Pipeline processor

Unit 2

Principles of linear pipelining

computers. The concept of pipeline processing in a computer is similar to assembly lines in an industrial plant. To achieve pipelining, one must subdivide the input task (process) into a sequence of subtasks, each of which can be executed by a specialized hardware stage that operates concurrently with other stages in the pipeline. Successive tasks are streamed into the pipe and get executed in an overlapped fashion at the subtask level. The subdivision of labor in assembly lines

A *linear pipeline* can process a succession of subtasks with a linear precedence graph.

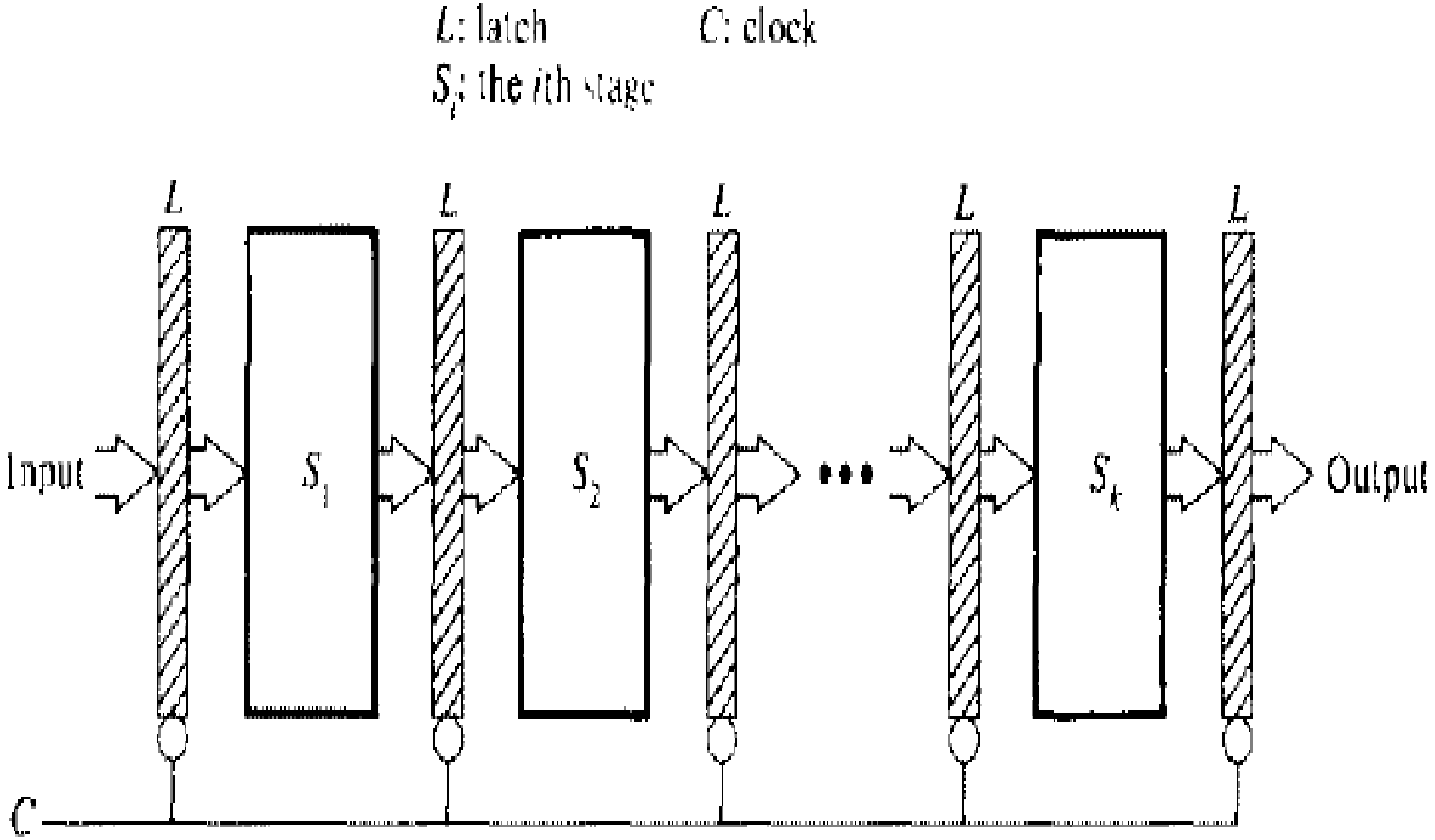
A basic linear-pipeline processor is depicted in Figure 3.1*a*. The pipeline consists of a cascade of processing stages. The *stages* are pure combinational circuits performing arithmetic or logic operations over the data stream flowing through the pipe. The stages are separated by high-speed interface *latches*. The latches are fast registers for holding the intermediate results between the stages. Information flows between adjacent stages are under the control of a common clock applied to all the latches simultaneously.

Clock period The logic circuitry in each stage S_i has a time delay denoted by τ_i. Let τ_l be the time delay of each interface latch. The *clock period* of a linear pipeline is defined by

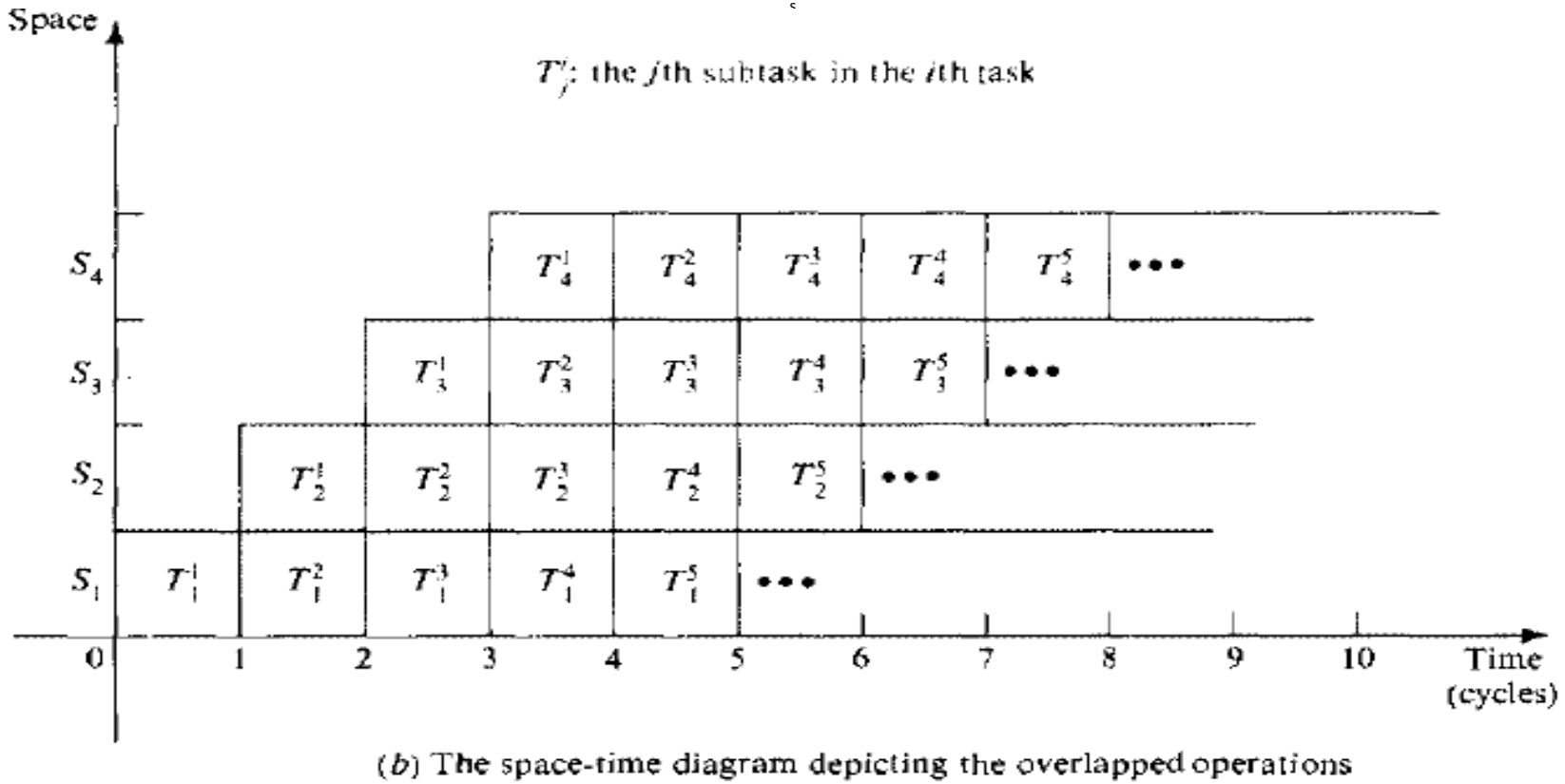
$$\tau = \max\{\tau_i\}_1^k + \tau_l = \tau_m + \tau_l \qquad (3.1)$$

The reciprocal of the clock period is called the *frequency* $f = 1/\tau$ of a pipeline processor.

s



(*a*) Basic structure of a linear pipeline processor



(*b*) The space-time diagram depicting the overlapped operations

Figure 3.1 Linear pipeline processor for overlapped processing of multiple tasks.

One can draw a *space-time diagram* to illustrate the overlapped operations in a linear pipeline processor. The space-time diagram of a four-stage pipeline processor is demonstrated in Figure 3.1*b*. Once the pipe is filled up, it will output one result per clock period independent of the number of stages in the pipe. Ideally, a linear pipeline with k stages can process n tasks in $T_k = k + (n - 1)$ clock periods, where k cycles are used to fill up the pipeline or to complete execution of the first task and $n - 1$ cycles are needed to complete the remaining $n - 1$ tasks. The same

The central processing unit (CPU) of a modern digital computer can generally be partitioned into three sections: the *instruction unit*, the *instruction queue*, and the *execution unit*. From the operational point of view, all three units are pipelined, as illustrated in Figure 3.3. Programs and data reside in the main memory, which usually consists of interleaved memory modules. The cache is a faster storage of copies of programs and data which are ready for execution. The cache is used to close up the speed gap between main memory and the CPU.

The instruction unit consists of pipeline stages for instruction fetch, instruction decode, operand address calculation, and operand fetches (if needed). The instruction queue is a first-in, first-out (FIFO) storage area for decoded instructions and fetched operands. The execution unit may contain multiple functional pipelines for arithmetic logic functions. While the instruction unit is fetching instruction

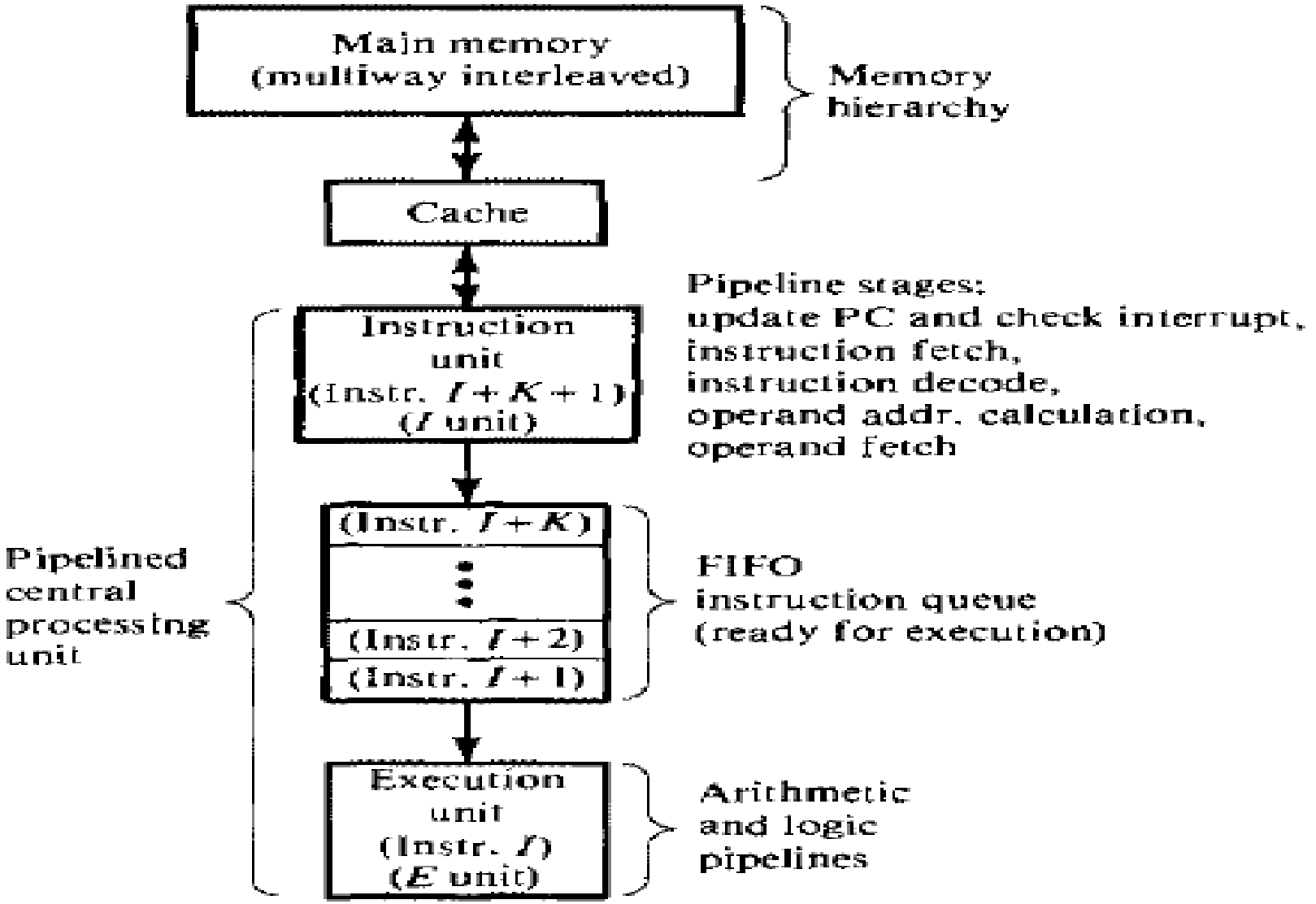


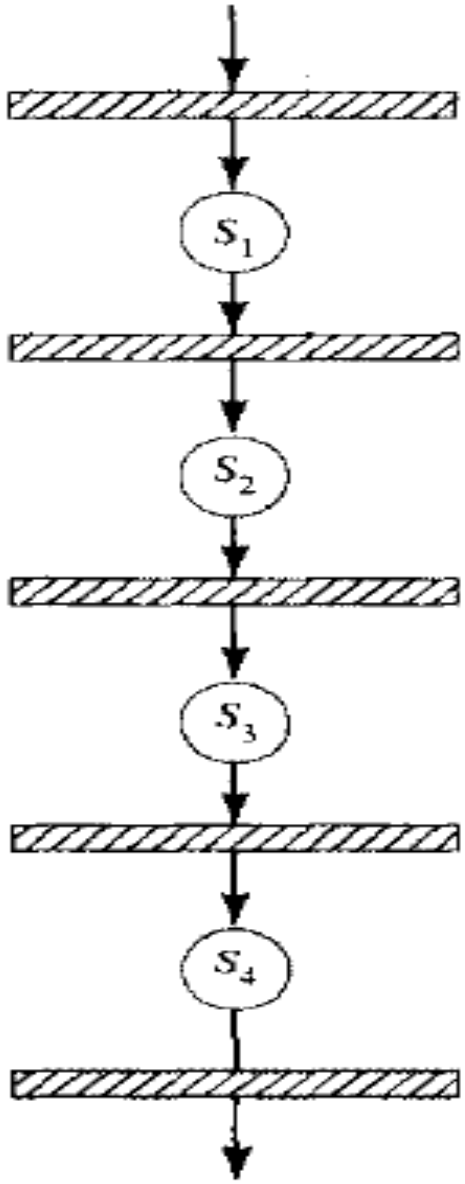
Figure 3.3 The pipelined structure of a typical central processing unit.

Classification of Pipeline processor

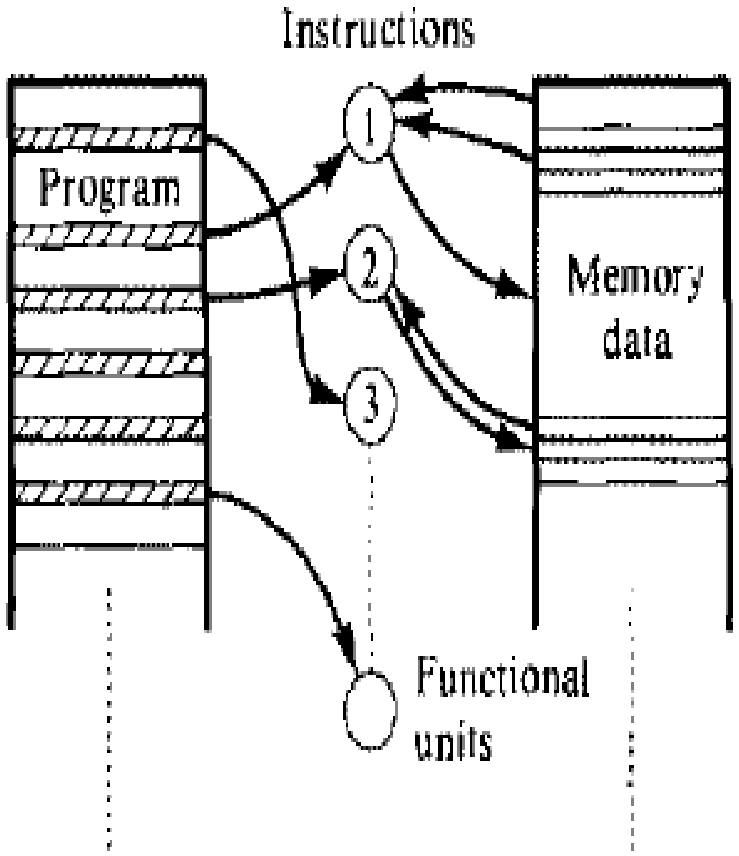
Arithmetic pipelining The arithmetic logic units of a computer can be segmentized for pipeline operations in various data formats (Figure 3.4*a*). Well-known arithmetic pipeline examples are the four-stage pipes used in Star-100, the eight-stage pipes used in the TI-ASC, the up to 14 pipeline stages used in the Cray-1, and the up to 26 stages per pipe in the Cyber-205. These arithmetic logic pipeline designs will be studied subsequently.

Instruction pipelining The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode, and operand fetch of subsequent instructions (Figure 3.4*b*). This technique is also known as *instruction lookahead*. Almost all high-performance computers are now equipped with instruction-execution pipelines.

s

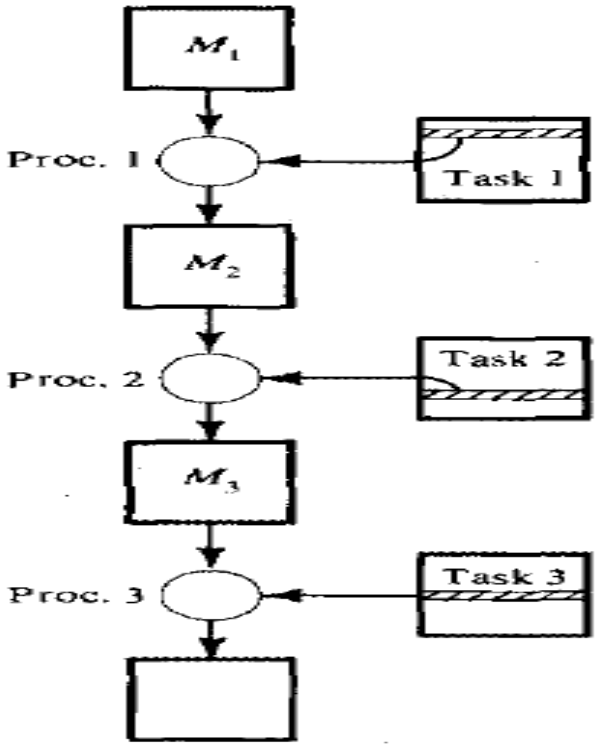


(a) Arithmetic pipelining



(b) Instruction pipelining

Processor pipelining This refers to the pipeline processing of the same data stream by a cascade of processors (Figure 3.4c), each of which processes a specific task. The data stream passes the first processor with results stored in a memory block which is also accessible by the second processor. The second processor then passes the refined results to the third, and so on. The pipelining of multiple processors is not yet well accepted as a common practice.



(c) Processor pipelining

s

According to pipeline configurations and control strategies, Ramamoorthy and Li (1977) have proposed the following three pipeline classification schemes:

Unifunction vs. multifunction pipelines A pipeline unit with a fixed and dedicated function, such as the floating-point adder in Figure 3.2, is called *unifunctional*. The Cray-1 has 12 unifunctional pipeline units for various scalar, vector, fixed-point, and floating-point operations. A *multifunction* pipe may perform different functions, either at different times or at the same time, by interconnecting different subsets of stages in the pipeline. The TI-ASC has four multifunction pipeline

Static vs. dynamic pipelines A *static pipeline* may assume only one functional configuration at a time. Static pipelines can be either unifunctional or multifunctional. Pipelining is made possible in static pipes only if instructions of the same type are to be executed continuously. The function performed by a static pipeline should not change frequently. Otherwise, its performance may be very low. A *dynamic pipeline* processor permits several functional configurations to exist simultaneously. In this sense, a dynamic pipeline must be multifunctional.

s

Scalar vs. vector pipelines Depending on the instruction or data types, pipeline processors can be also classified as scalar pipelines and vector pipelines. A *scalar pipeline* processes a sequence of scalar operands under the control of a DO loop. Instructions in a small DO loop are often prefetched into the instruction buffer. The required scalar operands for repeated scalar instructions are moved into a data cache in order to continuously supply the pipeline with operands. The IBM

Vector pipelines are specially designed to handle vector instructions over vector operands. Computers having vector instructions are often called *vector processors*. The design of a vector pipeline is expanded from that of a scalar pipeline. The handling of vector operands in vector pipelines is under firmware and hardware controls (rather than under software control as in scalar pipelines). Pipeline

Instruction pipeline design

- Instruction Pipeline: IBM 360/91
- IBM 360/91 incorporates high degree of pipelining with instruction preprocessing and instruction execution.
- It is a 32-bit machine designed for scientific computation in fixed and floating point data format.
- Multiple pipeline functional units are built into the system to allow parallel arithmetic computations.

CPU in IBM 360/91 consist of 4 major parts:

1]main storage control unit
2]instruction unit
3]fixed point execution unit
4]floating point execution unit

Block diagram

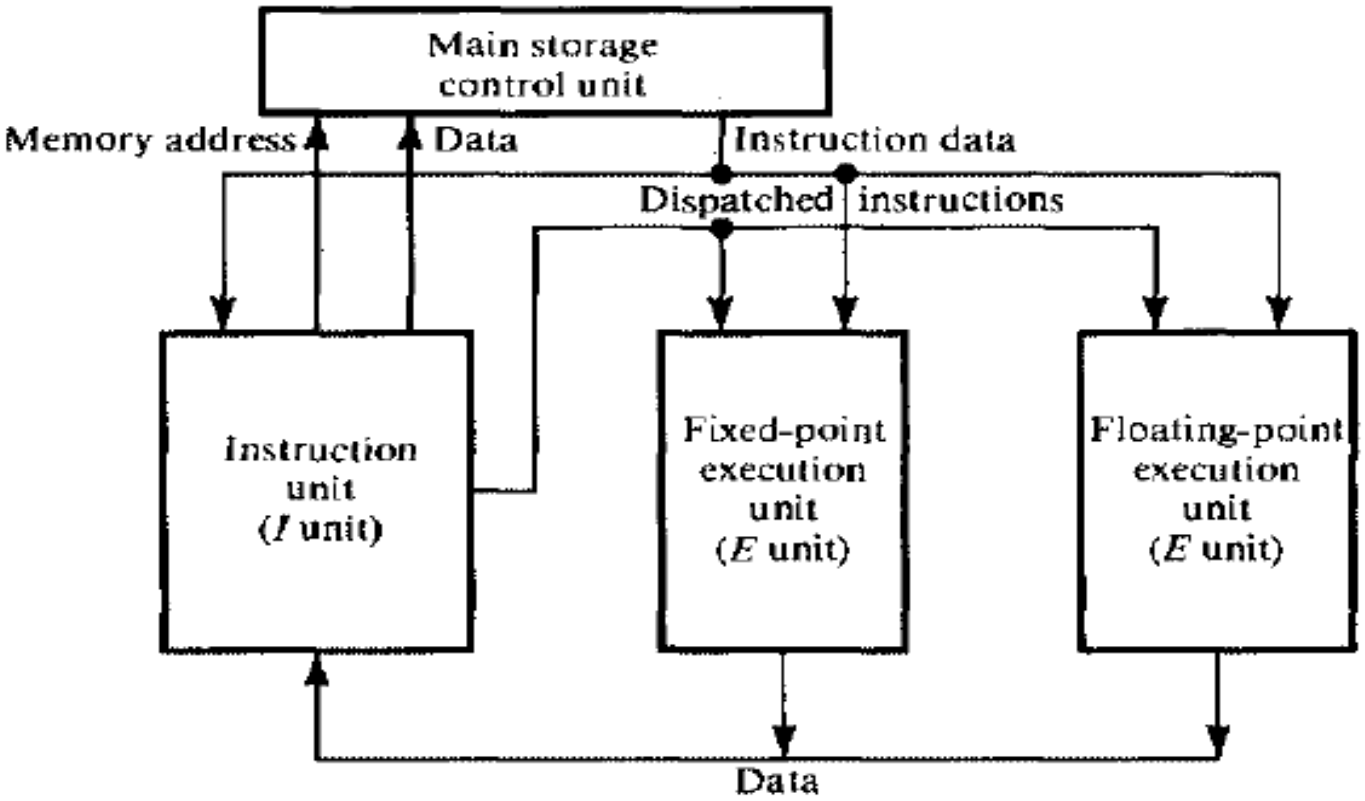


Figure 3.11 The central processing unit (CPU) of IBM System 360/Model 91.

- Instruction unit(I unit) is pipelined with a clock period of 60ns.
- This CPU is designed to issue instructions at a rate of **one instruction /clock cycle.**
- **Storage control unit(SCU)-**supervises information exchange between CPU and main memory major functions of the I-unit including fetch, decode and delivery to appropriate E unit, operand add calculation and operand fetch.
- The two E-units are responsible for the fixed point and floating point arithmetic logic operations needed in execution phase.

From memory access to instruction decode and execution, the CPU is fully pipelined across the four units shown in Figure 3.11. Concurrency among successive instructions in the Model 91 is illustrated in Figure 3.12. It is desirable to overlay separate instruction functions to the greatest possible degree. The shaded boxes correspond to circuit functions and the thin lines between them refer to delays caused by memory access. Obviously, memory accesses for fetching either instructions or operands take much longer time than the delays of functional circuitry. Following the delay caused by the initial filling of the pipeline, the execution results will begin emerging at the rate of one per 60 ns.

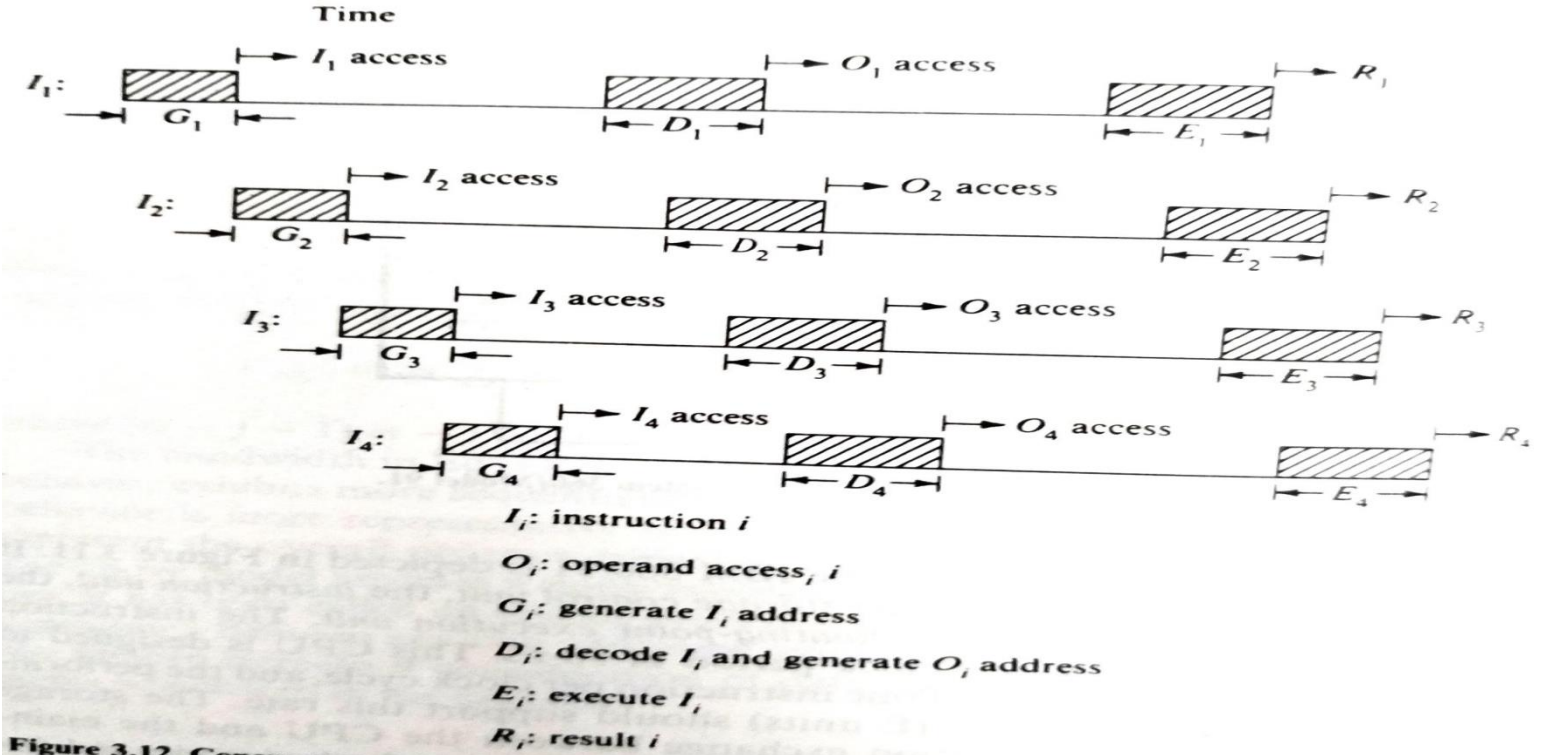


Figure 3.12 Concurrency among successive instruction fetch-decode-execute in the IBM 360/91.

For floating point storage to register instruction—functional segmentation of the pipeline along with the clock time divisions.

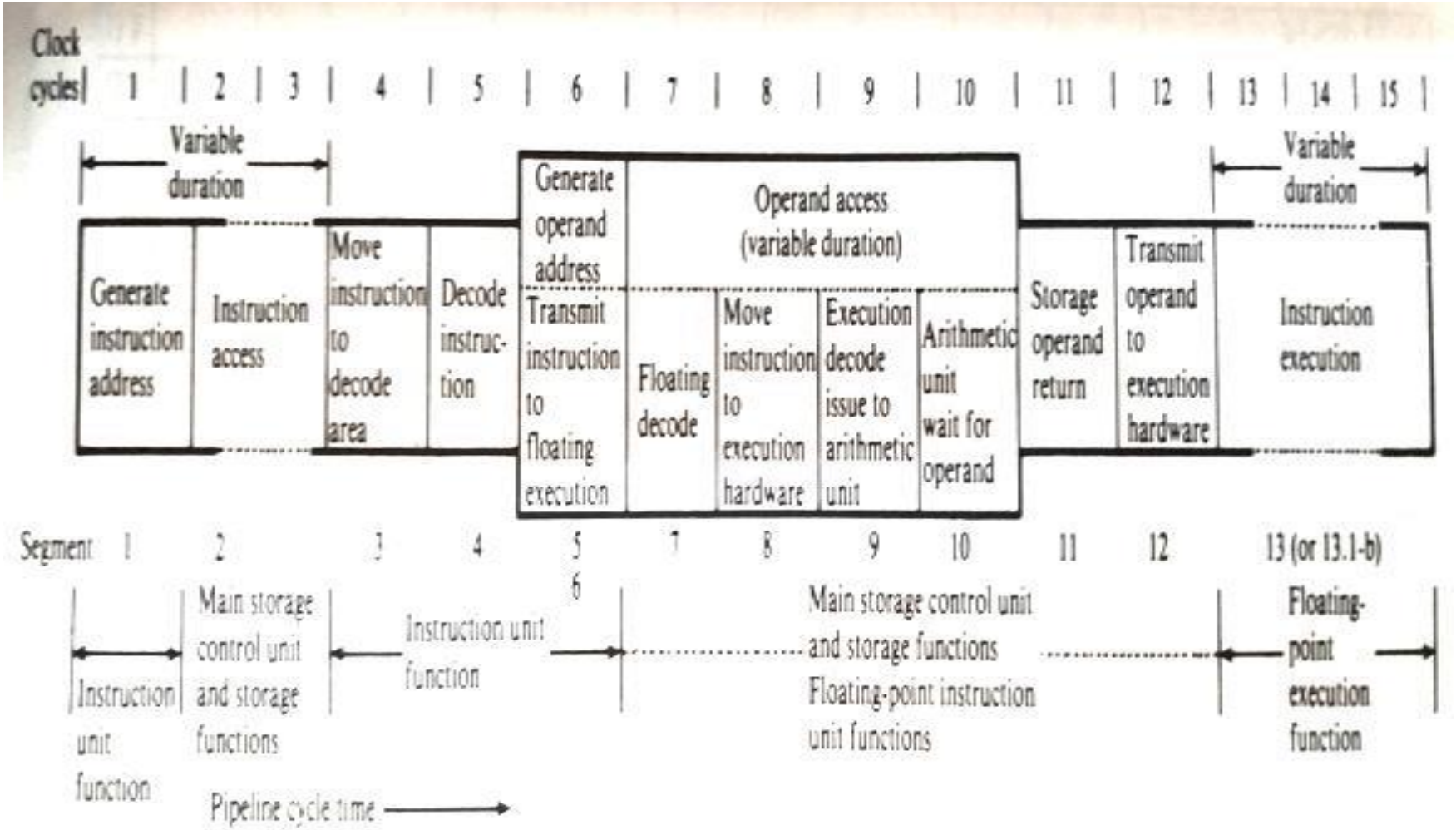


Figure 3.13 Functional segmentation of a typical storage-to-register floating-point instruction in the IBM 360/91.

s

- Memory and many execution functions require variable no of pipeline cycles.
- In model 91 by providing two separate units for fixed and floating point execution , instructions of two classes to be executed in parallel–Multiply and divide can operate in parallel.
- Time disparities between various instruction types, the model 91 utilizes the organizational techniques of memory interleaving , parallel arithmetic functions, data buffering and internal forwarding to overcome the speed gap problem.
- Interleaving is function of the memory cycle time, CPU storage request rate and desired effective access time.
- Performance of pipeline processor relies on continuous supply of instruction and data to the pipeline

Vector processing concept

3.4.1 Characteristics of Vector Processing

A vector operand contains an ordered set of n elements, where n is called the *length* of the vector. Each element in a vector is a scalar quantity, which may be a floating-point number, an integer, a logical value, or a character (byte). Vector instructions can be classified into four primitive types:

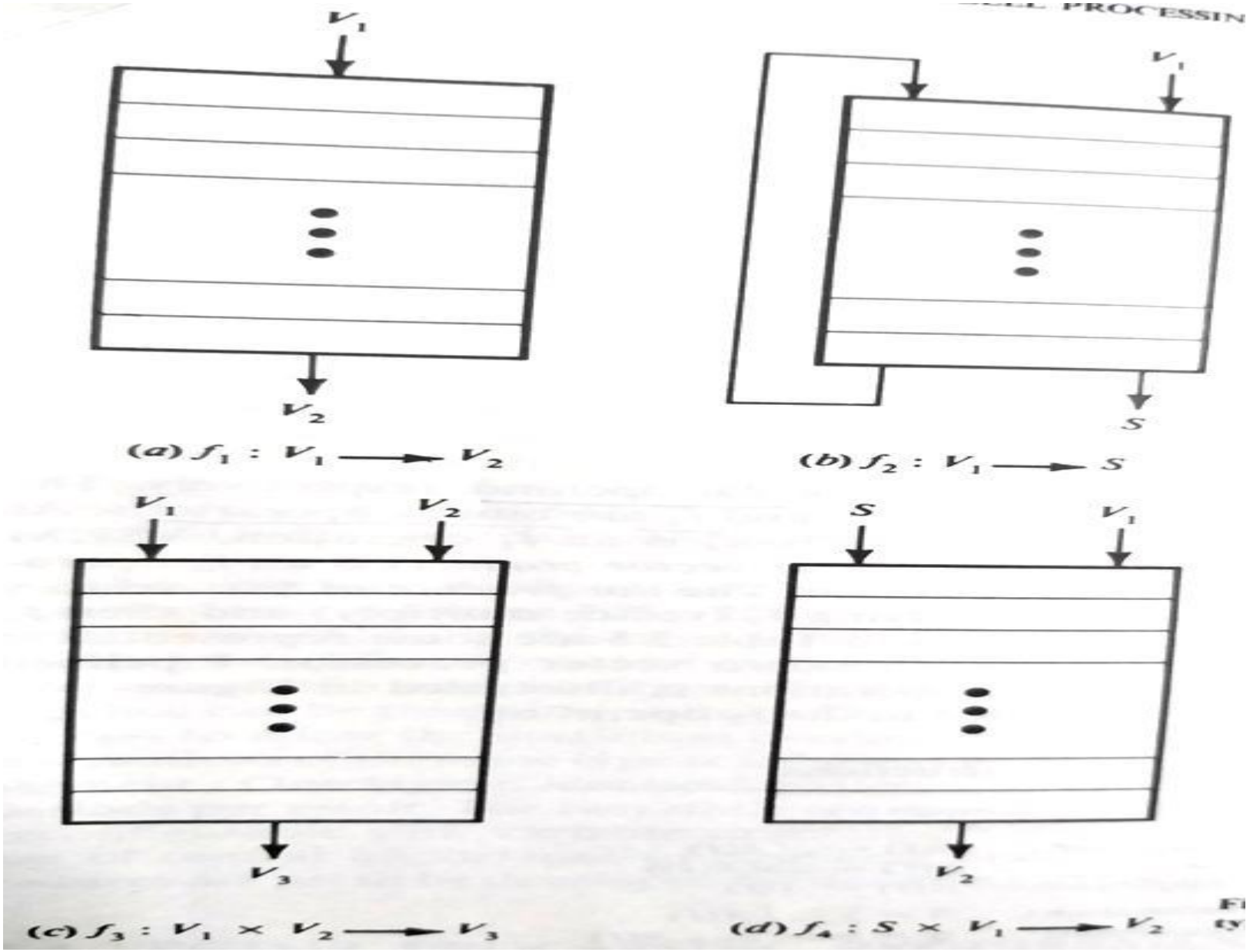
$$\begin{aligned} f_1 &: V \rightarrow V \\ f_2 &: V \rightarrow S \\ f_3 &: V \times V \rightarrow V \\ f_4 &: V \times S \rightarrow V \end{aligned} \tag{3.20}$$

where V and S denote a vector operand and a scalar operand, respectively. The mappings f_1 and f_2 are unary operations and f_3 and f_4 are binary operations. As shown in Table 3.5, the VSQR (*vector square root*) is an f_1 operation, VSUM (*vector summation*) is an f_2 operation, SVP (*scalar-vector product*) is an f_4 operation, and VADD (*vector add*) is an f_3 operation. The *dot product* of two vectors

Table 3.5 Some representative vector instructions

Type	Mnemonic	Description (I = 1 through N)	
f_1	VSQR	Vector square root:	$B(I) \leftarrow \sqrt{A(I)}$
	VSIN	Vector sine:	$B(I) \leftarrow \sin(A(I))$
	VCOM	Vector complement:	$A(I) \leftarrow \overline{A(I)}$
f_2	VSUM	Vector summation:	$S = \sum_{I=1}^{N} A(I)$
	VMAX	Vector maximum:	$S = \max_{I=1,N} A(I)$
f_3	VADD	Vector add:	$C(I) = A(I) + B(I)$
	VMPY	Vector multiply:	$C(I) = A(I) * B(I)$
	VAND	Vector and:	$C(I) = A(I)$ and $B(I)$
	VLAR	Vector larger:	$C(I) = \max(A(I), B(I))$
	VTGE	Vector test $>$:	$C(I) = 0$ if $A(I) < B(I)$
			$C(I) = 1$ if $A(I) > B(I)$
f_4	SADD	Vector-scalar add:	$B(I) = S + A(I)$
	SDIV	Vector-scalar divide:	$B(I) = A(I)/S$

Four vector instruction types for pipelined processor



Some special instructions may be used to facilitate the manipulation of vector data. A *boolean vector* can be generated as a result of comparing two vectors, and can be used as a *masking vector* for enabling or disabling component operations in a vector instruction. A *compress* instruction will shorten a vector under the control of a masking vector. A *merge* instruction combines two vectors under the control of a masking vector. Compress and merge are special f_1 and f_3 operations

Example 3.3 Let $X = (2, 5, 8, 7)$ and $Y = (9, 3, 6, 4)$. After the *compare* instruction $B = X > Y$ is executed, the boolean vector $B = (0, 1, 1, 1)$ is generated.

Let $X = (1, 2, 3, 4, 5, 6, 7, 8)$ and $B = (1, 0, 1, 0, 1, 0, 1, 0)$. After the execution of the *compress* instruction $Y = X(B)$, the compressed vector $Y = (1, 3, 5, 7)$ is generated.

Let $X = (1, 2, 4, 8)$, $Y = (3, 5, 6, 7)$, and $B = (1, 1, 0, 1, 0, 0, 0, 1)$. After the *merge* instruction $Z = X, Y, (B)$, the result is $Z = (1, 2, 3, 4, 5, 6, 7, 8)$. The first 1 in B indicates that $Z(1)$ is selected from the first element of X. Similarly, the first 0 in B indicates that $Z(3)$ is selected from the first element of Y.

- m/c operations suitable for pipelining have the following properties:

a)Identical processes are repeatedly invoked many times , each of which can be subdivided into subprocesses.

b)Successive operands are fed through pipeline segments and require as few buffers and local controls as possible

c)Operations executed by distinct pipelines should be able to share expensive resources such as memory and buses in the system.

- **Vector v/s scalar instructions**
- Vector instructions need to perform the same operation on different data sets repeatedly
- Elimination of the overhead caused by the loop control mechanism.
- Vector instructions perform better with longer vector

Vector instructions are specified with following fields:

The *operation code* must be specified in order to select the functional unit or to reconfigure a multifunctional unit to perform the specified operation.

-Memory reference instruction : base address

Address increment:
–Address offset:
–Vector length:
•Vector length register can be used to control the vector operation.
•Vector length affects the processing efficiency because of the additional overhead caused by subdividing a long vector.
•Vector length is needed to determine the termination of a vector instruction.

s

- Pipeline vector computers –architectural configuration
- –Memory to memory
- –Register to register
- To enhance vector processing capability, optimized object code must be produced to maximize the utilization of pipeline resources.
- –Following approaches suggested
- •Enrich the vector instruction set:
- •Combine scalar instruction:

–Group scalar instructions of the same type together.

•Choose suitable algorithm:

•Use a vectorizing compiler:

–to detect concurrency among instructions

Architecture of Vector processor

- Main memory is interleaved to minimize the access time of vector operands
- Instruction processing unit (IPU) fetches and decodes scalar and vector instructions.
- **Task system** —set of vector instructions (tasks) with precedence relation determined only by data dependencies.
- A long vector task can be partitioned into many subvectors, to be processed by several pipelines concurrently.

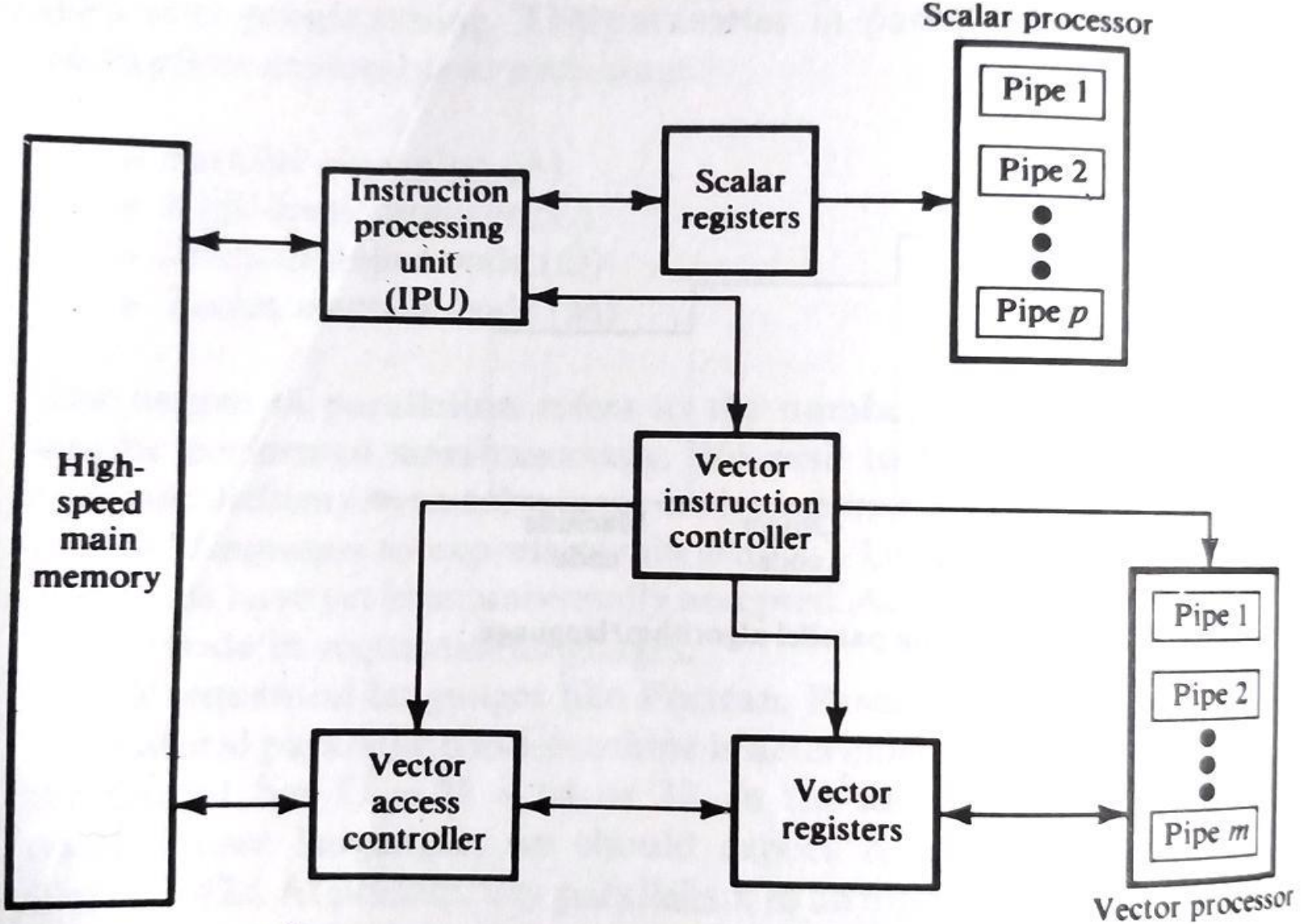


Figure 3.43 The architecture of a typical vector processor with multiple functional pipes.

After vector instruction is recognized by IPU

- Vector instruction controller:

–Supervising vector instruction execution

–Decoding vector instructions

–Calculating effective vector operand address

–Setting up Vector access controller and vector processor

–Monitoring execution of vector instructions

–Partition a vector task and schedule different instructions to different functional pipelines.

s

- Commercial vector processor

–Identical pipelines must execute the same vector instruction at the same time.

- Vector access controller:

–Is responsible for fetching vector operands by a series of main memory access.

- Vector registers:

–Used to close the speed gap between main memory and vector processor.

- m homogeneous vector pipelines in the vector processor , each of which is unior multi-functional.

s

- Vector instruction controller is capable of scheduling several vector instructions simultaneously .

$t_0+\tau$= time required to complete the execution

of single vector task

–t_0 is pipeline overhead time due to startup and flushing delays

- $\tau=t_1$.L---is the average latency between two successive operand pairs.

L= vector length

- Schedule the vector tasks among ‘m’ identical pipelines so total execution time is minimized.

- Assume equal overhead time t0for all vector tasks.

- vector tasks system can be characterized by

$V=(T, <, \tau)$

1.$T=\{T1,T2.....Tn\}$ ---set of n vector tasks

2.< ---is a ordering relation, specifying the precedence relationship among the tasks in set

3.τ ---time function defining production delay

s

3.4.3 Pipelined Vector Processing Methods

Vector computations are often involved in processing large arrays of data. By ordering successive computations in the array, we can classify vector (array) processing methods into three types:

1. *Horizontal processing*, in which vector computations are performed horizontally from left to right in row fashion
2. *Vertical processing*, in which vector computations are carried out vertically from top to bottom in column fashion
3. *Vector looping*, in which segmented vector loop computations are performed from left to right and top to bottom in a combined horizontal and vertical method

Cray type vector processor-

namely Cray Research's Cray-1, Control Data's Cyber-205, and Fujitsu's VP-200. All three are commercial supercomputers with multiple pipelines for concurrent scalar and vector processing. Possible extensions to these vector supercomputers

4.4.1 The Architecture of Cray-1

The Cray-1 has been available as the first modern vector processor since 1976. The architecture of Cray-1 consists of a number of working registers, large instruction buffers and data buffers, and 12 functional pipeline units. With the "chaining" of pipeline units, interim results are used immediately once they become available. The clock rate in the Cray-1 is 12.5 ns. The Cray-1 is not a "stand-alone" computer. A front-end host computer is required to serve as the system manager. A Data

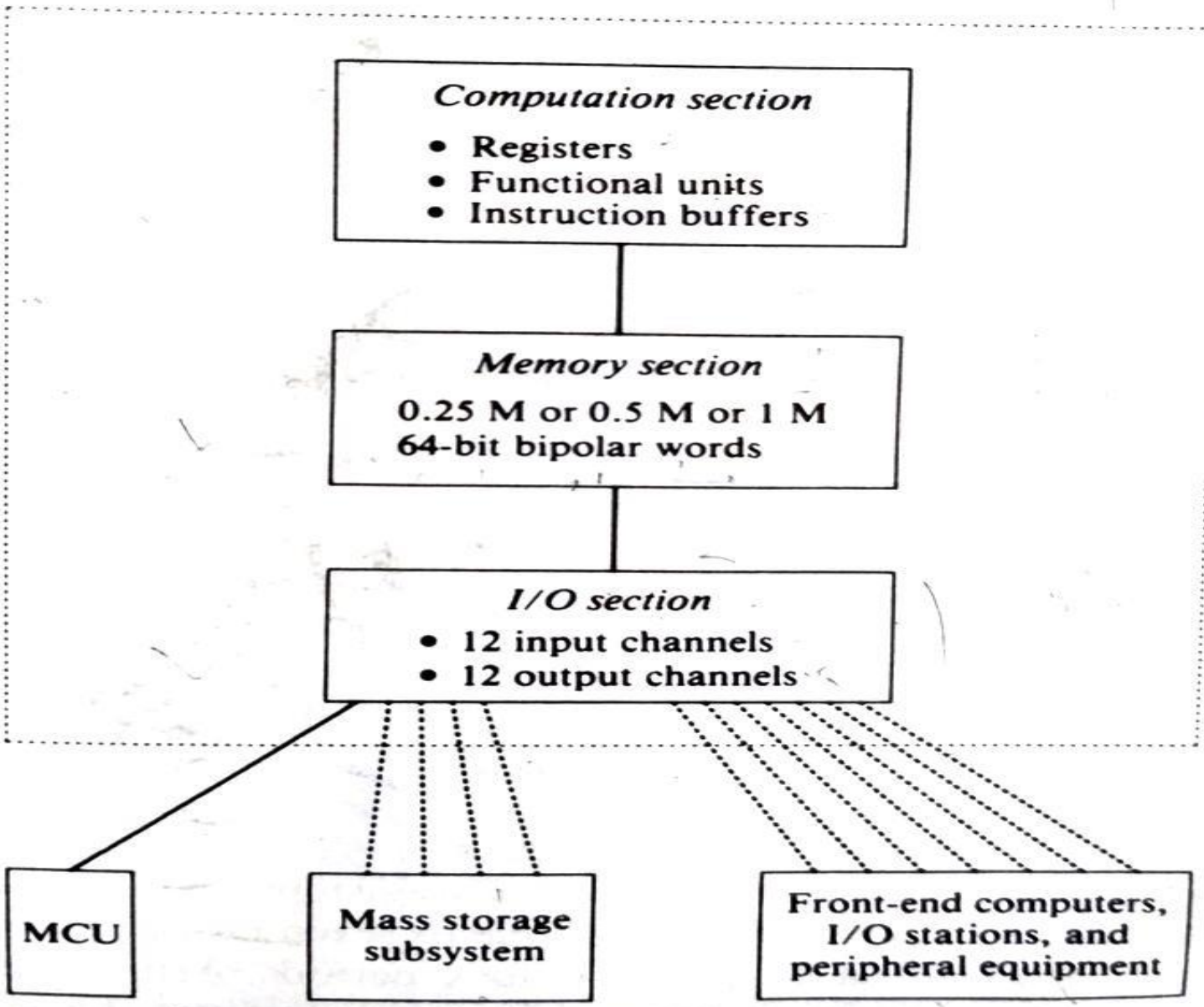


Figure 4.17 Front-end system interface and Cray-1 connections.

shows the front-end system interface and the Cray-1 memory and functional sections. The CPU contains a *computation section, a memory section*, and an *I/O section*. Twenty-four I/O channels are connected to the front-end computer, the I/O stations, peripheral equipment, the mass-storage subsystem, and a *maintenance control unit* (MCU). The front-end system will collect data, present it to the Cray-1 for processing, and receive output from the Cray-1 for distribution to slower devices. Table 4.9 summarizes the key characteristics of the three sections in the CPU of the Cray-1.

The memory section in the Cray-1 computer is organized in 8 or 16 banks with 72 modules per bank. Bipolar RAMs are used in the main memory with, at most, one million words of 72 bits each. Each memory module contributes 1 bit of a 72-bit word, out of which 8 bits are parity checks for *single error correction*

I/O section

- Consists of 12 input and 12 output channels
- Each channel has max transfer rate of 80 Mbytes/s
- Channels are grouped into six input and six output channel groups and served equally by all memory banks.
- One 64 bit word can be transferred per channel during each clock period.
- 4 input and 4 output channels operate simultaneously to achieve max transfer of instructions to computation section.

MCU: Maintenance control unit

- Handles system initiation and monitors system performance
- Mass storage system: provides large secondary storage in addition to the 1 million main memory words.

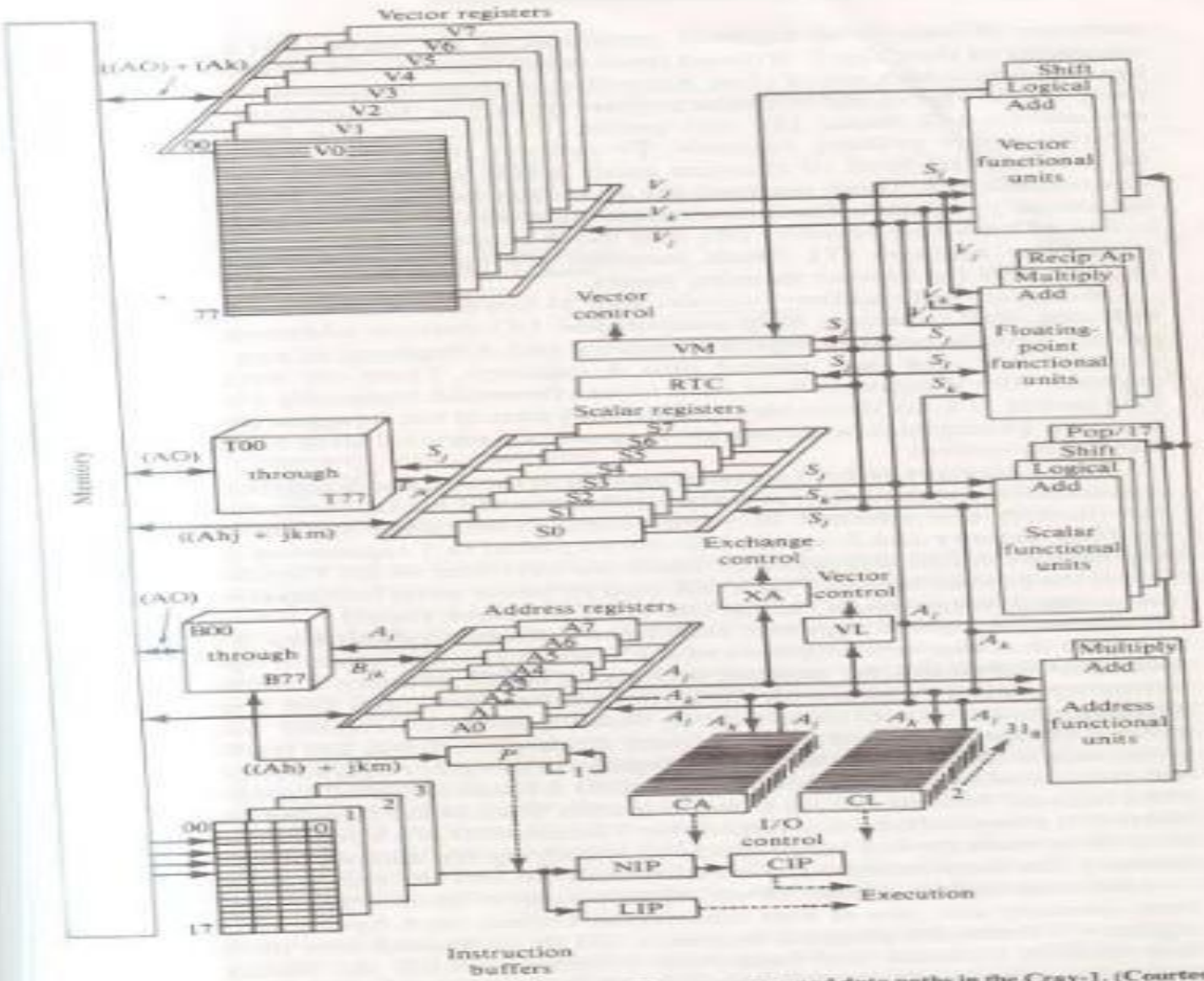


Figure 4.18 **Arithmetic logic pipelines, registers, buffers, memory, and data paths in the Cray-1. (Courtesy of Cray Research, Inc.)**

s

- It contains 64x4 instruction buffers
- 12 functional units are all pipelines with 1 to 7 clock delays.
- Arithmetic operations include 24bit integer and 64 bit floating point computations
- High speed registers for scalar and vector processing.

- Primary registers: s

–Address register(A)

–Scalar register(S)

–Vector register(V)

- Intermediate registers that supports address registers are **Address register(B)**
- And supports to scalar registers are **Scalar save registers(T)**

- Block transfer is possible between B and T registers

- **Address register(A):** s

8 address registers with 24 bits each

- Used for memory addressing , indexing, shift counting, loop control, and I/O channel addressing
- S registers:

Eight ,64 bit S registers serving for storage of source and destination operands of scalar arithmetic and logical instructions

- V registers:

Each has 64 component registers.

Group of data is stored in component registers of a V register to form a vector operand.

s

- All instructions are either 16 or 32 bit long.

- P register:

Is a 22 bit program counter indicating the next parcel of program code to enter the next instruction parcel(NIP)--holds a parcel of program code prior to entering the CIP register.

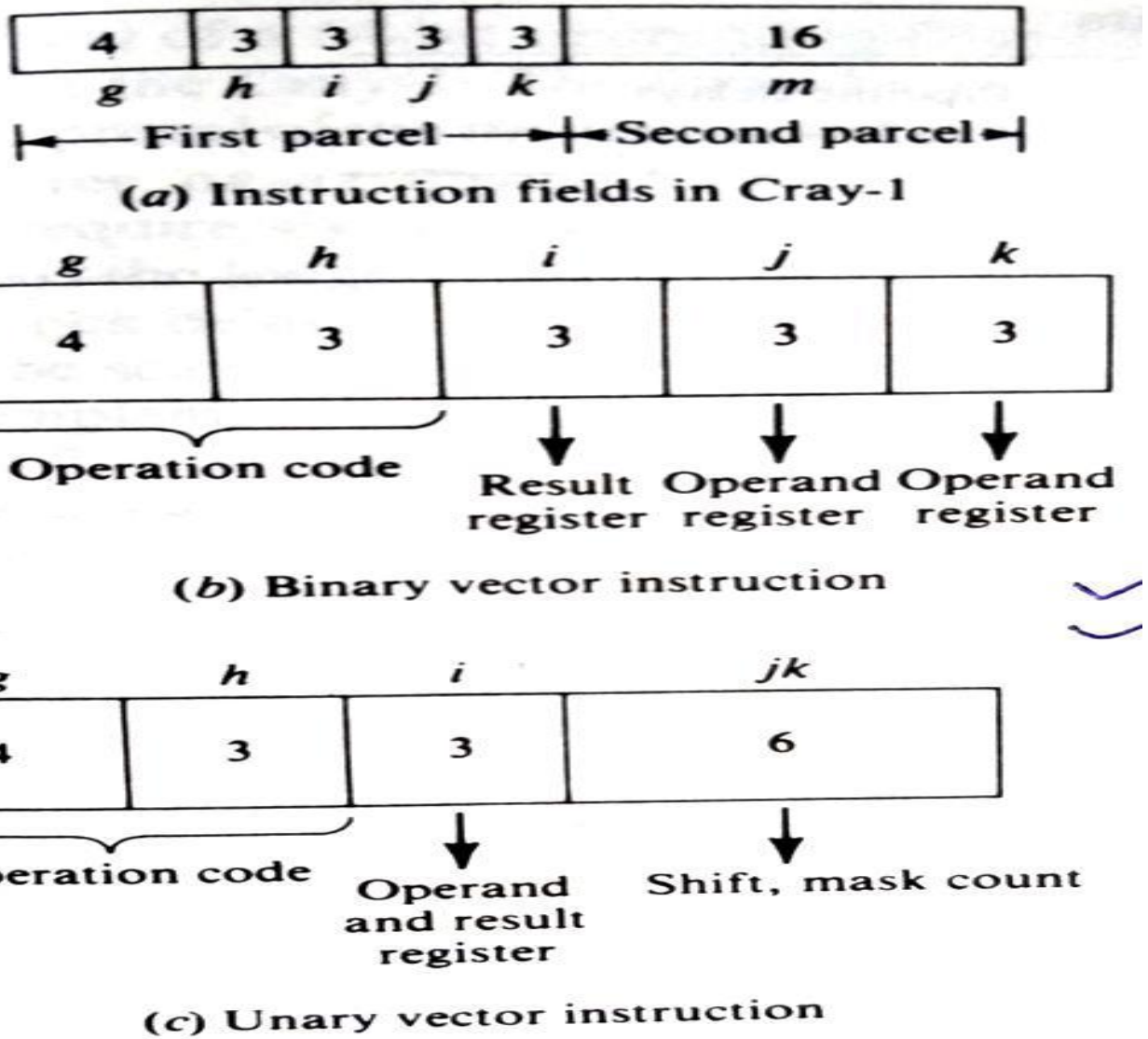
- **CIP :** current instruction parcel ---

holds lower half of the instruction

is a 16 bit register holding the instruction waiting to be issued.

- **LIP:** lower instruction parcel: holds lower half of the instruction

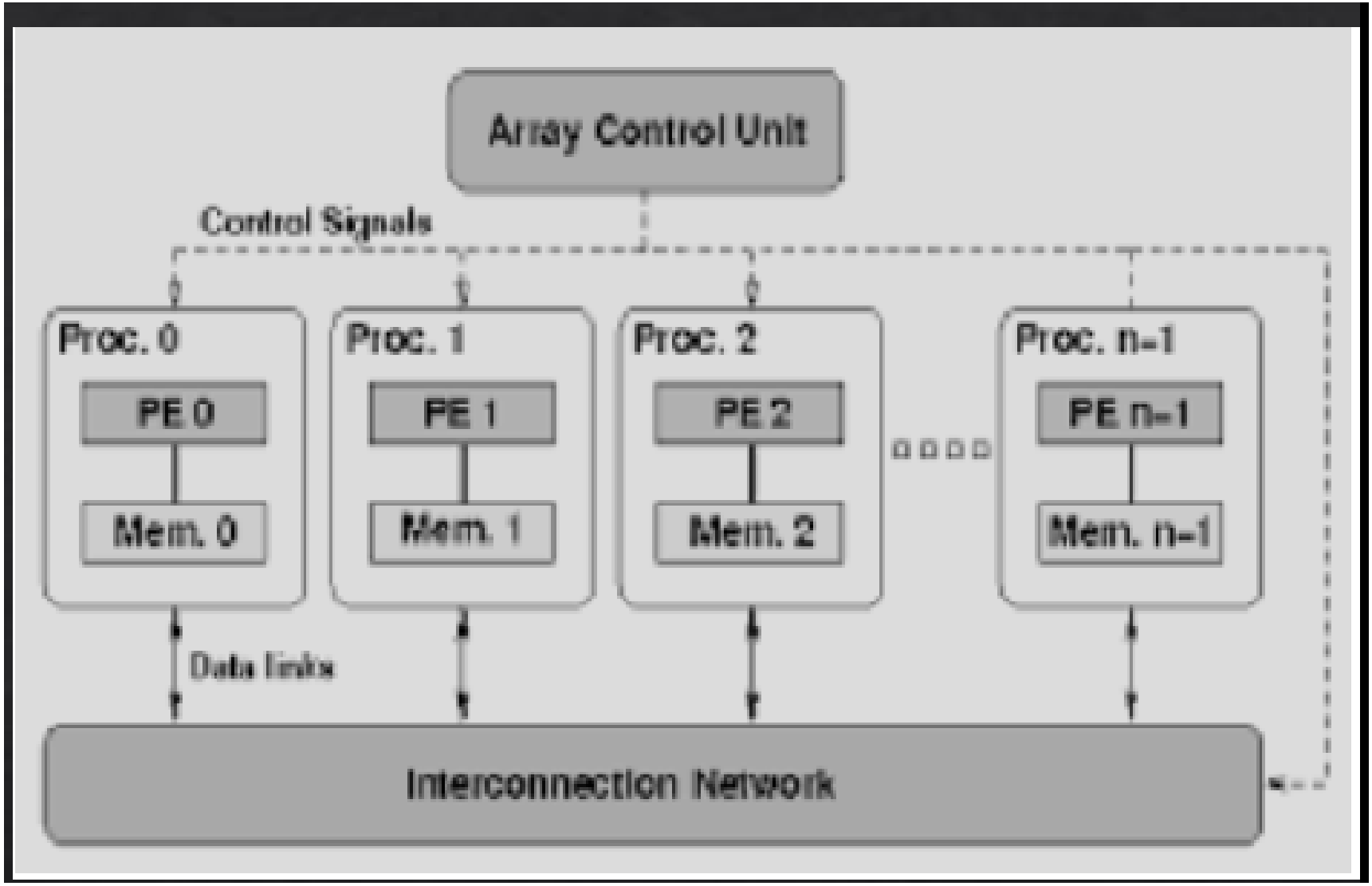
Instruction format of the Cray-1



Array processors

- Definition
- A synchronous array of parallel processors is called an array processor, which consists of multiple PE under the supervision of one control unit.
- Array processor is also known as SIMD computes
- It consists of N synchronized PE's all of which are under control of one CU.
- Each PE is ALU with attached working registers and local memory(PEM) for storage of distributed data.
- The system and user program are executed under the control of CU
- The user programs are loaded into CU memory from external source
- The function of CU is to decode all instructions
  - Scalar type instructions are directly executed inside the CU.
  - Vector type instructions are broadcast to all PE's for distributed execution to achieve spatial parallelism.

Array processors



Tutorial No. 2

Q1] Explain classification of pipeline processor?

Q2]Explain instruction pipeline design with example?

Q3]List characteristics of vector processing? Explain architecture of vector processor?

Q4]Write a note on Cray-1 architecture?

Unit No.3 Multithreading

- Introduction to associative memory processors