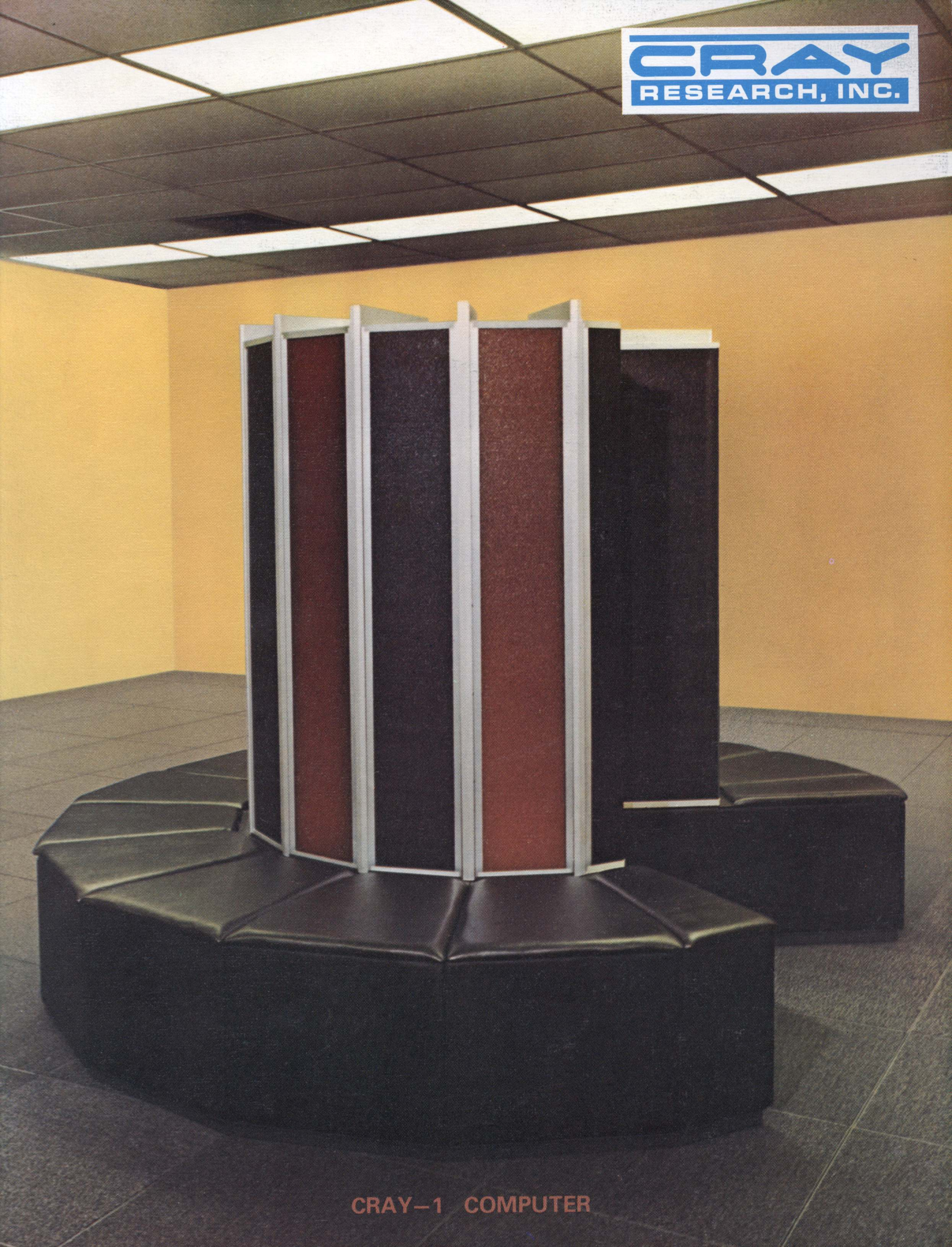


CRAY–1 COMPUTER

AN INTRODUCTION

TO THE

CRAY-1 COMPUTER

The Cray Research CRAY-1 computer is a powerful general purpose computer incorporating scalar and vector capabilities and a large, fast bi-polar memory. Vector processing provides result rates greatly exceeding the result rates of conventional scalar processing. Several unique design features of the CRAY-1 provide formidable competition for other existing vector machines. This paper describes the structure of the CRAY-1 and enumerates some of the important unique design features which give the machine its power. The paper concludes with examples to illustrate the speed of the CRAY-1 in vector applications.

Operating registers

The primary operating registers of the CRAY-1 are the scalar and vector registers, called S and V registers, respectively. Each of the eight V registers has sixty-four elements. A scalar instruction may perform some function, such as addition, obtaining operands from two S registers and entering the result into another S register. The analagous vector instruction performs the same function, obtaining each clock period (12.5 nanoseconds) a new pair of operands from two V registers. Results are entered into elements of another V register. The contents of the vector length (VL) register determines the number of operations performed by the vector instruction. Eight 24-bit A registers are used as address registers for memory references and as index registers. The A and S registers are each supported by sixty-four rapid-access temporary storage registers, called B and T registers, respectively. Data can be transferred between the A, B, S, T, or V registers and memory.

Memory

The CRAY-1 memory is constructed of bi-polar 1024-bit LSI chips. Up to one 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. One parity bit per word is maintained in 16 modules of the CPU. There is no inherent memory degradation for machines with less than one million words of memory.

Instruction buffers

All instructions, which may be 16 or 32 bits, are executed from four instruction buffers, each consisting of sixty-four 16-bit registers. Associated with each instruction buffer is a base address register that is used to determine if the current instruction resides in a buffer. Since the four instruction buffers are large, substantial program segments may reside in the buffers. Forward and backward branching within the buffers is possible and the program segments may be discontiguous. When the current instruction does not reside in a buffer, one of the instruction buffers 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 among the first to be read.

Input/Output

There are twenty-four I/O channels, twelve of which are input and twelve output. Any number of channels may be active at a given time. Each channel has a maximum transfer rate of 640 megabits. At most one 64-bit word per clock period can be transferred to or from memory and this is attained when four input channels and four output channels are simultaneously operating at their maximum rate. In practice this theoretical transfer rate is limited by the speed of peripheral devices and by memory reference activity of the CPU.

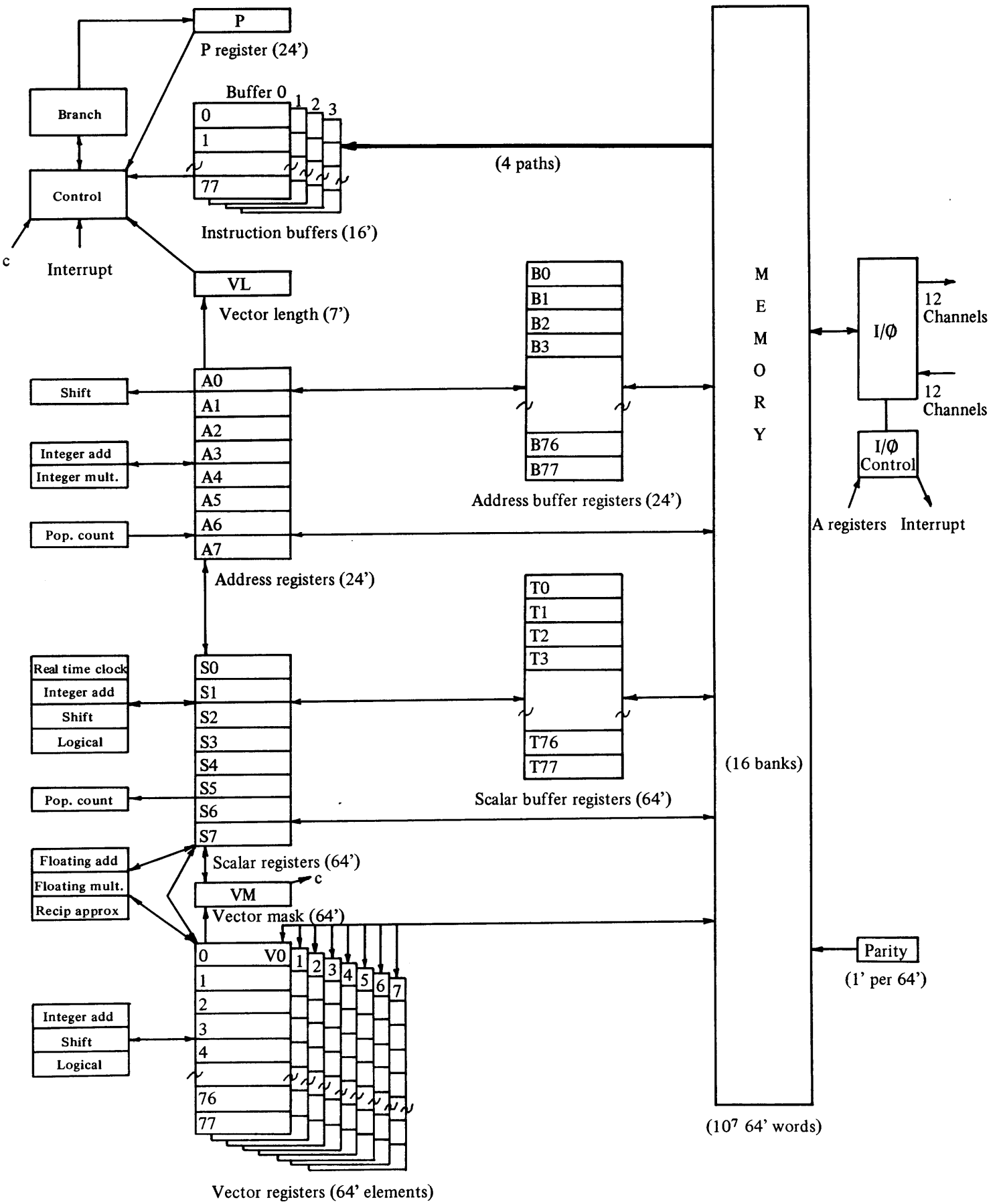


Figure 1. Registers block diagram

Functional units

There are twelve functional units in the CPU. Each is a specialized unit implementing algorithms for a portion of the instructions. Each unit is independent of the other units and a 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. There is no information retained in a functional unit for reference in subsequent instructions. These units operate essentially in three-address mode with limited source and destination addressing.

Three functional units provide 24-bit results to the A registers only:

— integer add
— integer multiply
— population count

Three functional units provide 64-bit results to the S registers only:

— integer add
— shift
— logical

Three functional units provide 64-bit results to the V registers only:

— integer add
— shift
— logical

Three functional units provide 64-bit results to either the S or V registers:

— floating add
— floating multiply
— reciprocal approximation

Integer arithmetic is performed in 2's complement mode. Floating point quantities have signed magnitude representation.

All functional units are fully segmented. This means that the information arriving at the unit, or moving within the unit, is captured and held in a new set of registers at the end of every clock period. It is therefore 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 the operands have been delivered to the unit. Functional units servicing the vector instructions produce one result per clock period.

Vector instructions

Vector instructions may be classified into four types. One type of vector instruction obtains operands from one or two V registers and enters results into another V register (Fig. 2a). Successive operand pairs are transmitted from Vj and Vk to the segmented functional unit each clock period and the corresponding results emerge from the function 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. The contents of the vector length (VL) register determines the number of operand pairs processed by the functional unit.

The second type of vector instruction obtains one operand from an S register and one from a V register (Fig. 2b). A copy of the S register is transmitted to the functional unit with each V-register operand.

The last two types of vector instruction transmit data between memory and the V registers (Fig. 2c and 2d). A path between memory and the V registers may be considered a functional unit for timing considerations.

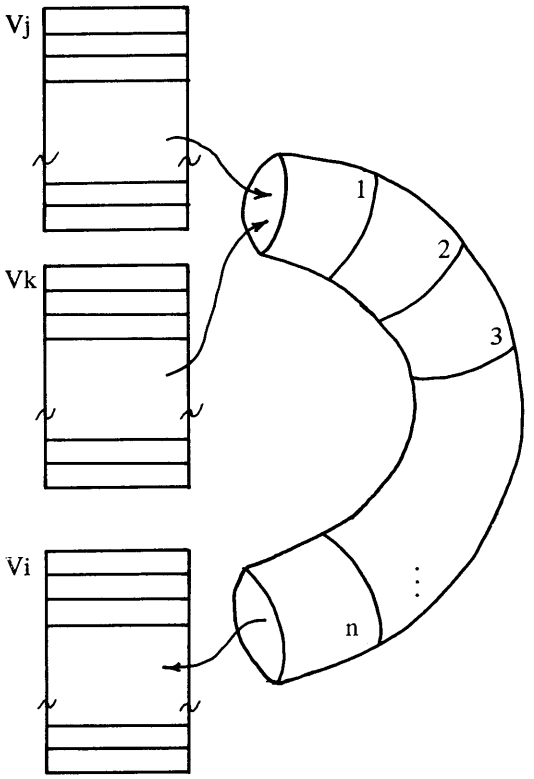


Figure 2a. Type I vector instruction

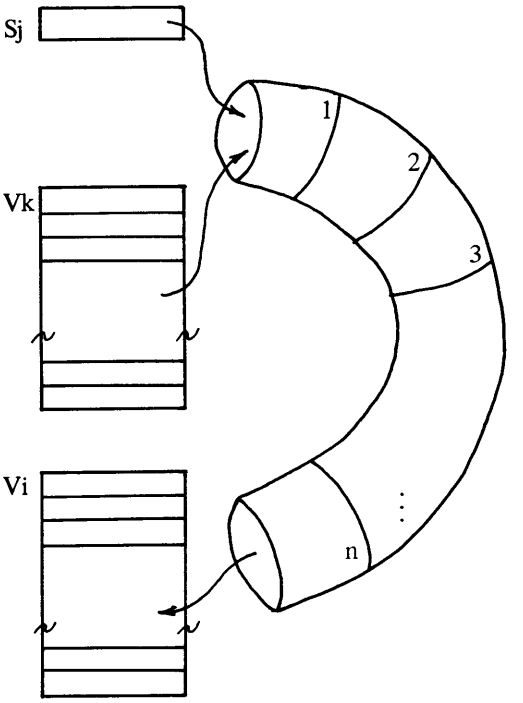


Figure 2b. Type II vector instruction

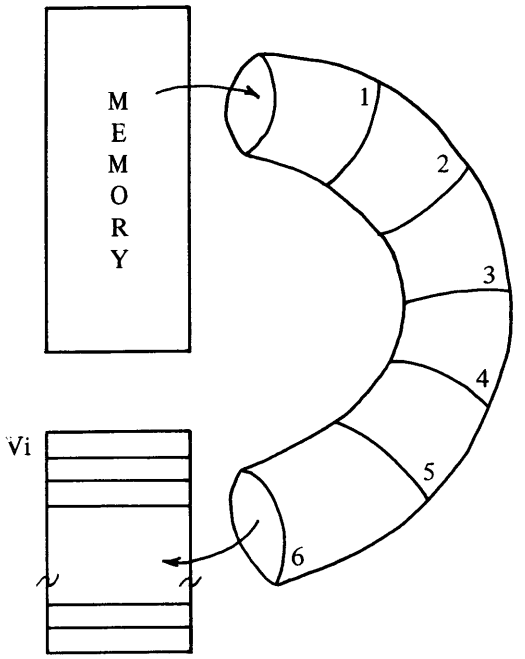


Figure 2c. Type III vector instruction

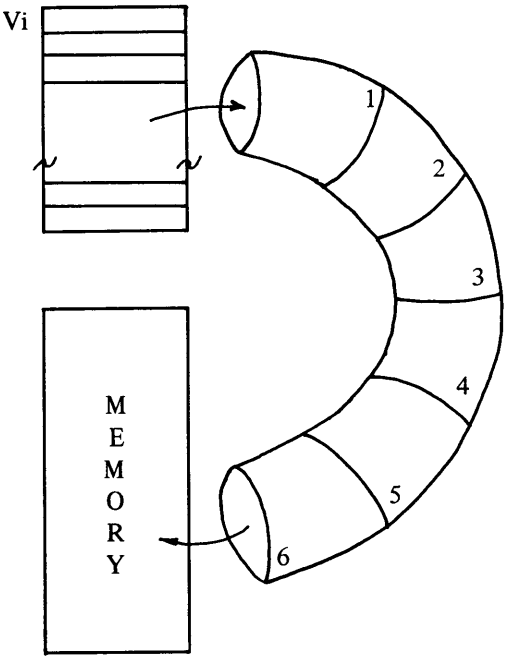


Figure 2d. Type IV vector instruction

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 sixty-four elements, the last element of V1 is $V1_{63}$. Figure 3 shows a timing chart 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. The dashed lines indicate transit to and from the functional unit. The functional unit time for this unit is six clock periods, so 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 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 the 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 which can be performed simultaneously by a functional unit is equal to the functional unit time.

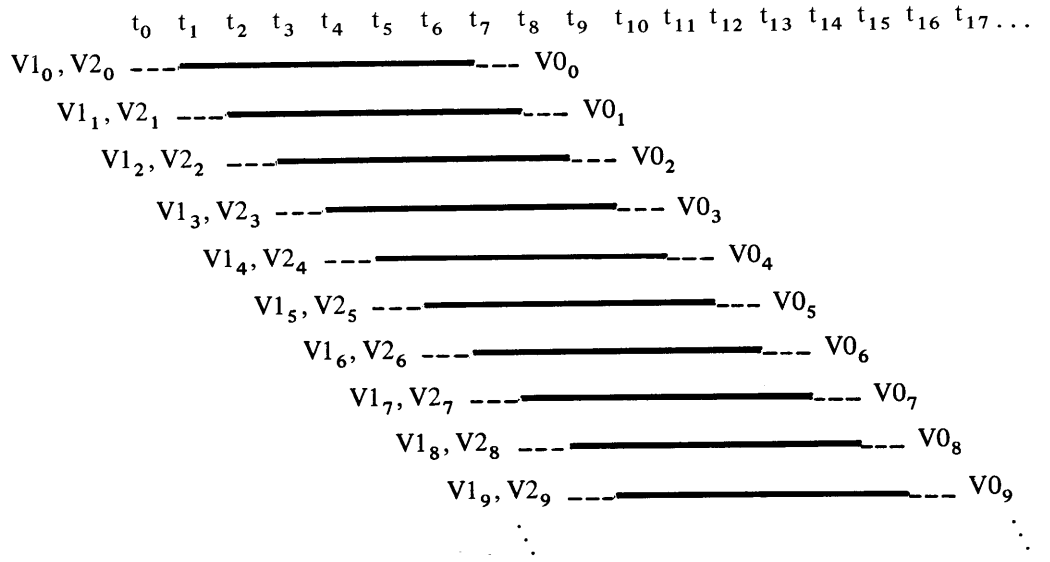


Figure 3. Vector instruction timing example (V0 ← V1 + V2)

The vector length determines the total number of operations performed by a functional unit. Although each vector register has sixty-four elements, only the number of elements specified by the vector length register is processed by a vector instruction. Vectors which have more than sixty-four 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 the operand registers are reserved for the number of clock periods determined by the vector length. A subsequent vector instruction which requires the same functional unit or operand register cannot issue until the reservations are released. When two vector instructions use different functional units and vector registers, they are independent and may issue in neighboring clock periods. Example 1a shows two independent instructions. Both execute concurrently with a one clock period difference in their issue times. Examples 1b-1d illustrate the effect of functional unit and operand register reservation when two instructions are not independent. Example 1b 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. Example 1c shows an add instruction followed by a multiply instruction. As in the previous example, 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 the operand register V1 is free. The second instruction in example 1d is delayed because of both functional unit and operand register reservations.

V0 ← V1 + V2
V3 ← V4 * V5

Ex. 1a. Independent instructions

V3 ← V1 + V2
V6 ← V4 + V5

Ex. 1b. Functional unit reservation

V3 ← V1 + V2
V6 ← V1 * V5

Ex. 1c. Operand register reservation

V0 ← V1 + V2
V3 ← V1 + V5

Ex. 1d. Functional unit and operand register reservation

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" and 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, then it must wait until the result register reservation is released. Figure 4a shows a chain of four instructions which read a vector of integers from memory, add that vector to another, shifting the sum, and finally forming the logical product of the shifted sum and a mask vector. The result of the four instructions is in vector register V5. Figure 4b graphically depicts the passage of information through the functional units. The diagram illustrates that the functional units may be considered links in a chain which works as a whole to produce the final result.

V0 ← memory	(memory read)	V3 ← V2<A3	(left shift)
V2 ← V0+V1	(integer add)	V5 ← V3&V4	(logical product)

Figure 4a. Chaining example

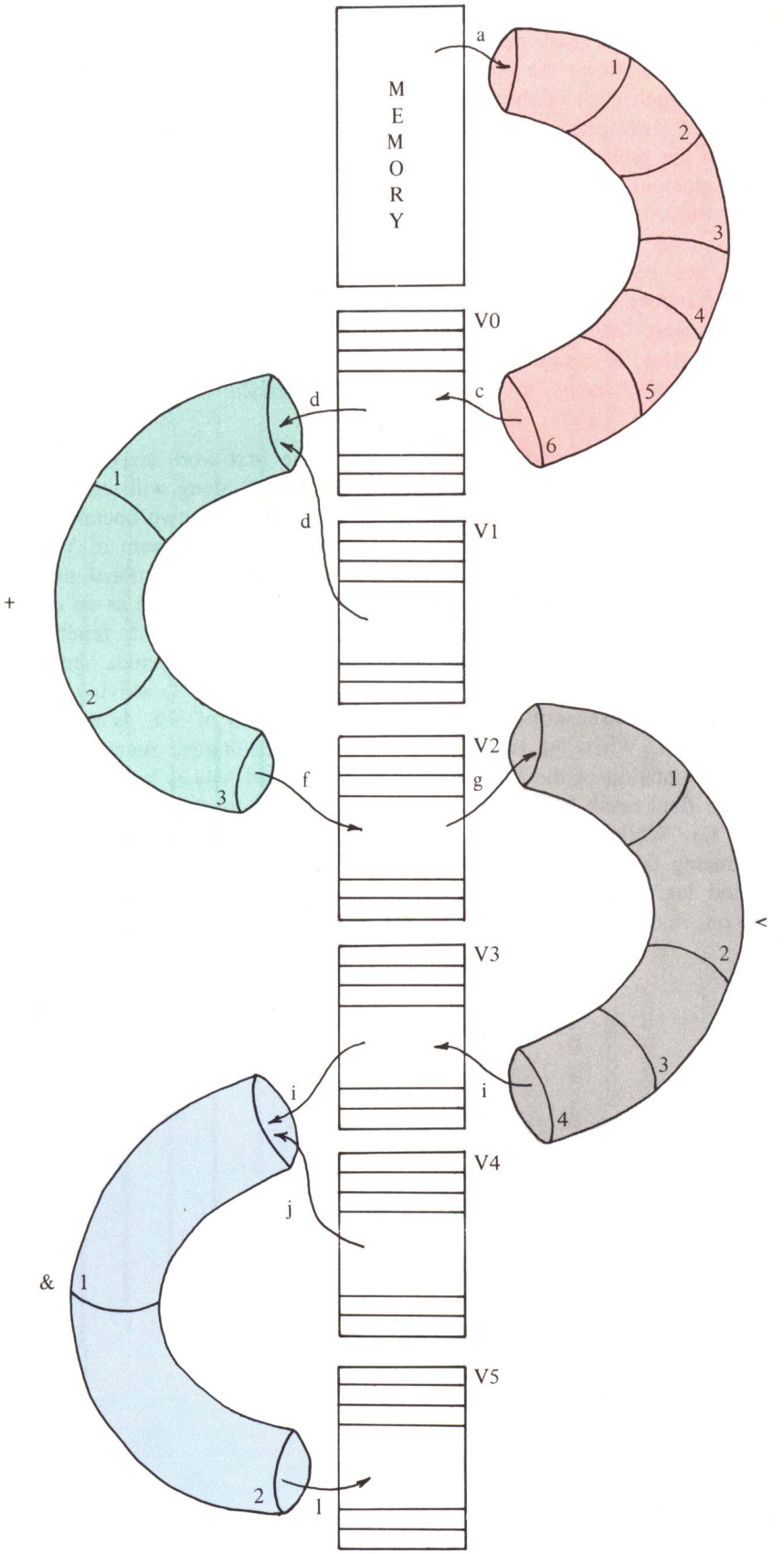
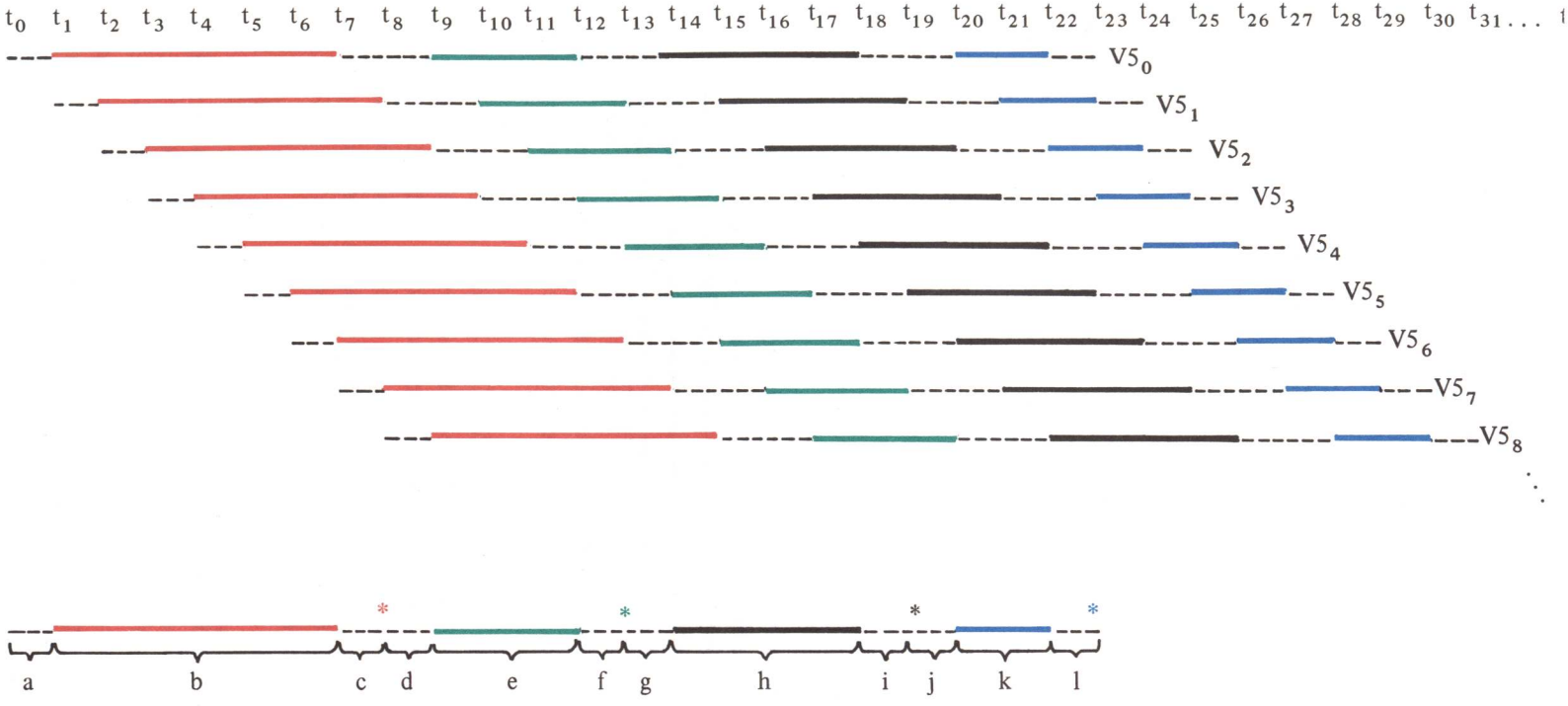


Figure 4b. Pictorial representation of chaining example

The timing diagram in Figure 4c will help clarify the concept of chaining. Gradations 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 is indicated by bars of corresponding length. The bars are color-coded to agree with Figures 4a and 4b. 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_{23} and a new result enters V5 each clock period thereafter. The first horizontal line, which shows the 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 marked by asterisks. A detailed description of the production of $V5_0$ may be illustrative; production of the 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 clock period t_8, 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_9, the computation of the sum of $V0_0$ and $V1_1$ is begun. Three clock periods later (t_{12}) the sum is sent from the functional unit to element zero of V2. It arrives at clock period t_{13} and is immediately transmitted as an operand to the shift functional unit. At clock period t_{14} the operand arrives at the shift functional unit and the shift operation is begun. The operation is completed four clock periods later (t_{18}) and the shifted sum is sent from the functional unit to element zero of V3, arriving the next clock period. It is immediately transmitted, along with element zero of V4, as an operand to the logical functional unit. When the two operands arrive at the logical functional unit at clock period t_{20}, the computation of the logical product of $V3_0$ and $V4_0$ is begun. Two clock periods later (t_{22}) the final result is sent from the functional unit to element zero of V5, arriving at clock period t_{23}. 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.



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

Figure 4c. Timing diagram for chaining example

Vector loops

Long vectors are processed in segments since the vector registers of the CRAY-1 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 twelve independent functional units and chaining to read the current vector segments from memory, execute the required functions and return the results to memory. Loop control is performed in the scalar registers concurrently with vector processing. Loop branch time is hidden by the vector operations.

The following simple FORTRAN loop can be used to illustrate processing of long vectors:

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

When N is 64 or less, all elements of the A array can be assigned a value with a sequence of seven instructions:

1.	S1 ← 5.	set constants	
2.	S2 ← 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 4-6 use different functional units (''memory'', multiply and add, respectively) and so they can be chained. When the V2 result register is free, the 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 the first pass through the loop. The loop performs instructions 4-7 for a segment of the A and B arrays. The last vector operation at the bottom of the loop is to store 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.

The concept of chaining and the use of register-to-register vector instructions are unique to the CRAY-1. The design minimizes the problem of speed degradation associated with memory-to-memory vector instructions. Start-up time for vector operations in the CRAY-1 are nominal; the benefits of vector processing are visible even for short vectors. The extensive concurrency attainable through the use of the 12 fully-segmented functional units can produce impressive results. A performance study of several subroutines for the CRAY-1 FORTRAN library and matrix multiplication illustrates this fact.

Fortran library

Each scalar FORTRAN library subroutine has a vector analog which employs the same algorithm in vector mode to produce several results at a time. The scalar subroutines must be called for each desired result, while the vector subroutines process an argument vector to obtain a vector of results. Performance studies on the CRAY-1 indicate that the vector subroutine outperforms its scalar counterpart whenever a vector of two or more results is required. Figure 5 depicts the behavior of the scalar and vector subroutines for several library functions. The cost of a result (in clock periods) is graphed as a function of vector length. The 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 the vector cost is less than the scalar cost when more than one result is produced.

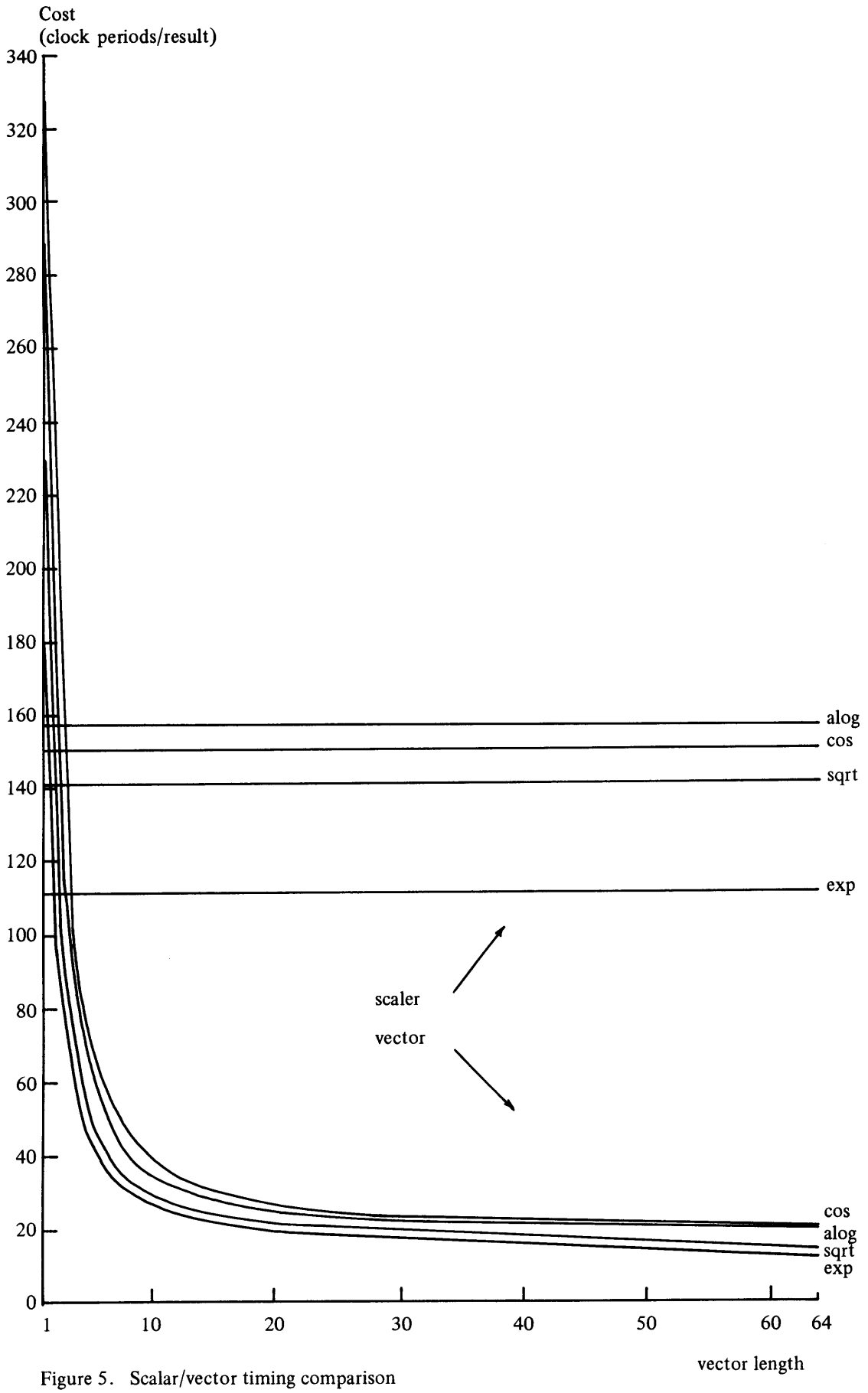


Figure 5. Scalar/vector timing comparison

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 the 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. Figure 6 shows the CRAY-1 execution rate for multiplication of square matrices as a function of matrix dimension. The 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 $n^2 \cdot (2n-1)$, since each of the n^2 elements of the result matrix is formed by summing n products.

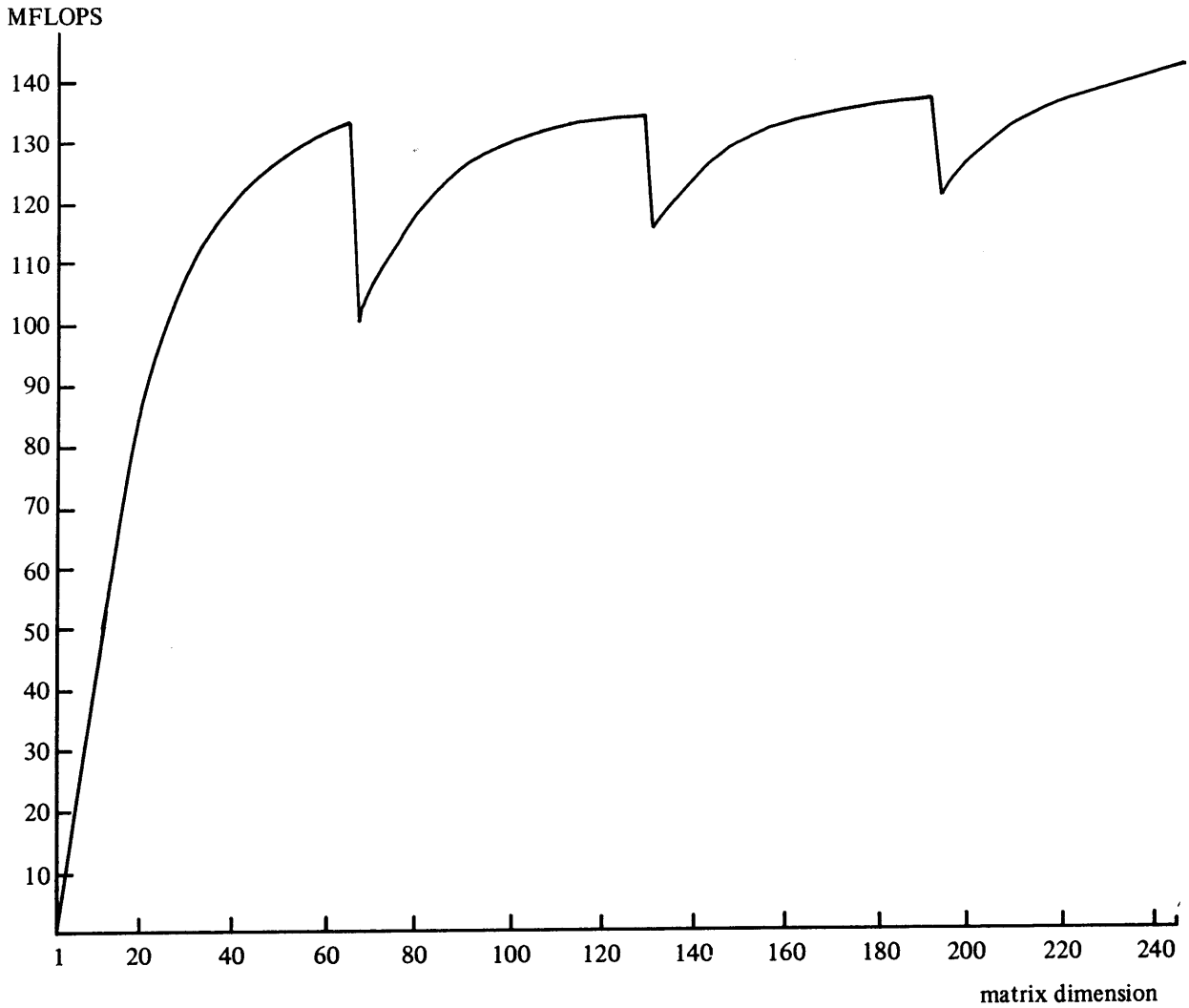


Figure 6. Matrix multiplication timing

Matrix multiplication is typical of the large class of problems which 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 twelve independent functional units and chaining can provide processing speeds exceeding those of other existing vector machines. Fields such as weather forecasting, nuclear research and seismic analysis will find the CRAY-1 a welcome extension to their workshops of computing tools.