Cray 1

Computer Systems

G. Bell, S. H. Fuller, and D. Siewiorek, Editors

The CRAY-1 Computer System

Richard M. Russell
Cray Research, Inc.

This paper describes the CRAY-1, discusses the evolution of its architecture, and gives an account of some of the problems that were overcome during its manufacture.

The CRAY-1 is the only computer to have been built to date that satisfies ERDA's Class VI requirement (a computer capable of processing from 20 to 60 million floating point operations per second) [1].

The CRAY-1's Fortran compiler (CFT) is designed to give the scientific user immediate access to the benefits of the CRAY-1's vector processing architecture. An optimizing compiler, CFT, "vectorizes" innermost DO loops. Compatible with the ANSI 1966 Fortran Standard and with many commonly supported Fortran extensions, CFT does not require any source program modifications or the use of additional nonstandard Fortran statements to achieve vectorization. Thus the user's investment of hundreds of man months of effort to develop Fortran programs for other contemporary computers is protected.

Key Words and Phrases: architecture, computer systems

CR Categories: 1.2, 6.2, 6.3

Introduction

Vector processors are not yet commonplace machines in the larger-scale computer market. At the time of this writing we know of only 12 non-CRAY-1 vector processor installations worldwide. Of these 12, the most powerful processor is the ILLIAC IV (1 installation), the most populous is the Texas Instruments Advanced Scientific Computer (7 installations) and the most publicized is Control Data's STAR 100

Author's address: Cray Research Inc., Suite 213, 7850 Metro Parkway, Minneapolis, MN 55420.

(4 installations). In its report on the CRAY-1, Auerbach Computer Technology Reports published a comparison of the CRAY-1, the ASC, and the STAR 100 [2]. The CRAY-1 is shown to be a more powerful computer than any of its main competitors and is estimated to be the equivalent of five IBM 370/195s.

Independent benchmark studies have shown the CRAY-1 fully capable of supporting computational rates of 138 million floating-point operations per second (MFLOPS) for sustained periods and even higher rates of 250 MFLOPS in short bursts [3, 4]. Such comparatively high performance results from the CRAY-1 internal architecture, which is designed to accommodate the computational needs of carrying out many calculations in discrete steps, with each step producing interim results used in subsequent steps. Through a technique called "chaining," the CRAY-1 vector functional units, in combination with scalar and vector registers, generate interim results and use them again immediately without additional memory references, which slow down the computational process in other contemporary computer systems.

Other features enhancing the CRAY-1's computational capabilities are: its small size, which reduces distances electrical signals must travel within the computer's framework and allows a 12.5 nanosecond clock period (the CRAY-1 is the world's fastest scalar processor); a one million word semiconductor memory equipped with error detection and correction logic (SECDED); its 64-bit word size; and its optimizing Fortran compiler.

Architecture

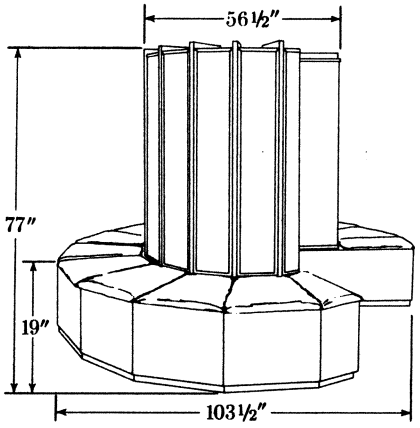
The CRAY-1 has been called "the world's most expensive love-seat" [5]. Certainly, most people's first reaction to the CRAY-1 is that it is so small. But in computer design it is a truism that smaller means faster. The greater the separation of components, the longer the time taken for a signal to pass between them. A cylindrical shape was chosen for the CRAY-1 in order to keep wiring distances small.

Figure 1 shows the physical dimensions of the machine. The mainframe is composed of 12 wedge-like columns arranged in a 270° arc. This leaves room for a reasonably trim individual to gain access to the interior of the machine. Note that the love-seat disguises the power supplies and some plumbing for the Freon cooling system. The photographs (Figure 2 and 3) show the interior of a working CRAY-1 and an exterior view of a column with one module in place. Figure 4 is a photograph of the interior of a single module.

An Analysis of the Architecture

Table I details important characteristics of the CRAY-1 Computer System. The CRAY-1 is equipped with 12 i/o channels, 16 memory banks, 12 functional units, and more than 4k bytes of register storage. Access to memory is shared by the i/o channels and high-speed registers. The most striking features of the CRAY-1 are: only four chip types, main memory speed, cooling system, and computation section.

Fig. 1. Physical organization of mainframe.



- Dimensions
  - Base–103½ inches diameter by 19 inches high
  - Columns–56½ inches diameter by 77 inches high including height of base
- 24 chassis
- 1662 modules; 113 module types
- Each module contains up to 288 IC packages per module
- Power consumption approximately 115 kw input for maximum memory size
- Freon cooled with Freon/water heat exchange
- Three memory options
- Weight 10,500 lbs (maximum memory size)
- Three basic chip types
  - 5/4 NAND gates
  - Memory chips
  - Register chips

Four Chip Types

Only four chip types are used to build the CRAY-1. These are 16 × 4 bit bipolar register chips (6 nanosecond cycle time), 1024 × 1 bit bipolar memory chips (50 nanosecond cycle time), and bipolar logic chips with subnanosecond propagation times. The logic chips are all simple low- or high-speed gates with both a 5 wide and a 4 wide gate (5/4 NAND). Emitter-coupled logic circuit (ECL) technology is used throughout the CRAY-1.

The printed circuit board used in the CRAY-1 is a 5-layer board with the two outer surfaces used for signal runs and the three inner layers for −5.2V, −2.0V, and ground power supplies. The boards are six inches wide, 8 inches long, and fit into the chassis as shown in Figure 3.

All integrated circuit devices used in the CRAY-1 are packaged in 16-pin hermetically sealed flat packs supplied by both Fairchild and Motorola. This type of package was chosen for its reliability and compactness. Compactness is of special importance; as many as 288 packages may be added to a board to fabricate a module (there are 113 module types), and as many as 72 modules may be inserted into a 28-inch-high chassis.

Fig. 2. The CRAY-1 Computer.

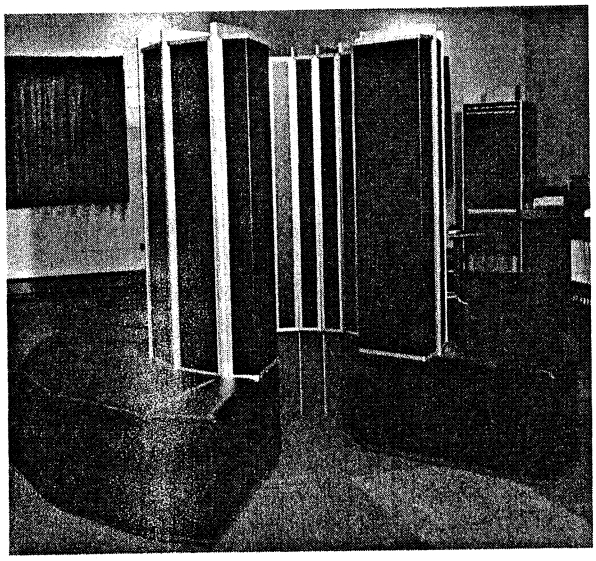
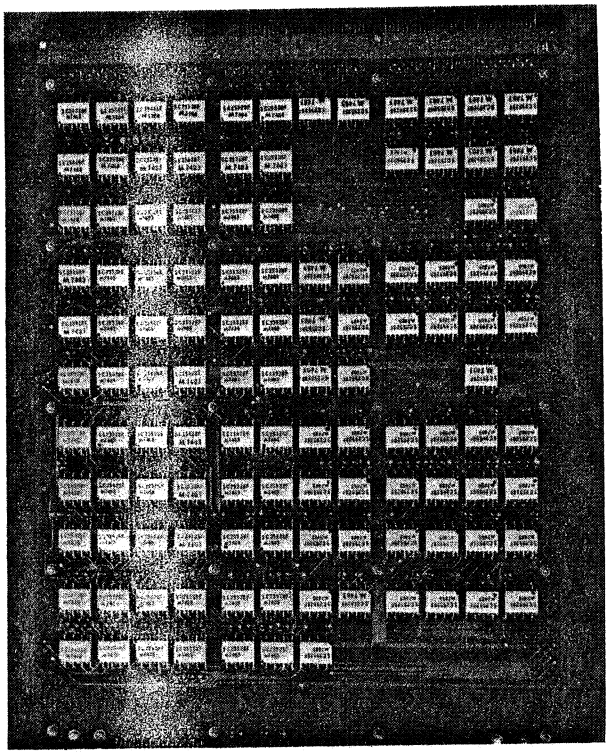


Fig. 3. CRAY-1 modules in place.



Such component densities evitably lead to a mammoth cooling problem (to be described).

Main Memory Speed

CRAY-1 memory is organized in 16 banks, 72 modules per bank. Each module contributes 1 bit to a 64-bit word. The other 8 bits are used to store an 8-bit check byte required for single-bit error correction, double-bit error detection (SECDED). Data words are stored in 1-bank increments throughout memory. This organization allows 16-way interleaving of memory accesses and prevents bank conflicts except in the case of memory accesses that step through memory with either an 8 or 16-word increment.

Fig. 4. A single module.

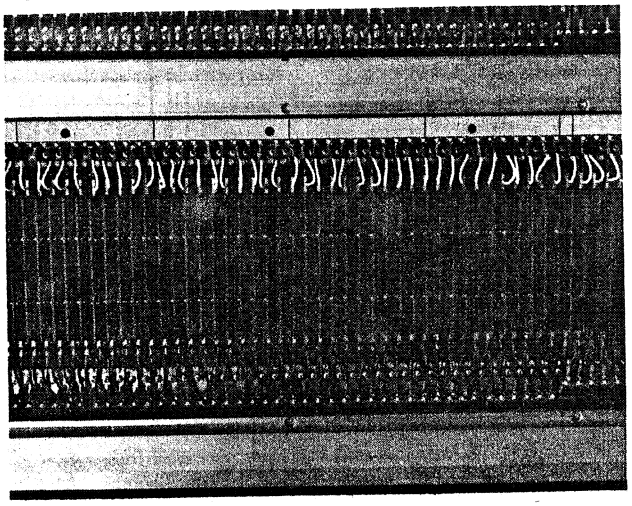


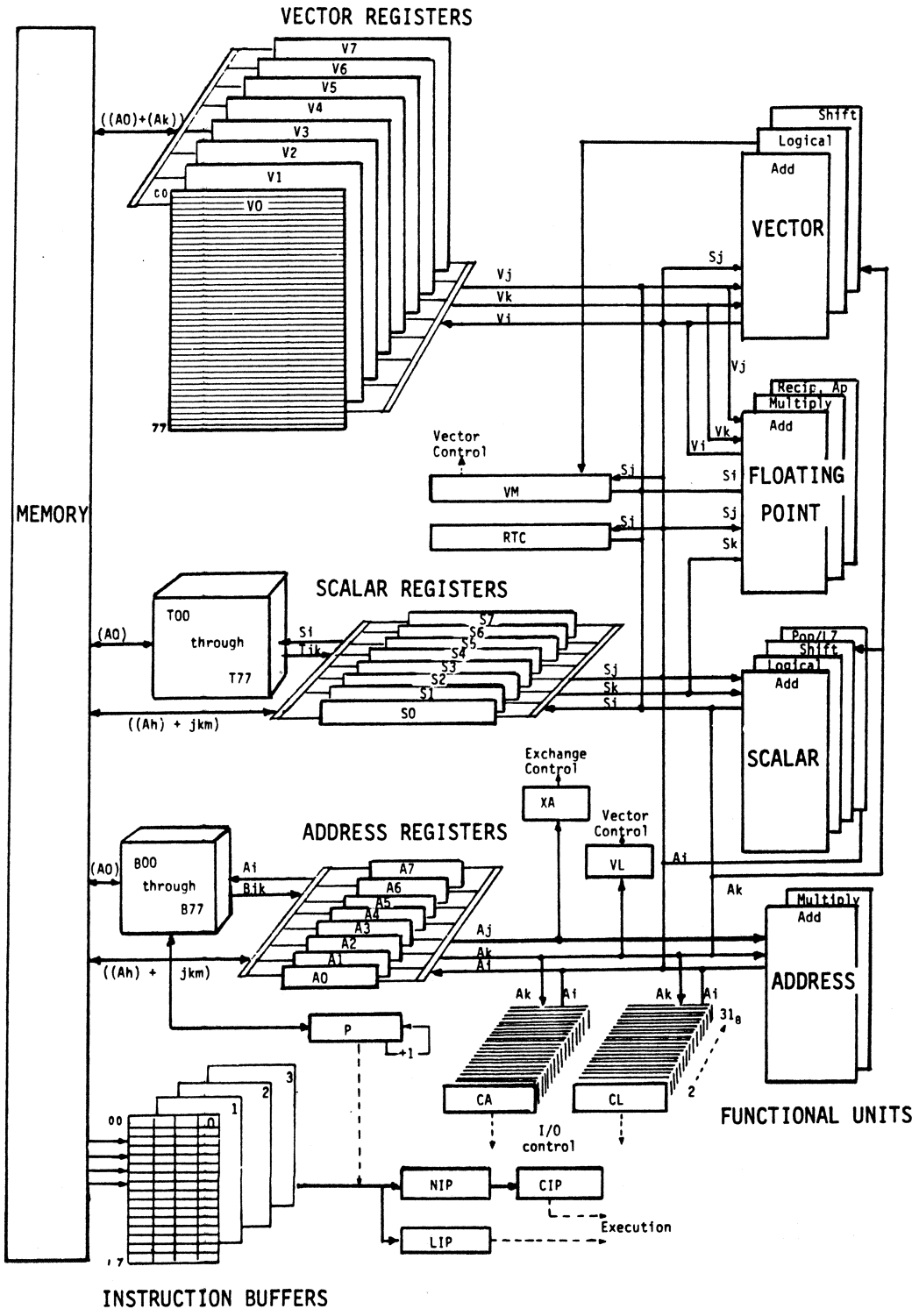
Table I. CRAY-1 CPU characteristics summary

Computation Section
Scalar and vector processing modes
12.5 nanosecond clock period operation
64-bit word size
Integer and floating-point arithmetic
Twelve fully segmented functional units
Eight 24-bit address (*A*) registers
Sixty-four 24-bit intermediate address (*B*) registers
Eight 64-bit scalar (*S*) registers
Sixty-four 64-bit intermediate scalar (*T*) registers
Eight 64-element vector (*V*) registers (64-bits per element)
Vector length and vector mask registers
One 64-bit real time clock (*RT*) register
Four instruction buffers of sixty-four 16-bit parcels each
128 basic instructions
Prioritized interrupt control
Memory Section
1,048,576 64-bit words (plus 8 check bits per word)
16 independent banks of 65,536 words each
4 clock period bank cycle time
1 word per clock period transfer rate for *B*, *T*, and *V* registers
1 word per 2 clock periods transfer rate for *A* and *S* registers
4 words per clock period transfer rate to instruction buffers (up to 16 instructions per clock period)
i/o Section
24 i/o channels organized into four 6-channel groups
Each channel group contains either 6 input or 6 output channels
Each channel group served by memory every 4 clock periods
Channel priority within each channel group
16 data bits, 3 control bits per channel, and 4 parity bits
Maximum channel rate of one 64-bit word every 100 nanoseconds
Maximum data streaming rate of 500,000 64-bit words/second
Channel error detection

Cooling System

The CRAY-1 generates about four times as much heat per cubic inch as the 7600. To cool the CRAY-1 a new cooling technology was developed, also based on Freon, but employing available metal conductors in a new way. Within each chassis vertical aluminum/stainless steel cooling bars line each column wall. The

Fig. 5. Block diagram of registers.



Freon refrigerant is passed through a stainless steel tube within the aluminum casing. When modules are in place, heat is dissipated through the inner copper heat transfer plate in the module to the column walls and thence into the cooling bars. The modules are mated with the cold bar by using stainless steel pins to pinch the copper plate against the aluminum outer casing of the bar.

To assure component reliability, the cooling system was designed to provide a maximum case temperature of 130°F (54°C). To meet this goal, the following temperature differentials are observed:

Temperature at center of module	130°F (54°C)
Temperature at edge of module	118°F (48°C)
Cold plate temperature at wedge	78°F (25°C)
Cold bar temperature	70°F (21°C)
Refrigerant tube temperature	70°F (21°C)

Functional Units

There are 12 functional units, organized in four groups: address, scalar, vector, and floating point. Each functional unit is pipelined into single clock segments. Functional unit time is shown in Table II. Note that all of the functional units can operate concurrently so that in addition to the benefits of pipelining (each functional unit can be driven at a result rate of 1 per clock period) we also have parallelism across the units too. Note the absence of a divide unit in the CRAY-1. In order to have a completely segmented divide operation the CRAY-1 performs floating-point division by the method of reciprocal approximation. This technique has been used before (e.g. IBM System/360 Model 91).

Table II. CRAY-1 functional units

	Register usage	Functional unit time (clock periods)
Address function units		
address add unit	*A*	2
address multiply unit	*A*	6
Scalar functional units		
scalar add unit	*S*	3
scalar shift unit	*S*	2 or 3 if double-word shift
scalar logical unit	*S*	1
population/leading zero count unit	*S*	3
Vector functional units		
vector add unit	*V*	3
vector shift unit	*V*	4
vector logical unit	*V*	2
Floating-point functional units		
floating-point add unit	*S* and *V*	6
floating-point multiply unit	*S* and *V*	7
reciprocal approximation unit	*S* and *V*	14

Registers

Figure 5 shows the CRAY-1 registers in relationship to the functional units, instruction buffers, i/o channel control registers, and memory. The basic set of programmable registers are as follows:

8 24-bit address (A) registers
64 24-bit address-save (B) registers
8 64-bit scalar (S) registers
64 64-bit scalar-save (T) registers
8 64-word (4096-bit) vector (V) registers

Expressed in 8-bit bytes rather than 64-bit words, that's a total of 4,888 bytes of high-speed (6ns) register storage.

The functional units take input operands from and store result operands only to A, S, and V registers. Thus the large amount of register storage is a crucial factor in the CRAY-1's architecture. Chaining could not take place if vector register space were not available for the storage of final or intermediate results. The B and T registers greatly assist scalar performance. Temporary scalar values can be stored from and reloaded to the A and S register in two clock periods. Figure 5 shows the CRAY-1's register paths in detail. The speed of the CFT Fortran IV compiler would be seriously impaired if it were unable to keep the many Pass 1 and Pass 2 tables it needs in register space. Without the register storage provided by the B, T, and V registers, the CRAY-1's bandwidth of only 80 million words/second would be a serious impediment to performance.

Instruction Formats

Instructions are expressed in either one or two 16-bit parcels. Below is the general form of a CRAY-1 instruction. Two-parcel instructions may overlap memory-word boundaries, as follows:

Fields	g	h	i	j	k	m
	0–3	4–6	7–9	10–12	13–15	16–31
Bit positions	(4)	(3)	(3)	(3)	(3)	(16)
			Parcel 1			Parcel 2

The computation section processes instructions at a maximum rate of one parcel per clock period.

For arithmetic and logical instructions, a 7-bit operation code (gh) is followed by three 3-bit register designators. The first field, i, designates the result register. The j and k fields designate the two operand registers or are combined to designate a B or T register.

The shift and mask instructions consist of a 7-bit operation code (gh) followed by a 3-bit i field and a 6-bit jk field. The i field designates the operand register. The jk combined field specifies a shift or mask count.

Immediate operand, read and store memory, and branch instructions require the two-parcel instruction word format. The immediate operand and the read and store memory instructions combine the j, k, and m fields to define a 22-bit quantity or memory address. In addition, the read and store memory instructions use the h field to specify an operating register for indexing. The branch instructions combine the i, j, k, and m fields into a 24-bit memory address field. This allows branching to any one of the four parcel positions in any 64-bit word, whether in memory or in an instruction buffer.

Operating Registers

Five types of registers—three primary (A, S, and V) and two intermediate (B and T)—are provided in the CRAY-1.

A *registers*—eight 24-bit A registers serve a variety of applications. They are primarily used as address registers for memory references and as index registers, but also are used to provide values for shift counts, loop control, and channel i/o operations. In address applications, they are used to index the base address for scalar memory references and for providing both a base address and an index address for vector memory references.

The 24-bit integer functional units modify values

(such as program addresses) by adding, subtracting, and multiplying A register quantities. The results of these operations are returned to A registers.

Data can be transferred directly from memory to A registers or can be placed in B registers as an intermediate step. This allows buffering of the data between A registers and memory. Data can also be transferred between A and S registers and from an A register to the vector length registers. The eight A registers are individually designated by the symbols A0, A1, A2, A3, A4, A5, A6, and A7.

B *registers* – there are sixty-four 24-bit B registers, which are used as auxiliary storage for the A registers. The transfer of an operand between an A and a B register requires only one clock period. Typically, B registers contain addresses and counters that are referenced over a longer period than would permit their being retained in A registers. A block of data in B registers may be transferred to or from memory at the rate of one clock period per register. Thus, it is feasible to store the contents of these registers in memory prior to calling a subroutine requiring their use. The sixty-four B registers are individually designated by the symbols B0, B1, B2, . . . , and $B77_8$.

S *registers* – eight 64-bit S registers are the principle data handling registers for scalar operations. The S registers serve as both source and destination registers for scalar arithmetic and logical instructions. Scalar quantities involved in vector operations are held in S registers. Logical, shift, fixed-point, and floating-point operations may be performed on S register data. The eight S registers are individually designated by the symbols S0, S1, S2, S3, S4, S5, S6, and S7.

T *registers* – sixty-four 64-bit T registers are used as auxiliary storage for the S registers. The transfer of an operand between S and T registers requires one clock period. Typically, T registers contain operands that are referenced over a longer period than would permit their being retained in S registers. T registers allow intermediate results of complex computations to be held in intermediate access storage rather than in memory. A block of data in T registers may be transferred to or from memory at the rate of one word per clock period. The sixty-four T registers are individually designated by the symbols T0, T1, T2, . . . , and $T77_8$.

V *registers* – eight 64-element V registers provide operands to and receive results from the functional units at a one clock period rate. Each element of a V register holds a 64-bit quantity. When associated data is grouped into successive elements of a V register, the register may be considered to contain a vector. Examples of vector quantities are rows and columns of a matrix, or similarly related elements of a table. Computational efficiency is achieved by processing each element of the vector identically. Vector merge and test instructions are provided in the CRAY-1 to allow operations to be performed on individual elements designated by the content of the vector mask (VM) register. The number of vector register elements to be processed is contained in the vector length (VL) register. The eight V registers are individually designated by the symbols V0, V1, V2, V3, V4, V5, B6, and V7.

Supporting Registers

The CPU contains a variety of additional registers that support the control of program execution. These are the vector length (VL) and vector mask (VM) registers, the program counter (P), the base address (BA) and limit address (LA) registers, the exchange address (XA) register, the flag (F) register, and the mode (M) register.

VL *register* – the 64-bit vector mask (VM) register controls vector element designation in vector merge and test instructions. Each bit of the VM register corresponds to a vector register element. In the vector test instruction, the VM register content is defined by testing each element of a V register for a specific condition.

P *register* – the 24-bit P register specifies the memory register parcel address of the current program instruction. The high order 22 bits specify a memory address and the low order two bits indicate a parcel number. This parcel address is advanced by one as each instruction parcel in a nonbranching sequence is executed and is replaced whenever program branching occurs.

BA *registers* – the 18-bit base address (BA) register contains the upper 18 bits of a 22-bit memory address. The lower four bits of this address are considered zeros. Just prior to initial or continued execution of a program, a process known as the "exchange sequence" stores into the BA register the upper 18 bits of the lowest memory address to be referenced during program execution. As the program executes, the address portion of each instruction referencing memory has its content added to that of the BA register. The sum then serves as the absolute address used for the memory reference and ensures that memory addresses lower than the contents of the BA register are not accessed. Programs must, therefore, have all instructions referencing memory do so with their address portions containing relative addresses. This process supports program loading and memory protection operations and does not, in producing an absolute address, affect the content of the instruction buffer, BA, or memory.

LA *register* – the 18-bit limit address (LA) register contains the upper 18 bits of a 22-bit memory address. The lower 4 bits of this address are considered zeros. Just prior to initial or continued execution of a program, the "exchange sequence" process stores into the LA register the upper 18 bits of that absolute address one greater than allowed to be referenced by the program. When program execution begins, each instruction referencing a memory location has the absolute address for that reference (determined by summing its address portion with the BA register contents) checked against the LA register content. If the absolute

address equals or exceeds the LA register content, an out-of-range error condition is flagged and program execution terminates. This process supports the memory protection operation.

XA *register*—the 8-bit exchange address (XA) register contains the upper eight bits of a 12-bit memory address. The lower four bits of the address are considered zeros. Because only twelve bits are used, with the lower four bits always being zeros, exchange addresses can reference only every 16th memory address beginning with address 0000 and concluding with address 4080. Each of these addresses designates the first word of a 16-word set. Thus, 256 sets (of 16 memory words each) can be specified. Prior to initiation or continuation of a program's execution, the XA register contains the first memory address of a particular 16-word set or exchange package. The exchange package contains certain operating and support registers' contents as required for operations following an interrupt. The XA register supports the exchange sequence operation and the contents of XA are stored in an exchange package whenever an exchange sequence occurs.

F *register*—the 9-bit F register contains flags that, whenever set, indicate interrupt conditions causing initiation of an exchange sequence. The interrupt conditions are: normal exit, error exit, i/o interrupt, uncorrected memory error, program range error, operand range error, floating-point overflow, real-time clock interrupt, and console interrupt.

M *register*—the M (mode) register is a three-bit register that contains part of the exchange package for a currently active program. The three bits are selectively set during an exchange sequence. Bit 37, the floating-point error mode flag, can be set or cleared during the execution interval for a program through use of the 0021 and 0022 instructions. The other two bits (bits 38 and 39) are not altered during the execution interval for the exchange package and can only be altered when the exchange package is inactive in storage. Bits are assigned as follows in word two of the exchange package.

Bit 37—Floating-point error mode flag. When this bit is set, interrupts on floating-point errors are enabled.

Bit 38—Uncorrectable memory error mode flag. When this bit is set, interrupts on uncorrectable memory parity errors are enabled.

Bit 39—Monitor mode flag. When this bit is set, all interrupts other than parity errors are inhibited.

Integer Arithmetic

All integer arithmetic is performed in 24-bit or 64-bit 2's complement form.

Floating-Point Arithmetic

Floating-point numbers are represented in signed magnitude form. The format is a packed signed binary fraction and a biased binary integer exponent. The fraction is a 49-bit signed magnitude value. The exponent is 15-bit biased. The unbiased exponent range is:

2^{-20000_8} to 2^{+17777_8},

or approximately

10^{-2500} to 10^{+2500}

An exponent equal to or greater than 2^{+20000_8} is recognized by the floating-point functional units as an overflow condition, and causes an interrupt if floating point interrupts are enabled.

Chaining

The chaining technique takes advantage of the parallel operation of functional units. Parallel vector operations may be processed in two ways: (a) using different functional units and V registers, and (b) chaining; that is, using the result stream to one vector register simultaneously as the operand set for another operation in a different functional unit.

Parallel operations on vectors allow the generation of two or more results per clock period. A vector operation either uses two vector registers as sources of operands or uses one scalar register and one vector register as sources of operands. Vectors exceeding 64 elements are processed in 64-element segments.

Basically, chaining is a phenomenon that occurs when results issuing from one functional unit (at a rate of one/clock period) are immediately fed into another functional unit and so on. In other words, intermediate results do not have to be stored to memory and can be used even before the vector operation that created them runs to completion.

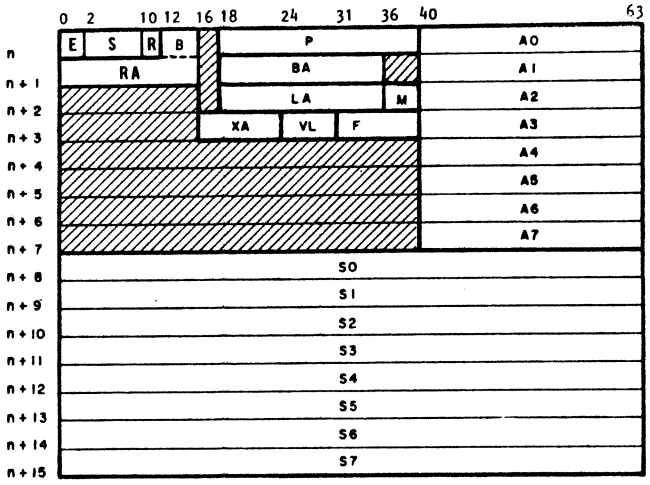
Chaining has been compared to the technique of "data forwarding" used in the IBM 360/195. Like data forwarding, chaining takes place automatically. Data forwarding consists of hardware facilities within the 195 floating-point processor communicating automatically by transferring "name tags," or internal codes between themselves [6]. Unlike the CRAY-1, the user has no access to the 195's data-forwarding buffers. And, of course, the 195 can only forward scalar values, not entire vectors.

Interrupts and Exchange Sequence

Interrupts are handled cleanly by the CRAY-1 hardware. Instruction issue is terminated by the hardware upon detection of an interrupt condition. All memory bank activity is allowed to complete as are any vector instructions that are in execution, and then an exchange sequence is activated. The Cray Operating System (cos) is always one partner of any exchange sequence. The cause of an interrupt is analyzed during an exchange sequence and all interrupts are processed until none remain.

Only the address and scalar registers are maintained in a program's exchange package (Fig. 6). The user's B, T, and V registers are saved by the operating system in the user's Job Table Area.

Fig. 6. Exchange package.



M - Modes†

36 Interrupt on correctable memory error
37 Interrupt on floating point
38 Interrupt on uncorrectable memory error
39 Monitor mode

F - Flags†

31 Console interrupt
32 RTC interrupt
33 Floating point error
34 Operand range
35 Program range
36 Memory error
37 I/O interrupt
38 Error exit
39 Normal exit

Registers

S Syndrome bits
RAB Read address for error (where B is bank)
P Program address
BA Base address
LA Limit address
XA Exchange address
VL Vector length

E - Error type (bits 0,1)

10 Uncorrectable memory
01 Correctable memory

R - Read mode (bits 10,11)

00 Scalar
01 I/O
10 Vector
11 Fetch

†Bit position from left of word

The CRAY-1's exchange sequence will be familiar to those who have had experience with the CDC 7600 and Cyber machines. One major benefit of the exchange sequence is the ease with which user jobs can be relocated in memory by the operating system. On the CRAY-1, dynamic relocation of a user job is facilitated by a base register that is transparent to the user.

Evolution of the CRAY-1

The CRAY-1 stems from a highly successful line of computers which S. Cray either designed or was associated with. Mr. Cray was one of the founders of Control Data Corporation. While at CDC, Mr. Cray was the principal architect of the CDC 1604, 6600, and 7600 computer systems. While there are many similarities with these earlier machines, two things stand out about the CRAY-1; first it is a vector machine, secondly, it utilizes semiconductor memories and integrated circuits rather than magnetic cores and discrete components. We classify the CRAY-1 as a second generation vector processor. The CDC STAR 100A and the Texas Instruments ASC are first-generation vector processors.

Both the STAR 100 and the ASC are designed to handle long vectors. Because of the startup time associated with data streaming, vector length is of critical importance. Vectors have to be long if the STAR 100 and the ASC vector processors are to be at all competitive with a scalar processor [3]. Another disadvantage of the STAR 100 architecture is that elements of a "vector" are required to be in consecutive addresses.

In contrast with these earlier designs, the CRAY-1 can be termed a short vector machine. Whereas the others require vector lengths of a 100 or more to be competitive with scalar processors, the cross-over point between choosing scalar rather than vector mode on the CRAY-1 is between 2 and 4 elements. This is demonstrated by a comparison of scalar/vector timings for some mathematical library routines shown in Figure 1 [7].

Also, the CRAY-1's addressing scheme allows complete flexibility. When accessing a vector, the user simply specifies the starting location and an increment. Arrays can be accessed by column, row, or diagonal; they can be stepped through with nonunary increments; and, there are no restrictions on addressing, except that the increment must be a constant.

Vector Startup Times

To be efficient at processing short vectors, vector startup times must be small. On the CRAY-1, vector instructions may issue at a rate of one instruction parcel per clock period. All vector instructions are one parcel instructions (parcel size = 16 bits). Vector instructions place a reservation on whichever functional unit they use, including memory, and on the input operand registers. In some cases, issue of a vector instruction may be delayed by a time (in clock periods) equal to vector length of the preceding vector operation + 4.

Functional unit times are shown in Table II. Vector operations that depend on the result of a previous vector operation can usually "chain" with them and are delayed for a maximum "chain slot" time in clock periods of functional unit time + 2.

Once issued, a vector instruction produces its first result after a delay in clock periods equal to functional unit time. Subsequent results continue to be produced at a rate of 1 per clock period. Results must be stored in a vector register. A separate instruction is required to store the final result vector to memory. Vector register capacity is 64-elements. Vectors longer than 64 are processed in 64-element segments.

Some sample timings for both scalar and vector are shown in Table III [8]. Note that there is no vector ASIN routine and so a reference to ASIN within a vectorized loop generates repetitive calls to the scalar ASIN routine. This involves a performance degradation but does allow the rest of the loop to vectorize (in a case where there are more statements than in this example). Simple loops 14, 15, and 16 show the

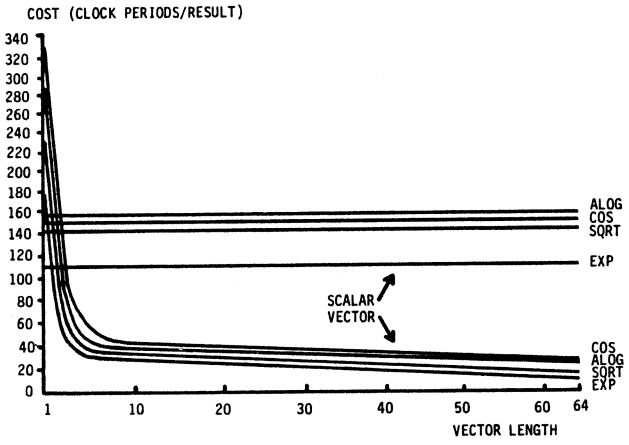
Table III.

Execution time in clock periods per result for various simple DO loops of the form

```
DO 10 I = 1,N
10 A(I) = B(I)
```

Loop Body	N = 1	10	100	1000	1000 Scalar
1. $A(I) = 1.$	41.0	5.5	2.6	2.5	22.5
2. $A(I) = B(I)$	44.0	5.8	2.7	2.5	31.0
3. $A(I) = B(I) + 10.$	55.0	6.9	2.9	2.6	37.0
4. $A(I) = B(I) + C(I)$	59.0	8.2	3.9	3.7	41.0
5. $A(I) = B(I)*10.$	56.0	7.0	2.9	2.6	38.0
6. $A(I) = B(I)*C(I)$	60.0	8.3	4.0	3.7	42.0
7. $A(I) = B(I)/10.$	94.0	10.8	4.1	3.7	52.0
8. $A(I) = B(I)/C(I)$	89.0	13.3	7.6	7.2	60.0
9. $A(I) = SIN(B(I))$	462.0	61.0	33.3	31.4	198.1
10. $A(I) = ASIN(B(I))$	430.0	209.5	189.5	188.3	169.1
11. $A(I) = ABS(B(I))$	61.0	7.5	2.9	2.6	
12. $A(I) = AMAX1(B(I), C(I))$	80.0	11.2	5.2	4.8	
13. $\left\{\begin{matrix} C(I) = A(I) \\ A(I) = B(I) \\ B(I) = CCI \end{matrix}\right\}$	90.0	12.7	6.3	5.8	47.0
14. $A(I) = B(I)*C(I) + D(I)*E(I)$	110.0	16.0	7.7	7.1	57.0
15. $A(I) = B(I)*C(I) + (D(I)*E(I))$	113.0	14.7	6.6	6.0	63.0
16. $A(I) = B(I)*C(I) + D(I)$	95.0	12.7	5.5	5.0	52.0

Fig. 7. Scalar/vector timing.



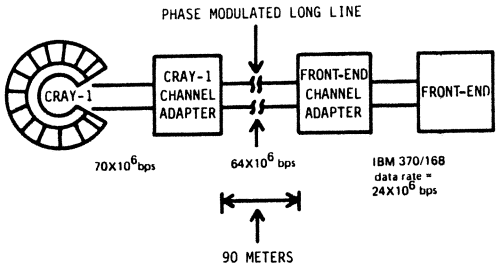
influence of chaining. For a long vector, the number of clock periods per result is approximately the number of memory references + 1. In loop 14, an extra clock period is consumed because the present CFT compiler will load all four operands before doing computation. This problem is overcome in loop 15 by helping the compiler with an extra set of parentheses.

Software

At the time of this writing, first releases of the CRAY Operating System (cos) and CRAY Fortran Compiler (CFT) have been delivered to user sites. cos is a batch operating system capable of supporting up to 63 jobs in a multiprogramming environment. cos is designed to be the recipient of job requests and data files from front-end computers. Output from jobs is normally staged back to the front-ends upon job completion.

CFT is an optimizing Fortran compiler designed to compile ANSI 66 Fortran IV to take best advantage of the CRAY-1's vector processing architecture. In its present form, CFT will not attempt to vectorize certain

Fig. 8. Front-end system interface.



loops which, due to dependence conditions, appear at first sight, unvectorizable.

However, future versions of CFT will be designed to eliminate as many dependency conditions as possible increasing the amount of vectorizable code. Basically, to be vectorizable, a DO loop should manipulate arrays and store the results of computations in arrays. Loops that contain branches such as GO TO's, IF's, or CALL statements are not currently vectorized. Loops may contain function references if the function is known to the compiler to have a vector version. Most of the mathematical functions in the CRAY library are vectorizable. By using the vector mask and vector merge features of the CRAY-1, future versions of the compiler will be able to vectorize loops containing IF and GO TO statements.

Early experience with CFT has shown that most Fortran loops will not run as fast as optimally hand-coded machine language equivalents. Future versions of CFT will show improved loop timings due mainly to improved instruction scheduling.

Other CRAY-1 software includes Cray Assembler Language (CAL) which is a powerful macro assembler, an overlay loader, a full range of utilities including a text editor, and some debug aids.

Front-End Computer Interface

The CRAY-1 was not designed for stand-alone operation. At the very minimum a minicomputer is required to act as a conduit between the CRAY-1 and the everyday world. Cray Research software development is currently being done using a Data General Eclipse computer in this category. The Cray Research "A" processor, a 16-bit, 80 MIPS minicomputer is scheduled to replace the Eclipse in early 1978. Front-end computers can be attached to any of the CRAY-1's 12 i/o channels.

The physical connection between a front-end computer and the CRAY-1 is shown in Figure 8. In this example an IBM 370/168 is assumed in the front-end role. Note that each computer requires a channel adapter between its own channel and a Cray Research phase-modulated long line. The link can only be driven at the speed of its slowest component. In this example it is the IBM block multiplexer channel speed of 3 megabytes/second. The discipline of the link is governed by the Cray Link Interface Protocol.

CRAY-1 Development Problems

Two of the most significant problems [9] encountered on the way to the CRAY-1 were building the first cold bar and designing circuits with a completely balanced dynamic load.

Building the Cold Bar

It took a year and a half of trial and error before the first good cold bar was built. The work was done by a small Minnesota company. A major problem was the discovery, quite early, that aluminum castings are porous. If there is a crack in the stainless steel tubing at the bond between the tubing and the elbow then the Freon leaks through the aluminum casing. The loss of the Freon is not itself a problem, but mixed with the Freon is a little oil, and the oil can cause problems if it is deposited on the modules. Aluminum also tends to get bubbles in it when it is cast, requiring a long process of temperature cycling, preheating of the stainless steel tube, and so on.

Designing the Circuits

CRAY-1 modules are 6 inches wide. The distance across the board is about a nanosecond which is just about the edge time of the electrical signals. Unless due precautions are taken, when electric signals run around a board, standing waves can be induced in the ground plane. Part of the solution is to make all signal paths in the machine the same length. This is done by padding out paths with foil runs and integrated circuit packages. All told, between 10 and 20 per cent of the IC packages in the machine are there simply to pad out a signal line. The other part of the solution was to use only simple gates and make sure that both sides of every gate are always terminated. This means that there is no dynamic component presented to the power supply. This is the principal reason why simple gates are used in the CRAY-1. If a more complex integrated circuit package is used, it is impossible to terminate both sides of every gate. So all of the CRAY-1's circuits are perfectly balanced. Five layer boards have one ground layer, two voltage layers, and then the two logic layers on the outside. Twisted pairs which interconnect the modules are balanced and there are equal and opposite signals on both sides of the pairs. The final result is that there is just a purely resistive load to the power supply!

Summary

The design of the CRAY-1 stems from user experience with first generation vector processors and is to some extent, evolved from the 7600 [2]. The CRAY-1 is particularly effective at processing short vectors. Its architecture exhibits a balanced approach to both scalar and vector processing. In [1], the conclusion is drawn that the CRAY-1 in scalar mode is more than twice as fast as the CDC 7600. Such good scalar performance is required in what is often an unvectorizable world.

At the time of this writing, Cray Research has shipped CRAY-1 systems to three customers (Los Alamos Scientific Laboratory, National Center for Atmospheric Research, and the European Center for Medium Range Weather Forecasts) and has contracts to supply three more systems, two to the Department of Defense, and one to United Computing Systems (UCS). Production plans already anticipate shipping one CRAY-1 per quarter. As the population of CRAY-1 computers expands, it will become clear that the CRAY-1 has made a significant step on the way to the general-purpose computers in the future.

Received February 1977; revised September 1977

Acknowledgments. Acknowledgments are due to my colleagues at Cray Research. G. Grenander, R. Hendrickson, M. Huber, C. Jewett, P. Johnson, A. La Bounty, and J. Robidoux, without whose contributions, this paper could not have been written.

References

1. CRAY-1 Final Evaluation by T. W. Keller, LASL, LA-6456-MS.
2. CRAY-1 Report, Auerbach Computer Technology Report, Auerbach Publisher's, 6560 North Park Drive, Pennsauken, N. J. 08109.
3. Preliminary Report on Results of Matrix Benchmarks on Vector Processors: Calahan, Joy, Orbits, System Engineering Laboratory, University of Michigan, Ann Arbor, Michigan 48109.
4. Computer Architecture Issues in Large-Scale Systems, 9th Asilomar Conference, Naval Postgraduate School, Monterey, California.
5. Computer World, August 1976.
6. The IBM 360/195 by Jesse O'Murphy and Robert M. Wade, Datamation, April 1970.
7. Work done by Paul Johnson, Cray Research.
8. Work done by Richard Hendrickson, Cray Research.
9. The section on CRAY-1 development problems is based on remarks made by Seymour Cray in a speech to prospective CRAY-1 users in 1975.

APPLICATIONS OF VECTOR PROCESSING

The vector-scalar processor, a unified supercomputer system that has very high speed computation capabilities, can execute 140 to 200 million operations per second in scientific and industrial vector signal processing applications. Analysis of algorithms associated with various applications leads to successful implementations

Lee Higbie Cray Research, Incorporated, Minneapolis, Minnesota

Scalar and vector signal processing and related algorithms, as implemented on the CRAY-1™ computer system, are applicable to many powerful general-purpose processing systems and illustrate the techniques used to code pipeline or array processors, as well as vector-scalar systems.

Two dichotomies of signal processing applications exemplify the spectrum of powerful supercomputer techniques. The first is active/passive; many signal types are reflections (usually) of signals that are transmitted, then received. Petroleum exploration seismology, radar, active sonar, and atmospheric sounding systems are in the active signal processing category. Typical passive signals are found in earthquake seismology, radio signal processing, passive sonar, radio telescopy, electroencephalography, and electrocardiography. The second dichotomy is acoustic/electromagnetic; acoustic signals are typified by those in seimology and sonar, while electromagnetic signals include radar and radio.

A large number of common computational procedures are used in many of these applications to process the signals received from multiple sensors. These procedures generally fall into one of several categories: direction finding and signal enhancement, frequency analysis and separation, and a plethora of "downstream" routines that are dependent on signal type, such as signature detection, analysis, and cataloging; clutter elimination; and resource scheduling.

Ed Note: This article complements "An Introduction to Vector Processing," by Paul M. Johnson, which appeared in *Computer Design*, Feb 1978, pp 89-97. While the previous article established basic principles of parallel vector processing, this follow-up demonstrates several real-world implementations as performed on a supercomputer system.

Computer Architecture

The vector-scalar signal processor contains both a very high speed vector processing unit and a very high speed scalar processing unit. It is difficult to characterize the raw computing power of a powerful general-purpose computer capable of extremely high processing rates; however, speeds of approximately 140 million floating-point operations per second (MFLOPS) are attainable when multiplying large matrices, and 200 MFLOPS are possible in bursts. These rates are achieved by combining scalar and vector capabilities into a single central pro-

™CRAY-1 is a registered trademark of CRAY Research, Inc, Minneapolis, Minn.

cessor which is joined to a large, fast, bipolar memory. Vector processing—performing iterative operations on sets of ordered data—provides result rates that greatly exceed those of conventional scalar processing. Scalar operations complement the vector capability by providing solutions to problems not readily adaptable to vector techniques.

Basic organization of a CRAY-1 computer system is discussed and illustrated in Johnson's article (see Bibliography). The central processor unit (CPU) is a single integrated processing unit consisting of a computation section, a memory section, and an input/output (I/O) section. Memory is currently expandable from 0.25 million 64-bit words to 1 million words (2M to 8M bytes). The 12 full-duplex I/O channels in the I/O section connect to a maintenance control unit (MCU), a mass storage subsystem, and a variety of I/O stations or peripheral equipment. The MCU provides for system initialization and for monitoring system performance. The mass storage subsystem provides secondary storage and consists of one to eight disc controllers, each with one to four disc storage units. Each storage unit has a capacity of 2.424 x 10^9 bits, so that a maximum mass storage configuration could hold 9.7 x 10^9 8-bit bytes.

Computation section consists of an instruction control network, operating registers, and functional units. The instruction control network performs all decisions related to instruction issue and coordinates activities for three types of processing—vector, scalar, and address. Associated with each type of processing are registers and functional units that support the processing mode. For vector processing, these include a set of 64-bit multielement vector registers (V0 to V7), three functional units—add, logical, and shift—dedicated to vector applications, and three floating-point functional units—add, multiply, and reciprocal approximation—supporting both scalar and vector operations. For scalar processing, there are two levels of 64-bit scalar registers (S0 to S77 and T0 to T77) and four functional units—add, logical, shift, and population/leading zero count—dedicated to scalar processing, in addition to the three floating-point functional units shared with vector operations. For address processing, there are two levels of 24-bit registers (A0 to A7 and B0 to B77) and two integer arithmetic functional units—add and multiply.

Vector and scalar processing are performed on data as opposed to address processing, which operates on internal control information such as addresses and indexes. Data flow in the computation section is generally from memory to registers and from registers to functional units. The flow of results is from functional units to registers and from registers to memory or back to functional units. Data flows along either the scalar or vector path depending on the mode of processing.

Flow of address information is from memory or control registers to address registers. Information in the address registers can then be distributed to various parts of the control network for use in controlling the scalar, vector, and I/O operations. Address registers can also supply operands to the two integer arithmetic functional units. These units generate address and index information and return the result to the address registers. Address information can also be transmitted to memory from the address registers.

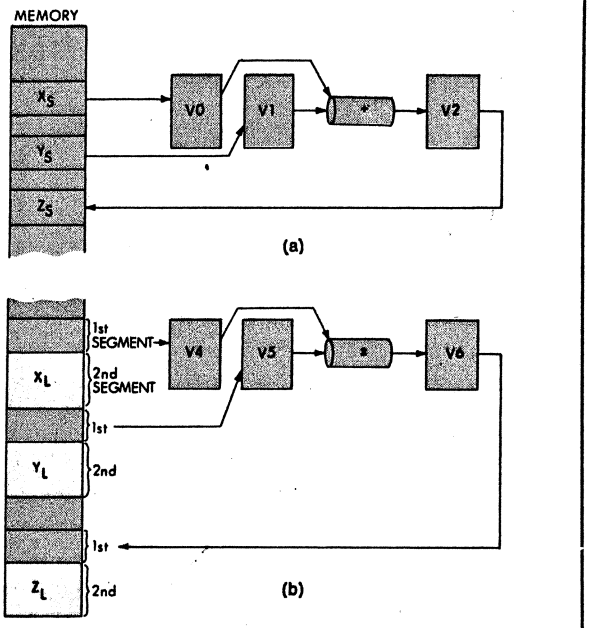


Fig 1 Vector segmentation. Segmenting of vector operations involves vectors of more than 64 elements. This processing procedure is required for computers with hardware limitations on vector length. The programmer never sees this segmentation. In short vector addition (a), $X_s + Y_s \rightarrow Z_s$, no segmenting is required. Long vector multiplication (b), $X_L * Y_L \rightarrow Z_L$, however, does require segmenting. In this diagram, the shaded segment is processed after the unshaded segment

Processing Techniques

All processing operations are performed between registers, allowing prefetching of operands and poststoring of results, as well as fast access to intermediate results. Since each of the eight V registers holds up to 64 elements (each is a 64-bit word) of a vector, long vectors are processed in blocks of 64 elements and a remainder, as shown in Fig 1. Here the sum of two 40-element vectors and the product of two 100-element vectors are illustrated. The 40-element vectors are added in one operation, while the 100-element vectors require two steps (this is transparent in FORTRAN, of course).

During most calculations, the scalar processor performs bookkeeping operations, such as indexing, counter incrementing, and checking to determine what code is next, while floating-point operations are streaming through the vector processor. Since these scalar operations do not involve floating-point operations, they proceed in parallel with vector processing. For codes that are "vectorizable", bookkeeping is generally done in a small fraction of the time required for vector operations; thus, there is no delay in starting a vector operation after the previous operation has finished.

Because of the independence and parallelism of the functional units, the processing of V1 + V2 * V3 proceeds as follows:

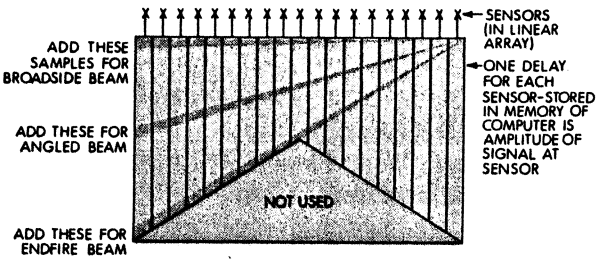


Fig 2 Beamforming. Delay line for each sensor is shown as a vertical subrectangle of the total beamformer memory. Each time a new set of amplitude samples is collected, all samples in the beamformer are logically moved down one row to make room for the next set of samples

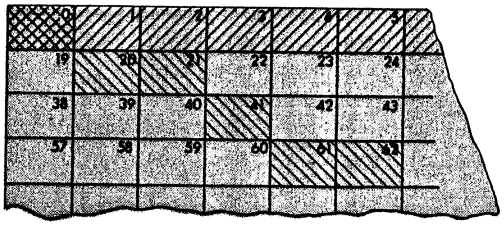


Fig 3 Broadband digital beamforming. Shaded cells (0 to 5) form the broadside beam and the beam value is computed using a vector memory fetch. Shaded cells (0, 20, 21, 41, 61, and 62) form an oblique beam and must be fetched serially because of nonlinearity of their subscript sequence. Vertical columns of boxes represent delay lines of Fig 1; each box shows one memory cell. Number in each box is its relative address in memory

	Process Step		Comment
LOAD	V1		Load 1st operand into register V1
LOAD	V2		Load 2nd operand into register V2
LOAD	V3	Chain	Load 3rd operand into register V3
MULTIPLY	V2 * V3 → V4	Chain	Multiply V2 by V3 and place result into V4 (temporarily)
ADD	V1 + V4 → V5	Chain	Add V4 to V1 and place result into V5
STORE	V5		Store result from V5

A delay exists in getting each pipe† started (memory accessing pipe, multiply pipe, and add pipe), but once started, all operations proceed together at a computational rate of 160 MFLOPS. Pipe startup time is eight or nine clock cycles, and the above operation takes 81 clock periods for vectors with 64 elements (eight clocks for add startup plus nine clocks for multiply startup plus 64 clocks for 64 elements). Concurrent vector operations are called "chaining," and 81 clock periods are equal to one chain time. Because of the short startup time of the vector functional units, there is little loss of speed when using short vectors. For typical operations, vector lengths of three elements or less run faster in scalar mode, while those of four elements or more run faster in vector mode. In addition, vector mode is about five times faster than scalar mode on long vectors such as those that must be segmented into blocks of length 64 or less. Transfers between memory and vector registers all proceed at one word per clock period, unless the increment between successive memory locations, which is arbitrary, is a multiple of ±8.

Clearly any vector operation can be processed as an iterated scalar operation, as it is on any scalar processor. The "scalar mode" referred to previously is for the operation performed in an iterated scalar loop instead of in a vector operation.

Application Algorithms

Several common algorithms that perform the operations required to process signals received from a seismic or passive sonar system using multiple sensors are presented as the type of application readily handled by a supercomputer. Usually, these signals are enhanced first by beamforming; then, they are filtered and frequency analyzed. Frequency analysis is normally accomplished with a fast Fourier transform (FFT). After the FFT, interpolation and signature analyses are usually performed.

Beamforming Operation

The initial signal-processing operation performed in a multiple-sensor system is the formation of a beam or beams. This operation provides two types of information enhancement on the signals: determine signal directional information and increase signal-to-noise ratio. For application simplicity, an equally spaced linear array of sensors is discussed. Fig 2 shows an array of sensors, each attached to a delay line so that its output signal is stored for a short period of time. If the delay line for each sensor is long enough for a signal to propagate to the farthest end of the sensor array in the time that a sampling signal passes down the delay line, beams can be formed in all directions. Beams in three directions—broadside, at an oblique angle, and end-on—are formed by adding the elements from the delay line, as shown by the shaded areas.

Computationally, it is necessary to be able to add the elements of a vector with as few as 12, or as many as several thousand, elements because arrays of such disparate numbers of elements are used. Furthermore, especially for large sensor arrays, it is frequently necessary to weight the signals (multiply by a weighting coefficient before summing) from some sensors more heavily than others so that the required operation is an inner product.

$$\sum_i w_i * x_i\,(t_{i\alpha})$$

where

w_i = weight for shaping
x_i = ith datum
$t_{i\alpha}$ = delay for beam at angle α for ith sensor

†The term "pipe" is used for a functional unit because it is pipelined.

```
CASE 1: NO SHAPING
LOAD ADDRESSES FOR BEAMS INTO REGISTERS
A1 <---- 1ST ADDRESS
S1 <---- (A1)  1ST BEAM ELEMENT
COMMENT: "INDIRECT" ADDRESSES AND 1ST
SUMMAND ARE NOW LOADED
FOR     I  <---- 2 TO M
        A1 <---- ITH ADDRESS
        S2 <---- (A1)
        S1 <---- S1 + S2
COMMENT: THIS LOOP SUMS ALL BEAM ELEMENTS
TO PRODUCE BEAM
CASE 2: SHAPED BEAM
LOAD ADDRESSES FOR BEAMS INTO REGISTERS
FOR     K  <---- 1 TO M/64 + 1
        I  <---- 1 to MIN (M, 64)
        A1 <---- ITH ADDRESS
        S1 <---- (A1)
        V1I <--- S1
*V4 <---- SHAPING WEIGHTS, SEGMENT k
 V5 <---- V4 * V1
 V5 <---- V5 + V4
COMMENT: HERE THE SHAPING MULTIPLY
IS DONE WITH A VECTOR OPERATION TO
SAVE TIME
RECURSIVE ADD OF V5
V5 <---- (V5 + V1) OR (V5 + V3)
COMMENT: THE RECURSIVE SUM ADDS THE
64-ELEMENT VECTOR ABOVE TO AN 8-ELEMENT
VECTOR
S2 <---- V56  56TH ELEMENT OF V5
FOR     I  <---- 57 TO 63          } SUMS
        S1 <---- V5 ITH ELEMENT OF V5 } OUTPUT OF
        S2 <---- S1 + S2           } RECURSIVE SUM
COMMENT: THIS LOOP SUMS THE LAST
EIGHT ELEMENTS
STORE   S2 = Σ (i=1 to N) xI  INTO SUM
```

Fig 4 Beamforming codes. Both shaped and unshaped beamformers are shown. Each case is for an oblique beam that requires irregular addressing of memory

The subscript i varies over all the sensors; weights may depend on the angle, be equal for all directions, or be unity. An additional complication is that indexing is not regular throughout the array, ie, i_a values do not form a linearly increasing sequence (Fig 3). Thus, a vector fetch of the data is not possible.

Beamforming Implementation

In beamforming, the first problem encountered is storing data in memory. It is assumed that data are stored as received; in other words, during each time interval, one datum from each sensor is stored in sensor order in a circular buffer. Beams are formed by computing the sum of a sample for each sensor, such as those samples shown by the shaded areas in Fig 3. Irregular accessing of data is required except for the case of very rapid sampling, when the increment between beam samples is uniform or when only one beam is formed, as in seismic operations. Fig 4 shows both implementations of this algorithm. In the first or unshaped beam case, a fast scalar loop is used because the arithmetic is simple; the second or shaped beam case uses vector operations because of the more complicated arithmetic. Irregular addressing and very high speed of the scalar processor, causes samples to be serially fetched from memory. If the beam is not being shaped, these samples can be added as they are fetched (Case 1); otherwise, they are loaded into a vector register for multiplication by shaping factors, and then added using the recursive-add operation (Case 2). For the special case where data accessing is regular, vector operations can be used throughout. This load chains into the shaping and summing operations; ie, it is concurrent with them.

Filtering Operation

Once beams are formed, it is common practice to filter the signals so that the sampling rate can be reduced; sampling at a rate more than double that of the highest

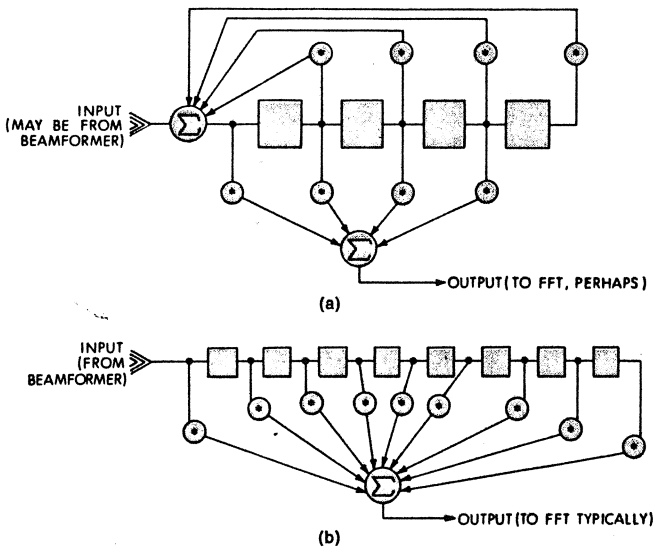


Fig 5 Two common types of digital filters. Efficient short vector operations allow easy implementation of both (a) recursive or infinite impulse response filter and (b) explicit or finite impulse response filter. Each box represents a unit time delay, each circled asterisk represents multiplication by a filter coefficient, and each circled Σ represents summation of all inputs

frequency in the signal yields no additional information. Reduction of the sample rate, called down-sampling or decimation, decreases the volume of data to be processed by succeeding operations. Fig 5 shows signal flow diagrams (not flowcharts) for two common filter types. The first type is called a recursive filter; it is the digital implementation of an analog infinite impulse response filter. Just as an analog filter can ring, this type of digital filter continues producing an output long after the input has gone to zero because of the recursive nature of the processed signal; once impulsed, this filter produces an output indefinitely. The second type is computationally explicit and, because no initial excitation will cause it to ring, is referred to as a finite impulse response or explicit filter.

The recursive filter cannot be calculated for large numbers of data points in parallel because each filter output value is used as an input for the next computation of the next filter output. In other words, computations of output n+1 cannot be initiated until that for output n is completed. Thus, the short vector capability is necessary for efficient recursive filter implementation.

However, the explicit filter can be calculated on large numbers of inputs or for large numbers of outputs at the same time. For this reason, it is often preferred to the recursive filter for digital implementations. Although not shown in Fig 5, the coefficients are frequently symmetrical about the midpoint of the delay line and, if the output is being down-sampled by a factor of two, the filter computation is performed on only half of the input points. Thus, even though the recursive filter is generally more efficient in terms of multiplies per input, the nonrecursive filter is both more efficient and easier to compute when down-sampling and symmetric filter coefficients are used. The explicit or nonrecursive filter is a running inner product:

$$\sum_i f_i x_{i+j} \qquad j = 0,1,2,\ldots,n$$

where f_i are filter coefficients and x_k is datum k.

Filtering Implementation

There are two major approaches to producing high order filters: cascading several low order filters (ie, processing the signal output from one through the next in a chain of filters) or using a filter with many poles and zeroes. In signal processing systems with short word lengths, cascading is preferred because it reduces computational errors; highly parallel systems run more efficiently with high order filters.

In a recursive filter (Fig 6), short vector operations allow efficient implementation even for the 4-pole, 4-zero filter illustrated, making cascading filters feasible. In this case, there are four poles plus four zeros or eight coefficients, ie, vector length (VL) = 8. For high order filters, ie, those with many segments for the delay line, processing is more efficient because of the longer vectors that can be used in the processor. Cascading low order filters is not necessary because of the accuracy of the 48-bit mantissa floating-point arithmetic in the computer system. Fig 6 shows that each pole coefficient is multiplied by a recursed output and each zero coefficient is multiplied by an input datum; then the products are summed to produce an output.

In the program for the nonrecursive filter (Fig 7), to preserve generality, no decimation of the output is shown nor is any symmetry of the filter coefficients used, even though both of these may be used at times. Particularly noteworthy in this implementation are the large number of chained operations and the low rate of data transfers from memory to processing unit. This results directly from using multiple registers in the vector processor.

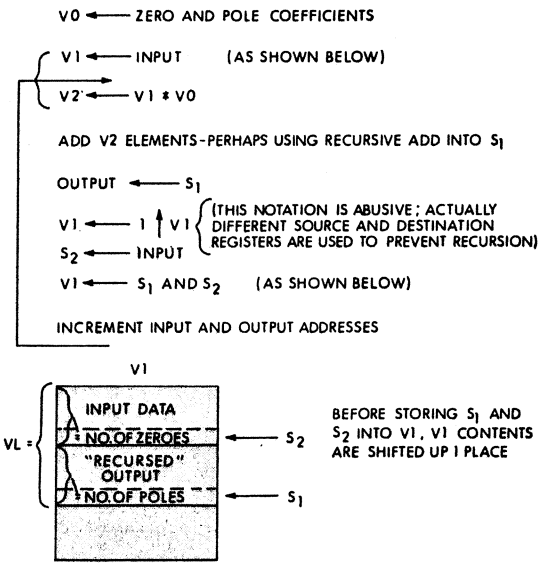


Fig 6 Code for infinite impulse response or recursive filter. A shift of V1 ← 1 ↑ V1 is shown; however, actual code requires ping-ponging between two registers. Code multiplies each input by a zero coefficient and each recursed output by a pole coefficient and sums products for next output

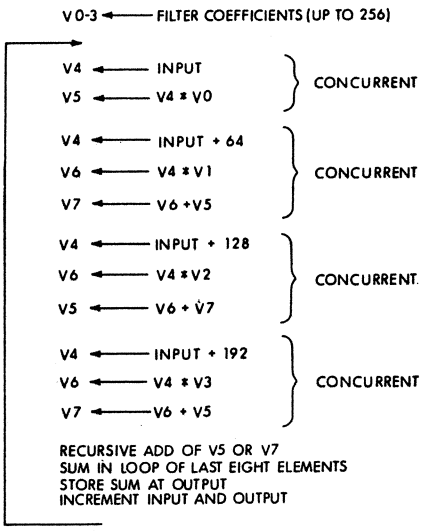


Fig 7 Code for finite impulse response filter. Large number of concurrent vector operations allows a 100-million operation per second execution rate of this code. Each input is multiplied by a coefficient and products are added to form an output

Fig 7 shows that each coefficient is multiplied by an input datum and the products are summed to produce an output.

Fast Fourier Transform Operation

Frequency analysis is generally applied to the filter output signals and is most commonly performed by an FFT algorithm. Figs 8 and 9 show an FFT algorithm similar to the original by Cooley (see Bibliography); however, Fig 9 shows the input amplitude samples on the left. The complicated signal routing and processing required for an FFT are illustrated in two equivalent signal flow graphs, with Fig 9 merely a rearrangement of Fig 8. Outputs and processing are identical; only the order is different. Note that the difference between the indices of successive input signals, which is the difference between memory locations, is usually divisible by a large power of two. Thus, this FFT algorithm tends to have many memory conflicts because there are 16 memory banks in the system. In addition, data must be shuffled into "bit-reverse" ordering (Fig 8), before the algorithm can be performed. Fig 9 shows a rearrangement of the signal flow diagram of Fig 8 with all inputs and outputs in numerical order, which saves rearranging the data; for this flow only, weights occur with subscript sequences that are separated by large powers of two. Operations in the FFT algorithm are all complex, except for those few that have weights of ± 1 or $\pm i = \pm\sqrt{-1}$. For these weights, the "multiplies" are all by ± 1.

FFT Implementation

For the typical FFT algorithm, a number of computationally equivalent procedures exist, some of which are much easier to implement on a vector processor than others (see Gold and Rader). The FFT algorithm shown in Fig 8 requires bit-reverse ordering of input data (left hand indexes), others generate bit-reverse ordered results. For a sequential processor, one of a number of special techniques can be used to reorder the data, such as a table lookup; however, these are not vectorizable, in general. Thus, as shown in Fig 9, the ordered-in, ordered-out FFT implementation is more straightforward. All data are accessed in natural order, in time sequence, and vectors in the processor are relatively long (more than 10 elements). Weights are accessed in groups where the exponents (subscripts, in effect, because the weights are precomputed) differ by large powers of two. In actual implementation, this difficulty is partially overcome by storing a small set of the weights a second time. This duplicate storing of weights is useful for those FFT stages where weights are used repeatedly.

The actual FFT is implemented with the first three stages (shown in Fig 8) done as special cases. In the first two stages, multiplication is not needed, thus saving computation; the third stage is also performed as a special case, because there is only one nontrivial coefficient, $\sqrt{2}$, ie, only one "multiplier" is different from ± 1 and $\pm i$.

By the fourth stage, there are four nontrivial weights, and it is easier to carry out the full multiplication. However, notice that the weights are used eight times over for 64-element vectors. Thus, the multiplication requires: (a) vectors of length eight, (b) serial fetching of weights and storing them repeatedly into a vector register, or (c) a block of weights repeated eight times. Because only a few weights are used in this stage, the solution is to repeat a few of the weights eight times and to use this set of repeated weights for the next two stages as well.

Memory accesses whose addresses are multiples of four proceed at full speed, thus the use of four times as many weights as necessary, each repeated eight times, allows all processing to be vectorized and all vector lengths to be a multiple of 64. With this scheme, the extra or repeated weights are used for the fourth, fifth, and sixth stages.

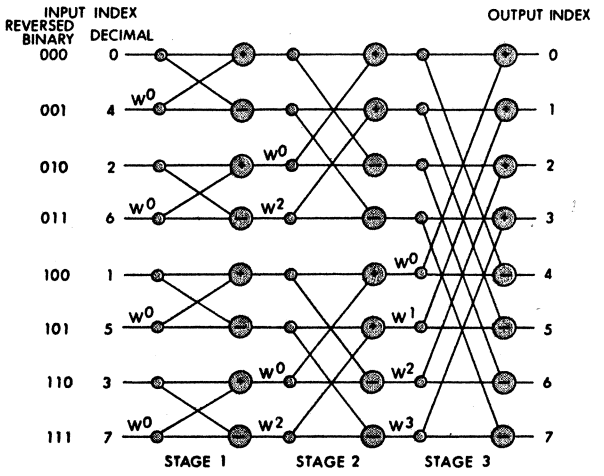


Fig 8 Original FFT signal flow diagram. Input indices differ by large powers of 2, producing frequent memory conflicts. For 2^n-point FFT, input indices differ by 2^{n-1} for every other pair of inputs, where w = 4th root of $1 = e^{2\pi i/4}$

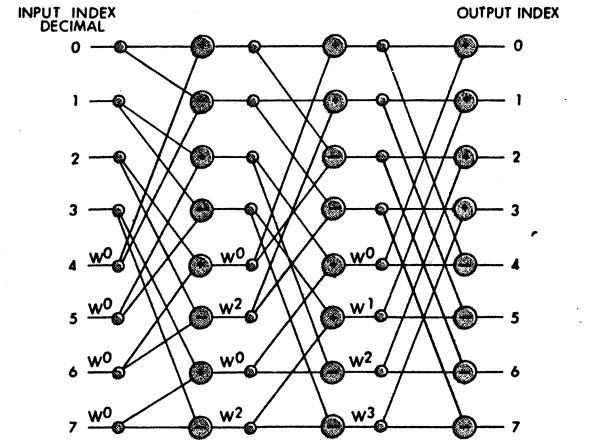


Fig 9 FFT with ordered-in, ordered-out arrangement. Inputs and outputs are in easy memory access locations and no preshuffling of data is required, where w = 4th root of $1 = e^{2\pi i/4}$

FORTRAN Usage

A FORTRAN programmer coding for a vector processor can take advantage of a number of possible techniques to increase code vectorizability. Loops that have all result names different from the operand names are likely to be vectorized by all FORTRAN compilers. Beyond this, compilers will vectorize depending on their sophistication and on the relative speeds of vector and scalar operations, which determine the importance of vectorization. For example, vector computer systems with poor short vector performance are more likely to have compilers that interchange program loops so that vector lengths are maximized; on vector-scalar computers this is not generally necessary. In any program, vectorizing compilers will attempt to code inner loops of programs as vector operations.

The following list presents inner loop constructs in order of increasing difficulty (approximately) for vectorization by compilers.

(1) Same variable names as operands and results
(2) Complicated subscript expressions
(3) Loop increments other than 1
(4) Functions with mixture of vector and scalar arguments, such as ATAN2 (X, Y(I))
(5) Scalar temporaries in loop
(6) Dependent or ambiguous subscripts
(7) Transfer out of loop
(8) Forward transfer in loop
(9) Subscripts defined before a numbered statement before a loop, yielding possibly nonlinear recursion, as

```
      IONE = 1
            .
            .
            .
50    CONTINUE
            .
            .
            .
      DO 100 I = 1, N
100   A(I) = A(I + IONE) * . . .
```

(10) Subroutine calls in loop (other than ones that are known to the compiler to be vectorized)
(11) Transfer into loop
(12) Backward transfer in loop

In item 9, the compiler cannot easily know whether the loop is nonlinearly recursive and, therefore, nonvectorizable.

Currently, the CRAY-1 FORTRAN compiler, (CFT), vectorizes loops that include items 1 to 5 and in some cases item 6. At this time, CFT generally produces better code for a few large loops rather than many small ones because there are only a few loop "overheads" instead of many. CFT recognizes common subexpressions and uses registers to avoid recomputation wherever possible. Using vector temporaries reduces speed by requiring more memory references; thus

```
      DO 100 I = 1,200
      TEMP(I) = A(I) * B(I) * SQRT (SIN (A(I) + K))
      X(I) = TEMP(I) + 1.0
200   Y(I) = TEMP(I) - 1.0
C     This code sets X_i = A_i*B_i*sqrt(sin(A_i+k)) + 1 and
C     Y_i = A_i*B_i*sqrt(sin(A_i+k)) -1 for i=1,2, . . . 200.
```

will execute more slowly than:

```
      DO 100 I = 1,200
      X(I) = A(I) *B(I) * SQRT(SIN(A(I) + K)) + 1.0
100   Y(I) = A(I) * B(I) *SQRT(SIN(A(I) + K)) -1.0
C     The same result as the previous case
```

The compiler will compute the common subexpression only once in either case but, in the second, it does not store this temporary result, resulting in faster execution.

Summary

Several signal processing algorithms and their implementations on a powerful vector-scalar processor have been described. Vector-scalar supercomputers are ideally suited to digital signal processing because of the very high processing rates that are required in scientific and industrial applications. These applications presently include energy, weather, and timesharing computations.

Because of tremendous speed requirements for vector signal processing applications, a final recapitulation of operating speeds for a vector-scalar supercomputer follows. Measurements are in millions of floating-point operations per second. Only required operations are counted; those needed by hardware or software are not included.

Operation	CRAY-1 Speed, MFLOPS
FFT	75
Nonrecursive filter	100
Recursive filter	60
Matrix multiply	140
Dot product	75
Constant Q interpolation	35
Search largest	30M compares/s

Bibliography

J. Cooley and J. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, No. 90, 1965, pp 297-301

Cray Research, "Cray-1 FORTRAN (CFT) Reference Manual," Pub 224009, Cray Research, Inc, Minneapolis, Minn, 1977

Cray Research, "Hardware Reference Manual," Pub 224004, Cray Research, Inc, Minneapolis, Minn, 1977

B. Gold and C. Rader, *Digital Processing of Signals*, McGraw-Hill, New York, 1969

L. Higbie, "Associative Processors: A Panacea or a Specific," *Computer Design*, July 1976, pp 75-82

P. Johnson, "An Introduction to Vector Processing," *Computer Design*, Feb 1978, pp 89-97

Lee Higbie, a marketing support analyst at Cray Research, has extensive high speed systems experience both as a designer and a user. He holds a BS degree in physics from St Lawrence University and an MS degree in mathematics from the California Institute of Technology.

AN INTRODUCTION TO VECTOR PROCESSING

Paul M. Johnson

Reprinted with permission from Computer Design — February, 1978 issue

AN INTRODUCTION TO VECTOR PROCESSING

Execution speed of scientific problems can be considerably enhanced by hardware and software design that provides for efficient execution of program loops. A large scale scientific computer incorporates vector processing for high speed execution of loops without sacrifice of processing speed in nonloop situations

Paul M. Johnson Cray Research, Incorporated, Minneapolis, Minnesota

Since program loops occur so frequently in scientific processing, providing for their efficient execution in both hardware and software design can considerably enhance the execution speeds of scientific problems. Within a loop, array indices are typically linear functions of the loop control variable. For example, the FORTRAN statements

```
    DO 100 I=1,21,2
100 A(I) = B(I+3) + 10
```

define a simple program loop that adds 10 to elements of array B and stores the sums in array A. The array indices—I in the case of array A and I+3 in the case of array B—are linear functions of I, the loop control variable; I ranges from 1 to 21 in steps of 2.

For most computers, the machine language equivalent of this FORTRAN loop is a sequence of instructions that reads a single element of the B array, adds the constant 10, and writes the result into the A array. The loop control variable is incremented and these steps are repeated until the variable equals the limit value. A single number, such as the individual elements of the B array or the constant 10, is called a scalar. Scalar processing is the application of arithmetic and logical operations on scalars. Some computers exploit the repetitive nature of loops through use of instructions that operate on series of numbers. One instruction reads a series of elements of the B array, another adds 10 to the elements, and yet another writes the series of sums into the A array. Such a series of numbers is called a vector, and vector processing is the application of arithmetic and logical operations on vectors. One major advantage of vector processing over scalar processing is elimination of overhead associated with maintenance of the loop control variable. In many cases, loops reduce to a simple sequence of instructions without backward branching.

However, not all aspects of a problem lend themselves to vector processing and, for these, scalar techniques should still be applied. Thus, one failing of early vector processors is their inability to compete successfully in scalar applications. Moreover, some vector processors require long vectors in order to show an advantage over conventional scalar processors (this is called the "start-up" time). Another failing is that memory conflicts due to the simultaneous reading of operand vectors from memory and writing of result vectors to memory often degraded vector performance.

Computer Architecture

Architecture of the CRAY-1® computer exhibits none of these objectionable traits. Conceptually, the machine is both a scalar and a vector processor, with instructions and registers for both applications. Start-up time for vector operations is short enough so that vector pro-

cessing is more efficient than scalar processing for vectors containing as few as two elements. Register to register vector instructions eliminate the problem of memory conflicts. Scalar and vector processing capabilities of the computer are characterized by high processing rates in vector applications.

Operating Registers

Primary operating registers are the scalar and vector registers, called S and V registers, respectively (see Fig 1). Each of the eight S registers has a single element; each of the eight V registers has 64 elements. Scalar instructions perform some function, such as addition, by obtaining operands from two S registers and entering the result into another S register. The analogous vector instruction repetitively performs the same function, obtaining new pairs of operands from elements of two V registers during each clock period (12.5 ns). Results are entered into elements of another V register. Contents of the vector length (VL) register determine the number of operations performed by vector instructions.

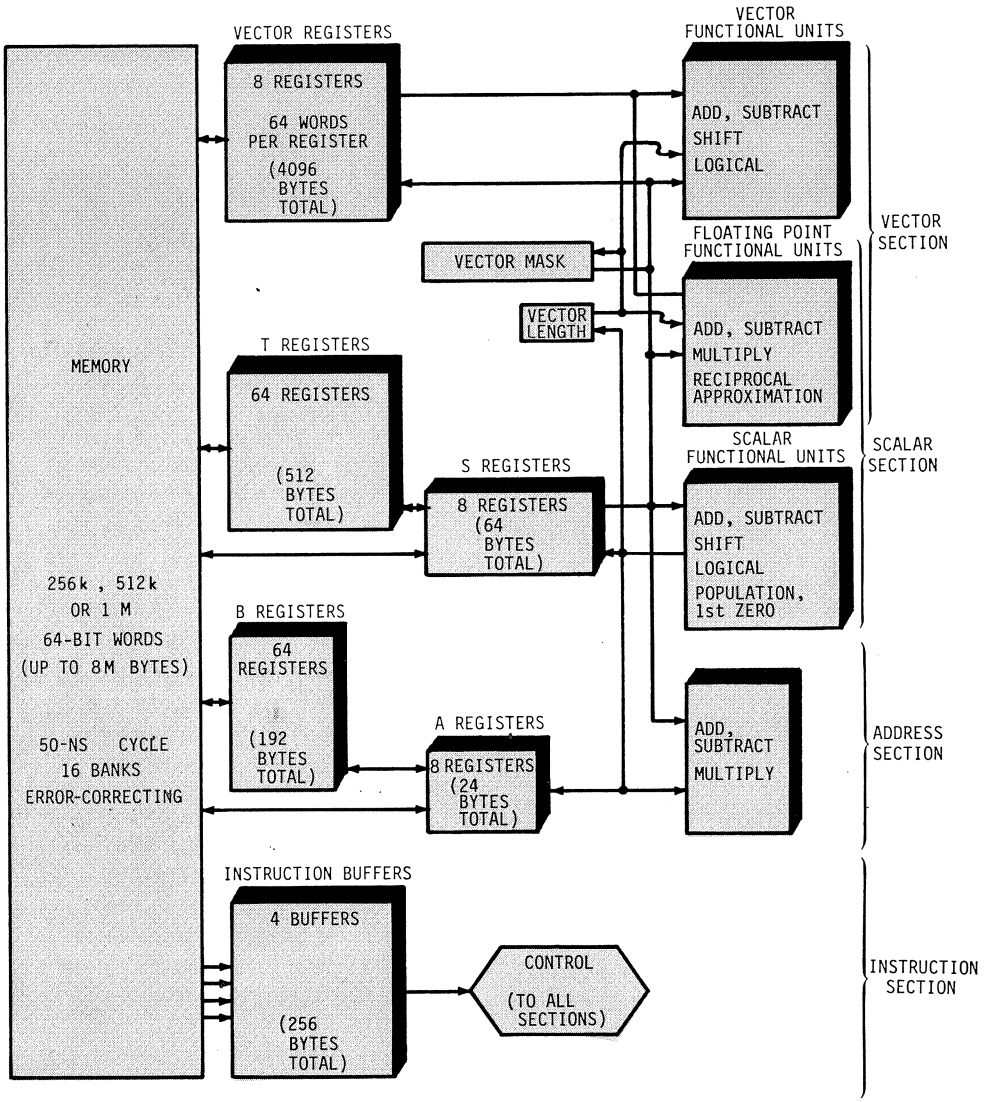


Fig 1 Register block diagram. Primary operating registers are scalar and vector registers, called S and V registers, respectively. Eight A registers are used as address registers for memory references and as index registers. A and S registers are each supported by 64 rapid-access temporary storage registers, called B and T registers, respectively. All registers can receive data from or send data to 1M-word bipolar memory. Instructions are executed from four instruction buffers. Arithmetic, logical, and shift operations are performed by 12 specialized functional units

Eight 24-bit A registers are used as address registers for memory references and as index registers. A and S registers are each supported by 64 rapid-access temporary storage registers, called B and T registers, respectively. All registers can receive data from or send data to memory.

Memory

Memory is constructed of bipolar 1024-bit large-scale integrated (LSI) chips. Up to 1 million 64-bit words are arranged in 16 banks with a bank cycle time of four clock periods. The short cycle time provides an extremely efficient random-access memory. Circuitry is provided for correction of all single-bit errors and detection of all double-bit errors.

Instruction Buffers

All instructions, which are 16- or 32-bits long, are executed from four instruction buffers, each consisting of sixty-four 16-bit registers. Since the four instruction buffers are large, substantial program segments may be stored. Forward and backward branching within the buffers is possible and program segments may be discontiguous. When the current instruction does not reside in a buffer, one instruction buffer is filled from memory. Four memory words are read per clock period to the least recently filled instruction buffer. To allow the current instruction to issue as soon as possible, the memory word containing the current instruction is the first to be read.

Input/Output

Any number of the 12 input and 12 output channels may be active at a given time. Each channel has a maximum transfer rate of 80M bytes/s. At most, one 64-bit word can be transferred to or from memory during each clock period; this is attained when four input channels and four output channels are operating simultaneously at their maximum rates. In practice, this theoretical transfer rate is limited by the speed of peripheral devices and by memory reference activity of the central processing unit (CPU).

Functional Units

Twelve specialized functional units in the CPU handle the arithmetic, logical, and shift operations. Each unit is independent of the others, and any number of functional units may be in operation at the same time. A functional unit receives operands from registers and delivers the result to a register when the function has been performed. These units operate essentially in a 3-address mode, with source and destination addressing limited to certain registers.

Three functional units, integer add, integer multiply, and population count, provide 24-bit results to A registers only. Integer add, shift, and logical units provide 64-bit results to S registers only. Sixty-four-bit results are provided to V registers only by integer add, shift, and logical units. Floating add, floating multiply, and reciprocal approximation units provide 64-bit results to either S or V registers.

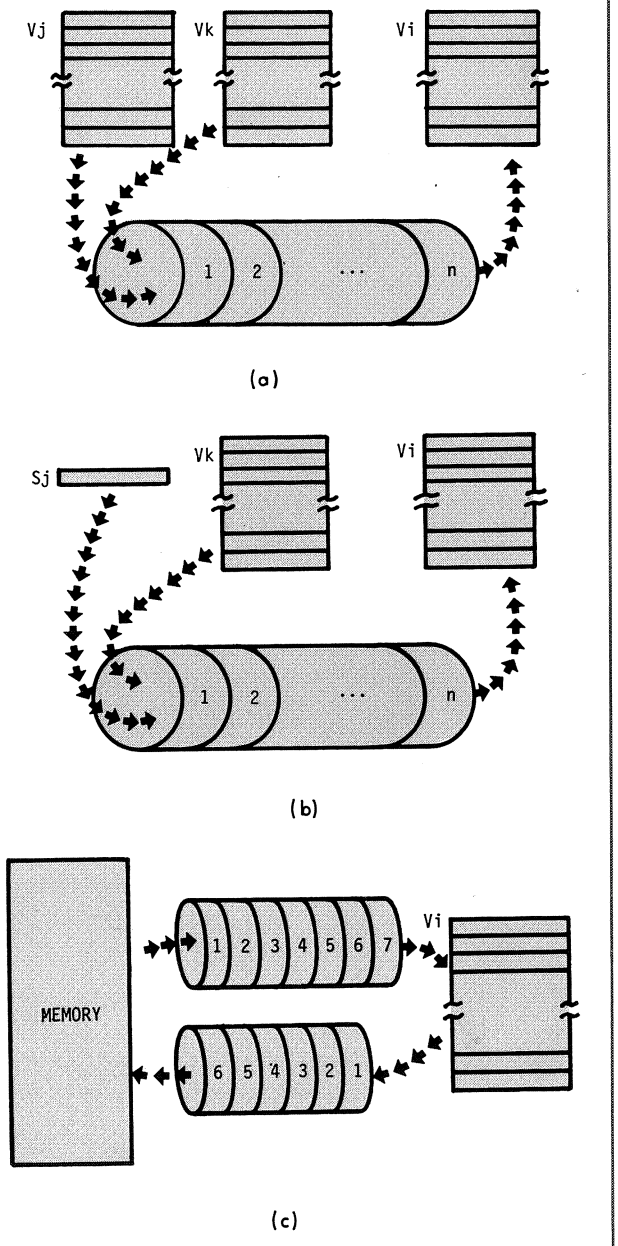


Fig 2 Vector instructions. Type I (a) instructions take operands from one or two vector registers. Type II (b) instructions take one operand, a constant, from scalar register, and second operand from vector register. Type III and IV instructions (c) read operands from memory or write operands to memory. Operands need not be contiguous and increment can be either positive or negative

All functional units are fully segmented. This means that information arriving at the unit or moving within it is captured and held in a new set of registers at the end of each clock period. Therefore, it is possible to start a new set of operands for unrelated computation into a functional unit each clock period even though the unit may require more than one clock period to complete the calculation. All functional units perform their

algorithms in a fixed amount of time. No delays are possible once operands have been delivered to the unit. Functional units servicing vector instructions produce one result per clock period.

Vector Instructions

Instructions that operate on vectors may be classified into four types. The first type of vector instruction obtains operands from one or two V registers and enters results into another V register [Fig 2(a)]. Successive operand pairs are transmitted from Vj and Vk to the segmented functional unit each clock period and corresponding results emerge from the functional unit n clock periods later. n is constant for a given functional unit and is called the functional unit time. Results are entered into result register Vi. Contents of the vector length (VL) register determine the number of operand pairs processed by the functional unit. A type II vector instruction obtains one operand from an S register and one from a V register [Fig 2(b)]. A copy of the S register is transmitted to the functional unit with each V-register operand. The last two types of vector instructions transmit data between memory and the V registers [Fig 2(c)]. A path between memory and the V registers may be considered a functional unit for timing considerations.

It is important to understand functional unit segmentation, especially as it relates to execution of vector instructions. Let a particular element of a V register be specified by adding the element number as a subscript to the register name. For example, the first three elements of register V1 are $V1_0$, $V1_1$, and $V1_2$, respectively. Since a vector register has 64 elements, the last element of V1 is $V1_{63}$.

Fig 3 shows a timing diagram for execution of a floating point addition instruction. This instruction is type I, since operands are obtained from two vector registers. When the instruction issues at clock period t_0, the first pair of elements ($V1_0$ and $V2_0$) is transmitted to the add functional unit where it arrives at clock period t_1. Dashed lines indicate transmit to and from the functional unit. Functional unit time for this unit is six clock periods; therefore, the first result, which is the sum of $V1_0$ and $V2_0$, exits from the functional unit at clock period t_7. The sum is transmitted to the first element of the result register V0, arriving at clock period t_8. Because the functional unit is fully segmented, the second pair of elements ($V1_1$ and $V2_1$) is transmitted to the add functional unit at clock period t_1. At clock period t_2 the functional unit is in the process of performing two additions simultaneously since addition of $V1_0$ and $V2_0$ was begun in the previous clock period. The second result, which is the sum of $V1_1$ and $V2_1$, is entered into the second element of result register V0 at clock period t_9. Continuing in this manner, a new pair of elements enters the functional unit each clock period and the corresponding sum emerges from the unit six clock periods later and is transmitted to the result register. Since a new addition is begun each clock period, six additions may be in progress at one time. In general, the number of operations that can be performed simultaneously by a functional unit is equal to the functional unit time.

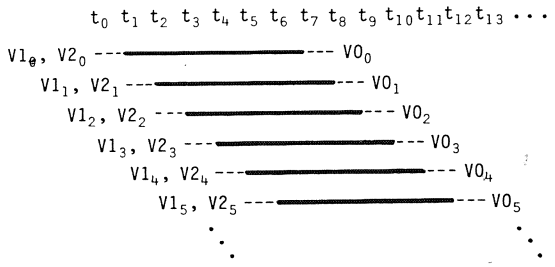


Fig 3 Vector instruction timing example (V0←V1+V2). New pair of operands is sent to functional unit each clock period. After first result emerges from functional unit, new result is sent to result register each clock period

Vector length determines the total number of operations performed by a functional unit. Although each vector register has 64 elements, only the number of elements specified by the vector length register is processed by a vector instruction. Vectors that have more than 64 elements are processed under program control in groups of 64 (with a possible residue). A later section on vector loops will illustrate the processing of long vectors.

Functional Unit and Operand Register Reservations

When a vector instruction issues, the required functional unit and operand registers are reserved for the number of clock periods determined by the vector length. A subsequent vector instruction that requires the same functional unit or operand register cannot be issued until the reservations are released. When two vector instructions use different functional units and vector registers, they are independent and may issue in consecutive clock periods. Some examples follow. Ex (1) shows two independent instructions. Both execute concurrently with a one clock period difference in their issue times. Ex (2) through (4) illustrate the effect of functional unit and operand register reservation when two instructions are not independent. Ex (2) shows two add instructions. When the first instruction issues, the floating add functional unit and operand registers V1 and V2 are reserved. Issue of a second add instruction is delayed until the functional unit is free. Ex (3) shows an add instruction followed by a multiply instruction. As in Ex (2), the floating add functional unit and operand registers V1 and V2 are reserved when the first instruction issues. Issue of the second instruction is delayed until operand register V1 is free. The second instruc-

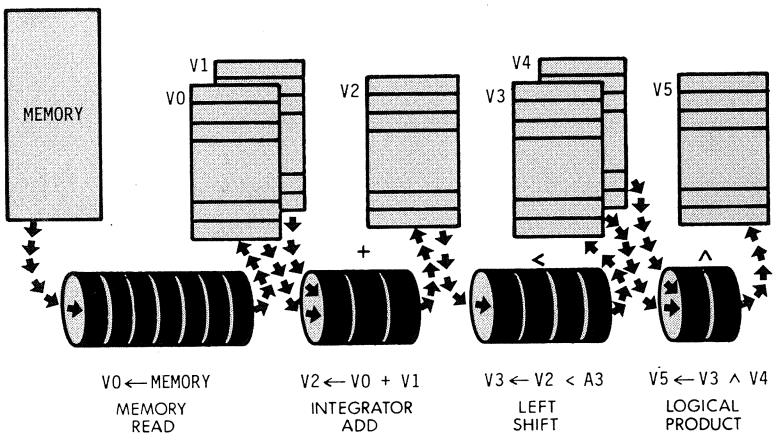


Fig 4 Chaining example. Chain of four instructions reads vector of integers from memory, adds that vector to another, shifts sum, and finally forms logical product of shifted sum and mask vector. Diagram illustrates how functional units may be considered links in chain which works as a whole to produce first result

tion in Ex (4) is delayed because of both functional unit and operand register reservations.

(1) Independent Instructions
V0 ← V1 + V2
V3 ← V4 * V5

(2) Functional Unit Reservation
V3 ← V1 + V2
V6 ← V4 + V5

(3) Operand Register Reservation
V3 ← V1 + V2
V6 ← V1 * V5

(4) Functional Unit and Operand Register Reservation
V0 ← V1 + V2
V3 ← V1 + V5

Result Register Reservations and Chaining

When a vector instruction issues, the result register is reserved for the number of clock periods determined by the vector length and functional unit time. This reservation allows the final operand pair to be processed by the functional unit and the corresponding result to be transmitted to the result register.

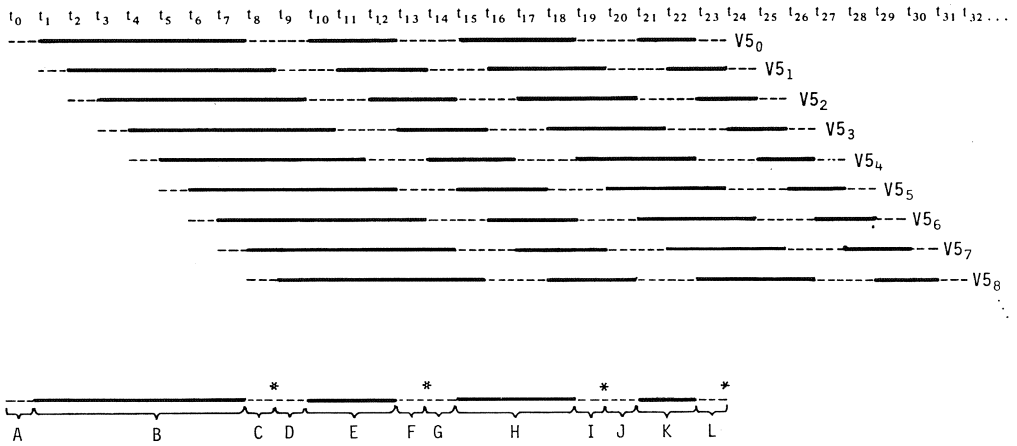
A result register becomes the operand register of a succeeding instruction. In the process called "chaining," the succeeding instruction issues as soon as the first result arrives for use as an operand. This clock period is termed "chain slot time"; it occurs only once for each vector instruction. If the succeeding instruction cannot issue at chain slot time because of a prior functional unit or operand register reservation, it must wait until the result register reservation is released.

Fig 4 shows a chain of four instructions which read a vector of integers from memory, add that vector to another, shift the sum, and finally form the logical product of the shifted sum and a mask vector. The result of the four instructions is in vector register V5. The diagram depicts passage of information through functional units, and illustrates the idea that functional units may be considered links in a chain which works as a whole to produce the final result.

The timing diagram in Fig 5 clarifies the concept of chaining. Graduations along the horizontal axis represent clock periods. The memory read instruction issues at clock period t_0. Each horizontal line shows the production of one element of the V5 result vector. Time spent in passing through each of the four functional units used in the instruction sequence (see Fig 4) is indicated by bars of corresponding length in the timing diagram (Fig 5). Note that the production of a new element of V5 begins each clock period. Production of the first element of V5 begins at clock period t_0 with the reading of the first word from memory, production of the second element of V5 begins at clock period t_1 with the reading of the second word from memory, and so on. The first result enters V5 at clock period t_{24} and a new result enters V5 each clock period thereafter. The first horizontal line, which shows production of the first element of V5 ($V5_0$), is reproduced below the timing diagram with segments lettered for identification. Chain slot times for each functional unit are indicated by asterisks.

A detailed description of the production of $V5_0$ serves for illustration; production of other elements of the result vector is identical except for the staggered start times.

The vector read instruction issues at clock period t_0. The first word arrives in element 0 of register V0 at



A - TRANSIT OF MEMORY WORD TO "READ FUNCTIONAL UNIT"

B - TRANSIT OF MEMORY WORD THROUGH "READ FUNCTIONAL UNIT"

C - TRANSIT OF MEMORY WORD FROM "READ FUNCTIONAL UNIT" TO ELEMENT OF V0

D - TRANSIT OF OPERAND ELEMENTS IN V0 AND V1 TO INTEGER ADD FUNCTIONAL UNIT

E - COMPUTATION OF SUM BY INTEGER ADD FUNCTIONAL UNIT

F - TRANSIT OF SUM FROM INTEGER ADD FUNCTIONAL UNIT TO ELEMENT OF V2

G - TRANSIT OF OPERAND ELEMENT IN V2 TO SHIFT FUNCTIONAL UNIT

H - SHIFT OPERATION PERFORMED BY SHIFT FUNCTIONAL UNIT

I - TRANSIT OF SHIFTED SUM FROM SHIFT FUNCTIONAL UNIT TO ELEMENT OF V3

J - TRANSIT OF OPERAND ELEMENTS IN V3 AND V4 TO LOGICAL FUNCTIONAL UNIT

K - LOGICAL OPERATION PERFORMED BY LOGICAL FUNCTIONAL UNIT

L - TRANSIT OF FINAL RESULT TO ELEMENT OF V5

Fig 5 Timing diagram for chaining. Each horizontal line shows production of one element of V5 vector result. Time spent in passing through each function unit used in instruction sequence is indicated by bar of corresponding length. Chain slot times for each functional unit are indicated by asterisks

clock period t_9, and is immediately transmitted along with element 0 of register V_1 as an operand to the integer add functional unit. When the two operands arrive at the integer add functional unit at clock period t_{10}, computation of the sum of $V0_0$ and $V1_1$ is begun. Three clock periods later (t_{13}) the sum is sent from the functional unit to element 0 of V2. It arrives at clock period t_{14} and is immediately transmitted as an operand to the shift functional unit. At clock period t_{15} the operand arrives at the shift functional unit and the shift operation is begun. The operation is completed four clock periods later (t_{13}) and the shifted sum is sent from the functional unit to element 0 of V3, arriving at the next clock period. It is immediately transmitted, along with element 0 of V4, as an operand to the logical functional unit. When the two operands arrive at the logical functional unit at clock period t_{21}, computation of the logical product of $V3_0$ and $V4_0$ is begun. Two clock periods later (t_{23}) the final result is sent from the functional unit to element 0 of V5, arriving at clock period t_{24}. While all this has been going on, production of the second element of V5 has been tracing the same path through the vector registers and functional units with a one clock period lag. Production of the third element of V5 lags one more clock period behind, and so on. A new result arrives at the V5 result register each clock period.

Vector Loops

Long vectors are processed in segments since the vector registers of the computer cannot accommodate vectors with more than 64 elements. The program construct created to process long vectors is called a vector loop. Each pass through the loop processes a 64-element (or smaller) segment of the long vectors. The general procedure is to compute the loop count based on the vector length before entering the loop. Inside the loop the program takes full advantage of the 12 independent functional units and chaining to read current vector segments from memory, execute required functions, and return results to memory. Loop control is performed in the scalar registers concurrently with vector processing. Loop branch time is hidden by vector operations.

Processing of long vectors is illustrated by the following simple FORTRAN loop.

```
    DO 100 I=1,N
100 A(I) = 5. * B(I) + C
```

The loop computes the Ith element of A by adding C to five times the Ith element of B, where I ranges from 1 to N. When N is 64 or less, all elements of the A array can be assigned a value with the following sequence of seven instructions.

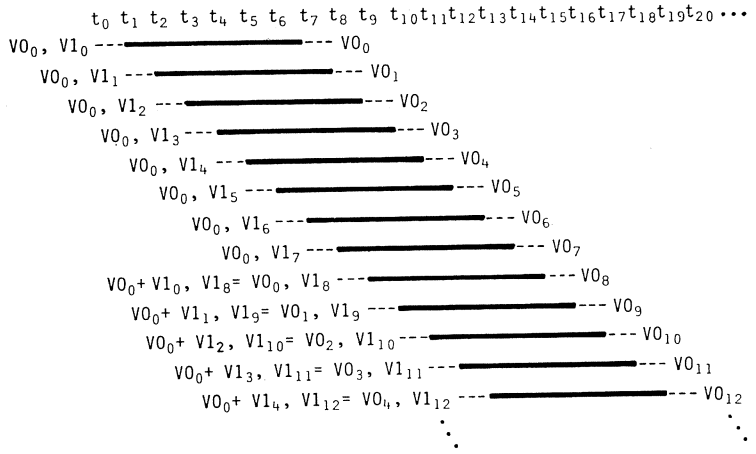


Fig 6 Timing diagram for recursive vector sum. Early results are re-sent to functional unit as operands. At instruction completion, dot product's 64 partial sums will have been reduced to eight, which will be held in elements $V0_{56}$ through $V0_{63}$

(1)	S1 ← 5.	Set constant 5	
(2)	S2 ← C	Set constant C	
(3)	VL ← N	Set vector length	
(4)	V0 ← B array	Read B array	chain
(5)	V1 ← S1*V0	Multiply elements by 5	chain
(6)	V2 ← S2+V1	Add C	chain
(7)	A array ← V2	Store A array	

Instructions (4) to (6) use different functional units ("memory," multiply, and add, respectively); therefore, they can be chained. When the V2 result register is free, results are stored in the A array.

When N exceeds 64, a vector loop is required to generate the entire A array. Before entering the loop, N is divided by 64 to determine the loop count. If there is a remainder, less than 64 elements of A are generated in one pass through the loop. The loop performs instructions (4) to (7) for a segment of the A and B arrays. The last vector operation at the bottom of the loop stores the current segment of the A array in memory. This operation must be completed before the next segment of the B array can be read in the next pass through the vector loop. The time required to decrement the loop counter, increment the current position in the arrays, and branch to the top of the loop is hidden because it is done in parallel with the store operation.

Dot Product and Recursive Vector Operations

The dot product of two vectors, $A = (a_0,a_1,...,a_N)$ and $B = (b_0,b_1,...,b_N)$ is defined by

$$A \cdot B = \sum_n a_n \cdot b_n$$

Computation of the dot product is achieved by a vector loop and a scalar loop. The vector loop, which contains a multiply-add chain, first computes 64 partial sums; element n of the result vector register contains the sum of every sixty-fourth product, $a_i \cdot b_i$, starting with $a_n \cdot b_n$. The scalar loop then adds partial sums to compute the complete dot product. An intermediate step that reduces the number of partial sums from 64 to eight may be interposed between the vector and scalar loops. This step executes a vector add instruction that has the register of partial sums as both operand and result register.

To see how this is done, observe that there is an element counter associated with each vector register. When a vector instruction issues, counters for its operand and result registers are zeroed. Normally, sending an operand from an operand register to a functional unit causes the counter associated with that register to be incremented; the counter always points to the next available operand. Similarly, a result arriving at the result register from a functional unit causes the counter

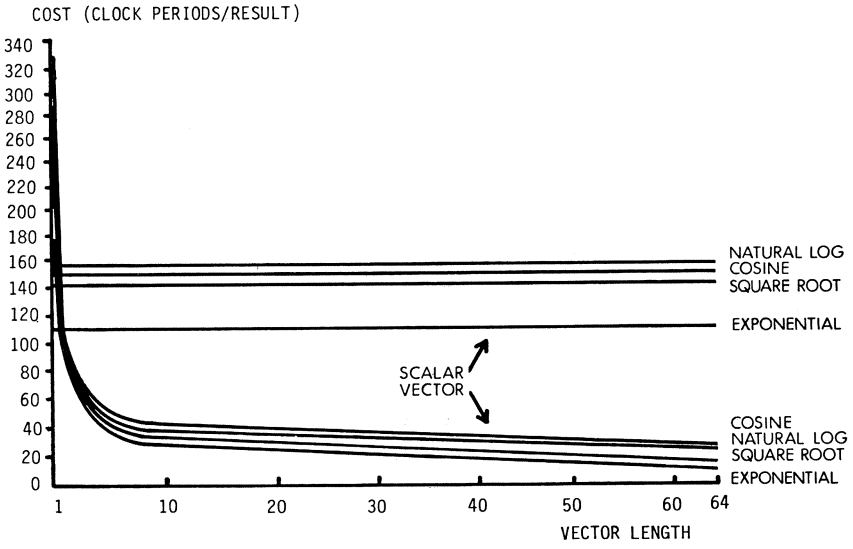


Fig 7 Scalar/vector timing comparison. Cost of result (in clock periods) is expressed graphically as function of vector length. Cost is constant for scalar subroutines since they must be called for each desired result; however, for vector subroutines, cost drops dramatically and rapidly approaches lower limit as vector length increases. Vector subroutine is always cheaper if more than one result is produced

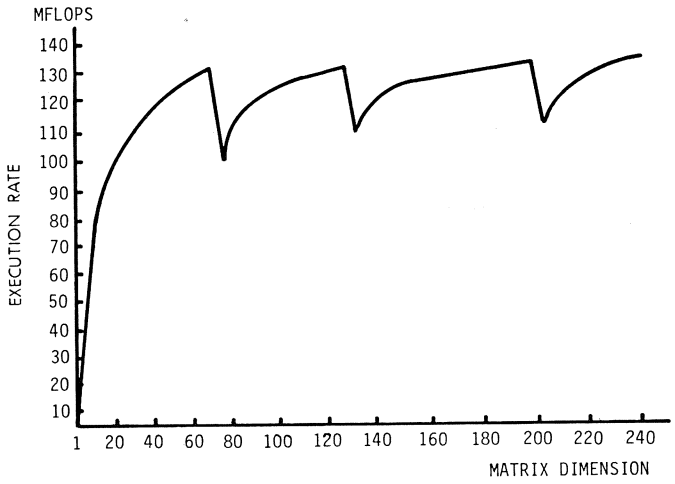


Fig 8 Matrix multiplication timing. Execution rate, expressed as millions of floating point operations per second, is plotted as function of vector length. Typical of problems that can be vectorized, matrix multiplication can be performed with significant increase in speed over conventional scalar processing

associated with that register to be incremented. However, when a register serves as both operand and result register, its counter does not begin advancing until the first result arrives from the functional unit. While the counter is held at 0, the contents of element 0 are repeatedly sent to the functional unit. Consider what happens if element 0 of register V0 is cleared and then V0 is used as both operand and result register for an add instruction. The register of partial sums, eg, V1, is used as the other operand register.

Refer to the timing diagram in Fig 6. When the add instruction issues at clock period t_0, operands from elements $V0_0$ and $V1_0$ are sent to the add functional unit. One clock period later, when the first addition begins, elements $V0_0$ and $V1_1$ are sent to the add functional unit. Note that, because V0 is both operand and result register, the element counter associated with V0 does not increment. Thus, the element of V0 that is sent at clock period t_0, $V0_0=0$, is sent again at clock period t_1. This holds true for six more clock periods. The counter for

V1 advances, but the counter for V0 remains at 0. Finally, at clock period t_8, the first sum arrives at element 0 of the result register, V0. The element counter for V0 now begins to advance once each clock period. At clock period t_8, elements $V0_0$ and $V1_8$ are sent to the add functional unit. Note that $V0_0$ is the sum of the original $V0_0$ and $V1_0$. Thus, since $V0_0$ was initially 0, summation of $V1_0$ and $V1_8$ is beginning. At clock period t_9, elements $V0_1$ and $V1_9$ are sent to the functional unit; since $V0_1$ is the sum of the original $V0_0$ and $V1_1$, summation of V_1 and $V1_9$ is beginning. The recursive character of the instruction should be becoming clear; as results are produced, they are re-sent to the functional unit as operands. At instruction completion, elements $V0_{56}$ through $V0_{63}$ contain the dot product's eight partial sums, reduced in number from the original 64. In this example, a vector length of 64 was assumed; however, the same technique is applicable for vectors of shorter length.

FORTRAN Library

The following performance study of matrix multiplication and several subroutines from the CRAY-1 FORTRAN library illustrates the high processing rates attainable through vector processing. Each scalar FORTRAN library subroutine has a vector analog which employs the same algorithm in vector mode to produce several results at a time. Scalar subroutines must be called for each desired result, while vector subroutines process an argument vector to obtain a vector of results. Performance studies on the CRAY-1 indicate that a vector subroutine outperforms its scalar counterpart whenever a vector of two or more results is required. Fig 7 depicts the behavior of the scalar and vector subroutines for several library functions. Cost of a result (in clock periods) is plotted as a function of vector length. Cost is constant for scalar subroutines since they must be called for each desired result; however, for vector subroutines, the cost drops dramatically and rapidly approaches a lower limit as vector length increases. In all cases vector cost is less than scalar cost when more than one result is produced.

Matrix Multiplication

Let [X] denote a matrix and let the element in row i, column j be denoted by x_{ij}. Given matrix [A] of dimension K by N and matrix [B] of dimension N by M, the product matrix [C] = [A] • [B] is defined by

$$c_{ij} = \sum_{n=1}^{N} a_{in} \cdot b_{nj}$$

Calculation of the product matrix is amenable to vector processing. The combination of multiplication and addition lends itself well to chaining. Fig 8 shows the computer's execution rate for multiplication of square matrices as a function of matrix dimension. Execution rate is defined in terms of "millions of floating point operations per second" (MFLOPS). This measure is more meaningful than the classical "millions of instructions per second" (MIPS), especially when comparing relative speeds of scalar and vector machines; a single vector instruction is equivalent to a loop of several scalar instructions. The number of floating point operations required to multiply two n-dimensional square matrices is $n^2(2n - 1)$, since each of the n^2 elements of the result matrix is formed by summing n products.

Matrix multiplication is typical of the large class of problems that can be vectorized. For these problems a significant increase in processing speed can be achieved over conventional scalar processing. Register to register vector instructions and the large amount of concurrency attainable through use of the 12 independent functional units and chaining provide high processing speeds presently unmatched. Fields such as weather forecasting, nuclear research, and seismic data analysis provide typical applications.

Summary

Capabilities of the CRAY-1 that contribute to its high processing rates for both scalar and vector applications include its large, fast random-access memory, instruction buffers, high bandwidth input/output channels, and full segmentation of the 12 independent functional units. Chaining techniques and the use of register to register vector instructions help eliminate the problem of speed degradation associated with memory to memory vector instructions. Additionally, start-up times for vector operations are nominal and the advantages of vector processing can be realized even for short vectors. Thus, the computer system's architecture meets its design goal for efficient execution of program loops by using vector processing, yet does not sacrifice scalar performance.

Bibliography

Cray Research, Inc, *CRAY-1 Computer System Hardware Reference Manual*, Minneapolis, Minn, Nov 1977

P. H. Enslow, Jr, "Multiprocessor Organization—A Survey," *ACM Computing Surveys*, Mar 1977, pp 103-129

P. M. Johnson, "An Introduction to Vector Processing," Cray Research, Inc, Minneapolis, Minn, 1975

D. J. Kuck, "A Survey of Parallel Machine Organization and Programming," *ACM Computing Surveys*, Mar 1977, pp 29-60

C. V. Ramamoorthy and H. F. Li, "Pipeline Architecture," *ACM Computing Surveys*, Mar 1977, pp 61-102

R. M. Russell, "The CRAY-1 Computer System," *Communications of the ACM*, Jan 1978, pp 63-72

Paul M. Johnson, senior systems analyst at Cray Research, has been involved in software development for several large scale scientific computers. He holds a BA degree in mathematics and German from Augustana College (I11) and an MS degree in computer science from the University of Minnesota.

CRAY RESEARCH, INC.

TECHNICAL COMMUNICATIONS
7850 Metro Parkway, Suite 213, Minneapolis, MN 55420 • (612) 854-7472

PUBLICATION CHANGE NOTICE

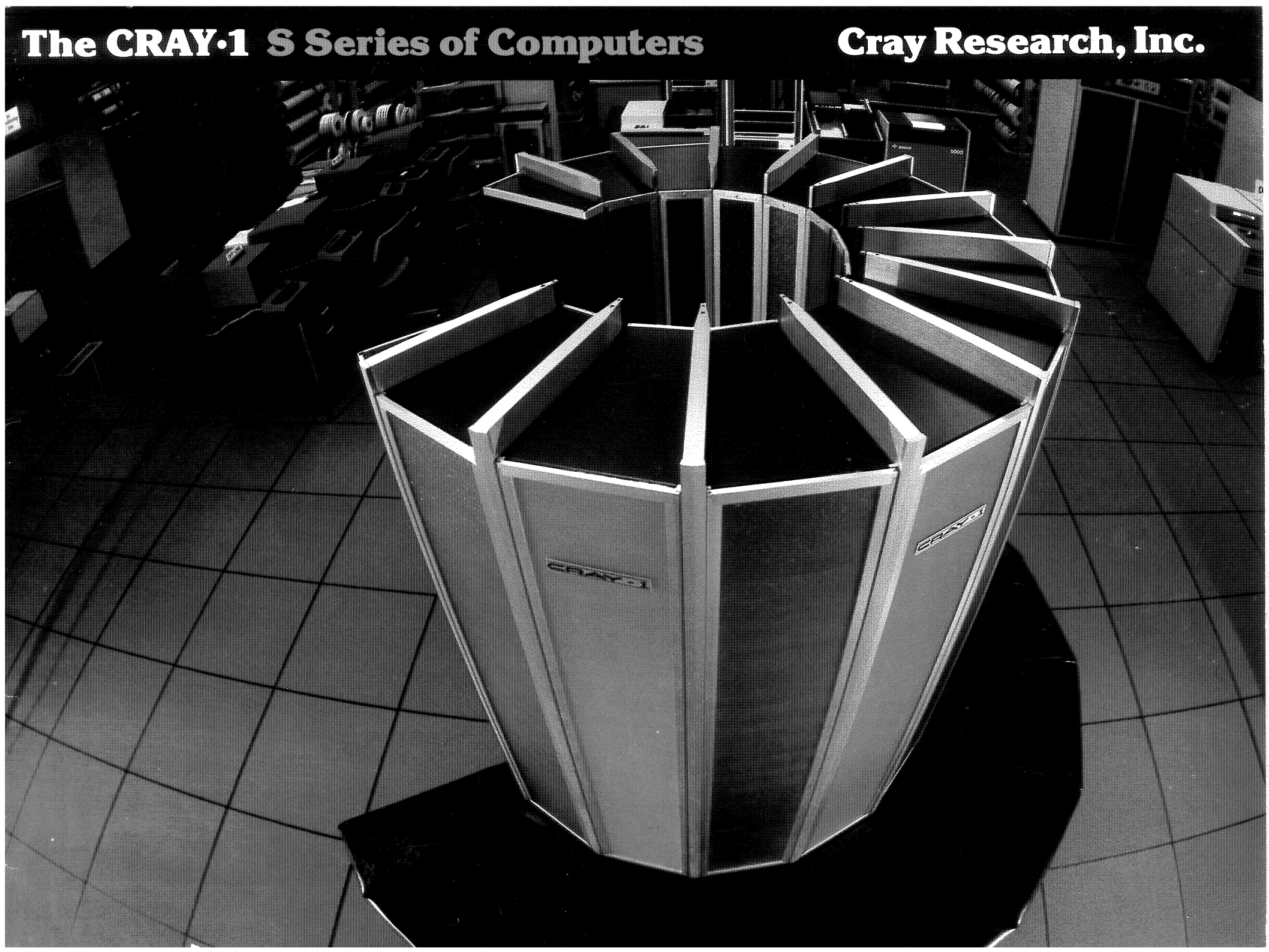
TITLE: CRAY-1 Reference Manual

PUBLICATION NO. 2240004 REV. B

Revision B applies to CRAY-1 computer systems starting with Serial No. 3. Revision A remains relevant for Serials 1 and 2.

The CRAY-1 S Series of Computers

Cray Research, Inc.



Solving Tomorrow's Problems Today

Weather forecasting and climatology... petroleum research... structural analysis... nuclear research... geophysics and seismic analysis... fluid dynamics... defense... medical research...

Until the advent of the CRAY-1 Computer Systems, solutions to problems in these and many other applications were not possible. The delivery of the first CRAY-1 in 1976 marked a turning point in computing power available. Now, because the CRAY-1 allows greater quantities of data to be processed and derives results more quickly, solutions are not only possible but economically practical as well.

Since its founding in 1972, Cray Research has dedicated itself to the design, development, and marketing of large-scale computers as tools for solving the complex problems facing the scientific, engineering, and technical communities. Science and technology are fields with nearly endless requirements for computing power. Their applications typically call for complex calculations to be performed on large quantities of data. Efficient solution of these sophisticated problems demands very high speed computations and extensive memory. The CRAY-1 Computer Systems are meeting these challenges, solving tomorrow's problems today.

Some Large-Scale Applications for a CRAY-1

- ☐ Weather forecasting and climatology
- ☐ Petroleum research
- ☐ Nuclear research
- ☐ Fluid dynamics
- ☐ National defense
- ☐ Geophysics and seismic analysis
- ☐ Structural analysis
- ☐ Medical research
- ☐ Electrical power distribution
- ☐ Graphics
- ☐ Automotive engineering
- ☐ Aerospace design
- ☐ Chemical engineering
- ☐ Particle physics
- ☐ Astronomy
- ☐ Economic analysis

Photo Credits (Clockwise from upper left): Medical Research, Gary Bistram; Petroleum Research, Webb Photos; Astronomy, Tim Larsen; Chemical Engineering, Rob Sheppard—Webb Photo; Weather Forecasting and Climatology, Garry McMichael—Webb Photo; Structural Analysis, Evans & Sutherland; Nuclear Research, David Frazier—Webb Photo; Aerospace Design, Erik Simonsen—Webb Photo.

Cover Photo: Courtesy Fairchild Camera and Instrument Co.

I/O Subsystem Buffer Memory

Buffer Memory is a separate independent storage unit accessible to all of the I/O Processors in the I/O Subsystem. It is a solid state device composed of NMOS (Negative Channel Metal Oxide Semiconductor) integrated circuits having a capacity of 0.5M to 1.0M 64-bit words.

The I/O Processors connect to the Buffer Memory through 850 megabit ports. For a 0.5M word memory, a maximum bandwidth of 1280 megabits per second is possible; for a 1M word MOS memory, a maximum bandwidth of 2560 megabits is possible.

Buffer Memory Characteristics

Data Storage

- ☐ 64-bit words
- ☐ 524,288 to 1,048,576 words
- ☐ 8 banks (524,288 words) or 16 banks (1,048,576 words)
- ☐ NMOS 16K bit integrated circuits
- ☐ 2 millisecond refresh rate
- ☐ 200 nanosecond access time
- ☐ 375 nanosecond cycle time

Error Correction

- ☐ SECDED within memory
- ☐ Error data available to separate error reporting channel

Interface

- ☐ 16-bit parcels
- ☐ Transfer rate:

  1280 Mbits/sec for 0.5M memory
  2560 Mbits/sec for 1 M memory
- ☐ Block sizes up to 16K 64-bit words
- ☐ Busy and done control signals

I/O Subsystem Mass Storage

The Buffer I/O Processor (BIOP) and at most two additional Disk I/O Processors (DIOP) can be dedicated to mass storage data transfers. The BIOP differs from a DIOP because it connects to the CRAY-1 CPU via a Memory Channel. Each BIOP or DIOP can have up to 4 DCU-4 Disk Control Units.

Each DCU-4 Controller supports up to 4 DD-29 Disk Storage Units. All units connected to a DCU-4 may be active simultaneously and can be directly accessible to the CPU rather than going through CPU memory. However, the number of concurrent data streams is limited by the Buffer Memory size, the BIOP transfer capacity, and software overhead. For example, on a Model S/x200, this limit might be 6 streams while on a larger system, it could be as many as 12 streams.

The DCU-4 Controller used on the I/O Subsystem is a combination of hardware and controlware.

The DCU-4 linkage to the I/O Subsystem is standard for Models 1200 and above.

DD-29 Disk Storage Unit Characteristics

Byte capacity	606×10^6
Tracks per surface	823
Sectors per track	18
Bytes per sector	4096
Data transfer rate (bytes per second)	4.4×10^6
Disk cylinder capacity	0.74×10^6
Access time	
Maximum	80 ms
Adjacent	15 ms
Latency	16.6 ms
Recording surfaces	40 per drive
Number of head groups	10

Mass Storage Subsystem

A Mass Storage Subsystem is composed of DCU-3 Disk Controllers and DD-29 Disk Storage Units.

The DCU-3 Disk Controllers connect to the CRAY-1 CPU through I/O Channels. Each controller requires 1 I/O channel and may control up to 4 disk storage units. The DCU-3 Controller is a Cray Research product implemented in ECL logic similar to that used in the CPU. The Controller is double-buffered to allow streaming of data to a DD-29 Disk Storage unit at full hardware rates.

A Mass Storage Subsystem composed of 2 to 8 DCU-3 Disk Controllers and 2 to 32 DD-29 Disk Storage Units is standard on the Models S/250, S/500, and S/1000. However, the actual maximum number of controllers and disk storage units depends on the number of available channels; e.g., if 4 channels are used for front-end computer systems (including one channel for the MCU), the remaining 8 can be used for a Mass Storage Subsystem that could support up to 32 DD-29 Disk Storage Units.

Front-end Interfaces

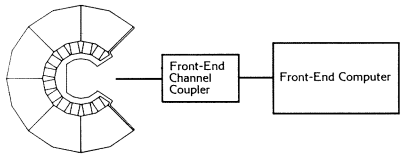
Each front-end computer system connected to the CRAY-1 executes under control of its own operating system in a mode asynchronous to the CRAY-1. The CRAY-1 is interfaced to front-end systems through special adapters that compensate for differences in channel widths, machine word size, electrical logic levels, and control protocols. These interfaces are Cray Research products implemented in logic compatible with the host system.

Cray Research provides external interfaces to a variety of other manufacturers' equipment. Cray Research is willing to work with customers to develop hardware interfaces for other computers.

The adjoining figures illustrate how front-end computer systems link directly to the CRAY-1 System I/O Channels or through the I/O Subsystem. The normal I/O Channel linkage is standard for Models S/250, S/500, and S/1000. The I/O Subsystem linkage is the standard mode for all other models.

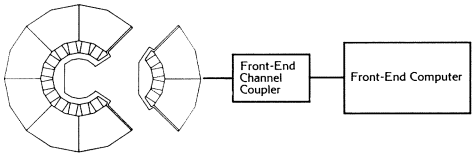
Direct Front-End Link

CRAY-1 Models S/250, S/500 and S/1000



I/O Subsystem Front-End Link

CRAY-1 Models S/1200 through S/4400

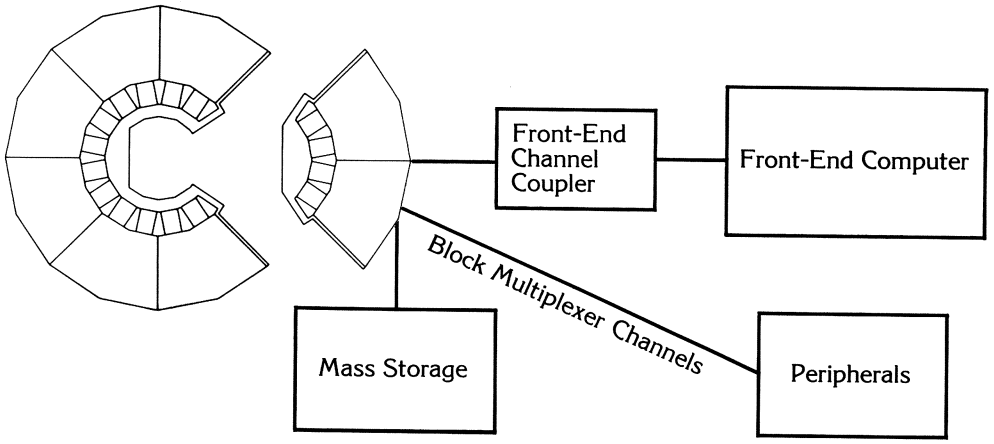


Operator Functions

Operator control of the CRAY-1 is through any front-end computer system designated as the system operator station. The station operator has available a versatile set of commands for controlling the flow of jobs and data files. A portion of the I/O Subsystem (i.e., the CRTs and the maintenance peripherals) can also be designated as the system operator station. The Maintenance Control Unit (MCU) provides similar capabilities for Models S/250, S/500, and S/1000. System startup is initiated at the I/O Subsystem (where present) or at the MCU (where present).

Maintenance Functions

An extensive set of diagnostic programs is available to field engineers to aid in quickly identifying problem areas in the hardware in event of a failure. Where the I/O Subsystem is present, these diagnostics are accessed via the operator consoles and the maintenance peripherals attached to the I/O Subsystem. Where the MCU is an integral part of the system, it serves maintenance functions.



up to 850 megabits per second. The 2 optional I/O Processors can be 2 Disk I/O Processors (DIOP) for driving additional disk storage units or a DIOP and a Block Multiplexer I/O Processor (XIOP) for controlling other devices. The BIOP and each DIOP contains 1 to 4 DCU-4 Disk Control Units, each of which controls up to 4 DD-29 Disk Storage Units. Similarly, an XIOP contains 1 to 4 Block Multiplexer Controllers, each of which contains up to 4 Block Multiplexer Channels.

The I/O Processors are all connected to each other and to Buffer Memory.

Software for the I/O Subsystem interfaces smoothly with the Operating System on the CPU. User interfaces are fully compatible for all models in the S Series.

The operating subsystem (Kernel) resident in each I/O Processor:

Handles interrupts,

Controls the disk units,

Supports station and front-end activities,

Dispatches messages to and from the CPU,

Handles interprocessor communications, and

Loads overlays.

The Kernel is the same for all I/O Processors and is modified by system parameters at time of installation.

I/O Processor

Each of the I/O Processors (2 minimum; 4 maximum) is a powerful minicomputer designed specifically for controlling and directing the flow of data at high rates from the CPU to peripheral devices and other computer systems. The 16-bit computation section is coupled with a fast bipolar local memory. The I/O Processors are ideally suited for mass storage access, network control, and computer interfacing.

Computation section —The computation section consists of registers and functional units having interconnecting data paths. Arithmetic is performed in a single adder in 2's complement mode. Floating-point arithmetic is not incorporated. The 128-instruction repertoire is purposely simple and includes a variety of branching and I/O functions. Shifts are either left or right and circular or end off. Several logical operations are available. Instructions occupy either 16 bits (1 parcel) or 32 bits (2 parcels).

Computation Section Characteristics

- ☐ 16-bit parcels
- ☐ Single address mode
- ☐ Addition/subtraction unit
- ☐ Shift unit
- ☐ 9-bit index register
- ☐ 512 operand registers
- ☐ 32-level instruction stack
- ☐ Subroutine return stack
- ☐ 128 operation codes

I/O section —Communication with an I/O Processor is through accumulator channels and through the 6 direct memory access (DMA) ports. Each port is full duplex and transfers a 16-bit data word each clock period. Ports are organized into three groups with a priority scheme within each group. One input port and one output port may be active at the same time as long as the ports belong to different groups. Thus, a DMA port can handle data at a theoretical rate of 850 megabits per second.

The ports are assigned to I/O Channels, possibly with several channels sharing one port. An I/O Processor includes an addressing capability for up to 512 separate channels; however, hardware restrictions impose a practical limit of 40. The slower the required data rate on the channels, the more channels that may be multiplexed onto one DMA port.

All I/O controlled by the I/O Subsystem originates with the CPU. Status is returned to the CPU to indicate request completion.

I/O Section Characteristics

- ☐ 6 full duplex DMA ports
- ☐ 850 megabits per second per DMA port
- ☐ 16 data bits
- ☐ Program selection of channel number
- ☐ Simultaneous input and output
- ☐ Interrupt driven under program control

Local memory —The local memory is composed of 65,536 16-bit parcels arranged in 4 sections of 4 banks each of bipolar LSI memory. There are 2 parity bits per parcel. All 16 memory sections are independent. Memory cycle time is 4 clock periods. The access time, that is, the time required to bring an operand from memory, is 7 clock periods.

A significant portion of an I/O Processor's local memory serves as I/O buffers.

Local Memory Characteristics

- ☐ 65,536 16-bit parcels
- ☐ 16 banks of 4,096 parcels each
- ☐ 4 clock period bank cycle time
- ☐ 7 clock period read time
- ☐ 1 instruction fetch per clock period
- ☐ Odd parity protection (2 parity bits per 16 bits)
- ☐ 3 data paths for reading; 2 data paths for writing

The 13 functional units can be thought of as forming four groups: address, scalar, vector, and floating-point. The first three groups act in conjunction with one of the three primary register types to support address, scalar, and vector modes of processing. The fourth group, floating-point, can support either scalar or vector operations and accepts operands from or delivers results to scalar or vector registers accordingly.

CRAY-1 CPU Functional Units

Unit	Register Usage	Time in Clock Periods
Address functional units		
Addition	A	2
Multiplication	A	6
Scalar functional units		
Addition	S	3
Shift		
Single	S	2
Double	S	3
Logical	S	1
Population, parity and leading zero	S	3
Vector functional units		
Addition	V	3
Shift	V	4
Logical	V	2
Population, parity	V	6
Floating-point functional units		
Addition	S and V	6
Multiplication	S and V	7
Reciprocal approx.	S and V	14

Interrupts and the Exchange Sequence

Instruction issue is terminated by the hardware upon detection of an interrupt condition. All memory bank and functional unit activity is allowed to complete. To switch execution in order to handle the interrupt, the CRAY-1 executes an exchange sequence. This causes program parameters for the next program to be exchanged with current information in the operating registers.

Each program in the system has associated with it a 16-word block called an exchange package containing the parameters used in its execution sequence. Only the address and scalar registers are maintained in a program's exchange package.

Exchange sequences may be initiated automatically upon occurrence of an interrupt condition or may be initiated voluntarily by the software.

Memory Characteristics

CRAY-1 Central Memory is large and fast with bipolar LSI chips as its basic elements. It is expandable depending on model type from 256K words to 4 million words. A word is composed of 64 data bits and 8 check bits.

Each memory word is associated with a unique address in memory. The bank cycle time of 50 nanoseconds (4 clock periods) enables 80 million words per second to be accessed in serially addressed blocks. Access time, the time required to fetch an operand from memory to an operation register, is 137.5 nanoseconds (11 clock periods).

Conflict detection and resolution enables simultaneous memory bank operations and prevents the loss of information when bank access conflicts occur. Because of the 4-clock-period memory cycle time, vector memory access can always proceed at one word each 4 clock periods; generally vector memory access is 1 word per clock period.

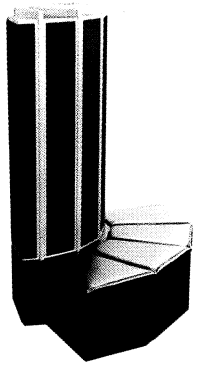
Memory Field Protection

Each object program is assigned a designated field of memory by the operating system. Field limits are defined by a base address register and a limit address register. Any attempt to reference instructions or data outside these limits results in a range error and an interrupt.

Memory field protection assures that no job can inadvertently modify another job in a multiprogramming environment.

CPU Memory Characteristics

Technology	Bipolar semiconductor
Word size	72 bits; 64 data 8 SECDED
Address space	4 million words
Cycle time	50 nsec
Size	0.25M to 4M words
Organization	8 or 16 banks interleaved
Error checking	Single error correction; double error detection



The I/O Subsystem

The CRAY-1 I/O Subsystem has been designed by Cray Research specifically to complement the CPU and to meet its high throughput demands.

The I/O Subsystem supports the following components:

- 2, 3, or 4 I/O Processors
- 0.5M or 1M 64-bit words of Buffer Memory
- 1 to 12 DCU-4 Disk Control Units
- 2 to 48 DD-29 Disk Storage Units
- 2 CRT consoles
- 1 to 4 Block Multiplexer Controllers
- 1 to 16 Block Multiplexer Channels
- One standard and two optional front-end interfaces
- A Peripheral Expander connected to:
  - A card reader
  - A printer/plotter
  - A magnetic tape unit

By using the I/O Subsystem, mass storage can be expanded to up to 48 disk storage units. Additionally, through use of a Memory Channel, the I/O Subsystem is capable of transferring data directly into memory at extremely high rates without interrupting the CPU.

A Block Multiplexer Controller allows easy addition of other vendors' peripheral equipment. Each controller supports 4 channels.

The Maintenance Peripherals provide for operational and maintenance functions.

I/O Subsystem Functions

Up to 4 I/O Processors comprise an I/O Subsystem. The Master I/O Processor (MIOP) and the Buffer I/O Processor (BIOP) are required; the other 2 processors are designed for moving data at high speeds and are optional. The MIOP connects to the Peripheral Expander, to the CRT consoles, to the CPU (through a normal channel), and to front-end computer systems. The BIOP is connected to a Memory Channel and moves data between Buffer Memory and Central Memory at rates of

Registers

The basic set of programmable registers are composed of:

- 8 24-bit address (A) registers
- 64 24-bit address-save (B) registers
- 8 64-bit scalar (S) registers
- 64 64-bit scalar-save (T) registers
- 8 64-word (4096-bit) vector (V) registers

Expressed in 8-bit bytes, the CRAY-1 operating registers represent a total of 4,888 bytes of very high speed (6 nanosecond) storage.

The 24-bit A registers are generally used for addressing and counting operations. Associated with them are 64 B registers, also 24 bits wide. Since the transfer between an A and a B register occupies only 1 clock period, the B registers assume the role of cache, storing information for fast access without tying up the A registers for long periods.

The 64-bit S registers are used for floating-point, logical, and some integer and character operations. The 64-bit T registers act as cache memory for the S registers.

Each of the 8 V registers is actually a set of sixty-four 64-bit registers. The V registers are used for vector operations. Successive elements from a V register enter a functional unit in successive clock periods. The effective length of a vector register for any operation is controlled by a program selectable vector length (VL) register. The vector employed in any calculation need not contain exactly 64 elements. A vector mask (VM) register allows for the logical selection of particular elements of a vector.

In addition to the operating registers, the CPU contains a variety of auxiliary and control registers. These are generally not accessible to a programmer.

Addressing

Instructions that reference data do so on a word basis. Branch instructions, on the other hand, reference parcels within words; the lower 2 bits of an address identify the location of an instruction parcel in a word. Significantly, the destination of a jump can be any instruction in the program; word alignment is not required.

Real-time Clock

Programs can be precisely timed with a real-time clock that increments once each 12.5 nanoseconds.

Programmable Clock

A programmable real-time clock that has a frequency of 80 Mhz, corresponding to an increment of 100 nanoseconds is a standard feature of a CRAY-1 S Series Computer System. This clock allows the operating system to force interrupts to occur at a particular time or frequency.

Instruction Set

The comprehensive CRAY-1 instruction set features over 100 operation codes and provides for both scalar and vector processing. Most instructions occupy 16 bits (1 parcel); certain branch instructions and memory reference operations occupy 32 bits (2 parcels).

Floating-point instructions provide for addition, subtraction, multiplication, and reciprocal approximation. The reciprocal approximation instruction enables the CRAY-1 to have a completely segmented divide operation through performance of a floating-point divide algorithm.

Integer addition, subtraction, and multiplication are provided for by the hardware. An integer multiply operation produces a 24-bit result; an addition or subtraction produces either a 24-bit or a 64-bit result. An integer divide is accomplished through a software algorithm using floating-point hardware.

The instruction set includes Boolean operations for OR, AND, exclusive OR, and for a mask-controlled merge operation. Shift operations allow for the manipulation of 64-bit or 128-bit operands to produce a 64-bit result. Similar 64-bit arithmetic capability is provided for both scalar and vector processing.

A programmer may index throughout memory in either scalar or vector processing mode. This full indexing capability allows matrix operations in vector mode to be performed on rows, on columns, on diagonals and, in general, on any set of data that is stored in memory with regular spacing between elements.

Instructions for population, parity, and leading zero counts (scalar only) return bit counts based on register contents.

Instruction Buffers

The CRAY-1 has 4 instruction buffers, each of which holds 64 consecutive 16-bit instruction parcels. The buffers are large enough to hold substantial noncontiguous program segments. Fetching of program steps does not interfere with data or I/O transfer to or from memory.

If a required instruction is not buffer resident, an out-of-buffer condition occurs, causing instructions to be fetched cyclically from the memory banks beginning always with the instruction required for execution. Buffers are loaded from memory starting with the buffer least recently filled at a rate of 4 words per clock period after a 10 clock period startup.

An important feature of the instruction buffers is that both forward and backward branching is possible within them. No reloading of buffers occurs if the instruction being branched to is buffer resident.

Data Structure

CRAY-1 internal character representation is in ASCII with each 64-bit word able to accommodate 8 characters.

All integer arithmetic is performed in 24-bit or 64-bit 2's complement mode. Floating-point numbers, 64-bit quantities, consist of a signed magnitude binary coefficient and a biased exponent. The unbiased exponent range is:

2^{-20000_8} to 2^{+17777_8}, or approximately

10^{-2466} to 10^{+2466}

An exponent greater than or equal to 2^{+20000_8} is recognized as an overflow condition and causes an interrupt if floating point interrupts are enabled.

Functional Units

Instructions other than simple transmit or control operations are performed on the CRAY-1 by hardware organizations known as functional units. Each functional unit specializes in implementing algorithms for a specific portion of the instruction set and operates totally independently of the other units.

A functional unit performs its operation in a fixed time called the functional unit time. No delays are possible once the operands have been delivered to a functional unit.

All functional units have 1-clock-period segmentation. As a result, information arriving at or moving within the unit is captured and held in a new set of functional unit registers at the end of every clock period. New pairs of operands can then enter the functional unit each clock period even though the unit may require more than 1 clock period to complete the calculation.

All functional units can operate concurrently so that in addition to the benefits of pipelining (each unit can be driven at a result rate of 1 per clock period), there is also parallelism across the units.

The Design of the CRAY-1

The CRAY-1 Central Processing Unit

From whatever level it is examined, the architecture of the CRAY-1 is clean and simple.

A 6.5 foot high hollow semi-cylinder occupying a mere 70 square feet of floor space, the CRAY-1 challenges the fundamental limitation placed on all computers by the speed of light. By keeping wire lengths short, signal propagation times are minimized. Within the CRAY-1, electronic modules comprising the CPU lie on the outside of the cylinder; the interconnections are as compact as possible on the inside.

The dense concentration of components requires new techniques to overcome the accompanying problems of heat dissipation. In the CRAY-1, liquid refrigerant is used to maintain internal temperatures of approximately 68° F.

The upholstered benches surrounding the CPU conceal the CRAY-1's power supplies.

Only one physical module type appears throughout the Central Processing Unit (CPU) —a module consisting of two 6" x 8" printed circuit boards mounted on opposite sides of a heavy copper heat transfer plate. Each circuit board, in turn, holds a maximum of 144 integrated circuit (IC) packages and approximately 300 resistors. A few basic chip types are used, allowing small field inventories for on-site module repair.

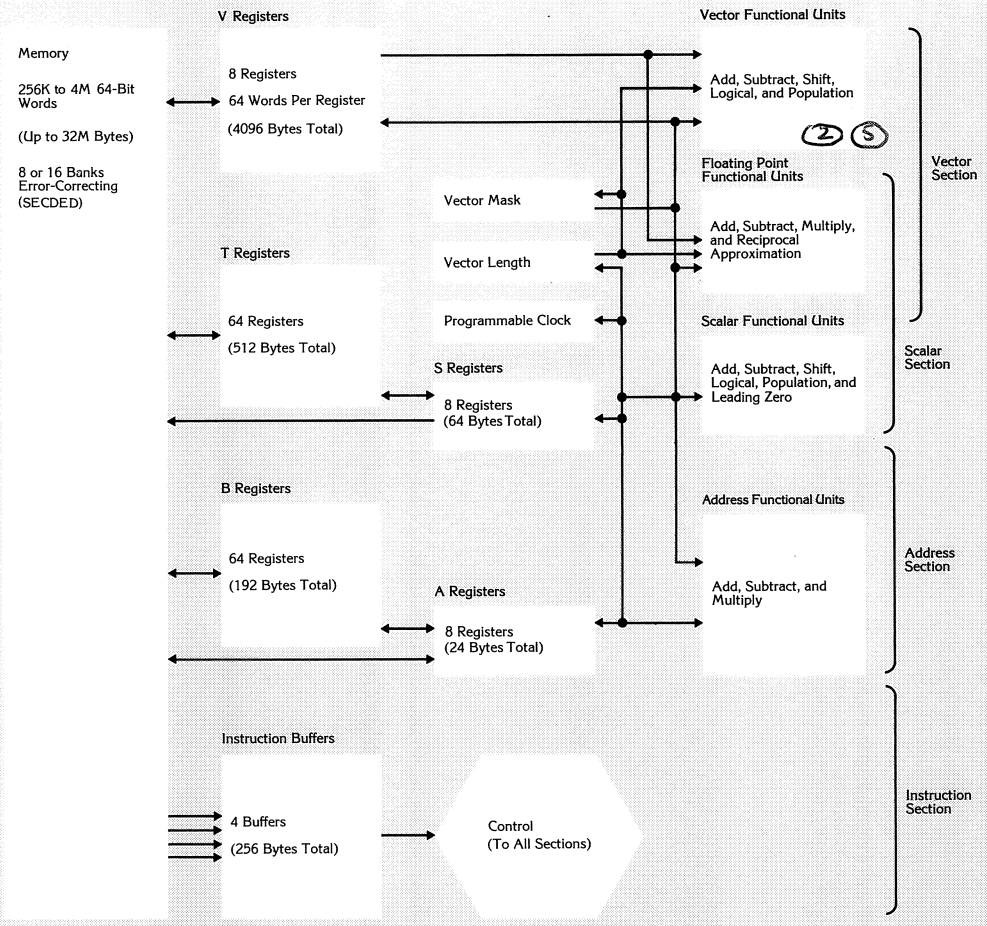
Features of the CRAY-1 CPU that contribute to its high speed and reliability include:

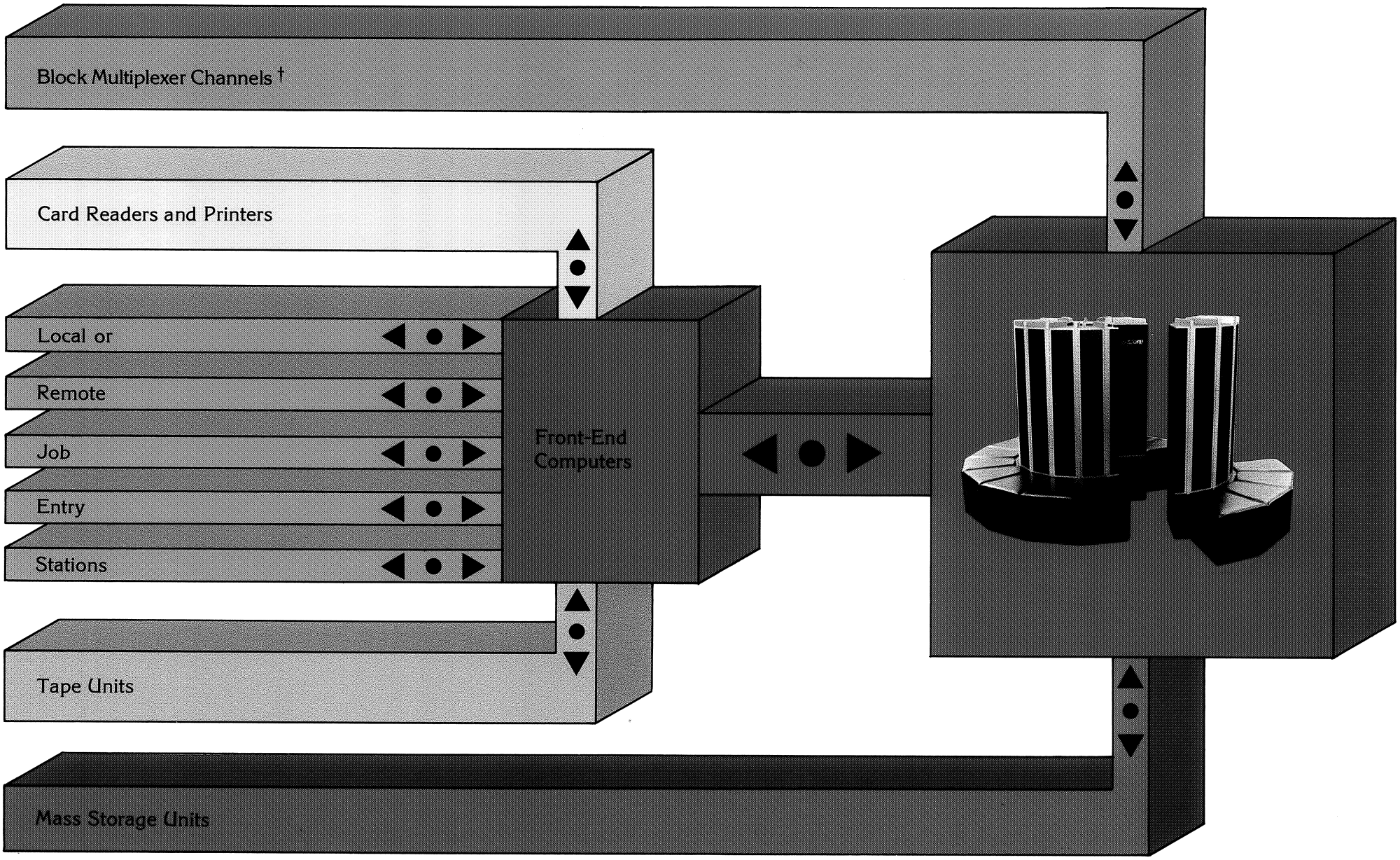
- ☐ Extreme compactness;
- ☐ 12.5 nanosecond clock period;
- ☐ Vector processing allowing up to 64 pairs of operands to be operated on by a single instruction;
- ☐ Random access semiconductor memory that transfers up to 320 million 64-bit words per second while performing single error correction/double error detection (SECDED);
- ☐ 13 fully segmented functional units that operate in parallel to support vector and scalar processing of fixed and floating point arithmetic as well as Boolean and related operations;
- ☐ 12 I/O channels which, with associated circuitry, transfer data at speeds determined by the peripheral devices while performing error checking and data assembly/disassembly; and
- ☐ A Memory Channel (Models S/1200 through S/4400) that allows I/O data transfers directly into memory at rates of up to 850 megabits per second.

Computation Section

Within the computation section are operating registers, functional units, and an instruction control network—hardware elements that cooperate in executing sequences of instructions. The instruction control network makes all decisions related to instruction issue as well as coordinating the three types of processing: vector, scalar, and address. Each of the processing modes has its associated registers and functional units.

The block diagram of the CRAY-1 CPU illustrates the relationship of the registers to the functional units, instruction buffers, I/O channel control registers, and memory.





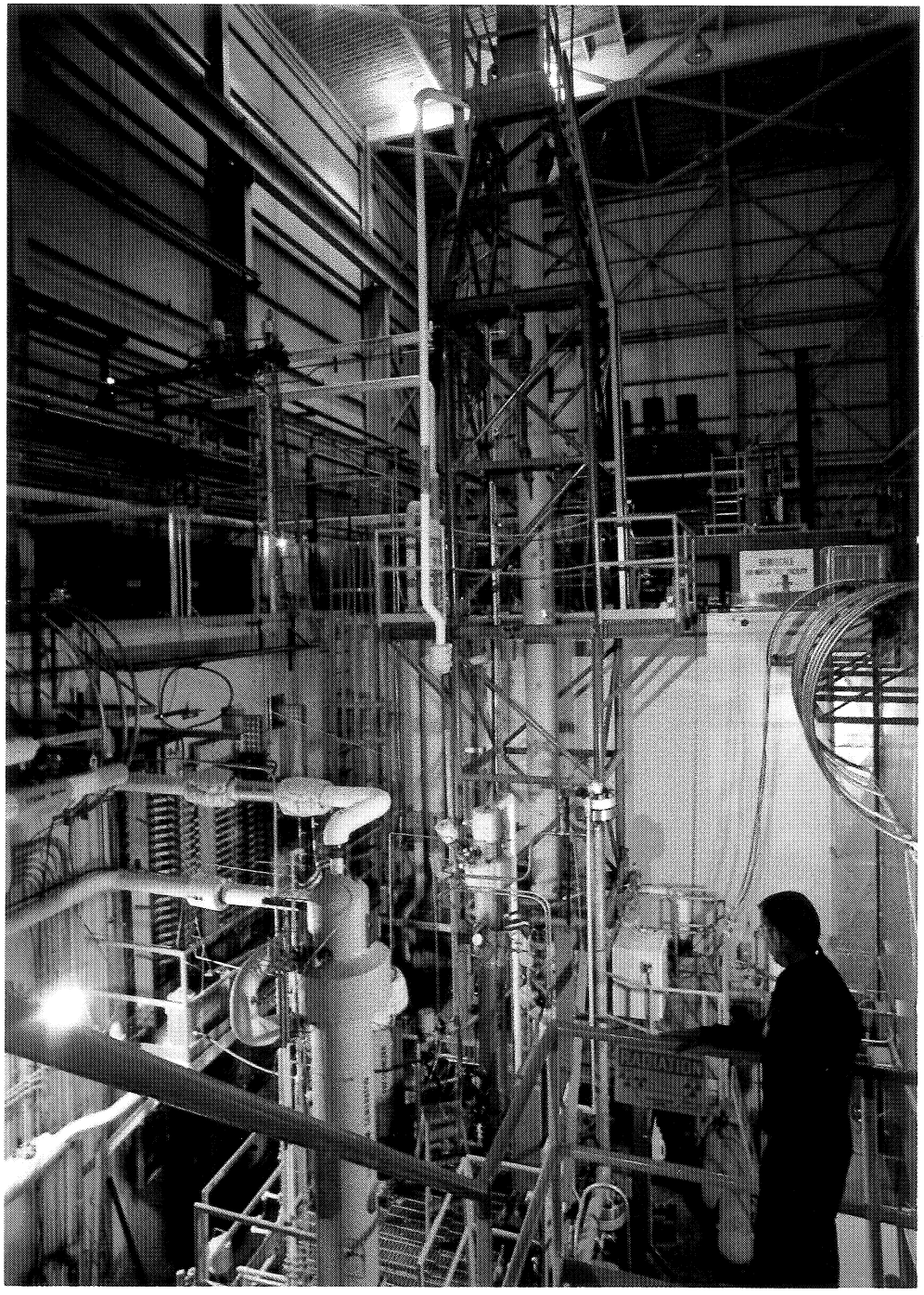
† The I/O Subsystem and Block Multiplexer Channels are available on Models S/1200 and above.

Job Flow

A job may originate from any of a variety of sources local to or remote from your front-end computer. Thus, your existing computers serve as entry stations for submitting jobs to the CRAY-1 Computer System or as data concentrators for multiplexing several remote stations or terminals. Your computer may also provide operator functions by passing commands and messages between the operator and the CRAY-1.

After submission to the CRAY-1, a job waits on mass storage until COS determines that the resources the job needs are available. Then, the system begins job processing by examining the associated job control statements. These statements are read, interpreted, and acted on sequentially. Output from the job is placed on mass storage. At job completion, output is transferred back to the front-end or terminal of job origin for additional processing such as printing or transfer to magnetic tape.

CRAY-1 Utility Programs



David Frazier—Webb Photo

Utility programs available to the CRAY-1 user include:

- ☐ LDR, a relocatable and overlay loader, which allows program modules to be loaded, relocated, and linked to externals in a single pass, and allows redefinition of programs into overlays, (separate modules called into execution when necessary)
- ☐ UPDATE, a program for maintaining program source code
- ☐ BUILD, a library generation and maintenance program
- ☐ Programs for the management and modification of datasets permanently resident on mass storage
- ☐ Programs for copying records, files, and datasets
- ☐ Programs for positioning datasets relative to records and files
- ☐ Compare programs
- ☐ Dump programs and other aids
- ☐ Programs for analyzing the system logfile

The CRAY-1 assists in nuclear process modeling and economic analysis.

The CRAY-1 Operating System (COS)

A vital ingredient of the CRAY-1 Computer System is the CRAY-1 Operating System (COS). COS is an advanced operating system offering a multiprogramming batch environment to the user. Up to 63 jobs can be in some stage of processing concurrently.

COS manages all system resources, supervises job processing, and performs input/output operations. Though it is mostly memory resident (all system utilities reside on mass storage), COS occupies less than 5% of memory on a one million word CRAY-1, leaving the bulk of memory available for user jobs. COS is straightforward and uncomplicated.

COS organizes and maintains information on system mass storage. Its dataset management capability provides for the highly efficient creation and maintenance of temporary and permanent datasets, taking full advantage of multichannel access to mass storage.

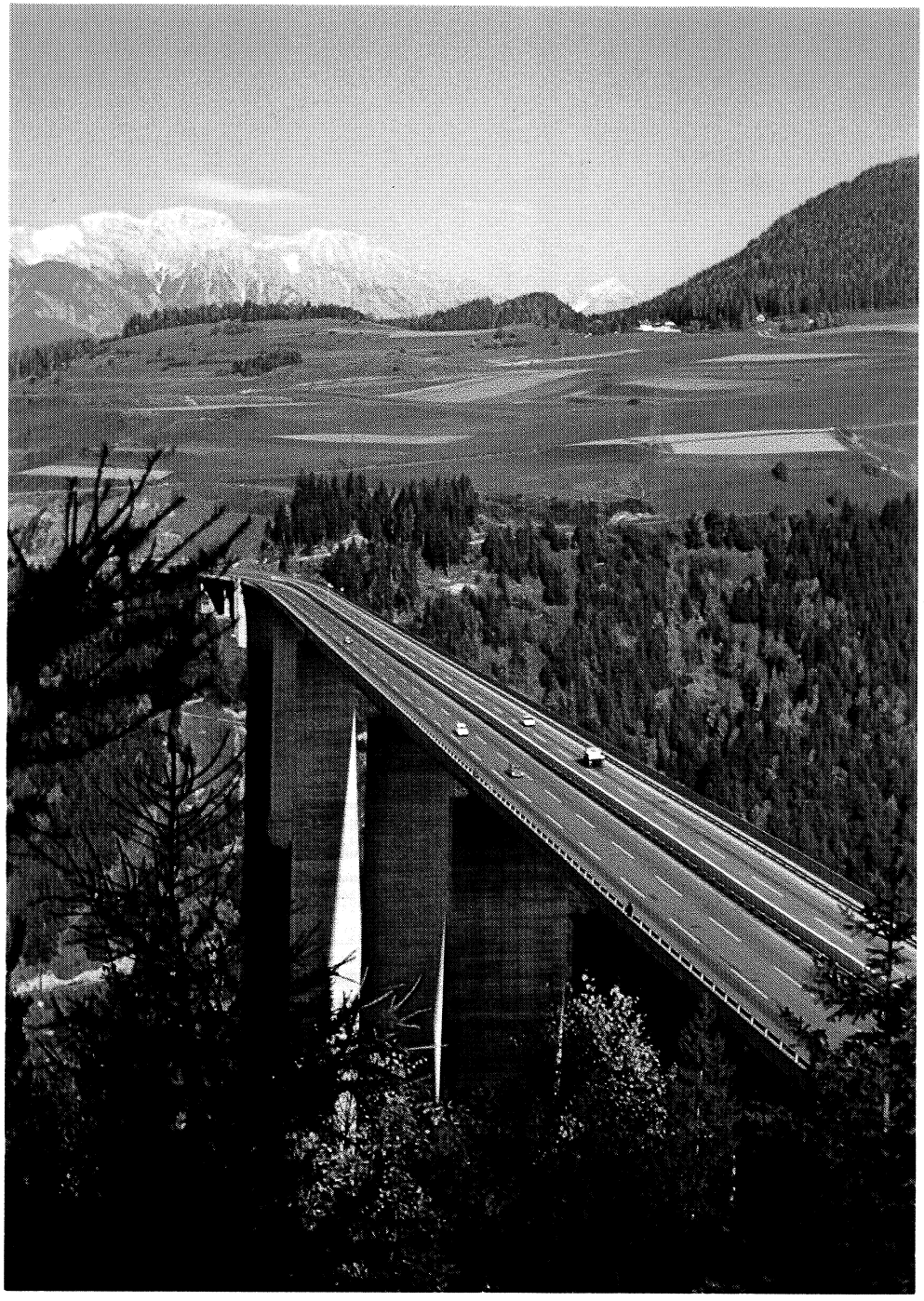
COS monitors and controls CRAY-1 Computer System resources by allocating memory and mass storage, by scheduling jobs, and by maintaining accounting records.

Jobs and job control information are supplied to the CRAY-1 via a front-end computer or at local or remote job entry stations. Results of CRAY-1 operations, including a logfile of the processed job control statements and accounting information, are returned to the front-end or station of job origin.

Primary features of the Operating System include:

- ☐ Resource management
- ☐ Remote or local job entry
- ☐ Multiprogramming of up to 63 jobs concurrently
- ☐ Recovery of jobs following a system interruption
- ☐ Printout of a chronological history (a logfile) of each job
- ☐ Communication with station operators
- ☐ Staging of data between system mass storage and front-end peripherals
- ☐ Program maintenance

The CRAY-1 Assembler (CAL)



Phillip MacMillan James—Webb Photo

The CRAY-1 assembly language enables a user to closely tailor a program to the architecture of the CRAY-1. Through CAL, a programmer may express symbolically all hardware functions of the CRAY-1. CAL allows the production of highly efficient machine-language programs. The user may designate program and data information to enable complete control of the CRAY-1 Central Processing Unit.

Augmenting the instruction repertoire is a set of versatile pseudo operations that provide for defining macro instructions and controlling the assembler. CAL applies extensive diagnostics to programs during their assembly and issues error codes where appropriate.

CAL, CFT, the operating system, and most standard software provided by Cray Research are coded in CRAY-1 assembly language.

The CAL assembler, like the CRAY-1 FORTRAN Compiler, is extremely fast. A typical assembly rate is about 250,000 lines per minute.

Complex structural analysis can be visualized in 3 dimensions using the CRAY-1.

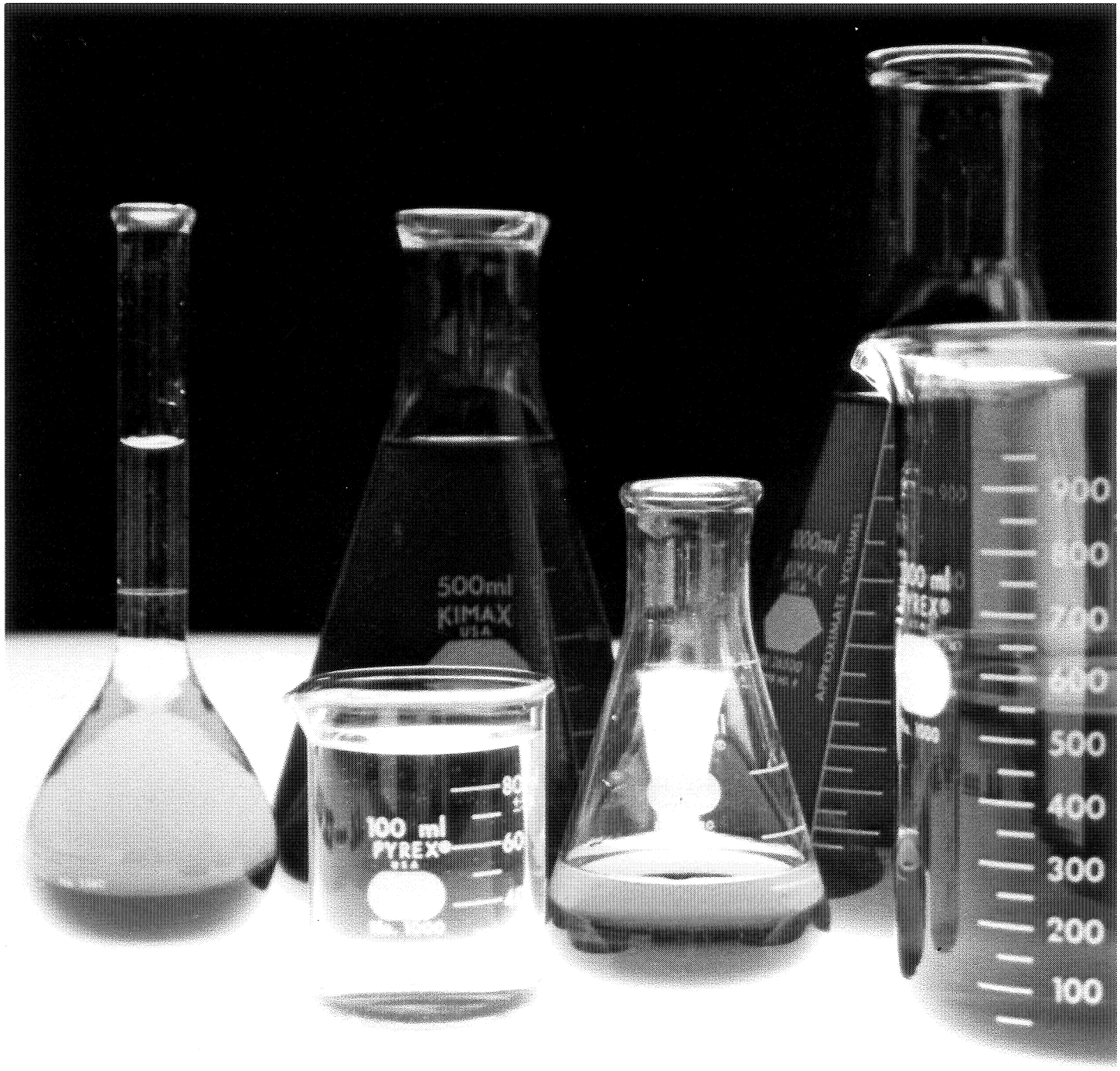
Subroutine Library

The existence of scalar and vector versions of all standard FORTRAN library routines enables CFT to automatically choose the appropriate version.

The possibilities for optimization are further enhanced by the Basic Linear Algebra Subroutines (BLAS), a comprehensive library of commonly used mathematical routines. The library currently includes fast Fourier transforms, matrix and linear algebra packages, and other FORTRAN callable subroutines. Many of these hand-coded routines use the pipeline/chaining properties of the CRAY-1 hardware. These routines are fast—often executing at over 140 million floating-point operations per second—and provide easy access to the full vector power of the CRAY-1.

The CRAY-1 aids in locating and producing fossil fuels through reservoir modeling and seismic analysis.

Webb Photo



Gary Bistram

CFT features include:

- ☐ Full ANSI X3.9-1966 compatibility with extensions toward and beyond ANSI X3.9-1978
- ☐ Acceptance of most dialects and syntaxes implemented for other large-scale computers
- ☐ Automatic detection and vectorization of inner loops
- ☐ Positive, negative, and zero integer and floating point DO-loop indices and limits
- ☐ Arbitrary subscript range
- ☐ ENCODE and DECODE statements
- ☐ BUFFER IN and BUFFER OUT statements
- ☐ NAMELIST statement
- ☐ Random I/O
- ☐ Descriptions of vectorized loops
- ☐ Flow analysis, assembly code, cross reference maps, and many other listing and debugging options
- ☐ A compilation rate of between 150,000 and 250,000 lines per minute

The CRAY-1 promises to help with exciting breakthroughs in medical and chemical research.

The CRAY-1 FORTRAN Compiler (CFT)

CFT makes the tremendous power of the CRAY-1 readily accessible at the user level by removing most of the burden of optimization and vectorization of code.

CFT is a mature compiler. Its development began concurrent with CRAY-1 hardware design in 1973. Compatible with the ANSI X3.9-1966 FORTRAN Standard, the compiler is rapidly nearing its goal of adherence to the X3.9-1978 Standard.

The compiler, itself, employs the unique features of the CRAY-1 architecture. It compiles FORTRAN language programs into machine language code that exploits the CRAY-1 processor at near optimal rates. This protects the user's investment in FORTRAN program development and eliminates the need for costly conversion of existing codes.

CFT is unmatched in the extent to which it automatically generates vectorized machine language code. Thus, for the first time, the power of a vector computer becomes immediately accessible to the user having no prior experience with vectorization techniques. CFT analyzes the innermost loops of a FORTRAN program to detect vectorizable sequences. It then generates code that takes full advantage of the segmentation and independence of the CRAY-1's functional units. A vectorized inner loop exploits the great speed of vector operations, generally executing three to ten times faster than its scalar equivalent.

CFT links vector computations, saving variables and subexpressions in intermediate registers rather than in memory. Doing so reduces the number of memory accesses.

A wide range of compiler options has been developed. One feature in particular, Flowtrace, provides the programmer requiring additional optimization with a valuable diagnostic tool. By enabling Flowtrace, the programmer obtains a complete analysis of execution time spent in each subroutine, the number of times each subroutine was called, and subroutine and linkage overhead.

Other compiler options enable listings of assembly code, cross reference maps, and other debugging aids.

Cray Research has made an extensive effort to allow most dialects of FORTRAN and to accept many nonstandard syntax structures that are common in programs written for other manufacturers' equipment.

Tim Larsen

Software for Solving Problems

Any power is useful only when correctly applied. Accordingly, the design and efficiency of CRAY-1 software supports the hardware computation rates which can exceed 140 million floating-point operations per second.

The CRAY-1's computing power is accessed at two different software levels—FORTRAN and the assembly language—and applied to a broad spectrum of scientific and engineering applications.

All Cray Research software is designed and documented for ease of application, extendability, and maintenance.

Cray Research software has matured through stringent testing at a variety of user sites. It has earned a reputation for both utility and reliability.

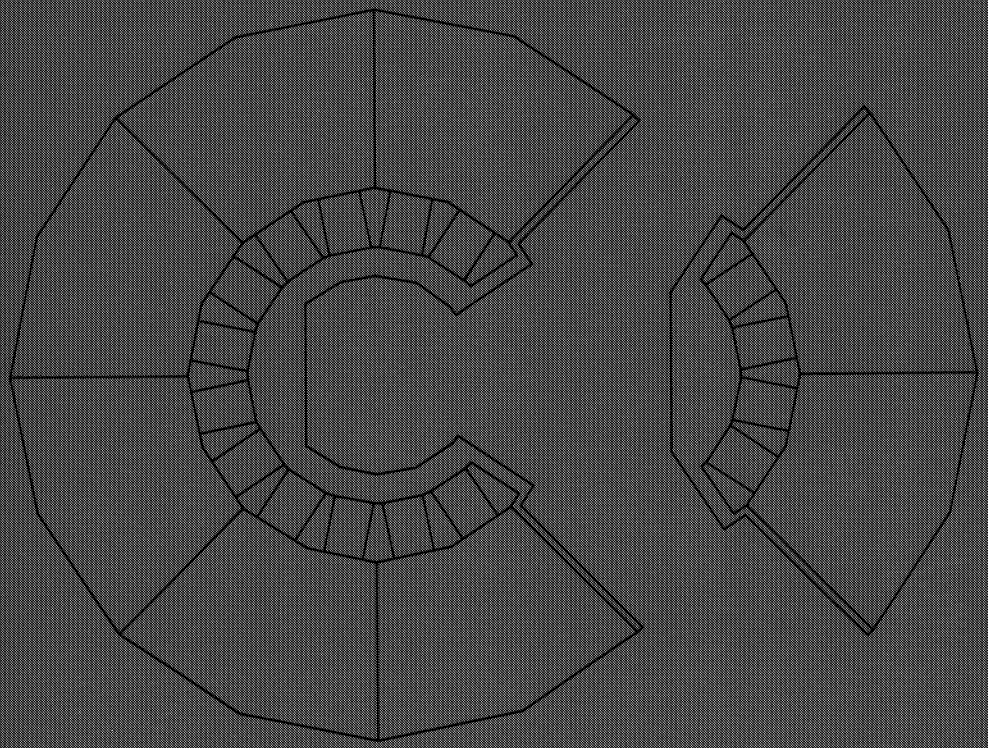
CRAY-1 Software

- ☐ CFT, a vectorizing and optimizing FORTRAN Compiler,
- ☐ FORTRAN library subroutines,
- ☐ CAL, a versatile assembler,
- ☐ COS, an advanced multiprogramming operating system,
- ☐ A variety of system utility programs, and
- ☐ Front-end interfaces for IBM 370 MVT and MVS and CDC NOS and NOS/BE

The CRAY-1 provides 7 to 10 day weather forecasts vital to farmers, oil drillers, fishermen, builders...

Features

- Front-end interfaces
- Memory with 1M, 2M or 4M 64 bit words (8M to 32M bytes)
- Field upgradability through the S Series
- Up to 16 block multiplexer channels
- 850 Mbit/sec I/O Subsystem to memory transfer rates
- Disk capacity up to 48 600Mbyte drives
- Disk transfer rates of 4.4 Mbytes/sec.
- 12 I/O Channels
- Software compatibility across the S Series



CRAY-1 Models S/1200 through S/4400

The CRAY-1 Models S/1200 through S/4400 systems are composed of a few basic hardware components. These are:

The Central Processing Unit (CPU) with:

- ☐ Either 1 M, 2 M, or 4 M words of Central Memory
- ☐ 12 I/O Channels

An I/O Subsystem composed of:

- ☐ 2, 3 or 4 high-speed I/O Processors
- ☐ 1 High Speed Memory Channel
- ☐ 0.5 M or 1 M words of I/O Buffer Memory
- ☐ 1 to 12 DCU-4 Disk Control Units
- ☐ 2 to 48 DD-29 Disk Storage Units
- ☐ 1 to 4 Block Multiplexer Controllers
- ☐ 1 to 16 Block Multiplexer Channels
- ☐ 2 CRT consoles

A Peripheral Expander connected to:

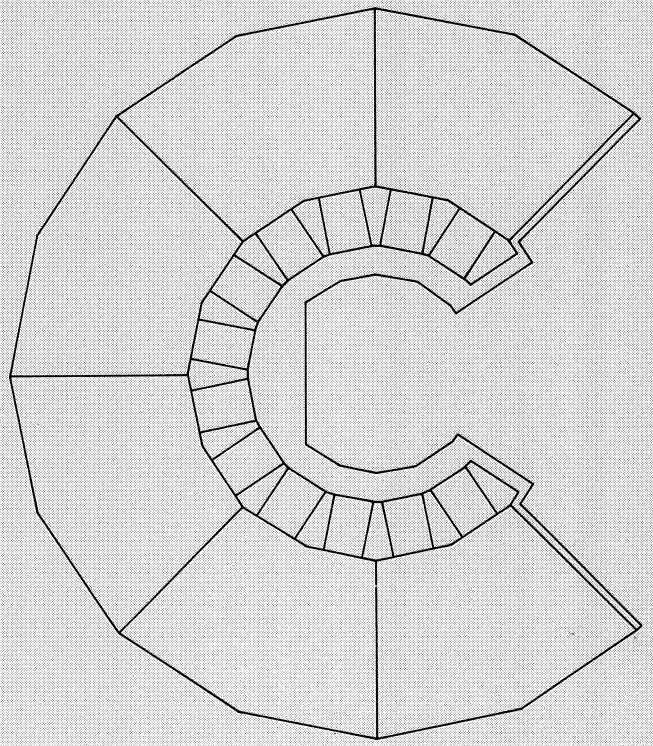
- ☐ A card reader
- ☐ A printer/plotter
- ☐ A magnetic tape unit

Power and cooling equipment

One standard, two optional front-end interfaces

Features

- Front-end interfaces
- Memory—with ¼M, ½M or 1M 64-Bit Words (2 M to 8 Mbytes)
- Field upgradability through the S Series
- Disk capacity up to 32 600 Mbyte drives
- Disk transfer rates of 4.4 Mbytes/sec.
- 12 I/O Channels
- Software compatibility across the S Series



CRAY-1 Models S/250, S/500, and S/1000

The CRAY-1 Models S/250, S/500, or S/1000 systems are composed of a few basic hardware components. These are:

The Central Processing Unit (CPU) with:

- ☐ Either 0.25M, 0.5M, or 1M words of Central Memory
- ☐ 12 I/O Channels

A Maintenance Control Unit composed of:

- ☐ A minicomputer
- ☐ A card reader
- ☐ A magnetic tape unit
- ☐ A removable pack disk drive
- ☐ A printer/plotter
- ☐ 2 CRT consoles

A Mass Storage (Disk) Subsystem consisting of:

- ☐ 2 to 8 DCU-3 Disk Control Units
- ☐ 2 to 32 DD-29 Disk Storage Units

Power and cooling equipment

One standard, two optional front-end interfaces

Model	S/250	S/500	S/1000	S/1200	S/1300	S/1400	S/2200	S/2300	S/2400	S/4200	S/4300	S/4400
CPU Central Memory size (64-bit words)	¼M	½M	1M	1M	1M	1M	2M	2M	2M	4M	4M	4M
Front-End Interfaces	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3
I/O Subsystem												
I/O Processors				2	3	4	2	3	4	2	3	4
Buffer Memory Size				½ or 1M	½ or 1M	½ or 1M	½ or 1M	½ or 1M	½ or 1M	½ or 1M	½ or 1M	½ or 1M
DCU-4 Disk Control Units				1-4	1-8	1-12	1-4	1-8	1-12	1-4	1-8	1-12
DD-29 Disk Storage Units				2-16	2-32	2-48	2-16	2-32	2-48	2-16	2-32	2-48
Block Multiplexer Controllers					1-4	1-4		1-4	1-4		1-4	1-4
Block Multiplexer Channels					1-16	1-16		1-16	1-16		1-16	1-16
Mass Storage Subsystem												
DCU-3 Disk Control Units	2-8	2-8	2-8									
DD-29 Disk Storage Units	2-32	2-32	2-32									

Introducing the CRAY-1 S Series of Computer Systems

The CRAY-1 Computer System has evolved into the CRAY-1 S Series of Computer Systems. Users now have a choice of several models and options. One of the S Series models is sure to meet your specific needs.

At the lower end of the S Series is the Model S/250, which has 256K words of Central Memory. The next two larger models differ from the Model S/250 primarily in memory capacity—the S/500 has 512K words and the S/1000 has 1024K (or 1 million) words. On these three models, front-end computer systems and mass storage link directly to I/O Channels on the CRAY-1 CPU, much as they do on the earlier models of the CRAY-1.

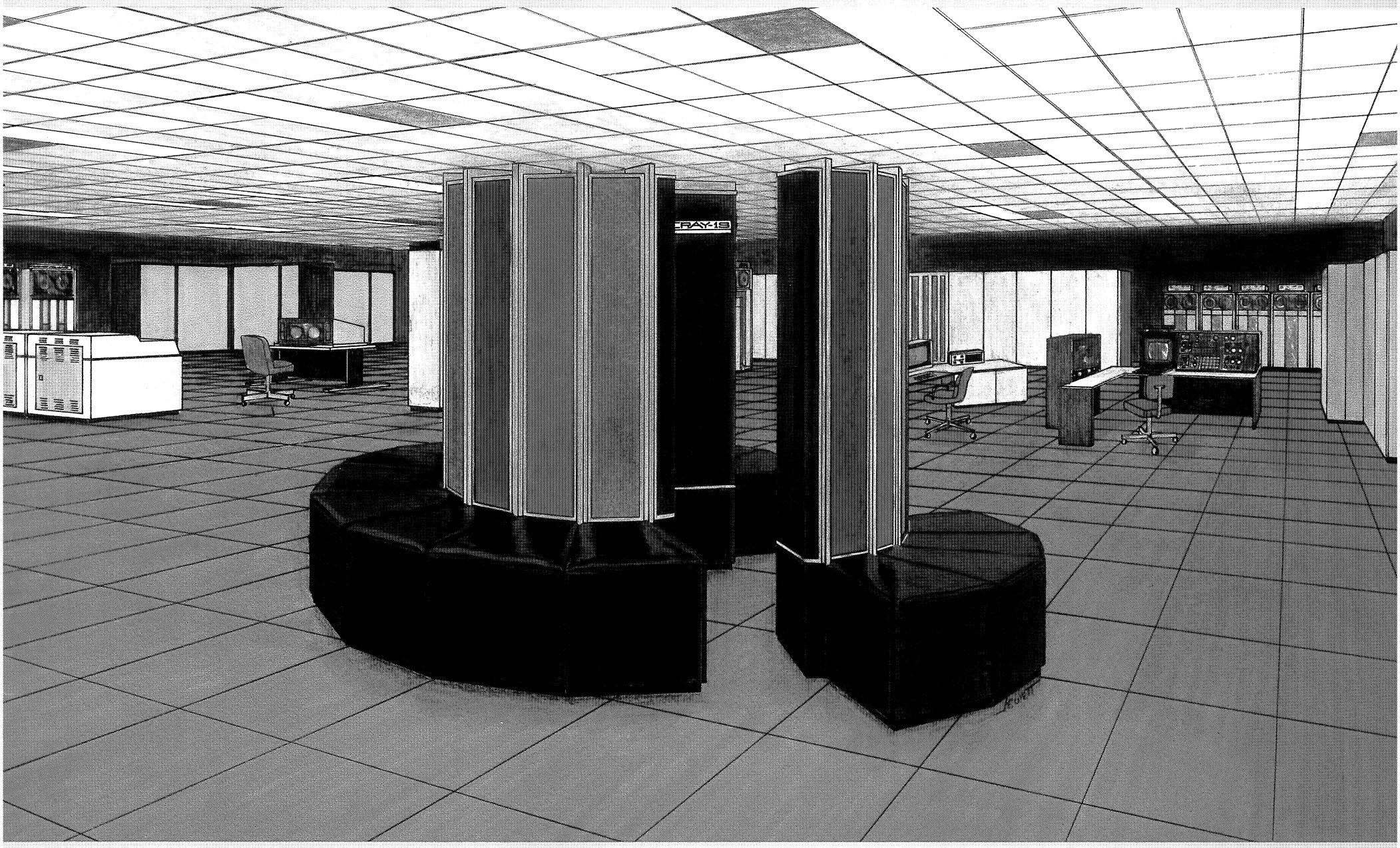
Starting with the S/1200 (also with 1 million words), I/O throughput to front-end computers and to mass storage devices has been significantly enhanced with the incorporation of an I/O Subsystem. The I/O Subsystem is a Cray Research product specifically designed to complement the CRAY-1 CPU requirements. A primary feature is the incorporation of a Memory Channel linking the I/O Subsystem to Central Memory. Maximum transfer rates of 850M bits per second are achievable on this channel. The power of the I/O Subsystem relates directly to the number of I/O Processors it contains. Two, three, or four I/O Processors may comprise the I/O Subsystem. With each addition of another I/O Processor, significant increases in mass storage capacity or the ability to drive peripheral devices is achieved.

As your computing needs increase, you can expand your system by upgrading your S Series CRAY-1 Computer System to a higher model by adding more memory or incorporating a more powerful I/O Subsystem. Upgrading is possible all the way to the Model S/4400, with a maximum of four million words and four I/O processors.

Reliability in hardware and software is a key ingredient in a successful processing environment. The clean architecture of the CRAY-1 is teamed with proven logic circuitry resulting in new levels of hardware reliability. The semicircular mainframe houses over 200,000 integrated circuits, 3400 printed circuit boards, and over 60 miles of wire, yet it takes up less than 70 square feet of floor space. CRAY-1 system up-time recorded in a wide range of production environments is unprecedented for systems of its class.

While you have been reading these lines, somewhere in the world a CRAY-1 Computer System performed several billion calculations. As today's most advanced scientific computer, a CRAY-1 Computer System offers, through its outstanding speed and power, a major computational resource providing new dimensions and capabilities to its users.

Our continuing commitment at Cray Research to be the industry leader of large-scale computer systems is represented by the CRAY-1 S Series, an evolutionary family of field-upgradable systems that can deliver significant price/performance solutions for the scientific and engineering user.


CRAY-1S

The CRAY-1 and Your System

A CRAY-1 Computer System complements your existing system, serving as a powerful component in a distributed system geared to solving complex problems and handling large amounts of data.

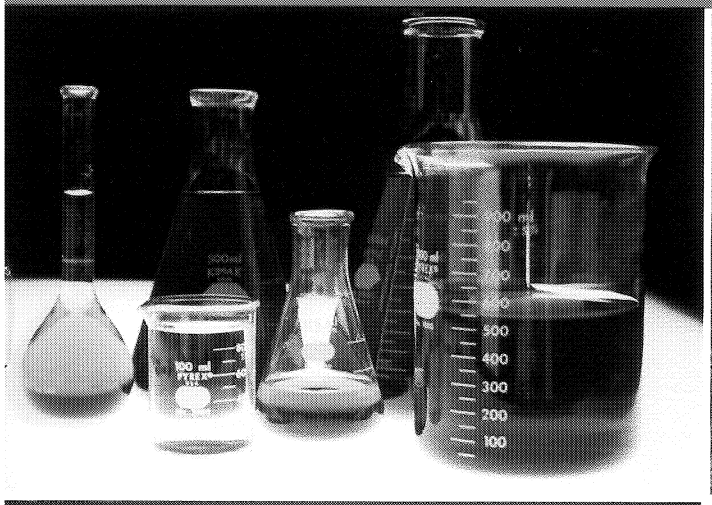
By adding a CRAY-1 to an existing facility you can achieve extremely cost effective computation. Dramatic improvements in throughput are possible when large CPU-bound jobs are off-loaded from your current system onto the CRAY-1. Services such as the operation of slow-speed peripherals may then be handled by a front-end computer operating under control of its own operating system in a mode asynchronous to the CRAY-1.

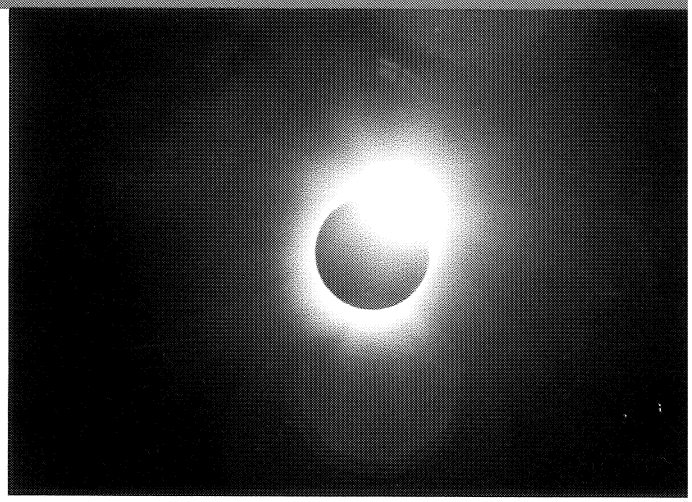
The CRAY-1 has been successfully interfaced with computers from a number of other manufacturers. A wide variety of computer systems are now serving as front-end processors for CRAY-1 systems. Cray Research offers software interfaces to link the CRAY-1 with such computers as the IBM 370 Series MVT or MVS based front-end computers and for CDC NOS and NOS/BE systems.

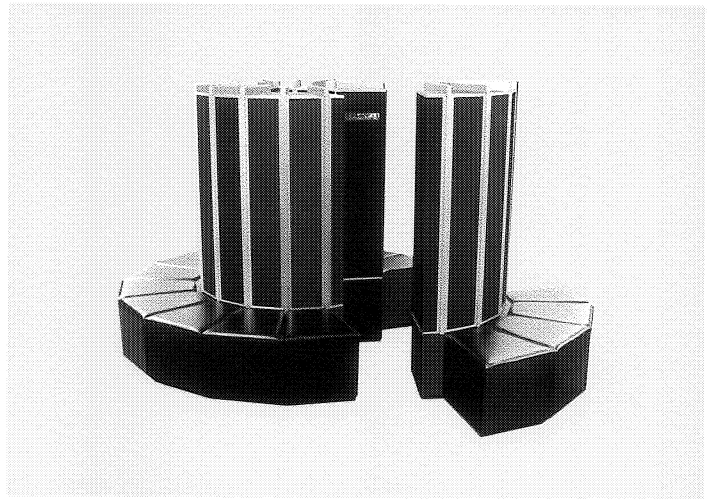
The CRAY-1 FORTRAN Compiler represents Cray Research's commitment to state-of-the art software. This extremely efficient and powerful program allows nearly transparent access to the features that make the CRAY-1 so powerful. Thus, new or existing FORTRAN programs can be compiled and run quickly on a CRAY-1, taking immediate advantage of the computer's potential.

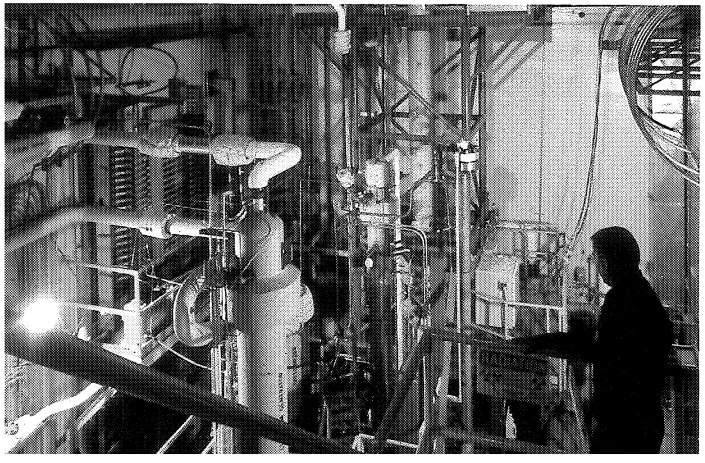
Current Front-End Systems for the CRAY-1

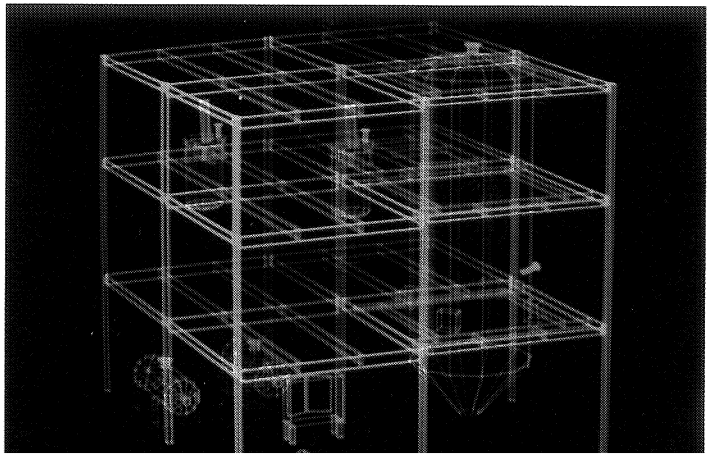
- ☐ CDC CYBER 70/170 and 6000/7000
- ☐ IBM 370
- ☐ Amdahl
- ☐ Honeywell
- ☐ DEC
- ☐ Data General
- ☐ Systems Engineering Laboratories

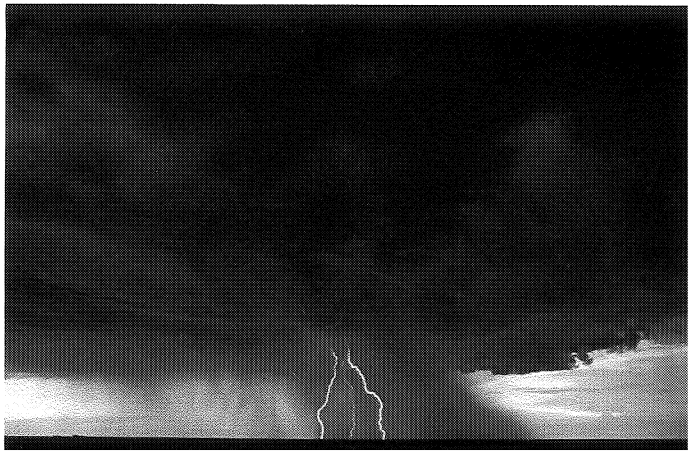

500
400
300
200
100











Cray Research, Inc.

Corporate Headquarters

1440 Northland Drive
Mendota Heights, MN 55120
Tel: (612) 452-6650
TLX: 298444

Sales Offices

Eastern Regional Sales Office
10750 Columbia Pike, Suite 602
Silver Spring, MD 20901
Tel: (301) 681-9626

Pittsburgh Sales Office
#1 Northgate Square
Greensburg, PA 15601
Tel: (412) 832-8120

Central Regional Sales Office
5330 Manhattan Circle, Suite F
Boulder, CO 80303
Tel: (303) 499-3055

Houston Sales Office
3121 Buffalo Speedway, Suite 400
Houston, TX 77098
Tel: (713) 877-8053

Austin Sales Office
3415 Greystone, Suite 201
Austin, TX 78731
Tel: (512) 345-7034

Chicago Sales Office
625 No. Michigan Avenue, Suite 500
Chicago, IL 60611
Tel: (312) 944-1183

Albuquerque Sales Office
6400 Uptown Blvd. N.E.,
Suite 361-W
Albuquerque, NM 87110
Tel: (505) 881-8278

Western Regional Sales Office
Sunset Office Plaza
1874 Holmes Street
Livermore, CA 94550
Tel: (415) 447-0201

Los Angeles Sales Office
101 Continental Blvd., Suite 456
El Segundo, CA 90245
Tel: (213) 640-2351

Seattle Sales Office
1750 112th Avenue N.E.
Suite C-237
Belleuve, WA 98004
Tel: (206) 454-4438

International (Subsidiaries)

Cray Research (UK) Limited
James Glaisher House
Grenville Place
Bracknell, UK
Tel: (344) 21515
TLX: 51 848841

Cray Research GmbH
Wartburgplatz 7
8000 Munich 40
West Germany
Tel: (893) 63076
TLX: 41 521 3211

Cray Research Japan, Limited
Shin Aoyama Building, West 1661
1-1 Minami-Aoyama 1-chome
Minato-ku, Tokyo 107 Japan
Tel: (03) 403-3471
TLX: J28893