Marwadi
education foundation

COMPUTER ORGANIZATION

SUBJECT CODE: 2140707

Prepared by : Prof. Hetal Dave

KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY

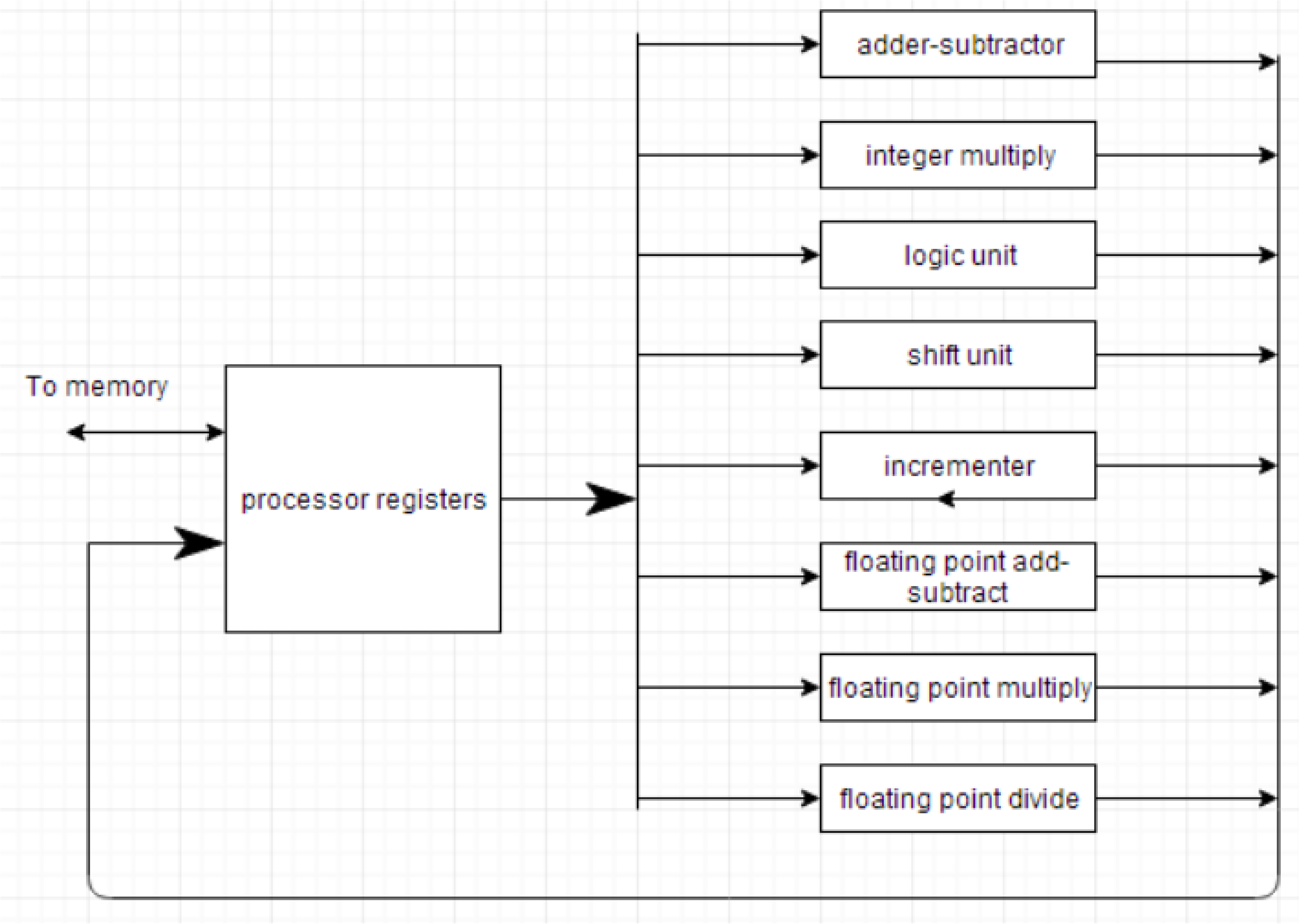
Chapter 6 - Pipeline And Vector Processing

- ✓ Parallel Processing
- ✓ Flynn's taxonomy
- ✓ Pipelining (Arithmetic Pipeline and Instruction Pipeline)
- ✓ RISC Pipeline
- ✓ Vector Processing
- ✓ Array Processors

Parallel Processing

- **Parallel processing** system provides **concurrent** data processing.
- It reduces the **execution time**.
- **Two or more ALU's** and should be **able to execute** two or more instructions at the same time.

Parallel Processing



Parallel Processing

- The execution unit is **separated into 8** functional unit operating in parallel.
- The data in the registers are applied to one of the unit depending on the operation specified by the instruction associated with the data.
- **Adder and multiplier** -- Arithmetic operation with integer numbers.
- **The floating point** operations are separated into 3 circuits operating in parallel.
- **Logic , shift and increment** can be done parallel on different data.
- All units are independent of each other.
- One number can be **shifted** while another number is **being incremented.**
- Complex control unit.

Application of Parallel Processing

- Numerical weather forecasting
- Computational aerodynamics
- Remote Sensing applications
- Genetic Engineering
- Tomography(Related to medical application)
- Weapon research and defense.

Explain Flynn's classification for computers.

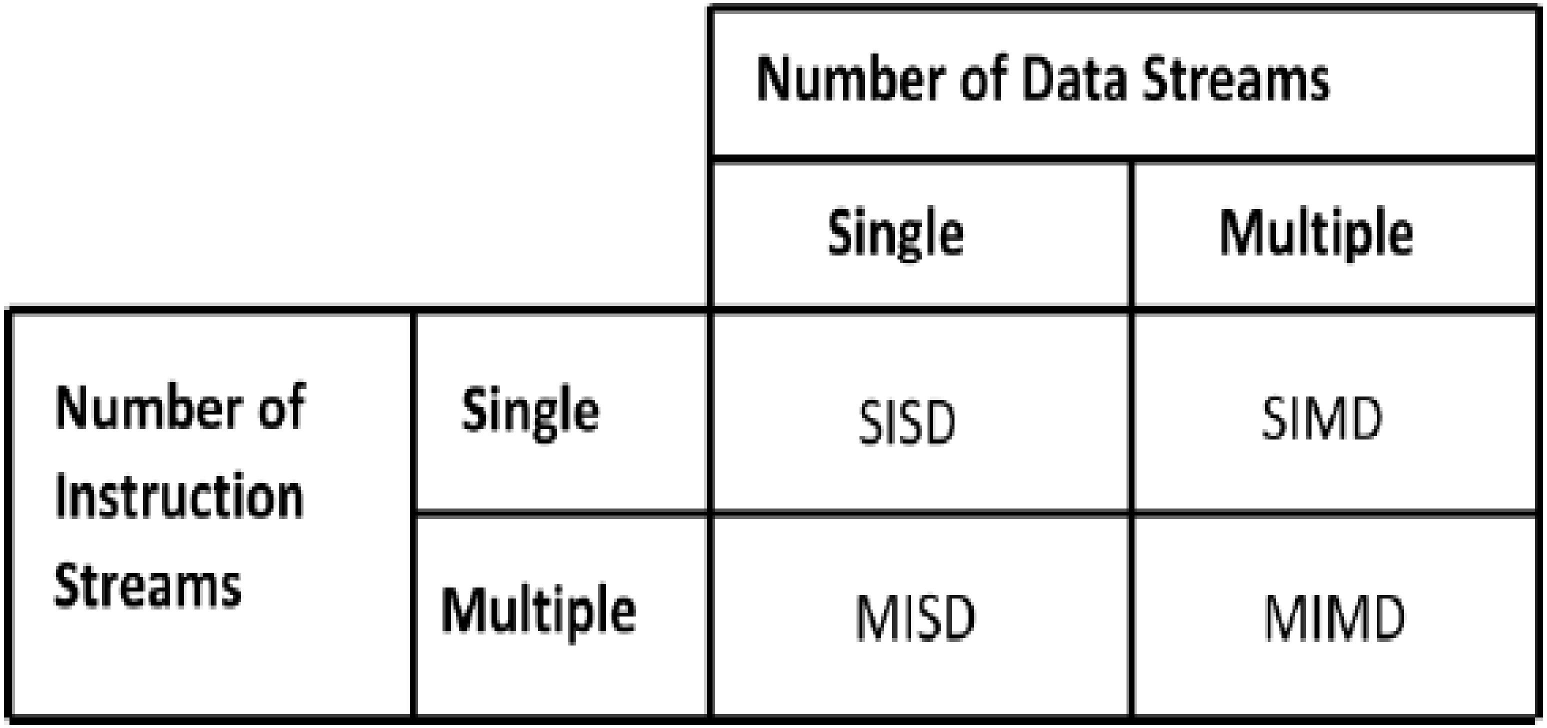
Based on the multiplicity of ***Instruction Streams*** and ***Data Streams.***

Instruction Stream

Sequence of Instructions read from memory

Data Stream

Operations performed on the data in the processor



		Number of Data Streams	
		Single	Multiple
Number of Instruction Streams	Single	SISD	SIMD
	Multiple	MISD	MIMD

Figure Flynn's Classification

Explain Flynn’s classification for computers.

1) SISD (Single Instruction - Single Data stream)

- A single computer containing **a control unit, processor unit and a memory unit.**
- Instructions are executed sequentially.

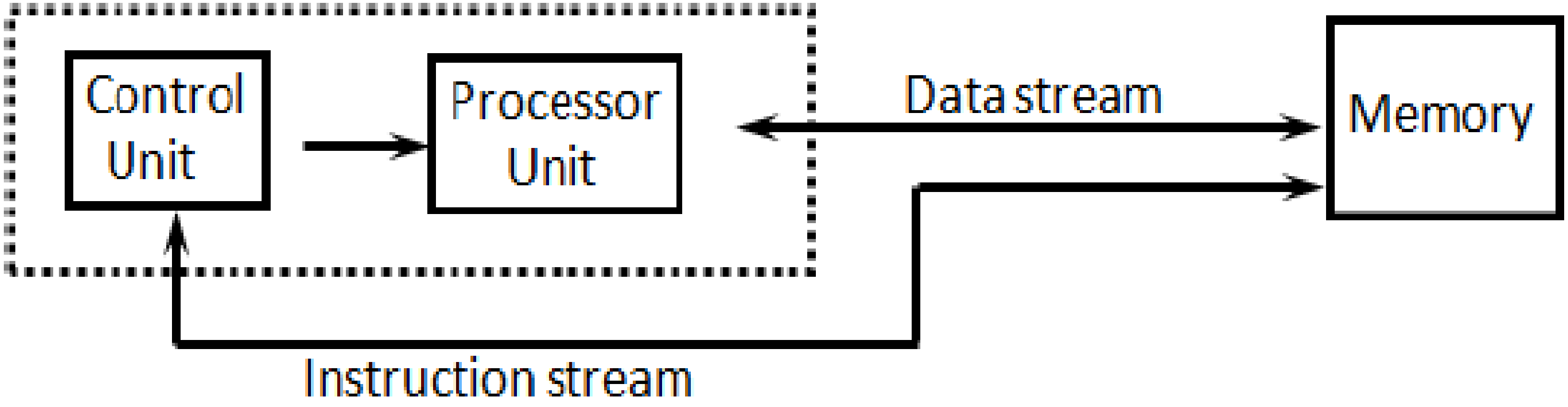
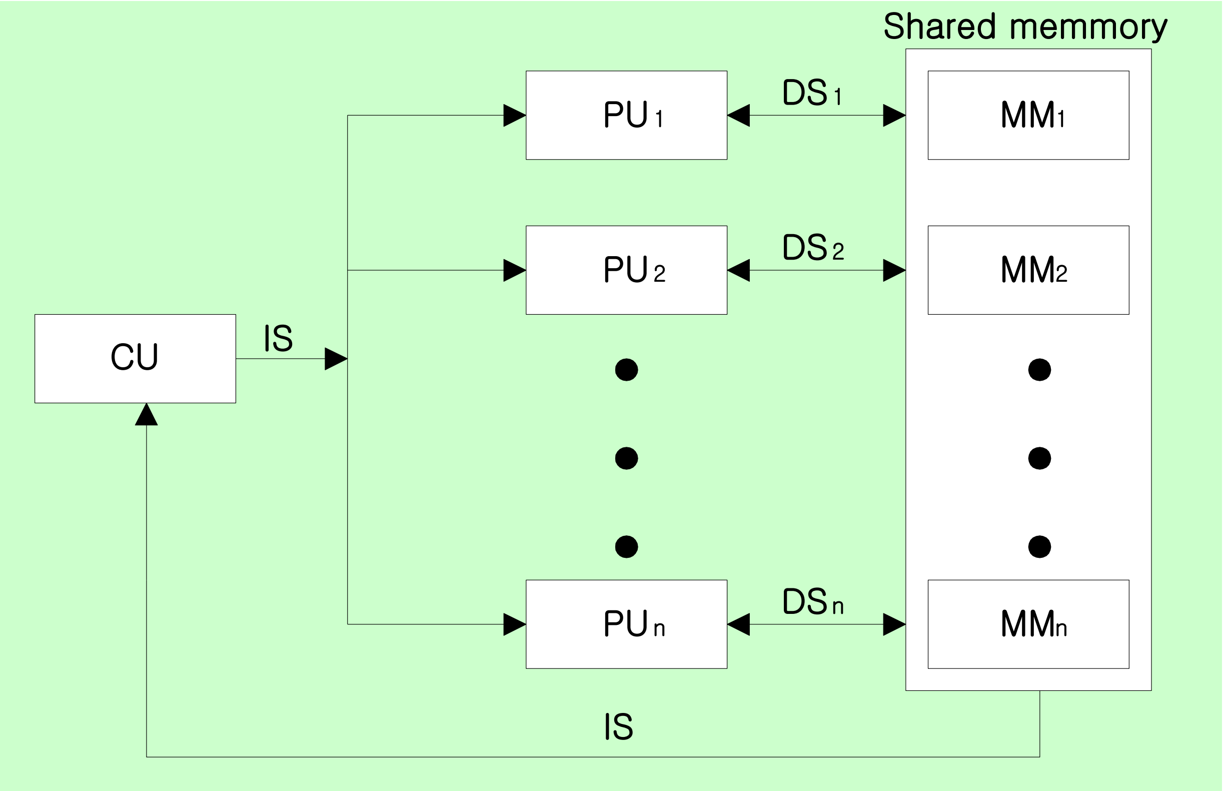


Figure : SISD Organization

Explain Flynn’s classification for computers.

2) SIMD (Single Instruction - Multiple Data stream)

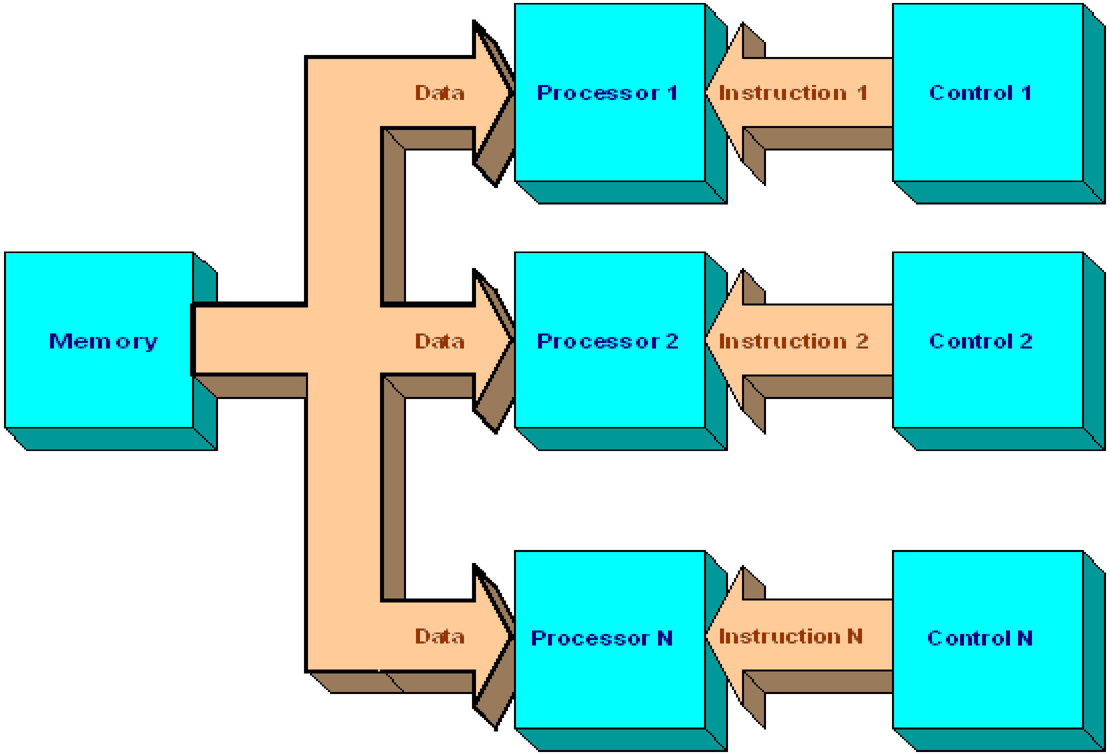
- **Multiple processing units** under the control of a **common control** unit.
- All processors receive the same instruction from control unit but operate on different parts of the data.
- PU = Processing Unit



Explain Flynn's classification for computers.

3) MISD (Multiple Instruction Single Data stream)

- No computer is structured in this category.

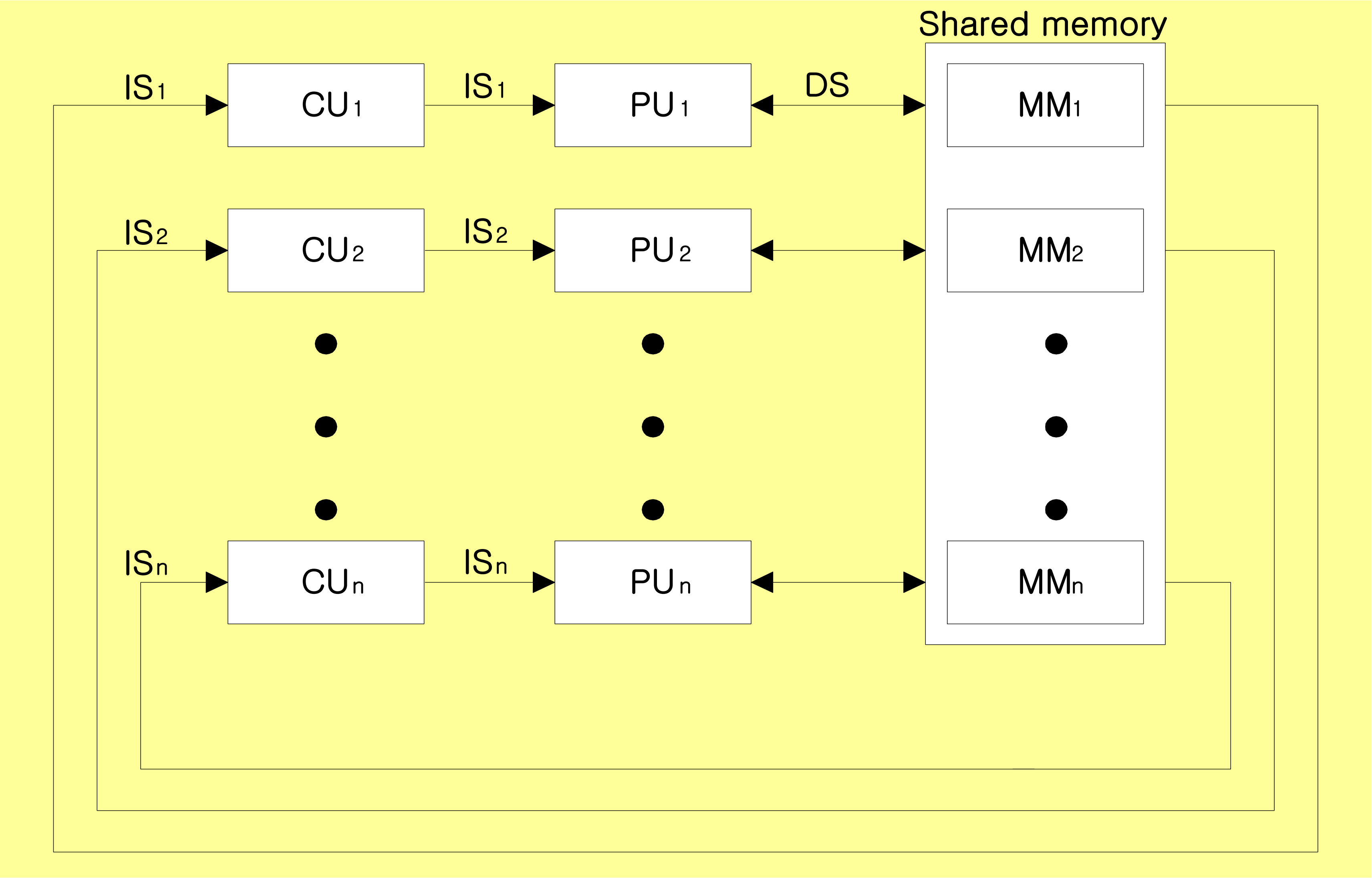


Explain Flynn's classification for computers.

4) MIMD (Multiple Instruction Multiple Data stream)

- **Many programs are** processed at the same time.
- **Multiple instructions** are executed on multiple data.
- Multiprocessors are classified in this.
- **Multiple processing** unit.

Explain Flynn’s classification for computers.



Multiple Instruction Multiple Data stream

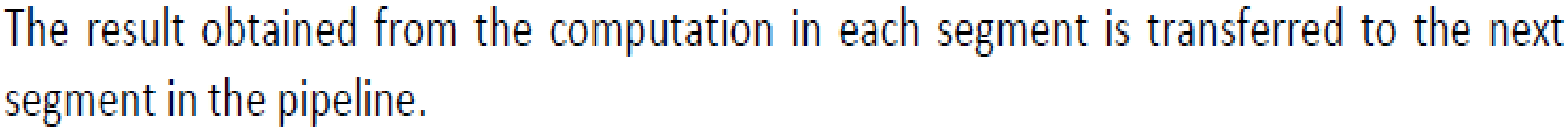
IS = Instruction Stream DS = Data stream

Pipelining :

- **A technique of decomposing** a sequential process into sub-operations.

- Each sub-process being executed in a partial dedicated segment.

- All operate concurrently with all other segments.



The result obtained from the computation in each segment is transferred to the next segment in the pipeline.

- Several computation can be in progress in distinct segment at the same time.

Pipelining

- Each segment consists of an input register to hold the data from memory.

- Combinational circuit performs the sub-operation.

- Output of the combinational circuit in a given segment is applied to the input register of the next segment.

- **Example** : combined operation: Multiplication and Addition with a stream of numbers.

$$A_i * B_i + C_i \qquad \text{for } i = 1, 2, 3, \ldots , 7$$

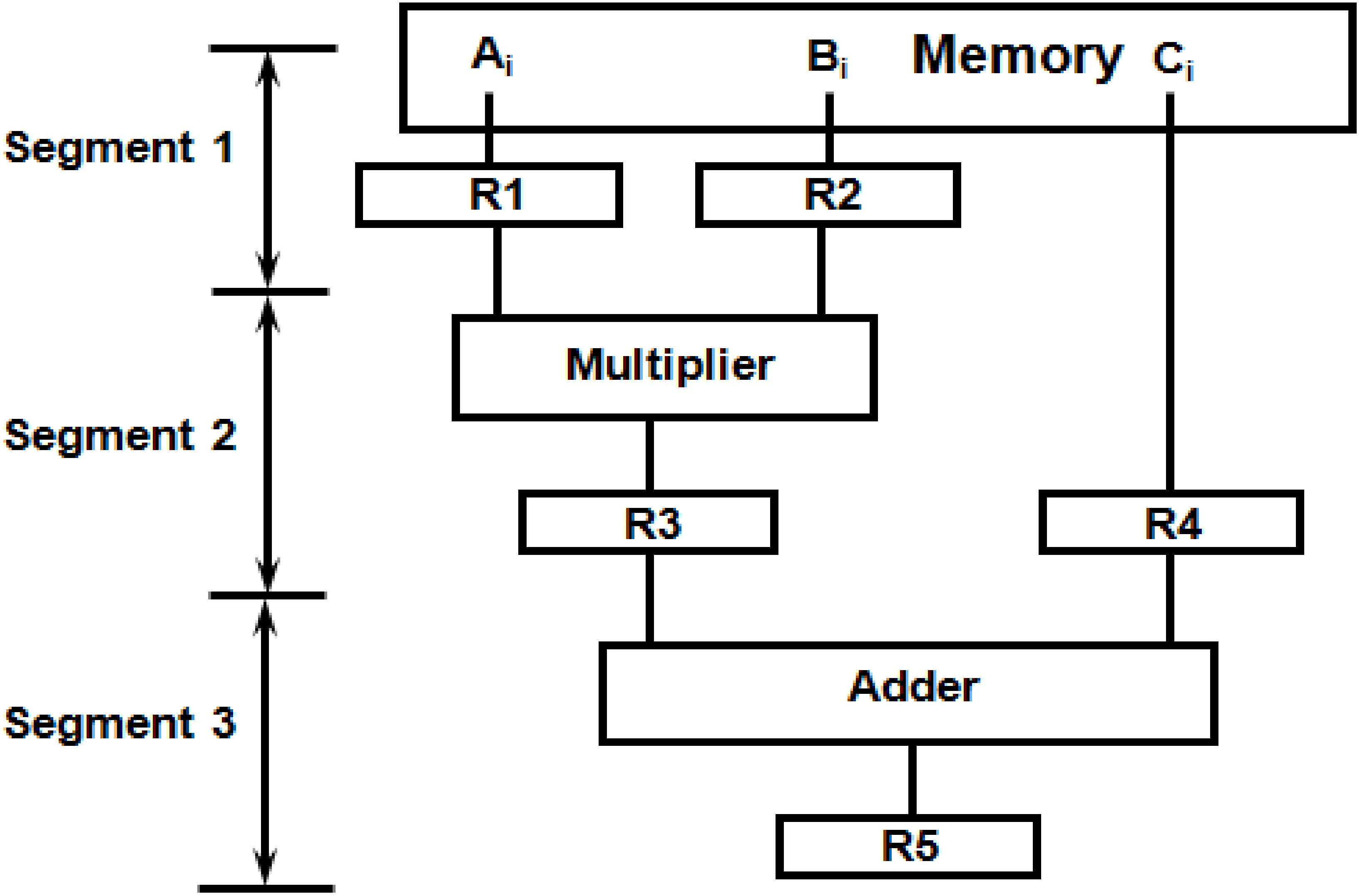
Pipelining- Example

- Each segment has one or two registers and a combinational circuit.

- R1 through R5 are registers that receive new data with every clock pulse.

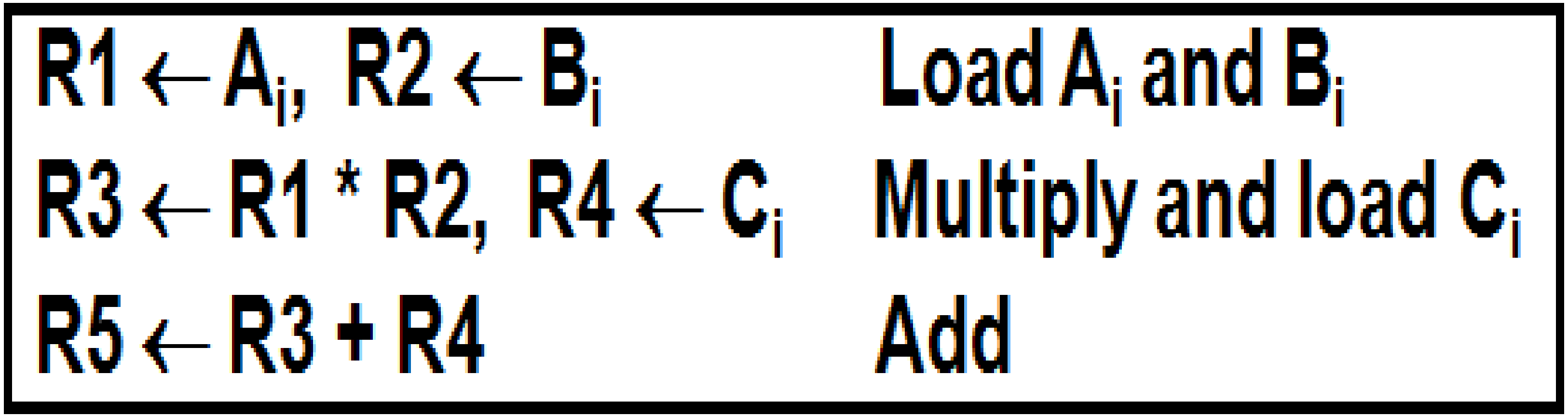
- The multiplier and adder are combinational circuits.

Pipelining- Example



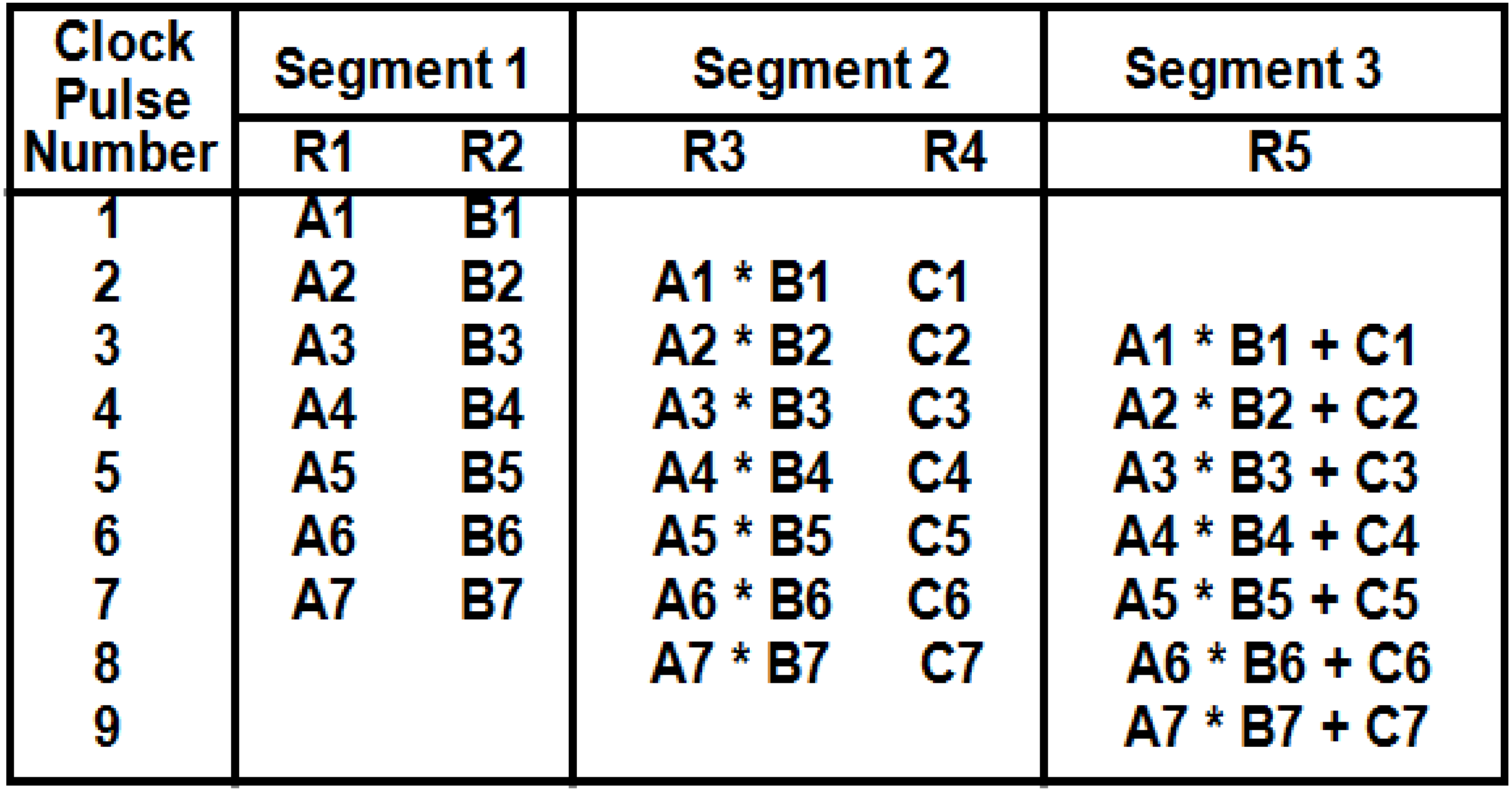
Pipelining- Example

The sub-operations performed in each segment of the pipeline are,



$R1 \leftarrow A_i,\ R2 \leftarrow B_i$	Load A_i and B_i
$R3 \leftarrow R1 * R2,\ R4 \leftarrow C_i$	Multiply and load C_i
$R5 \leftarrow R3 + R4$	Add

Pipelining- Example



Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1			
2	A2	B2	A1 * B1	C1	
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8			A7 * B7	C7	A6 * B6 + C6
9					A7 * B7 + C7

OPERATIONS IN EACH PIPELINE STAGE

Pipelining- Example

- It takes three clock pulses to fill up the pipe and get the first output from R5.

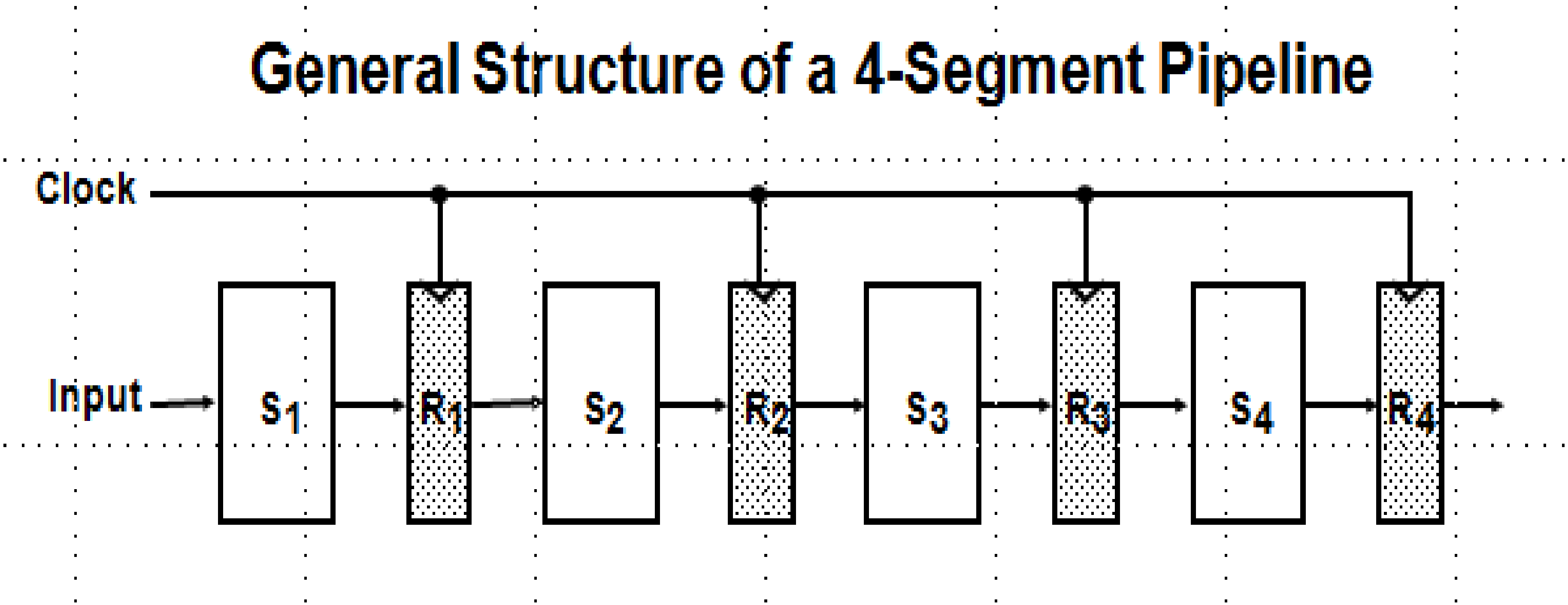
- From there for each clock produces a new output .

Pipelining(Continue…)

- The data pass through all four segment in a fixed sequence.
- Segment consists of combinational circuit Si.

The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

General Structure of a 4-Segment Pipeline



Pipelining(Continue…)

-- The operands pass through all four segments in a fixed sequence.

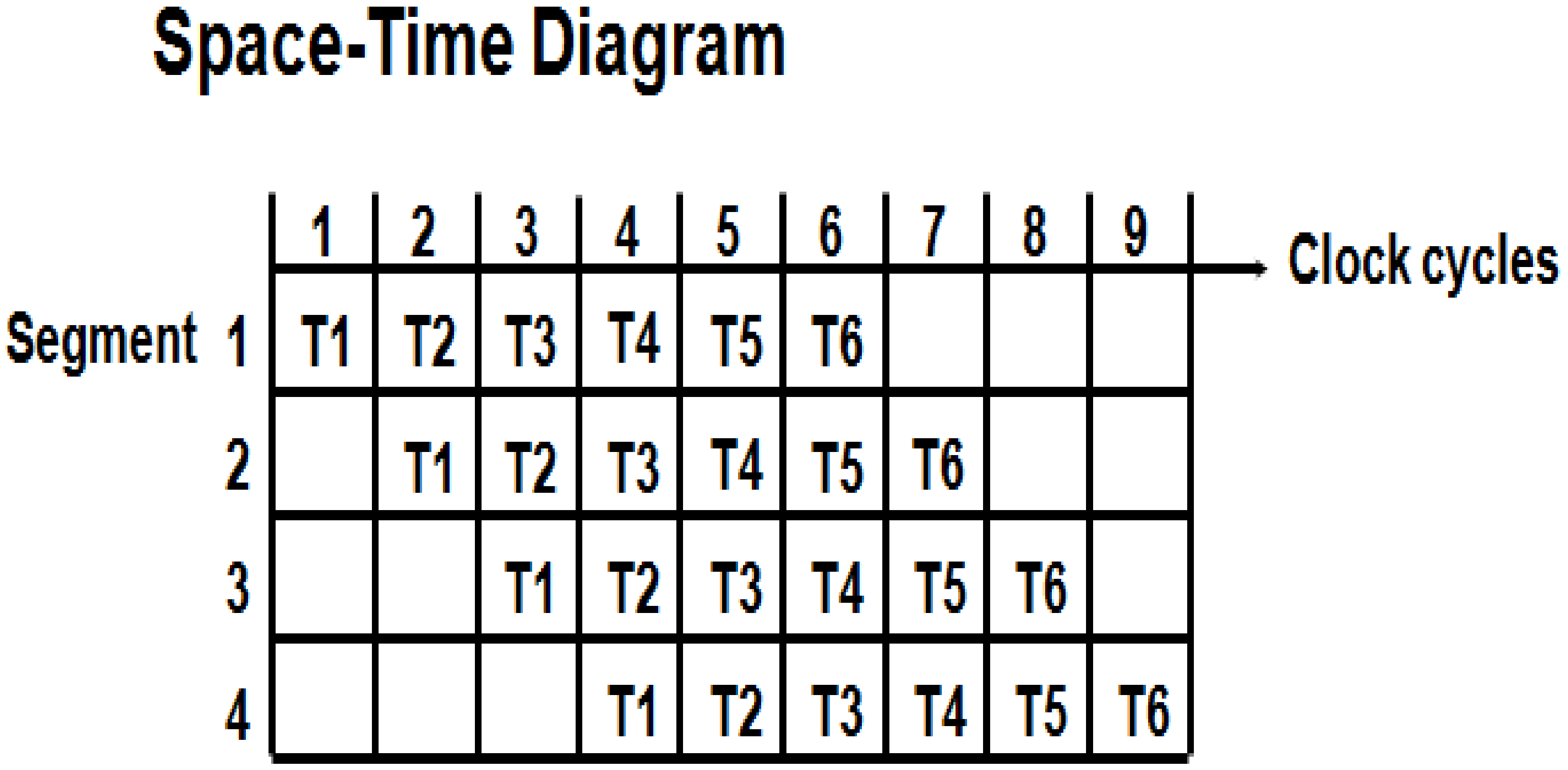
-- Each segment consists of a combinational circuit S, which performs a sub operation over the data stream flowing through the pipe.

-- Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.

-- a task as the total operation performed going through all the segments

Pipelining(Continue…)

Diagram shows the **segment utilization as a function of time** called **Space time diagram**.



Space-Time Diagram

	1	2	3	4	5	6	7	8	9
Segment 1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

Clock cycles

Space time diagram for 4 segment pipeline

Space time diagram for 4 segment pipeline

- Horizontal axis – Time in clock.
- Vertical axis – The segment number
- **Six Task** T1 through T6 and **4** segments.
- Initially **task T1** is handled by **segment 1 .**
- After the **first clock segment 2** is busy **with T1** while **segment 1 is busy with task 2**.
- First task T1 is completed after 4th clock cycle.
- From then the pipe completes a task every clock cycle.
- Once the pipeline is full , it takes only one clock period to obtain an output.

Pipeline Speed Up

Lets consider,

k-segment pipeline with a clock cycle time t_p is used to execute n tasks.

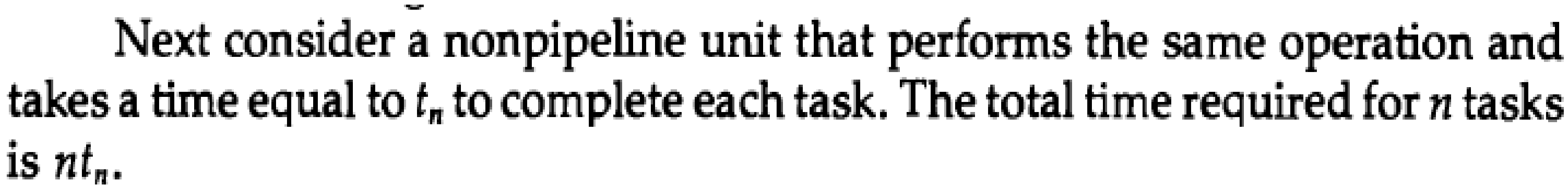
- To Complete **n** tasks using **k** segment pipeline, [k+(n-1)]tp.

if tp=1 then ,

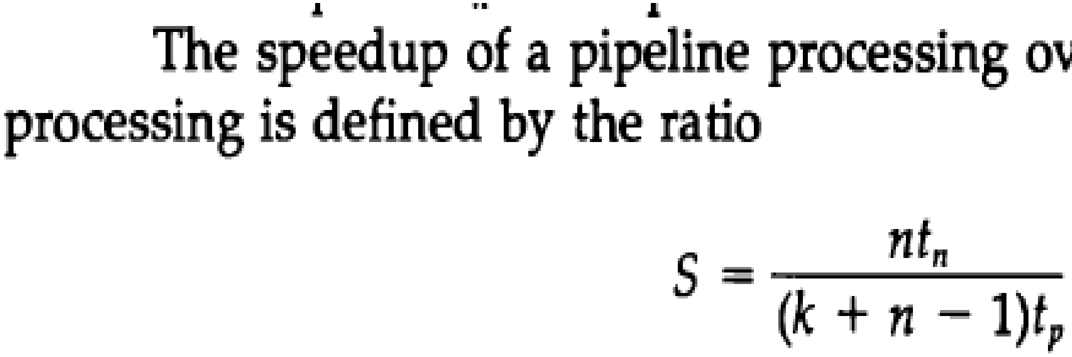
$$k + (n - 1) \text{ clock cycles.}$$

to complete all the operations is $4 + (6 - 1) = 9$ clock cycles, as indicated in the diagram.

Pipeline Speed Up



Next consider a nonpipeline unit that performs the same operation and takes a time equal to t_n to complete each task. The total time required for n tasks is nt_n.



The speedup of a pipeline processing over an equivalent nonpipeline processing is defined by the ratio

$$S = \frac{nt_n}{(k + n - 1)t_p}$$

Pipeline Speed Up(Example)

- **4-stage** pipeline

- - sub-opertion in each stage; **t_p = 20nS**
- - **100 tasks** to be executed
- - 1 task in non-pipelined system; 20*4 = 80nS
-
- Non-Pipelined System
- **$n*k*t_p$ = 100 * 80 = 8000nS**

- Pipelined System
- **$(k + n - 1)*t_p$ = (4 + 99) * 20 = 2060nS**

- Speedup ratio:
- **S_k = 8000 / 2060 = 3.88**

Arithmetic Pipeline

Floating-point Adder Pipeline Example :

- Add / Subtract two normalized floating-point binary number

» $X = A \times 2^a = 0.9504 \times 10^3$

» $Y = B \times 2^b = 0.8200 \times 10^2$

- 4 segments sub-operations

1) Compare exponents by subtraction :

$3 - 2 = 1$

2) Align mantissas

■ $X = 0.9504 \times 10^3$

■ $Y = 0.08200 \times 10^3$

3) Add mantissas

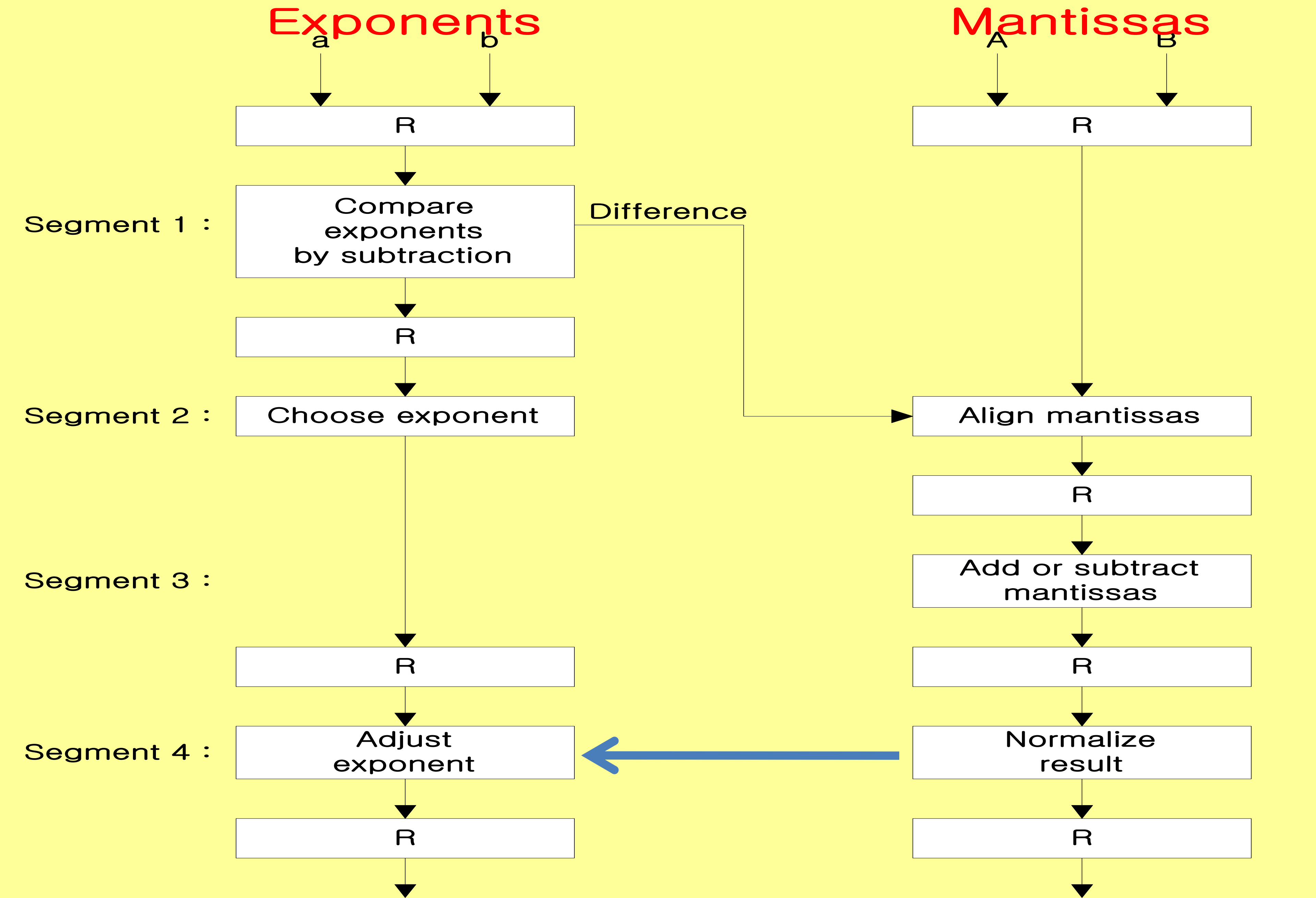
larger exponent 3 is chosen as the exponent of the result.

■ $Z = 1.0324 \times 10^3$

4) Normalize result

■ $Z = 0.1324 \times 10^4$

Figure Pipeline for floating-point addition and subtraction



Instruction Pipeline

❖ *6 phases of Instruction Cycle*

1) **Fetch** the instruction from memory
2) **Decode** the instruction
3) Calculate the **effective address**
4) **Fetch the operands** from memory
5) **Execute** the instruction
6) **Store the result** in the proper place

- ✓ Some instructions **skip** some phases.
- ✓ **Effective address** calculation can be done in the part of decoding phase
- ✓ **Storage of the operation** result into a register is done automatically in the execution phase

Instruction Pipeline-

Four-segment(phase/Stage) Instruction Pipeline

[1] FI: **Fetch** an **I**nstruction from memory

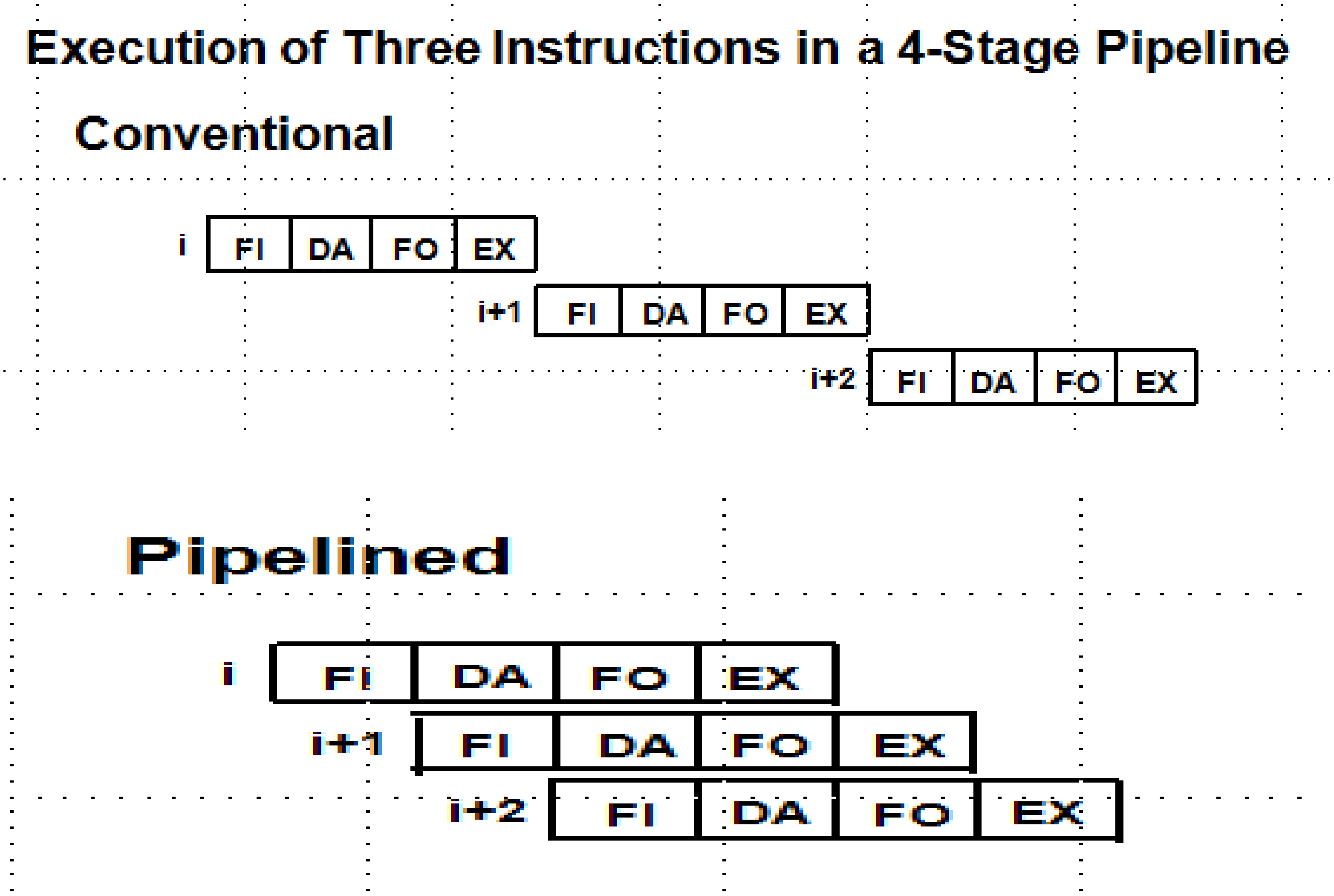
[2] DA: **Decode** the instruction and **calculate the effective Address** of the operand

[3] FO: **F**etch the **O**perand

[4] EX: **Ex**ecute the operation

Instruction Pipeline-

Four-segment(phase/Stage) Instruction Pipeline



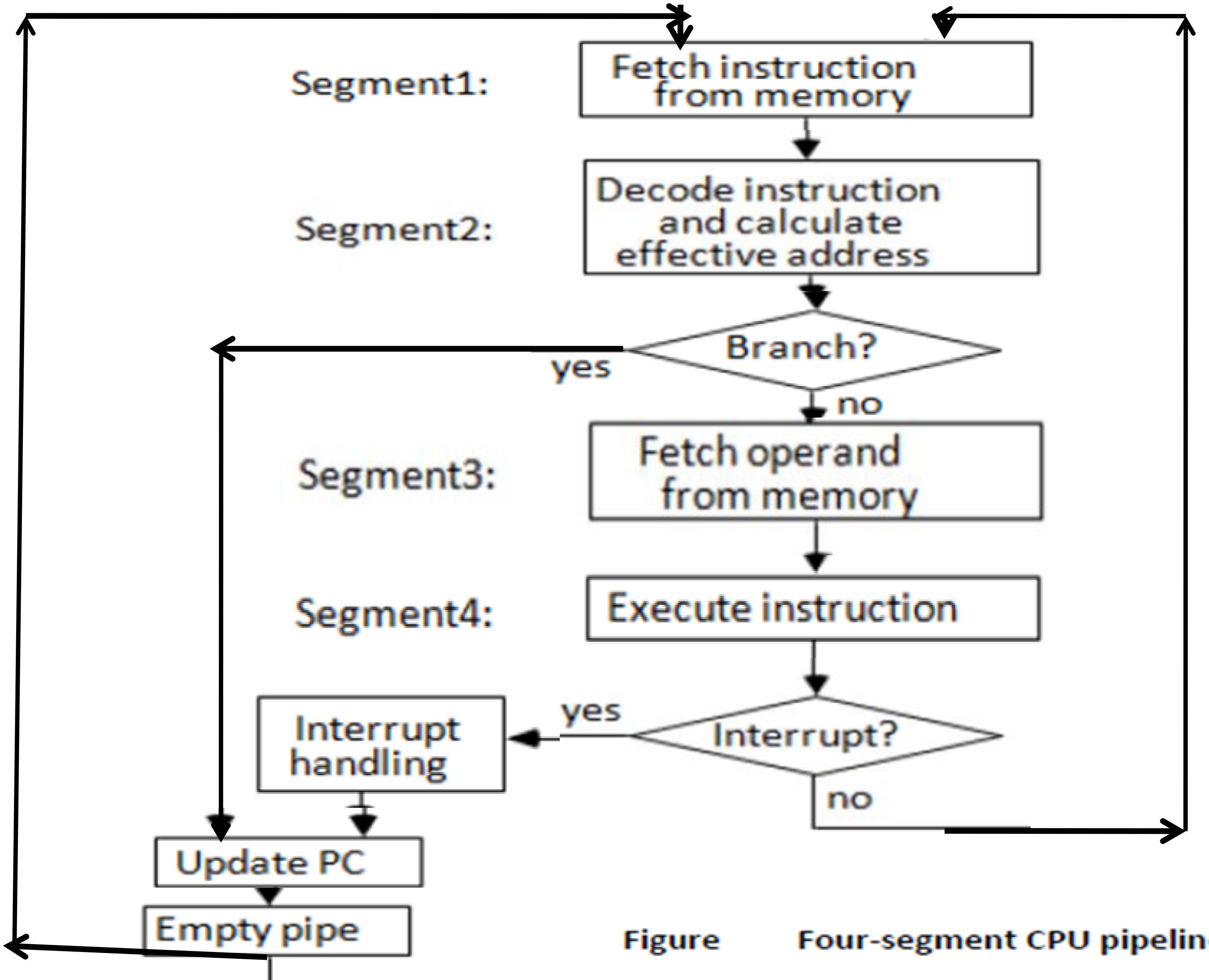


Figure Four-segment CPU pipeline

Instruction Pipeline-

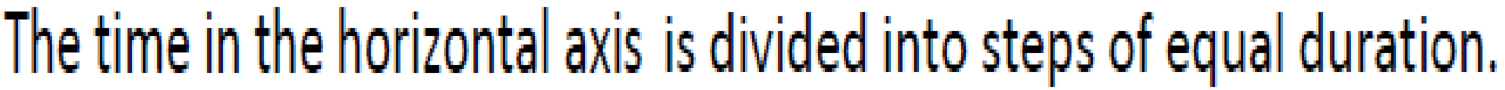
Four-segment(phase/Stage) Instruction Pipeline

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Figure Timing of Instruction Pipeline

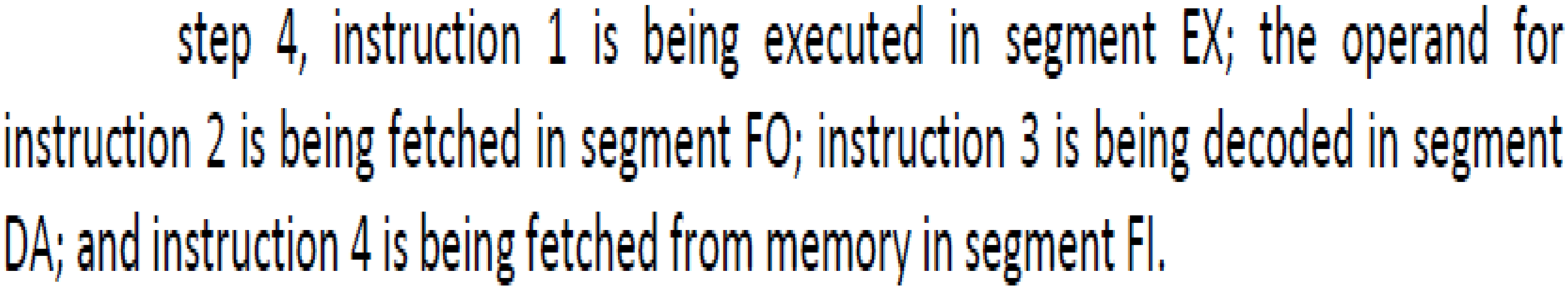
Instruction Pipeline-

Timing of Instruction Pipeline



The time in the horizontal axis is divided into steps of equal duration.

- Four segments are indicated by FI,DA,FO and EX.



step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FI.

Assume now that instruction 3 is a branch instruction.

Instruction Pipeline-

Timing of Instruction Pipeline

As soon as this instruction is decoded in segment DA in step 4, the transfer from FI to DA of the other instructions is halted until the branch instruction is executed in step 6.

If the branch is taken, a new instruction is fetched in step 7

If the branch is not taken, the instruction fetched previously in step 4 can be used.

The pipeline then continues until a new branch instruction is encountered.

RISC PIPELINE

- RISC is its ability to **use an efficient instruction** pipeline.
- Pipeline with small number of sub-operations with each being executed in **one clock cycle.**
- No need o calculate **an effective address.**
- Simple Instruction Set.
- Fixed Length Instruction Format.
- Register-to-Register Operations.

RISC PIPELINE

- So instruction pipeline can be implemented with **two or three segments.**

- One segment **fetches** the instruction from program memory.

- Other segment **executes** the instruction in **ALU.**

- A third segment is busy **to store** the result of the ALU operation in a destination register.

RISC PIPELINE

Instruction Cycles of Three-Stage Instruction Pipeline

Data Manipulation Instructions

I: Instruction Fetch
A: Decode, Read Registers, ALU Operations
E: Write a Register

Load and Store Instructions

I: Instruction Fetch
A: Decode, Evaluate Effective Address
E: Register-to-Memory or Memory-to-Register

Program Control Instructions

I: Instruction Fetch
A: Decode, Evaluate Branch Address
E: Write Register(PC)

RISC PIPELINE

DELAYED LOAD

LOAD :	R1 ← M[address 1]
LOAD :	R2 ← M[address 2]
ADD :	R3 ← R1 + R2
STORE:	M[address 3] ← R3

The concept of **delaying** the **use of the data loaded** from memory is referred to as delayed load.

RISC PIPELINE

1) DELAYED LOAD

Three-segment pipeline timing

Pipeline timing with data conflict

clock cycle	1	2	3	4	5	6
Load R1	I	A	E			
Load R2		I	A	E		
Add R1+R2			I	A	E	
Store R3				I	A	E

Pipeline timing with delayed load

clock cycle	1	2	3	4	5	6	7
Load R1	I	A	E				
Load R2		I	A	E			
NOP			I	A	E		
Add R1+R2				I	A	E	
Store R3					I	A	E

The data dependency is taken care by the compiler rather than the hardware

RISC PIPELINE

Three-segment pipeline timing

2) DELAYED Branch

Compiler **analyzes** the instructions before and after the branch and rearranges the program sequence by inserting useful instructions in the delay steps.

LOAD from memory to R1

Increment R2

Add R3 to R4

Subtract R5 from R6

Branch to address X

Next instruction

-

-

Instruction in X (subroutine)

RISC PIPELINE

Three-segment pipeline timing

2) DELAYED Branch

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. NOP						I	A	E		
7. NOP							I	A	E	
8. Instr. in X								I	A	E

Pipeline Conflict(HAZARDS)

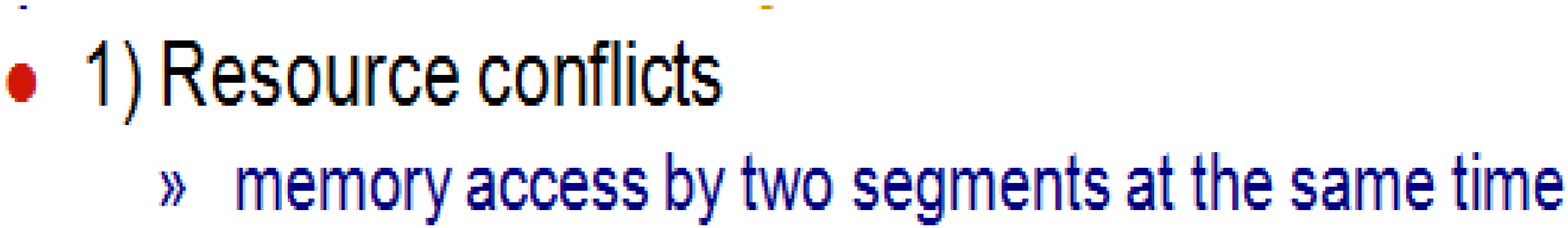
- Pipeline Conflicts : 3 major difficulties

- 1) Resource conflicts
  - » memory access by two segments at the same time

- 2) Data dependency
  - » when an instruction depend on the result of a previous instruction, but this result is not yet available

- 3) Branch difficulties
  - » branch and other instruction (interrupt, ret, ..) that change the value of PC

Pipeline Conflict(HAZARDS)



- 1) Resource conflicts
  - » memory access by two segments at the same time

- Solution: Separate Data and instruction memory

Pipeline Conflict(HAZARDS)

2)

A data dependency occurs when an instruction needs data that are not yet available.

- Solution of Data Hazard
  1) Hardware Interlocked:
  2) Operand Forwarding
  3) Delayed Load Software technique

Pipeline Conflict(HAZARDS)

1)**Hardware Interlocked:**

Circuit detects instructions whose source operands are destination of instructions. Dependent **instruction will be delayed.**

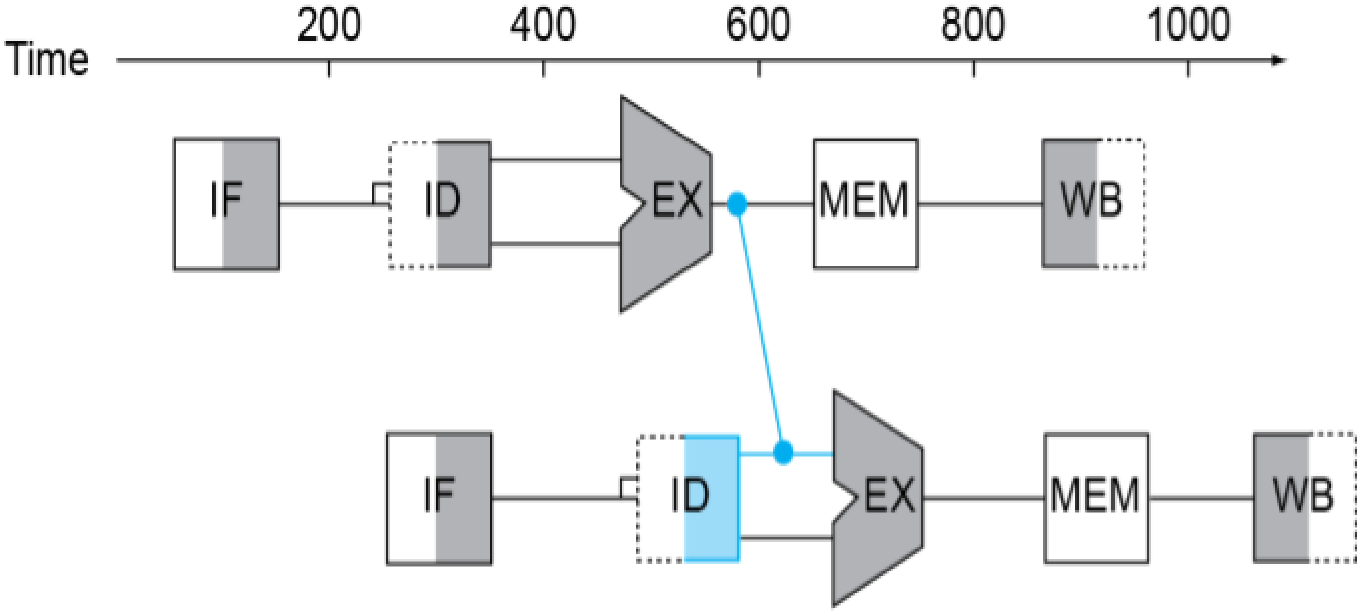
2) Operand Forwarding:

Use special hardware to **detect conflict** and then avoid it by **routing the data through special paths** between pipeline segments.

3) Delayed Load Software technique:

Uses compiler to resolve the problem.

Operand Forwarding



Solution of Branch Hazard

1) ***Pre-fetch target Instruction***

When a conditional branch is recognized, the target of the branch is pre-fetched, in addition to the instruction following the branch.

2) ***Branch Target Buffer(BTB)***

- Memory include in fetch segment.
- It stores address of a previously executed branch instruction and target instruction .

3) ***Loop Buffer***

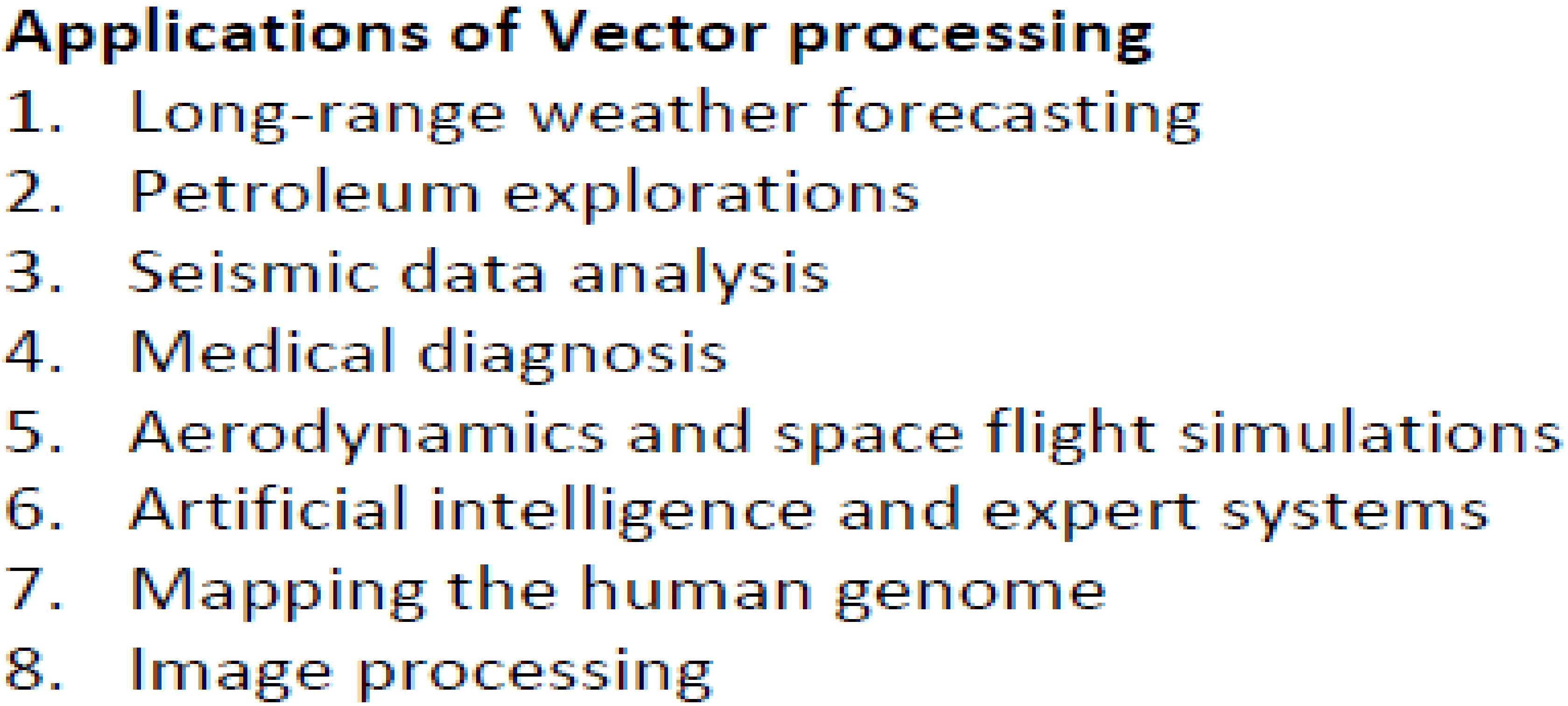
- A small, very-high-speed memory .Designed in instruction fetch stage .A program loop is stored in the loop buffer.

4) ***Delayed branch***

Taken care by compiler using rearranging the instruction .i.e. NOP

Vector Processing

- Ability to process **vectors,** and related data structures such as matrices and **multi-dimensional arrays**, **much faster than conventional computers.**



Applications of Vector processing

1. Long-range weather forecasting
2. Petroleum explorations
3. Seismic data analysis
4. Medical diagnosis
5. Aerodynamics and space flight simulations
6. Artificial intelligence and expert systems
7. Mapping the human genome
8. Image processing

VECTOR Processor

- A Vector processor is **a processor** that can operate on an **entire vector in one instruction.**

- A scalar processor processes only one data item at a time(i.e. SISD).

- **Vector computer- single vector instruction:**

C(1:100) = A(1:100) + B(1:100)

Vector Processing

- Vector Instruction Format :

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length
ADD	A	B	C	100

- Matrix Multiplication
  - 3 x 3 matrices multiplication : $n^2 = 9$ inner product

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

» $c_{11} = a_{11}\,b_{11} + a_{12}\,b_{21} + a_{13}\,b_{31}$: inner product

Vector Processing

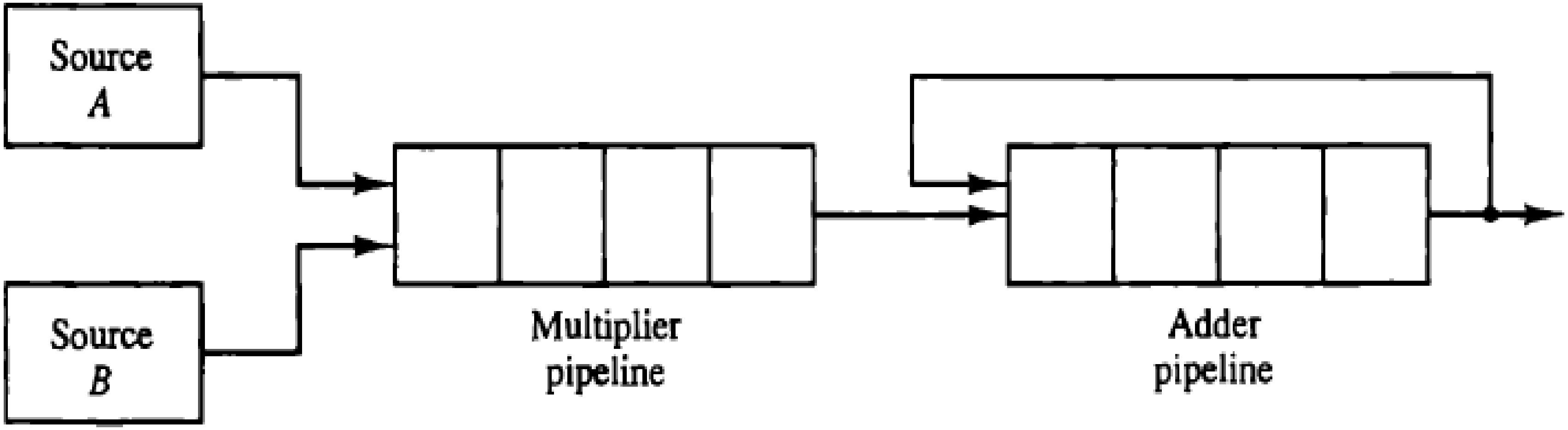
- Inner product equation:

$$\begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = A_x B_x + A_y B_y + A_z B_z$$

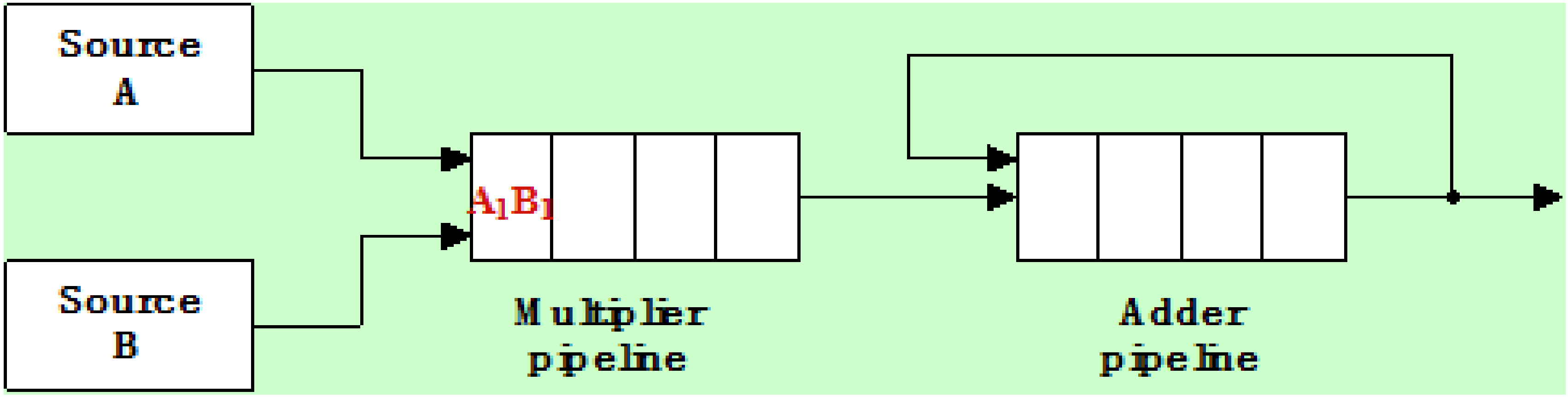
Vector Processing

$$
\begin{aligned}
C = {} & A_1 B_1 + A_5 B_5 + A_9 B_9 + A_{13} B_{13} + \cdots \\
& + A_2 B_2 + A_6 B_6 + A_{10} B_{10} + A_{14} B_{14} + \cdots \\
& + A_3 B_3 + A_7 B_7 + A_{11} B_{11} + A_{15} B_{15} + \cdots \\
& + A_4 B_4 + A_8 B_8 + A_{12} B_{12} + A_{16} B_{16} + \cdots
\end{aligned}
$$

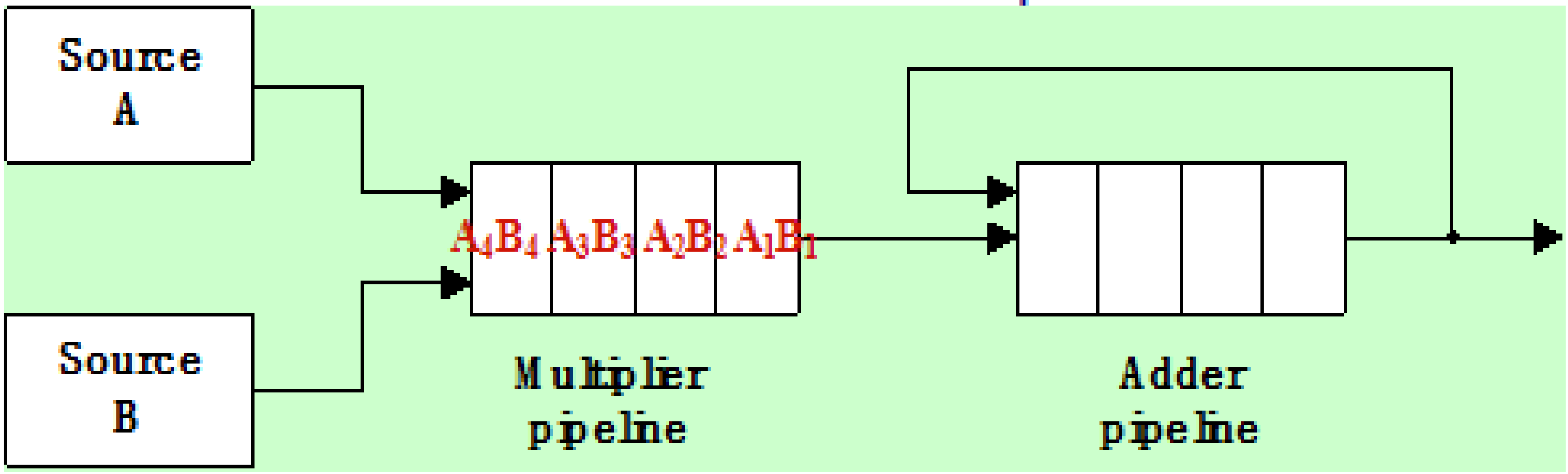
Figure Pipeline for calculating an inner product.



» after 1st clock input

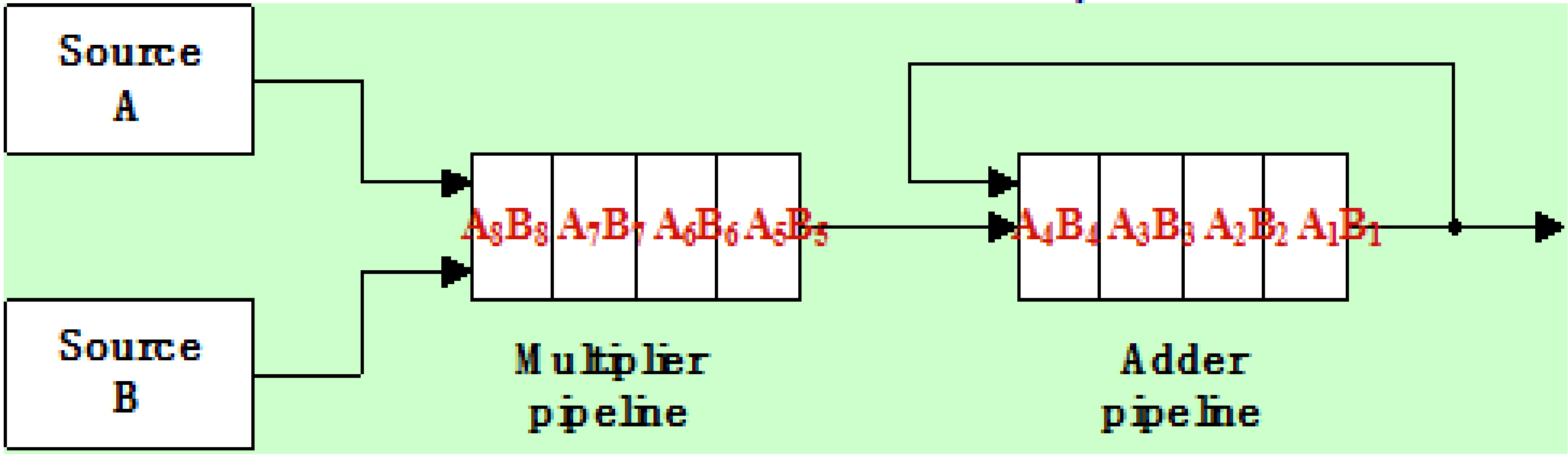


» after 4th clock input

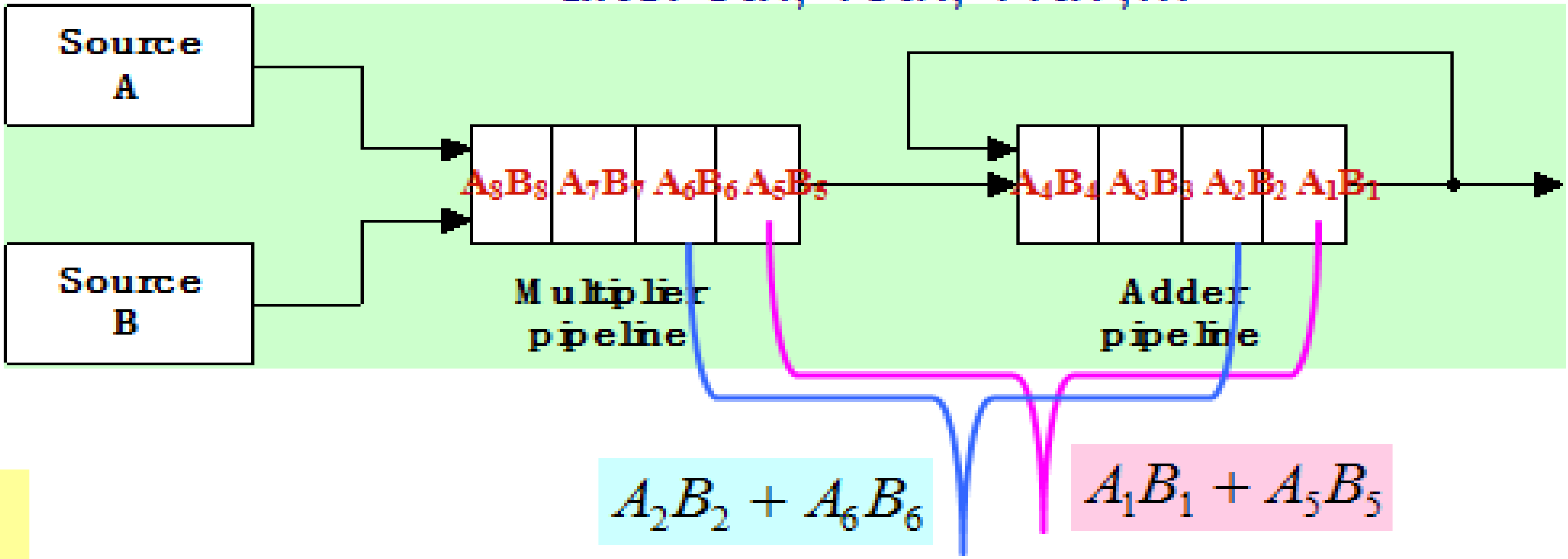


***For Understanding**

» after 8th clock input


Source A
Source B
A_8B_8 A_7B_7 A_6B_6 A_5B_5
A_4B_4 A_3B_3 A_2B_2 A_1B_1
Multiplier pipeline
Adder pipeline

» after 9th, 10th, 11th ,…



*For Understanding

Vector Processing

- Vector processing explain here as per previously explained from all figures.

Array Processors

- Performs computations on large arrays of data
- Two Types

1) Attached array processor :

   - **Auxiliary processor attached** to a general purpose computer

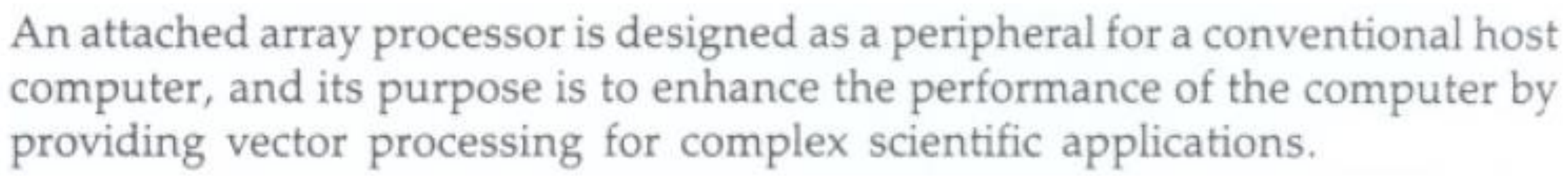
2) SIMD array processor :

   - **Computer with multiple processing units operating in parallel**

Array Processor

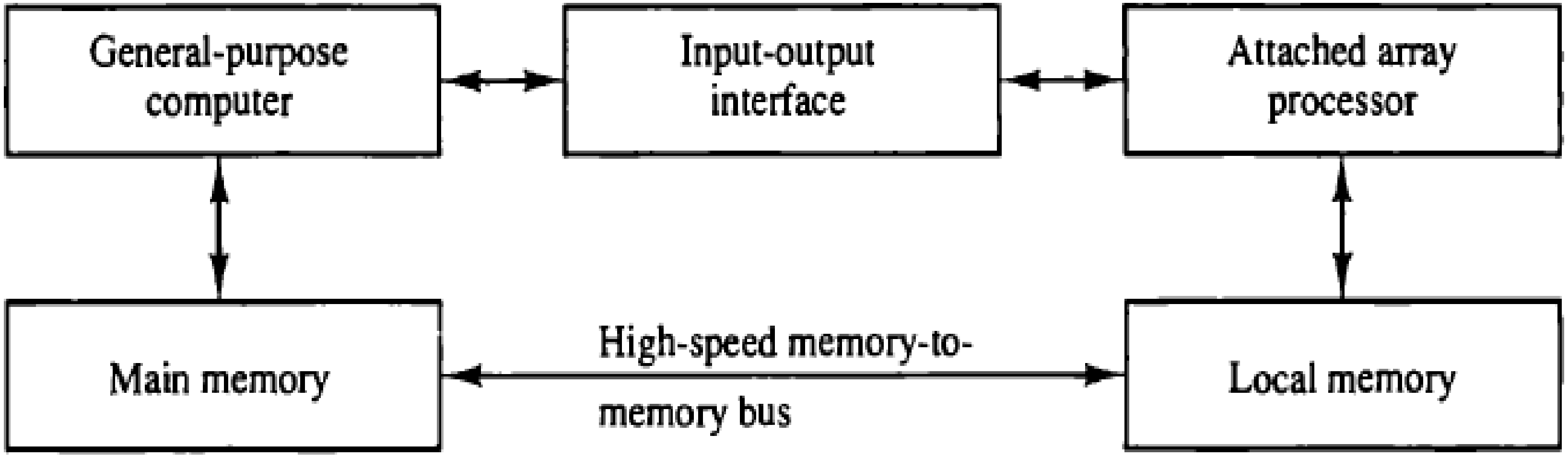
- Performs computations on large arrays of data- Attached array processor and SIMD array processor.

Attached Array Processor :



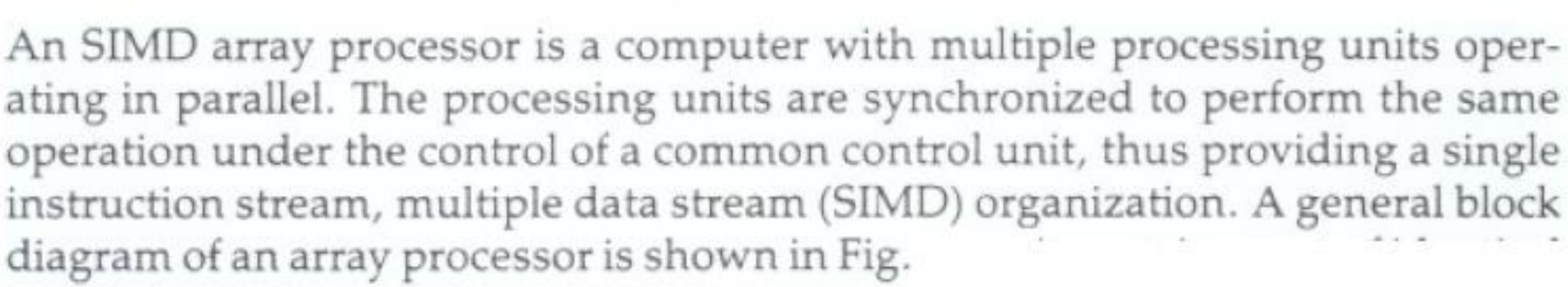
An attached array processor is designed as a peripheral for a conventional host computer, and its purpose is to enhance the performance of the computer by providing vector processing for complex scientific applications.

Figure Attached array processor with host computer.



Array Processor

SIMD Array Processor :



An SIMD array processor is a computer with multiple processing units operating in parallel. The processing units are synchronized to perform the same operation under the control of a common control unit, thus providing a single instruction stream, multiple data stream (SIMD) organization. A general block diagram of an array processor is shown in Fig.

- SIMD array processor : Computer with multiple processing units operating in parallel.
- The best known SIMD array processor is the

  ILLIAC IV

Array Processor

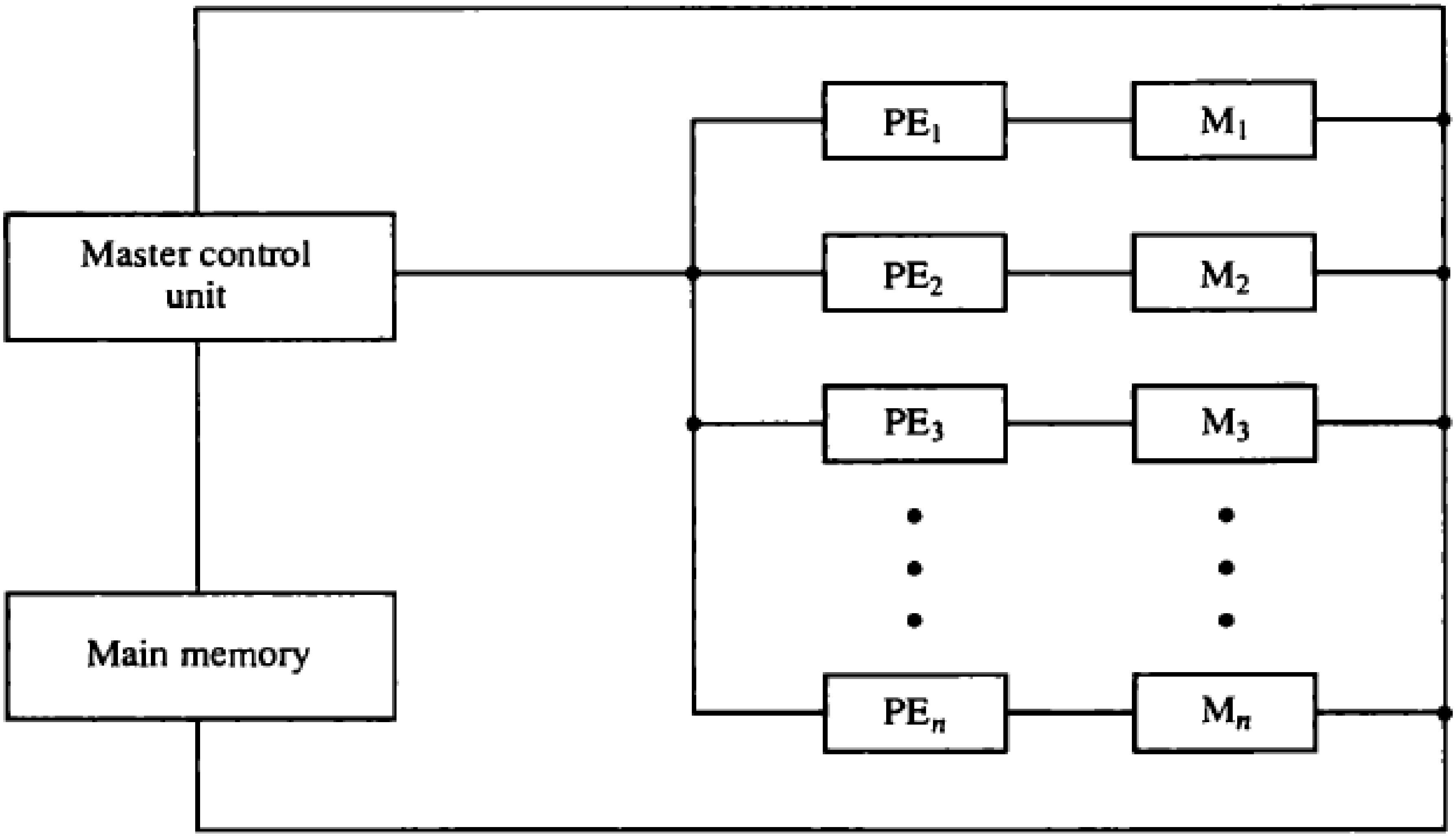


Figure SIMD array processor organization.

Supercomputer

- Supercomputer = Vector Instruction + Pipelined floating-point arithmetic

  - Performance Evaluation Index
    - MIPS : Million Instruction Per Second
    - FLOPS : Floating-point Operation Per Second
      - megaflops : 10^6, gigaflops : 10^9

  - Cray supercomputer : Cray Research
    - Cray-1 : 80 megaflops, 4 million 64 bit words memory
    - Cray-2 : 12 times more powerful than the cray-1

  - VP supercomputer : Fujitsu
    - VP-200 : 300 megaflops, 32 million memory, 83 vector instruction, 195 scalar instruction
    - VP-2600 : 5 gigaflops

PARAM 8000 is considered India's first supercomputer