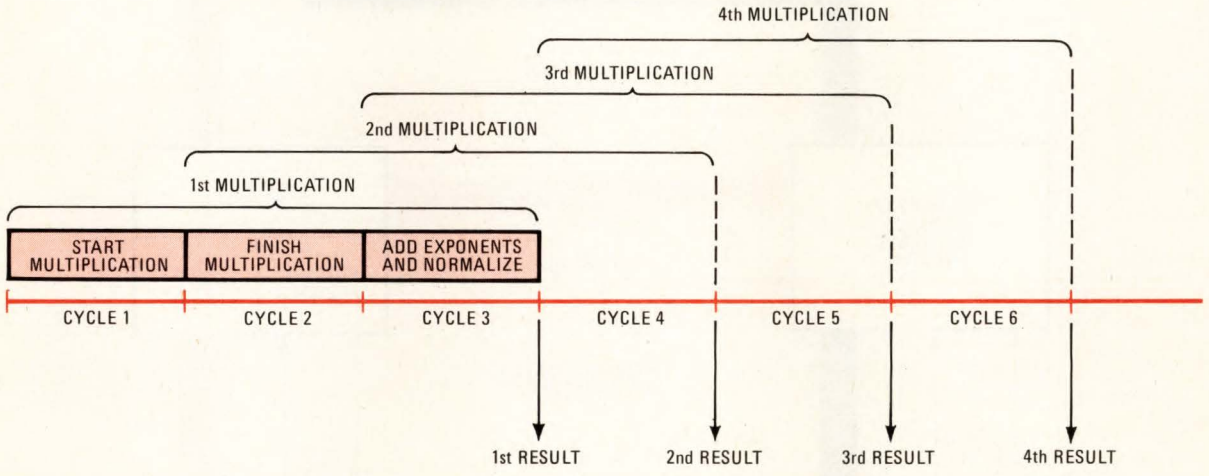
When preparing programs for an arithmetic processor, the differences between synchronous and asynchronous hardware architectures become apparent. As has been seen, array processors derive much of their high throughput from parallel hardware elements. Asynchronous designs offer a potential speed advantage since each of the parallel elements can operate at its own maximum speed, unconstrained by a common system clock cycle. Offsetting this advantage, however, is the necessity to write coordinated parallel programs for each of the

Pumping a full pipeline

In the operation of an oil pipeline, a second shipment is started down the line just after the last drops of the preceding shipment have entered the line; it would make no sense to wait until all of the first shipment had arrived at its destination, leaving the pipe empty. Similarly, hardware pipelining is a widely used technique for speeding up a computer by performing operations concurrently.

For example, a floating-point multiplication has three stages, each of which takes a machine cycle to complete. The product of the mantissas of the two operands starts in the first stage and is completed in the second stage, while the exponents of the two operands are added and any normalization and rounding of the product is performed during the third stage.

By pipelining, or staggering and overlapping, the sequential operations of many multiplications, the speed of the multiplier hardware is improved. As seen in the figure, although there is a delay of three machine cycles before the completion of the first multiplication, once the pipelining begins, a new result emerges each cycle.



asynchronous sections. In addition, the number of states in an asynchronous machine is indeterminate, making it impossible to write an exact simulator for debugging programs outside the machine.

Choosing appropriate parallel hardware elements and a clock cycle close to that of the fastest element, synchronous designs can provide throughputs equivalent to those of asynchronous designs in most applications. But the synchronous array processor is more easily programmed since there are no independent functions to coordinate. An exact simulator can also be written for a synchronous machine, since it has a fixed number of states. This makes it possible to develop software concurrently with machine hardware development and therefore to offer applications software such as math libraries at the same time as the hardware announcement.

Software support offered with the FPS-100 for speeding system design and reducing design costs includes special libraries of mathematical routines for signal and image processing, an interactive debugger, diagnostic routines, a linking loader, and programming aids for the I/O processors.

Real-time capability

While not all applications of arithmetic processors require real-time capability, the number of those that do make availability of such capability important. The real-time operating system software and hardware optionally available for the FPS-100 set it apart from the other models in the line.

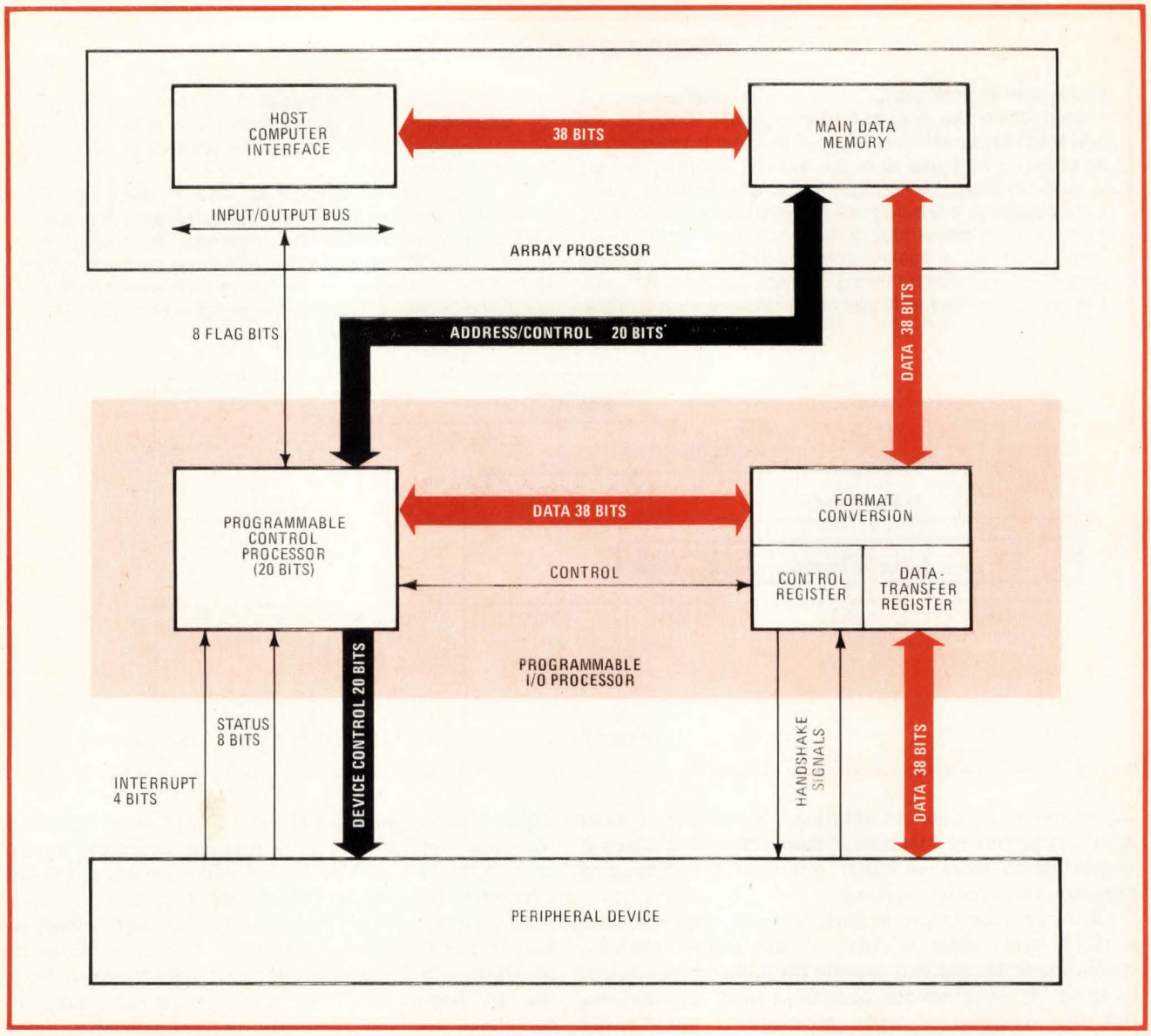
Central to the machine's real-time capabilities is the Super 100 real-time supervisor. In addition to supporting real-time applications, this resident operating system enables the FPS-100 to operate independently of the host computer for extended periods. Programs covering several computational tasks can be transferred from the host to the arithmetic processor at one time. When the programs are finished running, the results are stored in the arithmetic processor. Only when all tasks are completed does the arithmetic processor need to report back to the host.

To support real-time operation, the Super 100 supervisor works in concert with the real-time clock and the machine's interrupt structure. These features not only allow synchronizing the arithmetic processor with an external process, they also allow the machine to act as a system control clock.

The real-time clock is an up/down counter with a floating-point format. A 4-bit exponent allows selection of 1 of 16 counting rates ranging from 1 MHz to 60 Hz (1-microsecond to 16-millisecond resolution). Counting takes place in a 16-bit mantissa. The 4-bit rate selection register may also be disabled and the clock synchronized to an external source.

As a simple illustration of operation, a signal from an outside process could initiate an interval count in the clock. At the end of the interval the clock would interrupt the arithmetic processor so that computation on data from the outside process could be performed. Or the real-time clock might be used in asking for samples of data from the external process at regular intervals.

The priority interrupt structure is divided into three



5. General-purpose. Because of its programmability, the general-purpose input/output processor (GPIOP) can control a variety of more complex external devices. Choice of source and destination within the array processor and data formatting are under program control.

internal priority levels and one external level. The first three interrupt levels are involved with internal control of the arithmetic processor. The fourth level of the interrupt is of more interest to the system designer because it allows the outside world to interrupt the processor. This level is subdivided into 15 priority sublevels. The highest priority in this sublevel interrupt structure is assigned to the real-time clock so that computations associated with a real-time process will always be performed first.

Priority sublevels 2 and 3 are typically assigned to requests from the host computer to perform additional calculations that are not in real time. In this way the arithmetic processor can be operated in the multitasking mode. Sublevels 4 through 15 are assigned to service requests from I/O devices attached directly to the arithmetic processor.

Interface between the FPS-100 and an external real-time process is accomplished through one of two I/O processor types. These I/O processors are not restricted to real-time applications, and can also be used whenever there is a need for a direct interface between the arithmetic processor and an external device. Functions of the I/O processors include control of the external device and data transfer.

The first type of I/O processor, called the IOP (Fig. 4), is designed to interface fixed-protocol devices such as cathode-ray-tube terminals. The interfacing procedure does not vary for such devices and address control is limited to a fixed set of algorithms; the data-transfer format is also fixed. For interfacing variable-protocol peripheral devices, the programmable general-purpose I/O processor (GPIOP, Fig. 5) is used. A microprocessor included in the GPIOP permits programmable addressing of the data source and data destination in the arithmetic processor and the external device. It also provides programmable data formatting and control of more complex external devices. □