

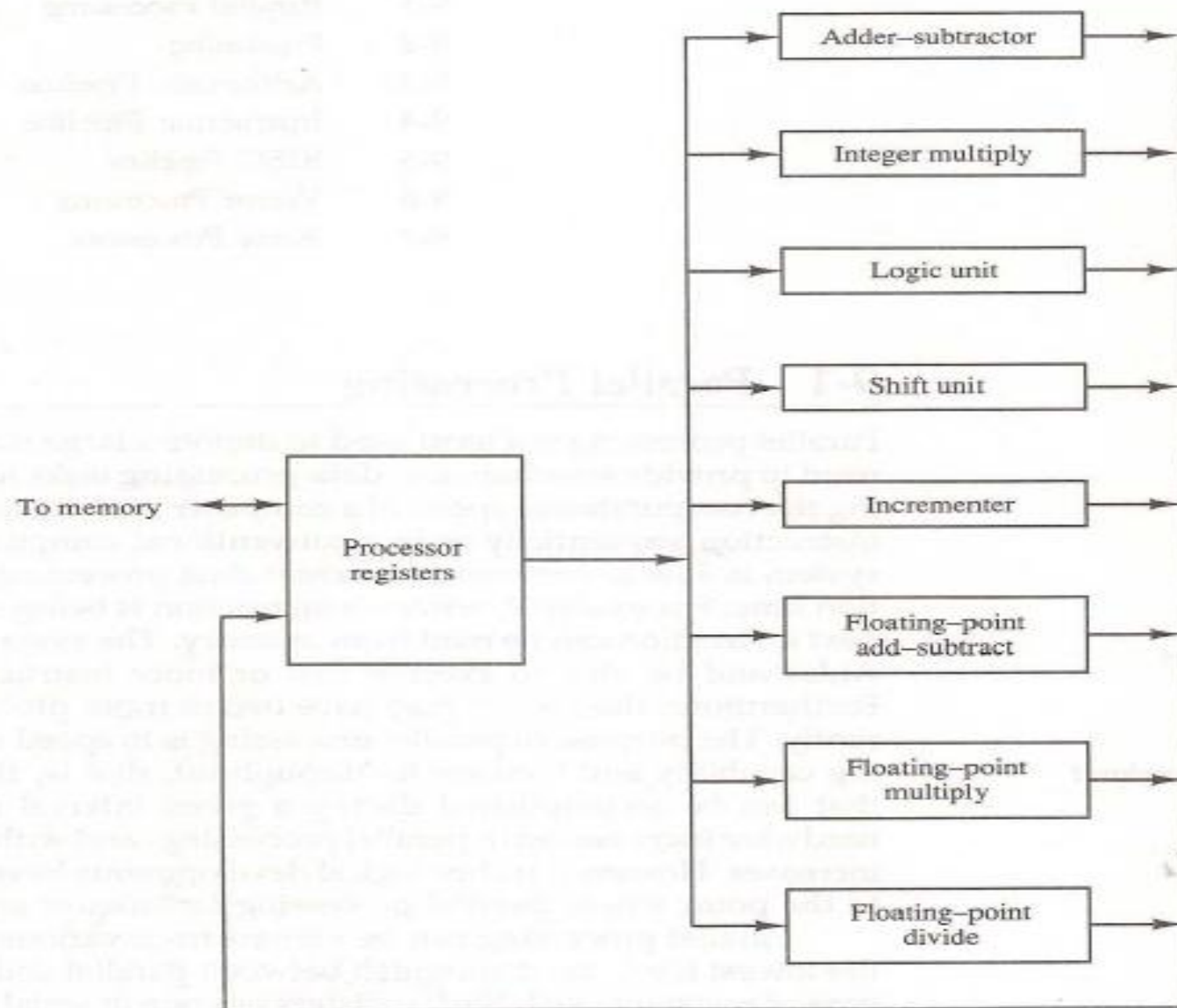
Unit – 5 : PIPELINING AND VECTOR PROCESSING

Parallel processing :

- A parallel processing system is able to perform concurrent data processing to achieve faster execution time.
- The system may have two or more ALUs and be able to execute two or more instructions at the same time.
- Goal is to increase the **throughput** – the amount of processing that can be accomplished during a given interval of time.
- **Throughput** : *No of inputs processed per unit of time.*
- Amount of hardware increases with parallel processing – cost increase
- Reduced hardware cost – parallel processing technique – economic

- Parallel processing can be viewed from various level of complexity.
- At lowest level – distinguish parallel & serial operation by the types of register used.
- Shift register operate in serial fashion one bit at a time.
- Register with parallel load operate with all the bits of the word simultaneously.
- At higher level – multiplicity of functional units perform identical or different operations simultaneously.
- Distributing the data among the multiple functional units.
- Ex – arithmetic, logic & shift operation can be separated between 3 units & operands diverted to each unit under the supervision of control unit.

Figure 9-1 Processor with multiple functional units.



- Adder & integer multiplier perform the arithmetic operation with integer numbers.
- The floating-point operations are separated into 3 circuits operating in parallel.
- The logic, shift & increment operation can be performed concurrently on different data.

PARALLEL COMPUTERS : Flynn's classification

Architectural Classification

- Flynn's classification
 - 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 <i>Data Streams</i>	
		Single	Multiple
Number of <i>Instruction Streams</i>	Single	SISD	SIMD
	Multiple	MISD	MIMD

Parallel processing classification

Single instruction stream, single data stream – SISD

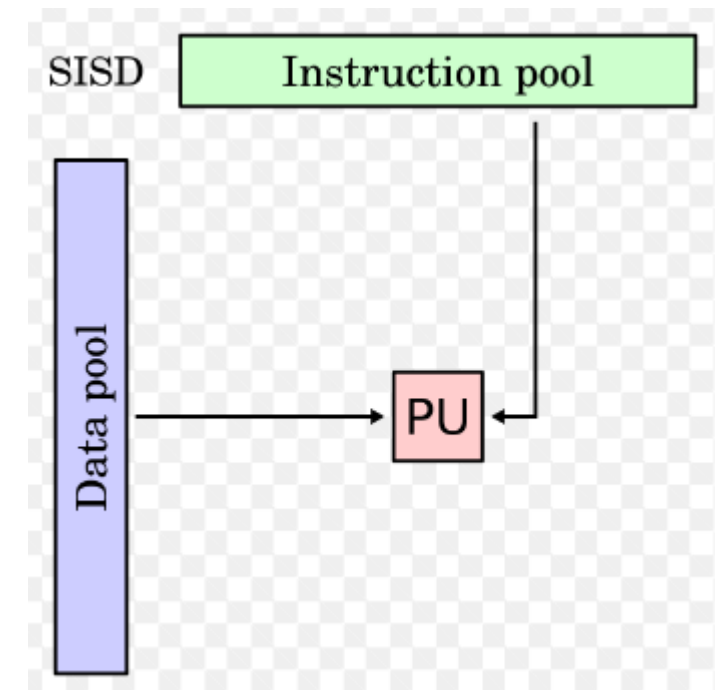
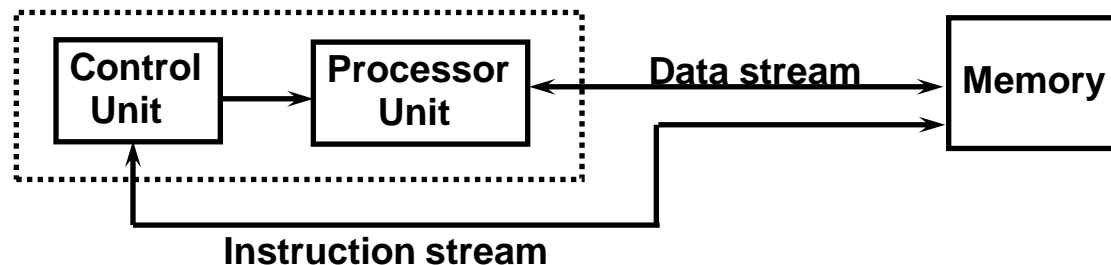
Single instruction stream, multiple data stream – SIMD

Multiple instruction stream, single data stream – MISD

Multiple instruction stream, multiple data stream – MIMD

Single instruction stream, single data stream – SISD

- Single control unit, single computer, and a memory unit.
- Instructions are executed sequentially.
- Parallel processing may be achieved by means of **multiple functional units** or by **pipeline processing**.
- Characteristics
 - Standard von Neumann machine
 - Instructions and data are stored in memory
 - One operation at a time



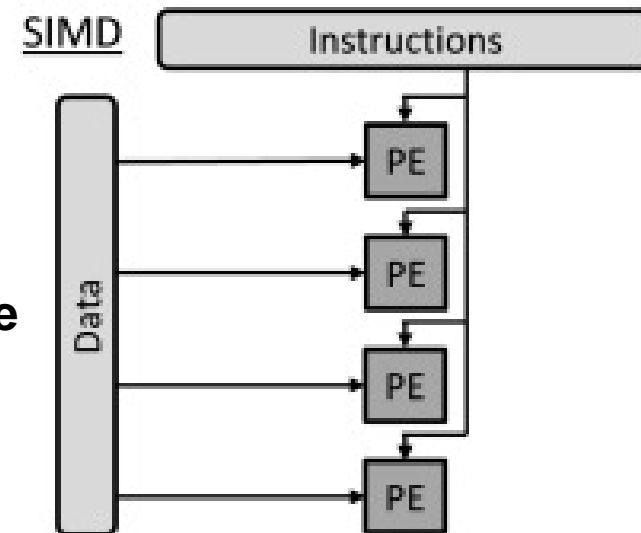
Single instruction stream, multiple data stream – SIMD

- Represents an organization that includes many processing units under the supervision of a common control unit.
- Includes multiple processing units with a single control unit.
- All processors receive the same instruction, but operate on different data.

Characteristics

- Only one copy of the program exists
- A single controller executes one instruction at a time

Ex – Pipeline Processor

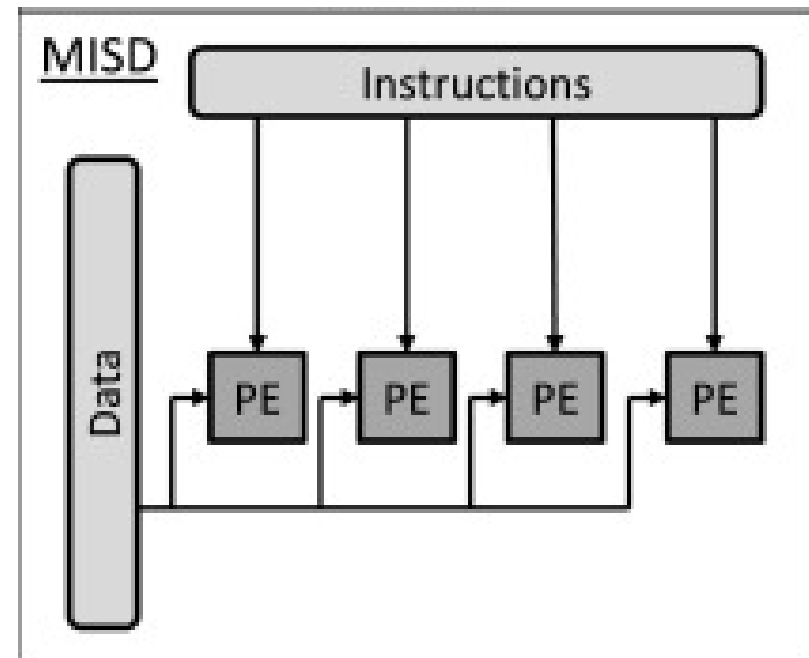


Multiple instruction stream, single data stream – MISD

- **Theoretical only**
- processors receive different instructions, but operate on same data.

Characteristics

- There is no computer at present that can be classified as MISD



Multiple instruction stream, multiple data stream – MIMD

- A computer system capable of processing several programs at the same time.
- Most multiprocessor and multicomputer systems can be classified in this category

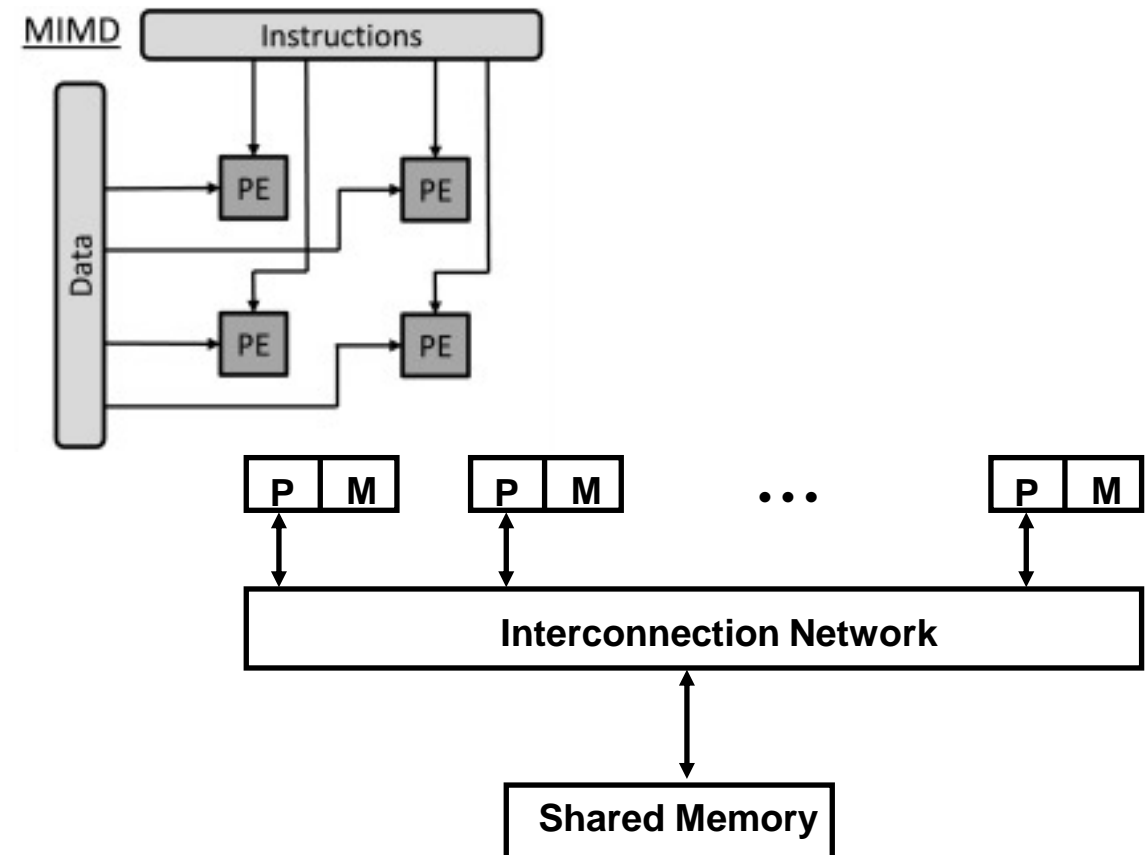
Characteristics

- Multiple processing units
- Execution of multiple instructions on multiple data

EX – Multiple pipeline (super scalar computer)

Types of MIMD computer systems

- Shared memory multiprocessors
- Message-passing multicomputer



PIPELINING :

- Pipelining is useful, when **same processing** is applied over **multiple inputs**.
- EX – Assembly line in manufacturing
- Technique to decompose a sequential process into sub-operations
- Sub-operations are performed in **Segments** (stage)
- **Task** : One operation performed in all segments
- Each segment can perform it's respective sub operation over different input in parallel to other segments.
- A technique of decomposing a sequential process into suboperations, with each subprocess being executed in a partial dedicated segment that operates concurrently with all other segments.

To perform multiply & add operation with a stream of numbers.

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

$R1 \leftarrow A_i, R2 \leftarrow B_i$

Load A_i and B_i

$R3 \leftarrow R1 * R2, R4 \leftarrow C_i$

Multiply and load C_i

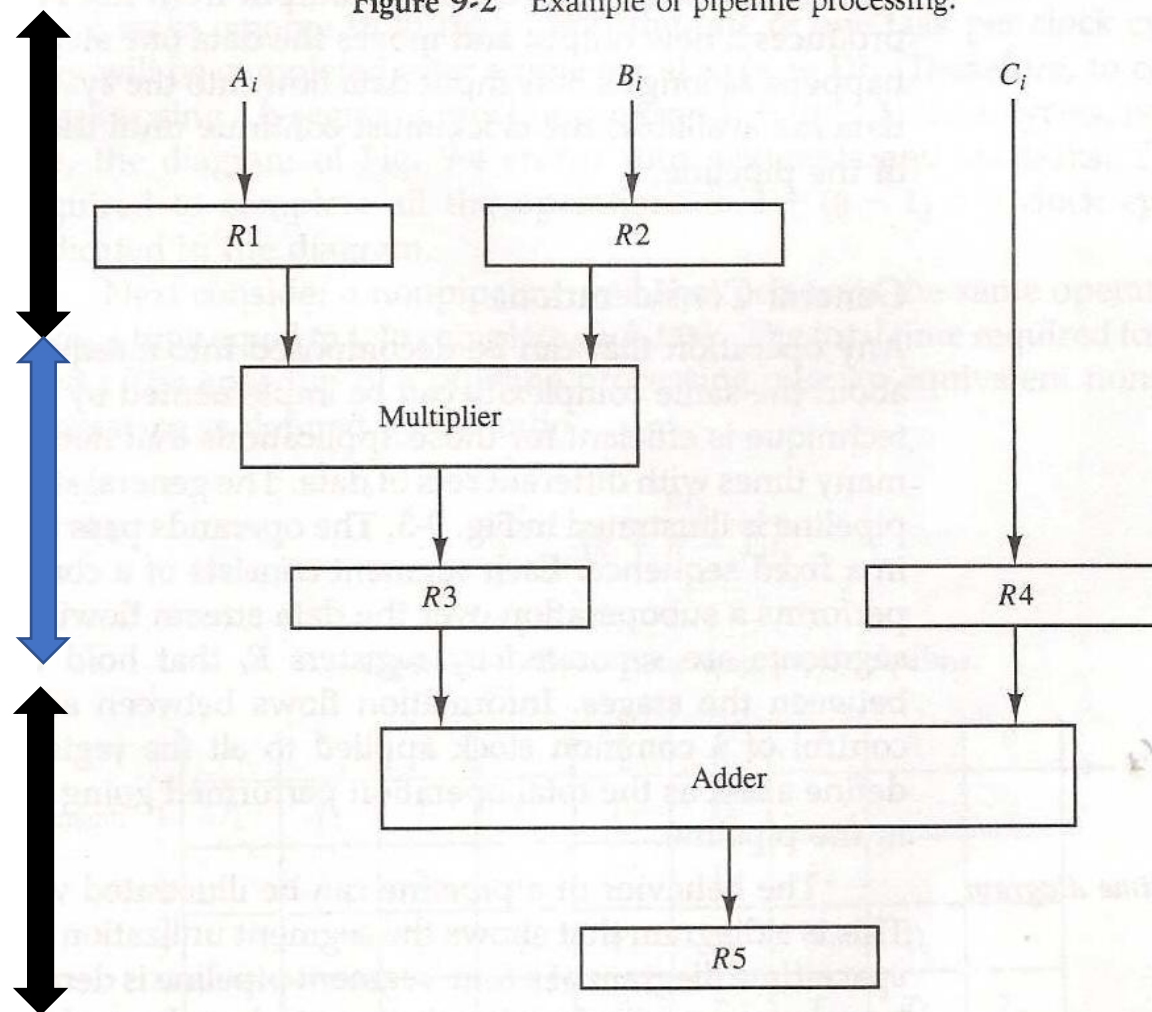
$R5 \leftarrow R3 + R4$

Add

TABLE 9-1 Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

Figure 9-2 Example of pipeline processing.



- **General consideration about Pipeline :**

Consider a k segments pipeline

With clock cycle time = t_p

To perform n tasks

Time required to perform 1st task = $k * t_p$

Time required to perform remaining $(n-1)$ tasks = $(n-1) * t_p$

Time required for all n tasks = $(k+n-1) * t_p$

- **General consideration about Non-Pipeline :**

Consider a non-pipeline system that takes t_n time to perform a tasks

Time required to perform n task = $n * t_n$

- Performance of a pipeline is given by **Speed up ratio**.

Speed up ratio = non-pipeline time / pipeline time

$$S = \frac{n * t_n}{(k+n-1) * t_p} \quad (k+n-1 = \text{Total no of cycle})$$

- Number of task n increase , $n \gg k$ (ignore k-1)

$$S_{\text{ideal}} = \frac{t_n}{t_p}$$

A non-pipeline system takes 50 ns to process a task. The same task can be processed in a six-segment pipeline with a clock cycle of 10ns. Determine the speedup ratio of the pipeline for 100 tasks. What is the maximum speedup that can be achieved?

$$\begin{aligned}t_n &= 50 \text{ nsec} & k &= 6 \\t_p &= 10 \text{ nsec} \\n &= 100\end{aligned}$$

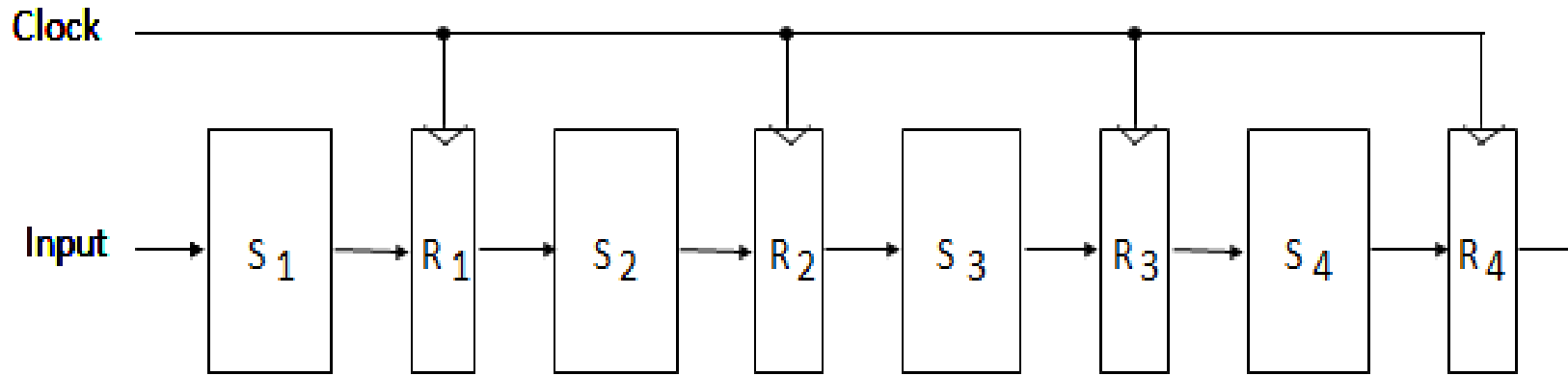
$$S = \frac{n * t_n}{(k+n-1)t_p}$$

$$= \frac{100 * 50}{(6+100-1)10}$$

$$= 4.76$$

$$S_{\max} = \frac{t_n}{t_p} = \frac{50}{10} = 5$$

General structure of four segment pipeline :



General Structure of Four-Segment Pipeline

- Each segment performs partial processing dictated by the way the task is partitioned.
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
- It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline.
- The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

- The technique is efficient for those applications that need to **repeat the same task** many times with **different sets of data**.
- 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.
- The segments are separated by registers R , which hold the intermediate results between the stages.
- Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.
- We define a task as the total operation performed going through all the segments in the pipeline.

Space-Time Diagram

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

Six task T1 To T6 executed in four segments. (Task $n - 6$ & Segment $k - 4$)

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

The first task T1 require a time equal to kt_p .

The remaining $n-1$ tasks require a time equal to $(n-1)t_p$.

Therefore to complete n -tasks using k -segment pipeline requires $k+(n-1)$ clock cycles .

Ex – 4 Segment 6 – tasks

Time required to complete all the operation = $(4 + 6 - 1) = 9$ clock cycles.

Speedup :

Non pipeline unit that perform same operation & takes a time equal to t_n to complete each task.

Total time required for n tasks is nt_n

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

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

As the number of tasks increase, n becomes much larger than $k-1$, and $k+n-1$ approaches the value of n .

The speedup becomes

$$S = \frac{t_n}{t_p}$$

Example

- 4-stage pipeline
- sub operation in each stage; $t_p = 20\text{nS}$
- 100 tasks to be executed
- 1 task in non-pipelined system; $20 \times 4 = 80\text{nS}$

Pipelined System

$$(k + n - 1) \cdot t_p = (4 + 99) \cdot 20 = 2060\text{nS}$$

Non-Pipelined System

$$n \cdot k \cdot t_p = 100 \cdot 4 \cdot 20 = 8000\text{nS}$$

Speedup

$$S_k = 8000 / 2060 = 3.88$$

4-Stage Pipeline is basically identical to the system with 4 identical function units

Floating Point Arithmetic Pipeline :

- Pipeline arithmetic units are usually found in very high speed computers
- They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems
- The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

$$X = A \times 2^a$$

$$Y = B \times 2^b$$

- A and B are two fractions that represent the mantissas
 - a and b are the exponents
- The floating-point addition and subtraction can be performed in four segments show in figure

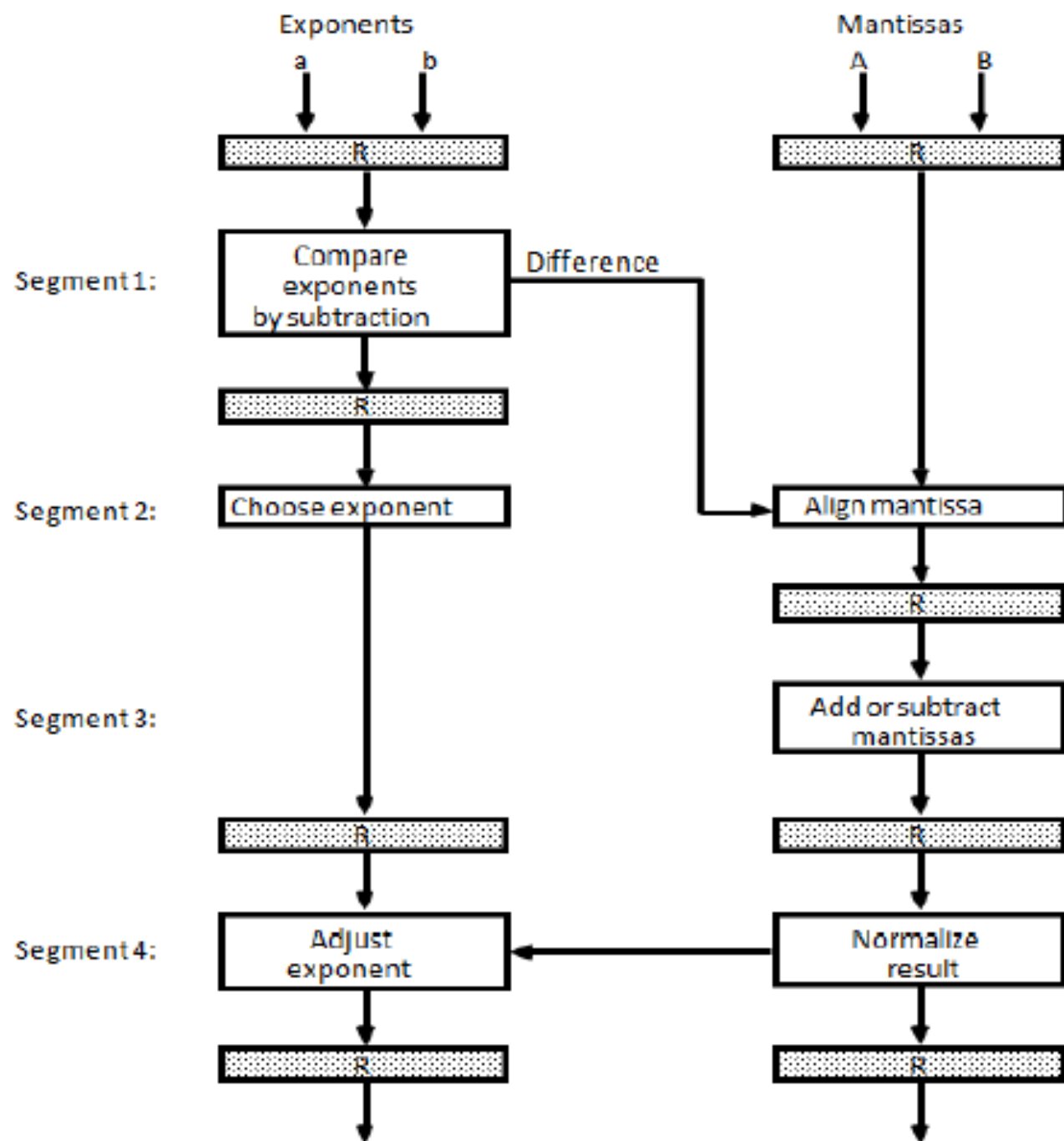


Figure 6.6: Pipeline for floating-point addition and subtraction

- The registers labeled R are placed between the segments to store intermediate results.
- The sub-operations that are performed in the four segments are:
 1. Compare the exponents
 2. Align the mantissas
 3. Add or subtract the mantissas
 4. Normalize the result
- The following numerical example ay clarify the sub-operations performed in each segment.
- Consider the two normalized floating-point numbers:
$$X = 0.9504 \times 10^3$$
$$Y = 0.8200 \times 10^2$$
- The two exponents are subtracted in the first segment to obtain $3 - 2 = 1$.
- The larger exponent 3 is chosen as the exponent of the result.

- The next segment shifts the mantissa of Y to the right to obtain

$$X = 0.9504 \times 10^3$$

$$Y = 0.0820 \times 10^3$$

- This aligns the two mantissas under the same exponent. The addition of the two mantissas in segment 3 produces the sum

$$Z = 1.0324 \times 10^3$$

- The sum is adjusted by normalizing the result so that it has a fraction with nonzero first digit. This is done by shifting the mantissa once to the right and incrementing the exponent by one to obtain the normalized sum.

$$Z = 0.10324 \times 10^4$$

Segment 1: The two exponents are subtracted in the first segment to obtain i.e. $3-2=1$

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

Segment 2: shifts the mantissa of Y to the right to obtain $Y = 0.0820 \times 10^3$

The mantissas are now aligned

Segment 3: produces the sum $Z = 1.0324 \times 10^3$

Segment 4: normalizes the result by shifting the mantissa once to the right and incrementing the exponent by one to obtain $Z = 0.10324 \times 10^4$

- Suppose that the time delays of 4 segments are $t_1 = 60 \text{ ns}$, $t_2 = 70 \text{ ns}$, $t_3 = 100 \text{ ns}$ & $t_4 = 80 \text{ ns}$ and the interface **register delay of $t_r = 10 \text{ ns}$** .
- The clock cycle is to be $t_p = \max(t_1, t_2, t_3, t_4) + t_r$
 $= 100 + 10 = 110 \text{ ns}$
- **The non pipeline floating point**
The clock cycle is to be $t_n = 60 + 70 + 100 + 80 + 10 = 320 \text{ ns}$
- **Speedup** $= 320 / 110 = 2.9$

Instruction Pipelining :

- Pipeline processing can occur in data stream as well as in instruction stream.
- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- This causes the instruction fetch and executes phases to overlap and perform simultaneous operations.
- One possible digression associated with such a scheme is that an **instruction may cause a branch** out of sequence.
- In that case the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded.
- Consider a computer with an instruction fetch unit and an instruction execution unit designed to provide a two-segment pipeline.
- The instruction fetch segment can be implemented by means of a first-in, first-out (FIFO) buffer.
- The buffer acts as a queue from which control then extracts the instructions for the execution unit.

Instruction Pipelining :

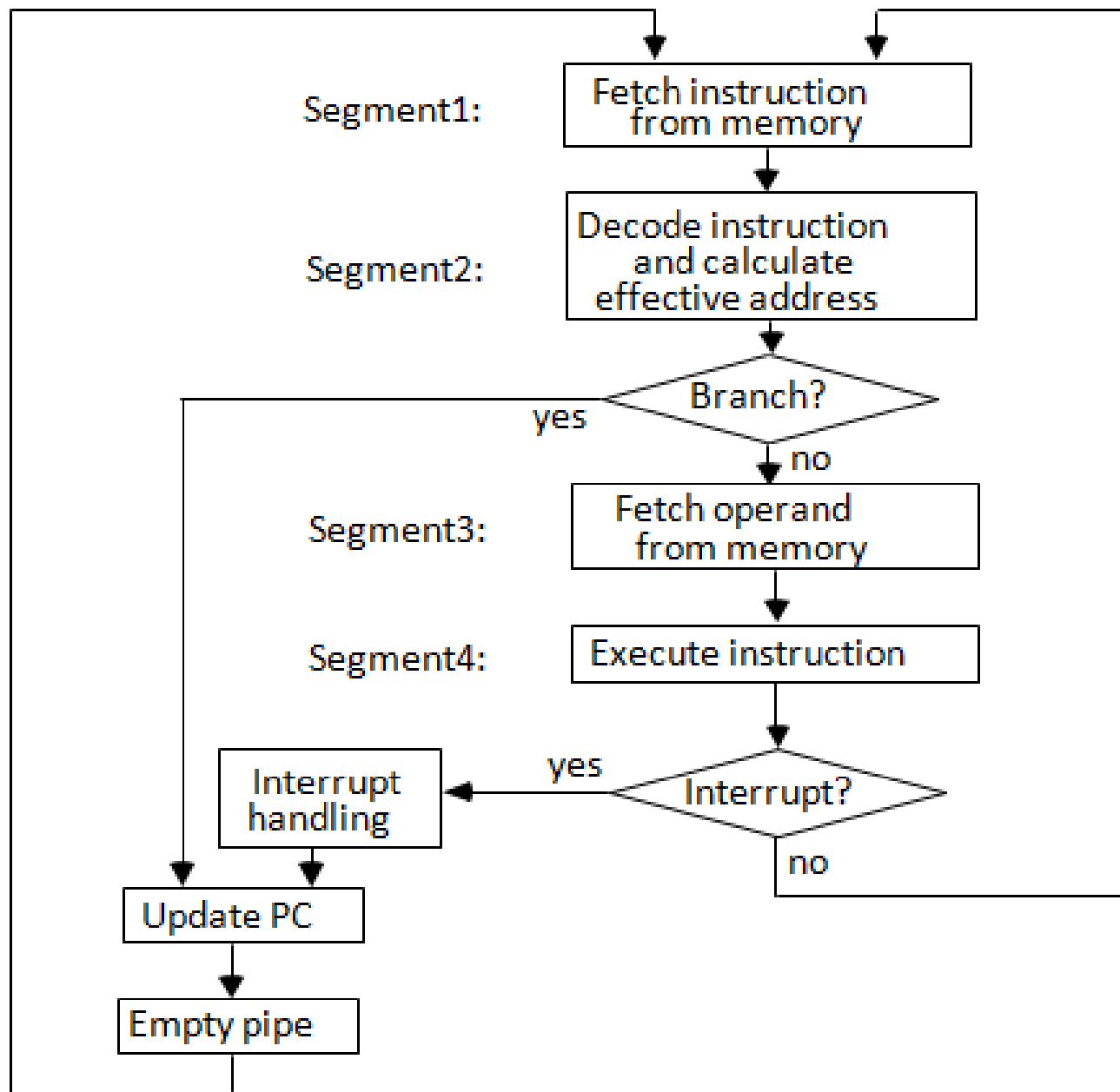
- Six Phases* in an Instruction Cycle

- [1] **Fetch** an instruction from memory
- [2] **Decode** the instruction
- [3] **Calculate the effective address of the operand**
- [4] **Fetch the operands** from memory
- [5] **Execute** the operation
- [6] **Store the result in the proper place**

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

==> 4-Stage Pipeline

- [1] FI: Fetch an instruction from memory
- [2] DA: Decode the instruction and calculate the effective address of the operand
- [3] FO: Fetch the operand
- [4] EX: Execute the operation



(a) Four-segment CPU pipeline

		Clock Cycle												
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

(b) Timing of Instruction Pipeline

- **Figure (a)** shows, how the instruction cycle in the CPU can be processed with a four segment pipeline.
- While an instruction is being **executed in segment 4**, the next instruction in sequence is busy **fetching an operand from memory in segment 3**.
- The effective address may be calculated in a separate arithmetic circuit for the third instruction, and whenever the memory is available, the fourth and all subsequent instructions can be fetched and placed in an instruction FIFO.
- Thus up to four sub operations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.
- **Figure (b)** shows the operation of the instruction pipeline. The time in the horizontal axis is divided into steps of equal duration. The four segments are represented in the diagram with an abbreviated symbol.
- It is assumed that the processor has separate instruction and data memories so that the operation in FI and FO can proceed at the same time.

- Thus, in 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**.
- 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.
- Another delay may occur in the pipeline if the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand.
- In that case, segment FO must wait until segment EX has finished its operation.

Pipeline Conflicts : Pipeline Hazards (difficulties)

- Situations that prevent the next instruction from being executing during its designated clock cycle.
- Hazards lead to have stall cycles.
- **Three major difficulties :**
 - (1) Resource conflict / Structural hazard
 - (2) Data dependency / Data hazard
 - (3) Branch Difficulty / Control hazard

(1) Resource conflict / Structural hazard

- Two different inputs try to use same resource at same time.

MUL IF ID OF EX EX EX WR

ADD IF ID OF --- --- EX WR

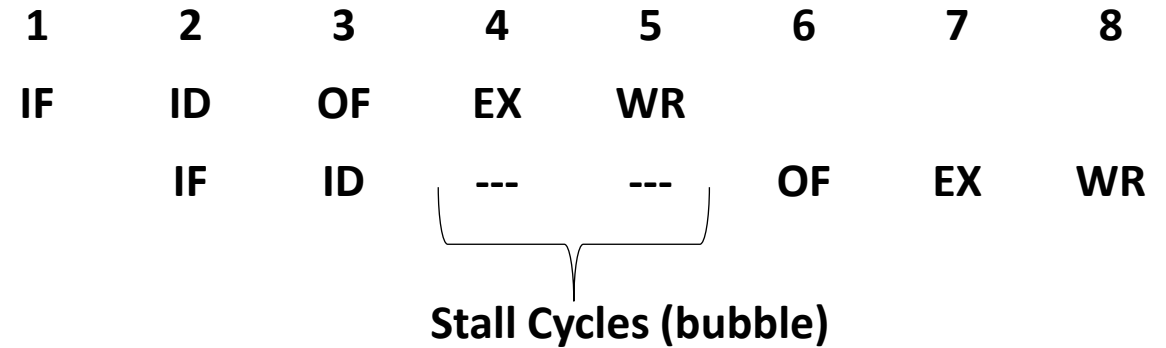
- Resource conflicts caused by access to memory by two segments at the same time.

(2) Data dependency / Data hazard :

- An instruction scheduled to be executed in the pipeline requires the result of a previous instruction, which is not yet available.
- Result of an instruction is used as input in next instruction.

I : R1 ← R2 * R3

I+1 : R4 ← R1 + R5



Solution :

By Software : Compiler → (1) Delayed Load

By Hardware :
(2) Hardware Interlock
(3) Operand forwarding

(1) Delayed Load :

- Either insert independent instructions or no operation instructions between dependent instruction.

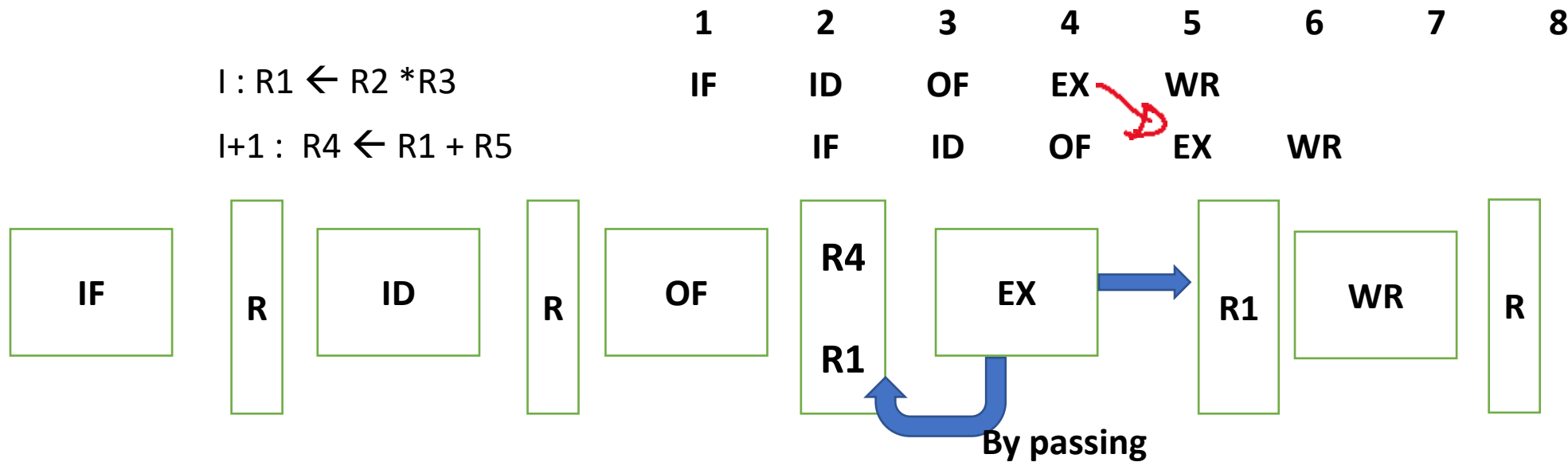
	1	2	3	4	5	6	7	8
$R1 \leftarrow R2 * R3$	IF	ID	OF	EX	WR			
No Operation		IF	ID	OF	EX	WR		
No Operation			IF	ID	OF	EX	WR	
$R4 \leftarrow R1 + R5$				IF	ID	OF	EX	WR

(2) Hardware Interlock :

- Stall cycles created

	1	2	3	4	5	6	7	8
I : $R1 \leftarrow R2 * R3$	IF	ID	OF	EX	WR			
I+1 : $R4 \leftarrow R1 + R5$		IF	ID	---	---	OF	EX	WR

(3) Operand forwarding : (bypassing)

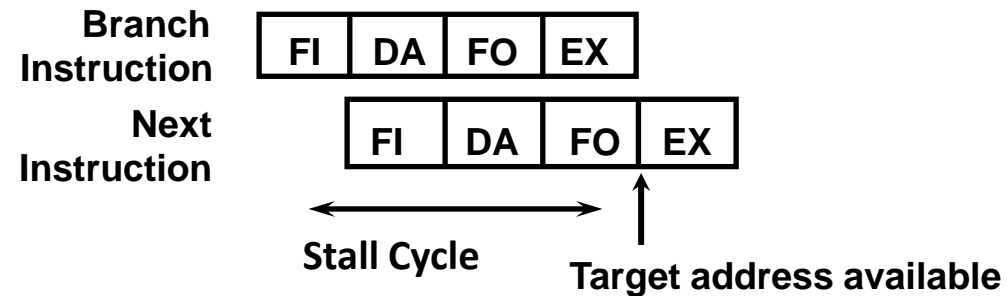


ALU to ALU data dependency only. (No stall cycle) → If Operand fetch from memory then stall cycle is there.

Expensive – Hardware required to take decision for dependency

(3) Branch Difficulty / Control hazard :

- Hazards because of branch instructions.
- Branch target address is not known until the branch instruction is completed

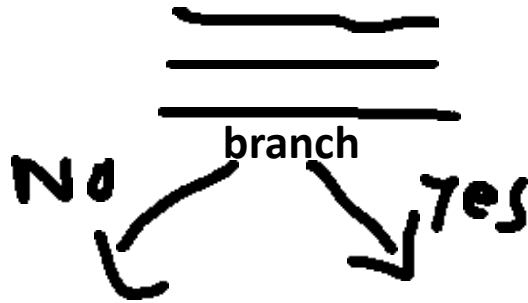


Dealing with Control Hazards

- * Prefetch Target Instruction
- * Branch Target Buffer
- * Loop Buffer
- * Branch Prediction
- * Delayed Branch

Prefetch Target Instruction :

- Fetch instructions in both streams, branch not taken and branch taken
- Both are saved until branch is executed. Then, select the right instruction stream and discard the wrong stream
- 2 pipeline will go together.



Branch Prediction :

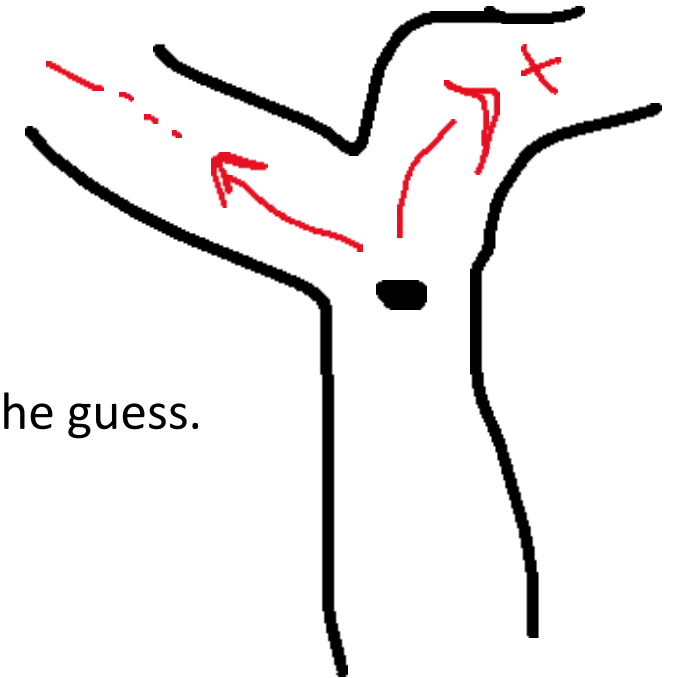
- Guessing the branch condition, and fetch an instruction stream based on the guess.
- Correct guess eliminates the branch penalty

Static : Branch not taken always

Branch taken always

Dynamic : Based on behaviour, current situation

80-90 % correct prediction



Loop Buffer :

- High Speed Register file
- Storage of entire loop that allows to execute a loop without accessing memory

Branch Target Buffer : BTB; Associative Memory

Entry: Address of previously executed branches;
Target instruction and the next few instructions

- When fetching an instruction, search BTB.
If found, fetch the instruction stream in BTB;
If not, new stream is fetched and update BTB

Delayed Branch :

- Compiler detects the branch and rearranges the instruction sequence by inserting useful instructions that keep the pipeline busy in the presence of a branch instruction

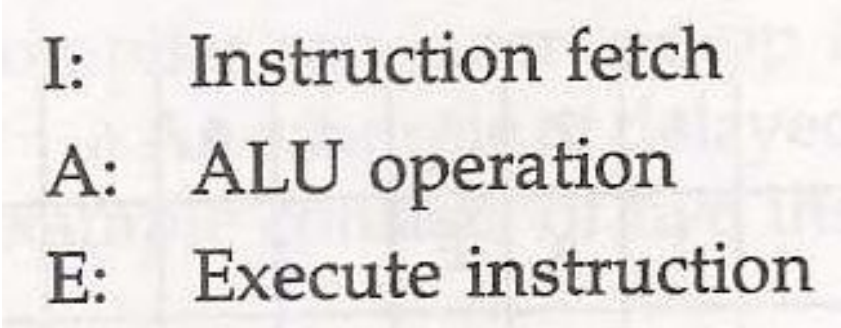
RISC Pipeline :

- Reduced Instruction Set Computer
- RISC is its ability to use an efficient instruction pipeline.
- Simplicity of instruction – small number of sub operations – executed in one clock cycle.
- Because of fixed-length instruction format, the decoding of the operation occur at the same time as the register selection.
- All **data manipulation** instruction have register-to-register operations.
- All operands in register – no calculate effective address or fetching from memory
- So instruction pipeline can be implemented with 2 or 3 segments.
- One segment – fetch instruction from memory
- Other segment – execute instruction in ALU
- Third segment – store result of ALU operation in a destination register.

- The **data transfer instruction** in RISC are limited to load & store instructions.
- These instruction use register indirect addressing mode.
- Need 3 or 4 segments in pipelining.
- To prevent conflicts – fetch instruction & load or store an operand – most RISC machines uses two separates buses with two separate memory – one storing instruction & other storing data.
- Major advantage of RISC is its ability to execute instructions at the rate of **one per clock cycle**.
- The **advantage RISC over CISC** is that RISC can achieve pipeline segments, requiring just one clock cycle, while CISC uses many segments in its pipeline, with longest segment requiring two or more clock cycles. (**Single-cycle instruction execution**)
- **Compiler support** – data conflicts & branch penalties

Example : Three-segment Instruction Pipeline

- The instruction cycle can be divided into 3 suboperation and implemented in 3 segments



I: Instruction fetch
A: ALU operation
E: Execute instruction

- I – segment fetch instruction from memory
- A – segment Instruction decoded & ALU operation is performed
- **ALU used for three function** – depending on decoded instruction
- Data manipulation instruction – effective address for load & store instruction
 - branch address for a program control instruction
- E – segment directs the **output of the ALU to one of three destinations**
 1. Register file, 2. it transfer the EA to a data memory for load or store
 3. transfer the branch address to PC.

Delayed LOAD

Consider now the operation of the following four instructions:

1. LOAD: $R1 \leftarrow M[\text{address } 1]$
2. LOAD: $R2 \leftarrow M[\text{address } 2]$
3. ADD: $R3 \leftarrow R1 + R2$
4. STORE: $M[\text{address } 3] \leftarrow R3$

Clock cycles:	1	2	3	4	5	6
1. Load $R1$	I	A	E			
2. Load $R2$		I	A	E		
3. Add $R1 + R2$			I	A	E	
4. Store $R3$				I	A	E

(a) Pipeline timing with data conflict

Clock cycle:	1	2	3	4	5	6	7
1. Load $R1$	I	A	E				
2. Load $R2$		I	A	E			
3. No-operation			I	A	E		
4. Add $R1 + R2$				I	A	E	
5. Store $R3$					I	A	E

(b) Pipeline timing with delayed load

Figure 9-9 Three-segment pipeline timing.

The advantage of the delayed load approach is that the data dependency is taken care by the **compiler** rather than the hardware.

Delayed Branch :

Load from memory to $R1$

Increment $R2$

Add $R3$ to $R4$

Subtract $R5$ from $R6$

Branch to address X

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. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in X								I	A	E

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

(b) Rearranging the instructions

Figure 9-10 Example of delayed branch.

Vector Processing :

- There is a class of computational problems that are beyond the capabilities of a conventional computer.
- These problems are characterized by the fact that they require a vast number of computations that will take a conventional computer days or even weeks to complete.
- In many science and engineering applications, the problems can be formulated in terms of vectors and matrices that lend themselves to vector processing.
- **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 (computer)

- Ability to process vectors, and related data structures such as matrices and multi-dimensional arrays, much faster than conventional computers
- Vector Processors may also be pipelined

Vector Operations

- Many scientific problems require arithmetic operations on large arrays of numbers.
- These numbers are usually formulated as vectors and matrices of floating-point numbers.
- A vector is an ordered set of a one-dimensional array of data items.
- A vector V of length n is represented as a row vector by $V = [V_1 \ V_2 \ V_3 \ \dots \ V_n]$ that may be represented as a column vector if the data items are listed in a column.
- A conventional sequential computer is capable of processing operands one at a time.
- Consequently, operations on vectors must be broken down into single computations with subscripted variables.
- The element of vector V is written as $V(I)$ and the index I refers to a memory address or register where the number is stored.


```
DO 20 I = 1, 100
20 C(I) = B(I) + A(I)
```

Conventional computer

```
Initialize I = 1
20 Read A(I)
   Read B(I)
   Store C(I) = A(I) + B(I)
   Increment I = i + 1
   If I ≤ 100 goto 20
```

Vector computer

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

Matrix Multiplication :

- Matrix multiplication is one of the most computational intensive operations performed in computers with vector processors.
- An $n \times m$ matrix of numbers has n rows and m columns and may be considered as constituting a set of n row vectors or a set of m column vectors.
- Consider, for example, the multiplication of two 3×3 matrices A and B .

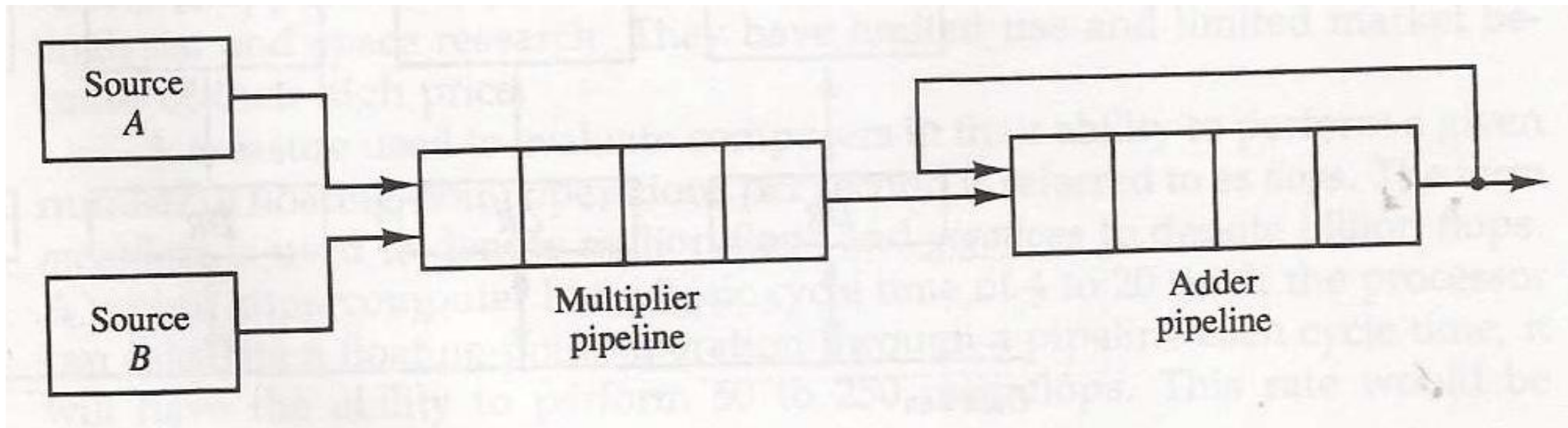
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23}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

- For example, the number in the first row and first column of matrix C is calculated by letting $i = 1, j = 1$, to obtain

$$C_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$

- This requires three multiplications and three additions.
- The total number of multiplications or additions required to compute the matrix product is $9 \times 3 = 27$.

- In general, the inner product consists of the sum of k product terms of the form
$$C = A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4 + \dots + A_kB_k$$
- In a typical application k may be equal to 100 or even 1000. The inner product calculation on a pipeline vector processor is shown in following figure.



- The values of A and B are either in memory or in processor registers.
- The floating point multiplier pipeline and the floating point adder pipeline are assumed to have four segments each.

- All segment registers in the multiplier and adder are initialized to 0.
- Therefore, the output of the adder is 0 for the first eight cycles until both pipes are full.

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

- When there are no more product terms to be added, the system inserts four zeros into the multiplier pipeline.
- The adder pipeline will then have one partial product in each of its four segments, corresponding to the four sums listed in the four rows in the above equation.
- The four partial sums are then added to form the final sum.

Memory Interleaving : Memory Module

- A single memory module cause sequentialization of a access.

Eg. Only one memory access can be performed at a time.

Hence, throughput may be drop.

- Memory interleaving is a technique to increase the throughput.
- Here, the system is divided into number of independent modules, which answers read or write requests independently in parallel.
- **How it is done?**
 - By spreading memory address evenly across memory modules.
 - By interleaving the address space (memory address) such that consecutive words in the address space are assigned to different memory modules.

{	0000	10
	0001	20
	0010	30
	0011	40
{	0100	50
	0101	60
	0110	70
	0111	80
{	1000	90
	1001	100
	1010	110
	1011	120
{	1100	130
	1101	140
	1110	150
	1111	160

MM : 00

00	10
01	20
10	30
11	40

MM : 01

00	50
01	60
10	70
11	80

MM : 10

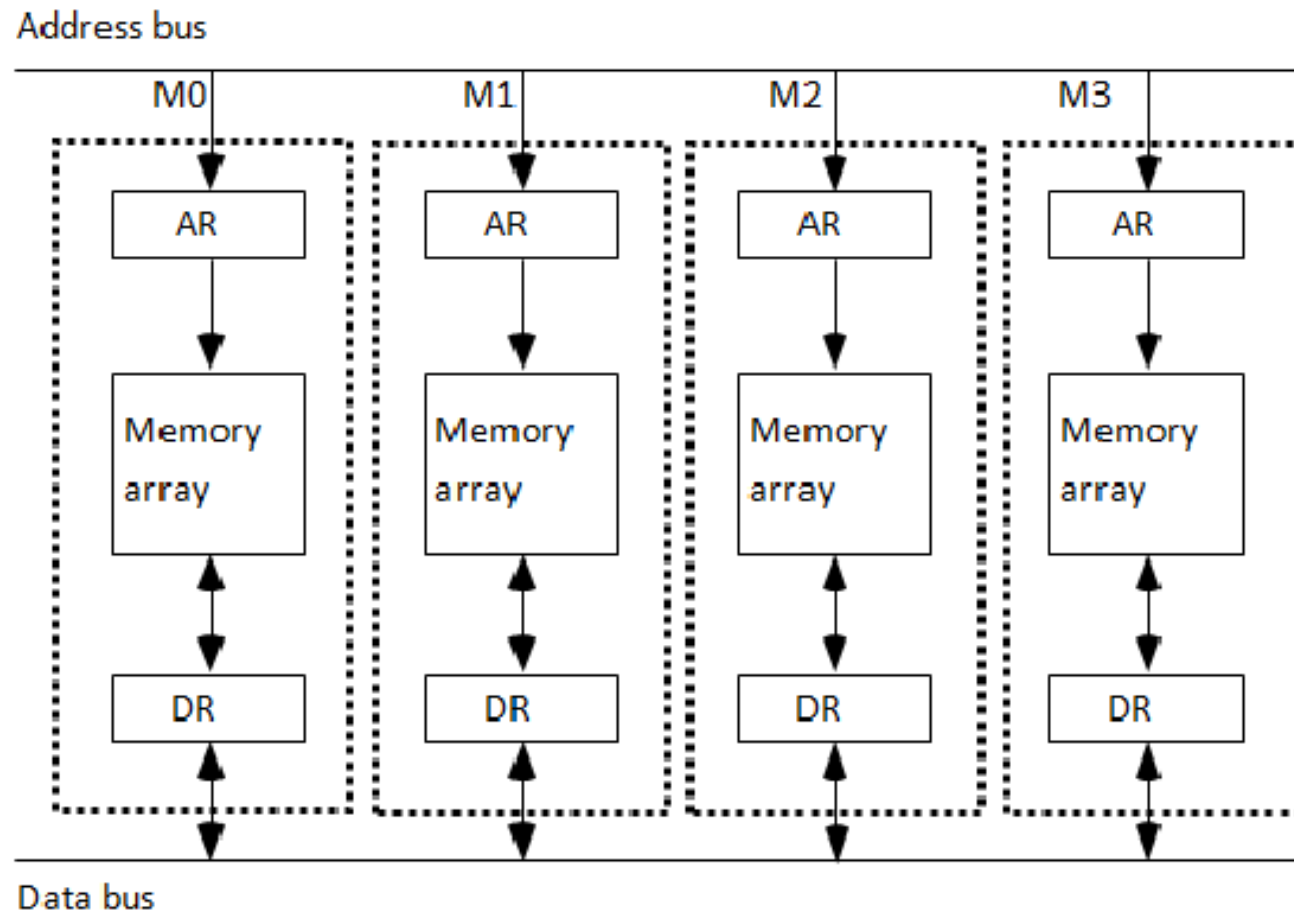
00	90
01	100
10	110
11	120

MM : 11

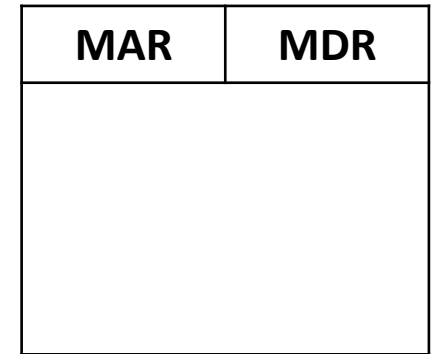
00	130
01	140
10	150
11	160

Example : 4-way interleaved memory (Divide memory into 4 module)

- Memory module is a memory array together with its address & data register.
- Instead of using two memory buses for simultaneous access, the memory can be partitioned into a number of modules connected to a common memory address and data buses.



Multiple module memory organization



Memory Module

- The modular system permits one module to initiate a memory access while other modules are in the process of reading or writing a word and each module can honor a memory request independent of the state of the other modules.
- In an interleaved memory, different sets of addresses are assigned to different memory modules.

Advantage :

System performance is enhanced because read & write activity occurs simultaneously across the multiple modules in a similar fashion.

- There are 2 address formats for memory interleaving the address space :

1. High-order Interleaving :

It uses the high-order bits as module address & the lower-order bits as the word address within each module.



MM : 00	
00	10
01	20
10	30
11	40

2. Low-order Interleaving :

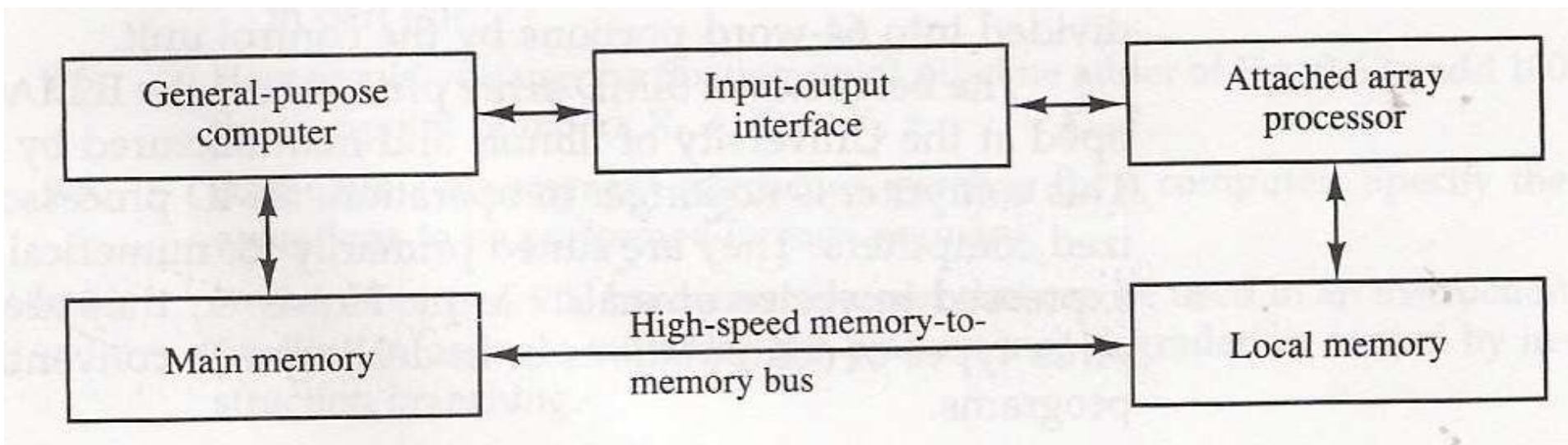
It uses the lower-order bits as module address & the high-order bits as the word address within each module.

Array Processor :

- An array processor is a processor that performs computations on large arrays of data.
- The term is used to refer to two different types of processors, attached array processor and SIMD array processor.
- An attached array processor is an auxiliary processor attached to a general-purpose computer.
- It is intended to improve the performance of the host computer in specific numerical computation tasks.
- An SIMD array processor is a processor that has a single-instruction multiple-data organization.
- It manipulates vector instructions by means of multiple functional units responding to a common instruction.

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.
- It achieves high performance by means of parallel processing with multiple functional units.
- It includes an arithmetic unit containing one or more pipelined floating-point adders and multipliers.
- The array processor can be programmed by the user to accommodate a variety of complex arithmetic problems.

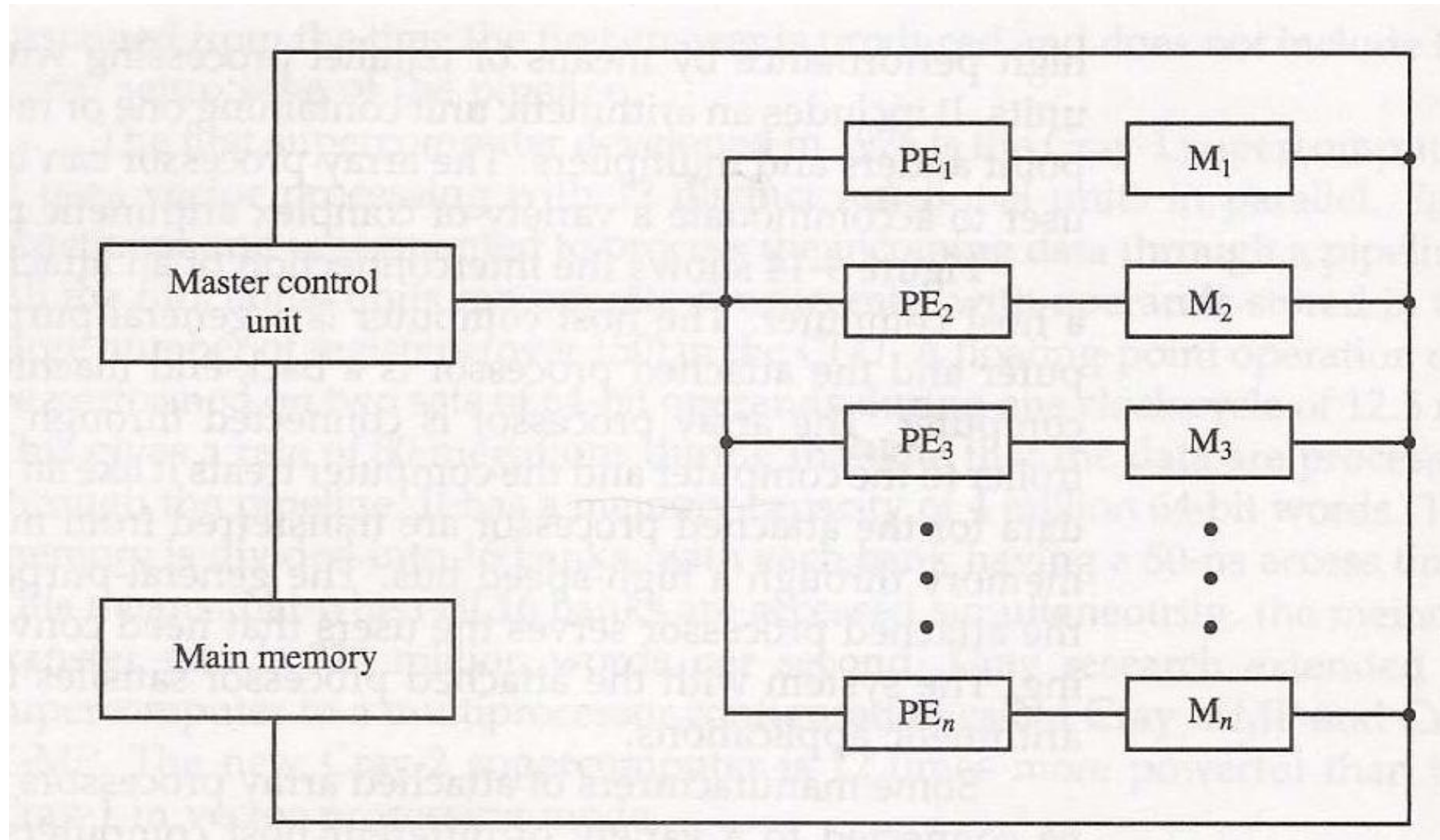


Attached array processor with host computer.

- The host computer is a general-purpose commercial computer and the attached processor is a back-end machine driven by the host computer.
- The array processor is connected through an input-output controller to the computer and the computer treats it like an external interface.
- The data for the attached processor are transferred from main memory to a local memory through a high-speed bus.
- The general-purpose computer without the attached processor serves the users that need conventional data processing.
- The system with the attached processor satisfies the needs for complex arithmetic applications.

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.



SIMD array processor organization.

- It contains a set of identical **processing elements** (PEs), each having a local memory M.
 - Each processor element includes an ALU, a floating-point arithmetic unit and working registers.
 - The master control unit controls the operations in the processor elements.
 - The main memory is used for storage of the program.
-
- The function of the master control unit is to decode the instructions and determine how the instruction is to be executed.
 - Scalar and program control instructions are directly executed within the master control unit.
 - Vector instructions are broadcast to all PEs simultaneously.
-
- Vector operands are distributed to the local memories prior to the parallel execution of the instruction.
 - Masking schemes are used to control the status of each PE during the execution of vector instructions.

- Each PE has a flag that is set when the PE is active and reset when the PE is inactive.
- This ensures that only that PE'S that needs to participate is active during the execution of the instruction.
- SIMD processors are highly specialized computers.
- They are suited primarily for numerical problems that can be expressed in vector matrix form.
- However, they are not very efficient in other types of computations or in dealing with conventional data-processing programs.