UNIT 5
Pipelining and vector Processing
LH 6

सरोज थापा

Parallel Processing

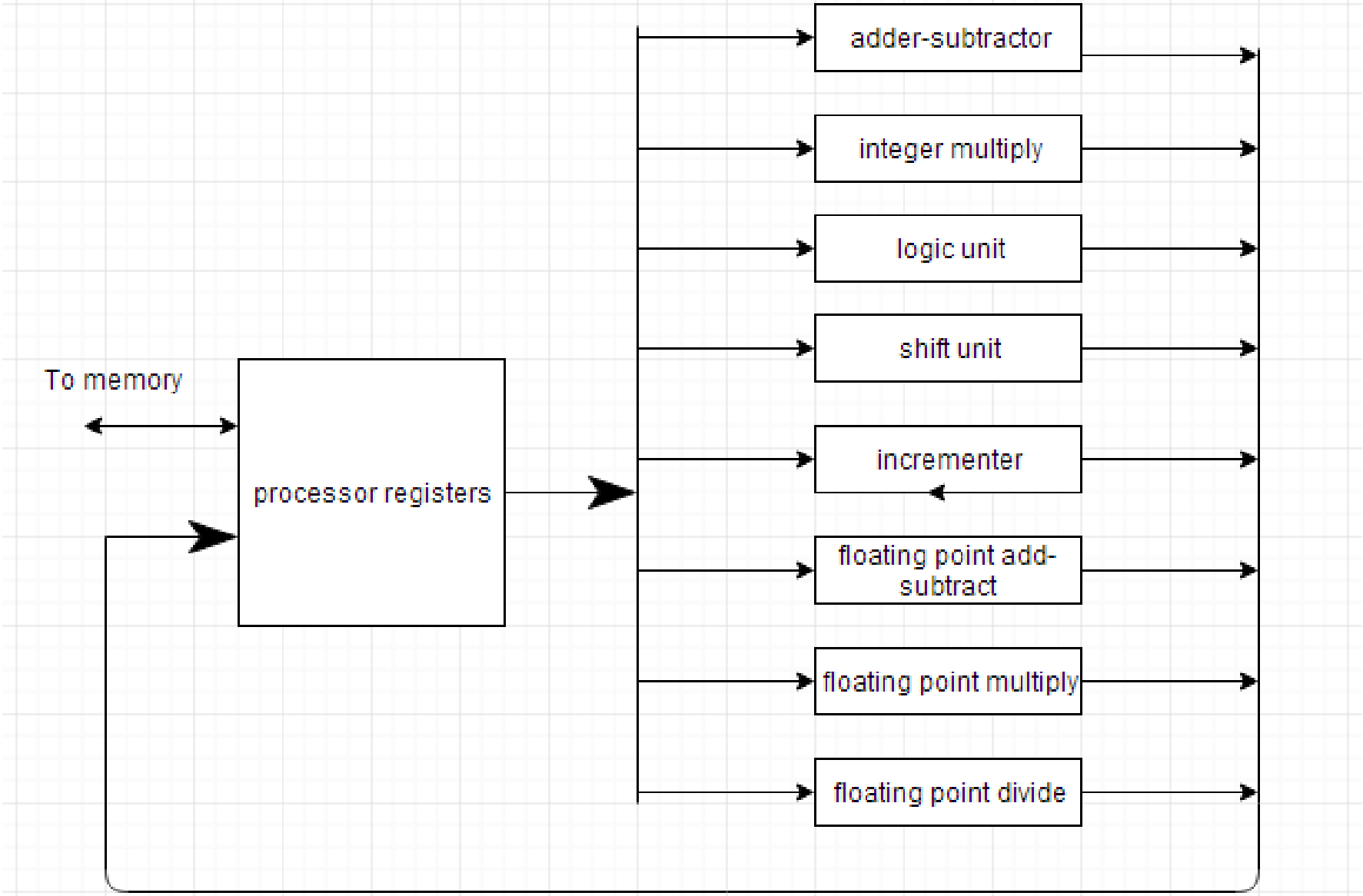
- Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.
- Instead of processing each instruction sequentially as in a conventional computer, a parallel processing system is able to perform concurrent data processing to achieve faster execution time.
- For example, while an instruction is being executed in the ALU, the next instruction can be read from memory. The system may have two or more ALUs and be able to execute two or more instructions at the same time..

Cont..

- Furthermore, the system may have two or more processors operating concurrently.
- The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time.
- The amount of hardware increases with parallel processing. and with it, the cost of the system increases.
- However, technological developments have reduced hardware costs to the point where parallel processing techniques are economically feasible

Processor with multifunctional units

- Parallel processing is established by distributing the data among the multiple functional units. For example, the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.
- The adder and integer multiplier perform the arithmetic operations with integer numbers.
- The floating-point operations are separated into three circuits operating in parallel.
- The logic, shift, and increment operations can be performed concurrently on different data.
- All units are independent of each other, so one number can be shifted while another number is being incremented.
- A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components.



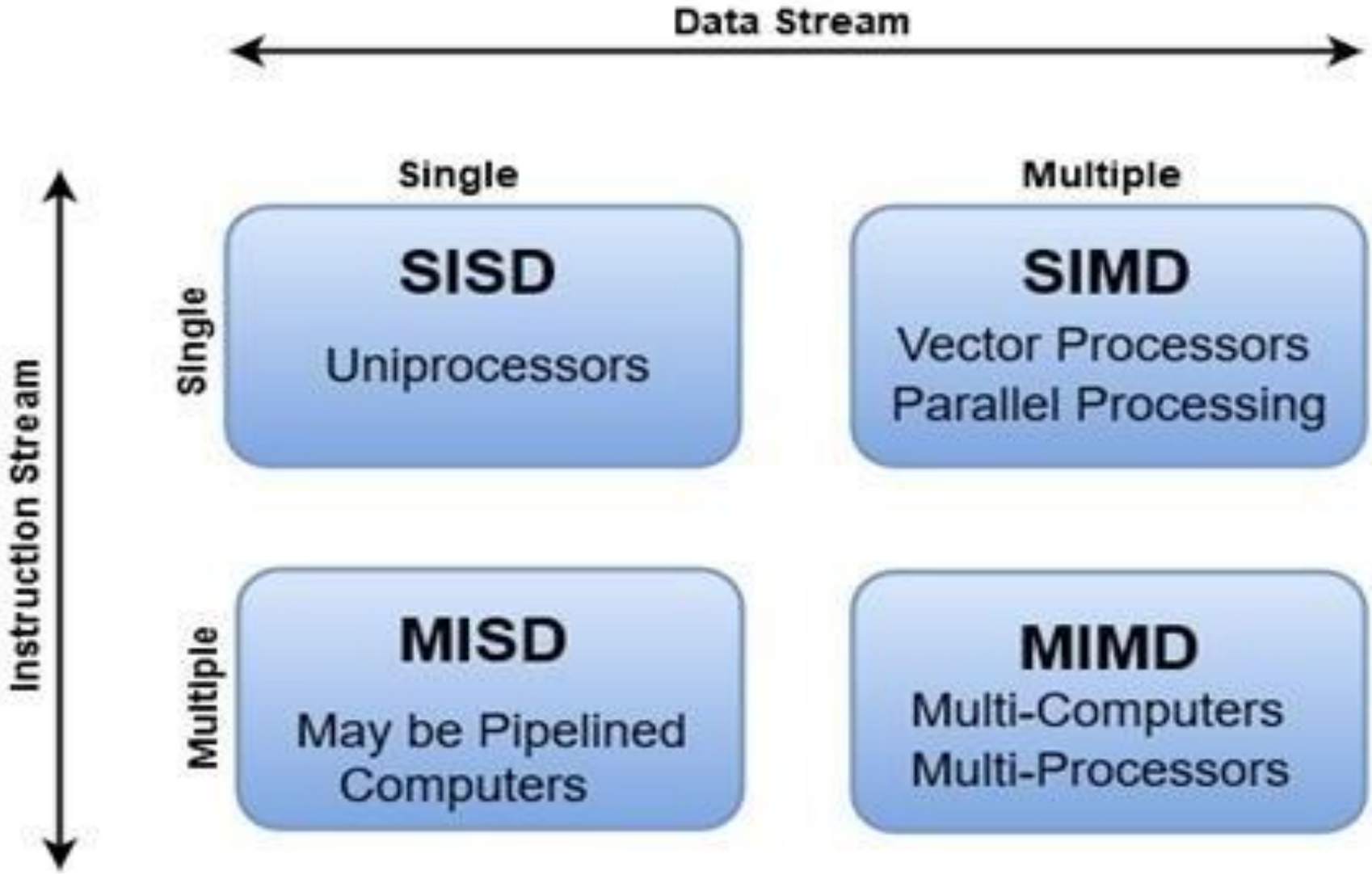
PROCESSOR WITH MULTIPLE FUNCTIONAL UNITS

Flynn's Classification of Parallel Processing

- There are a variety of ways that parallel processing can be classified. It can be considered from the internal organization of the processors, from the interconnection structure between processors, or from the flow of information through the system.
- One classification introduced by **M. J. Flynn** considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

- The normal operation of a computer is to fetch instructions from memory and execute them in the processor. The sequence of instructions read from memory constitutes an instruction stream. The operations performed on the data in the processor constitutes a data stream. Parallel processing may occur in the instruction stream, in the data stream, or in both.
- Flynn's classification divides computers into four major groups as follows:

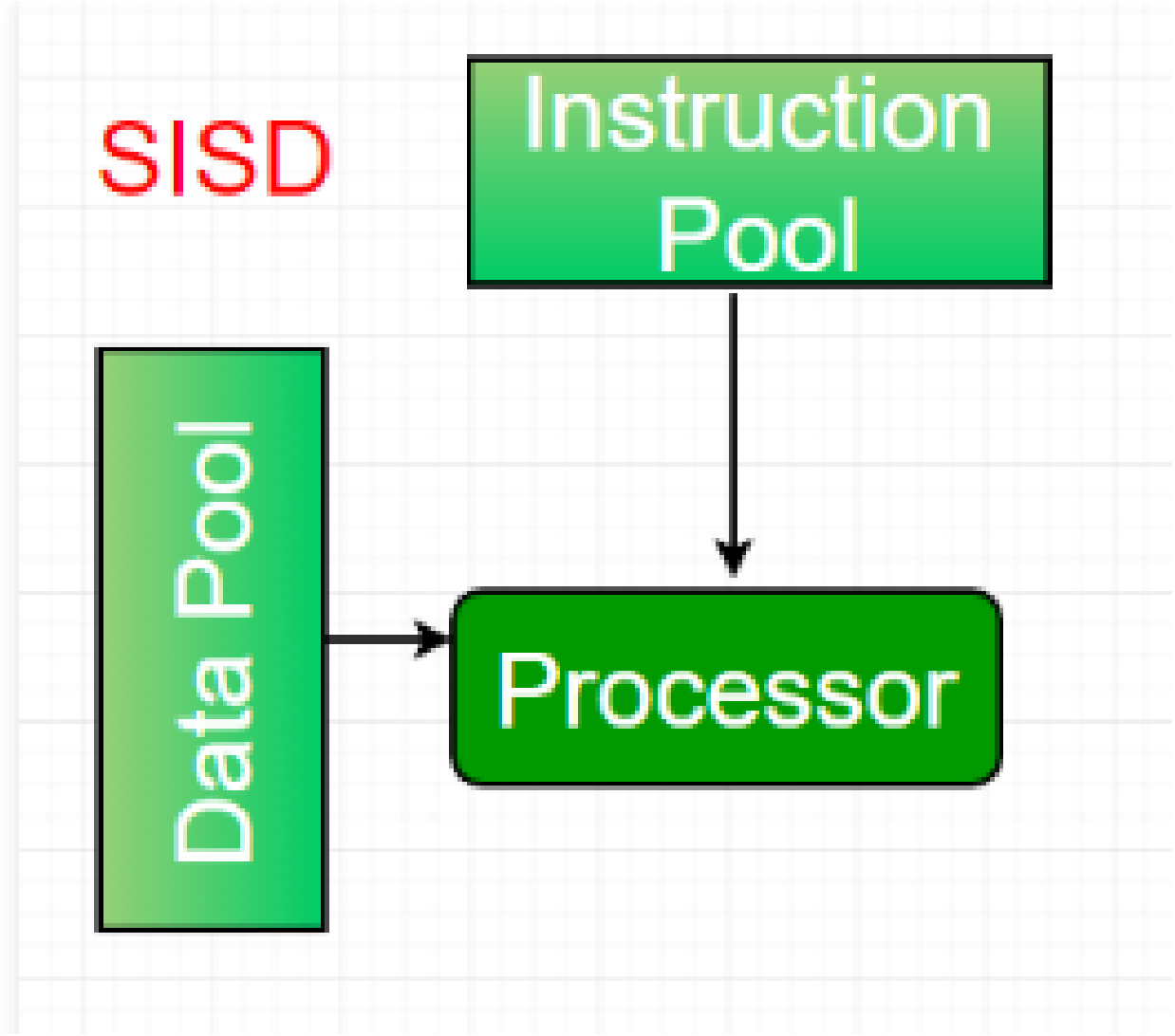
1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction stream, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)


Data Stream
Single
Multiple
Instruction Stream
Single
SISD
Uniprocessors
SIMD
Vector Processors
Parallel Processing
Multiple
MISD
May be Pipelined
Computers
MIMD
Multi-Computers
Multi-Processors

सरोज थापा

1. Single-instruction, single-data (SISD) systems –

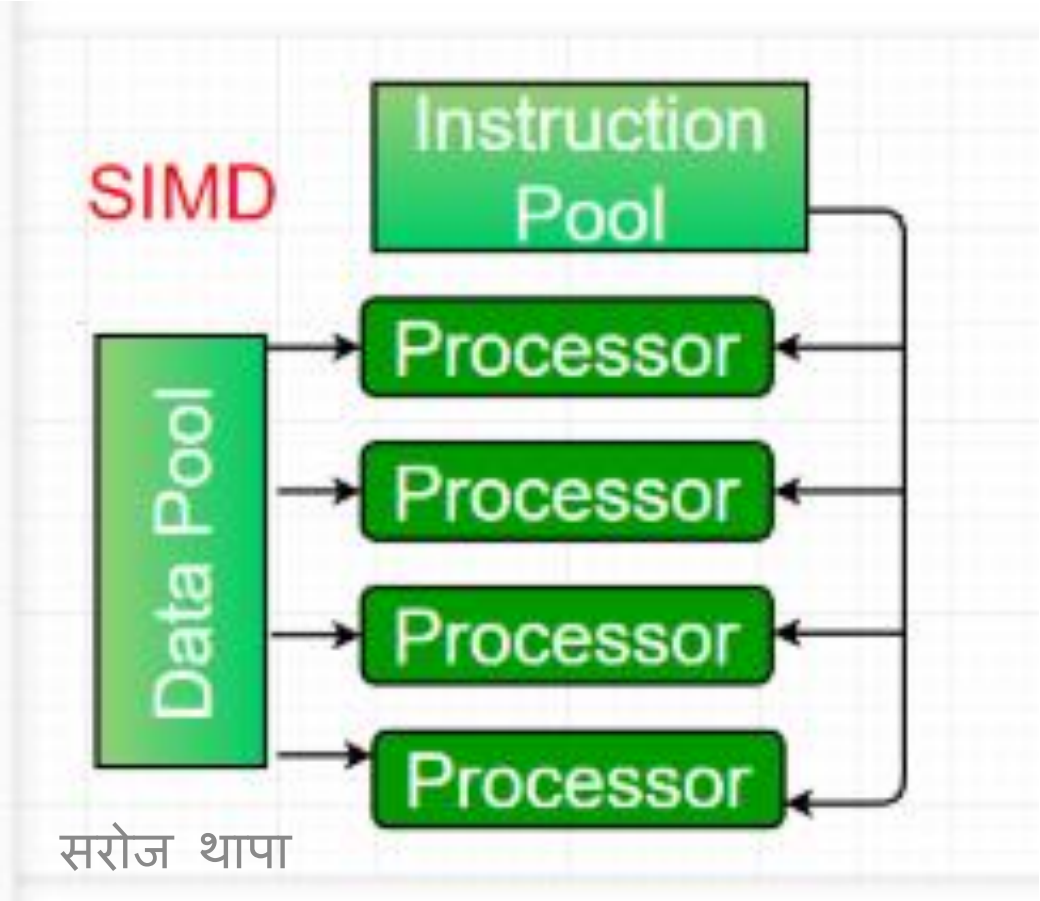
- An SISD computing system is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream.
- In SISD, machine instructions are processed in a sequential manner and computers adopting this model are popularly called sequential computers.
- Most conventional computers have SISD architecture.
- All the instructions and data to be processed have to be stored in primary memory.
- The speed of the processing element in the SISD model is limited(dependent) by the rate at which the computer can transfer information internally.
- Dominant representative SISD systems are IBM PC, workstations.


SISD
Instruction Pool
Data Pool
Processor

2. SIMD represents an organization that includes many processing units under the supervision of a common control unit.

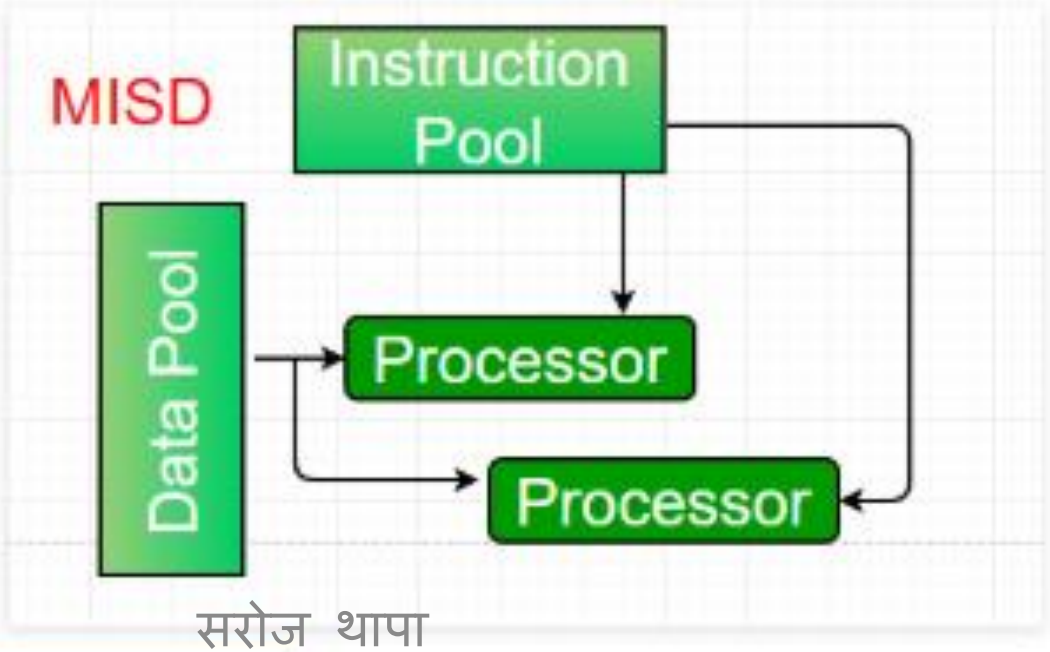
- All processors receive the same instruction from the control unit but operate on different items of data.
- The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

SIMD systems is Cray's vector processing machine.

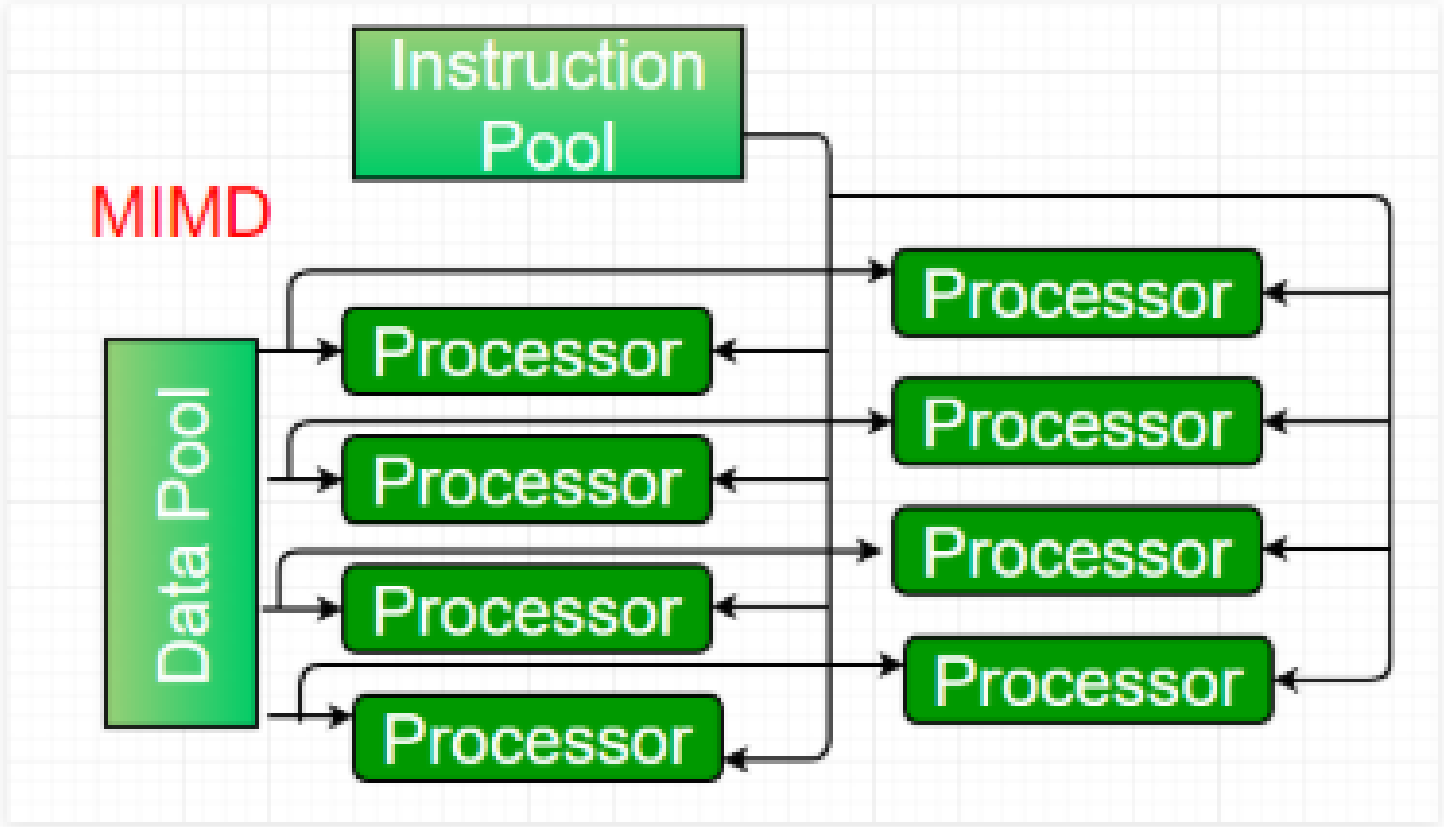


3. An MISD computing system is a multiprocessor machine capable of executing different instructions on different PEs but all of them operating on the same dataset .

- **MISD** structure is only of theoretical interest since no practical system has been constructed using this organization.
- Example Z = sin(x)+cos(x)+tan(x)



4. An **MIMD** system is a multiprocessor machine which is capable of executing multiple instructions on multiple data sets Most multiprocessor and multicomputer systems can be classified in this category.



Pipelining

- Pipelining is a technique of decomposing a sequential process into sub-operations, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments.
- A pipeline can be visualized as a collection of processing segments through which binary information flows.
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.
- 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.

Pipelining Example

The pipeline organization will be demonstrated by means of a simple example. Suppose that we want to perform the combined multiply and add operations with a stream of numbers.

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

Each sub-operation is to be implemented in a segment within a pipeline. Each segment has one or two registers and a combinational circuit as shown in Fig.

R1 through R5 are registers that receive new data with every clock pulse. The multiplier and adder are combinational circuits. The sub-operations performed in each segment of the pipeline are as follows:

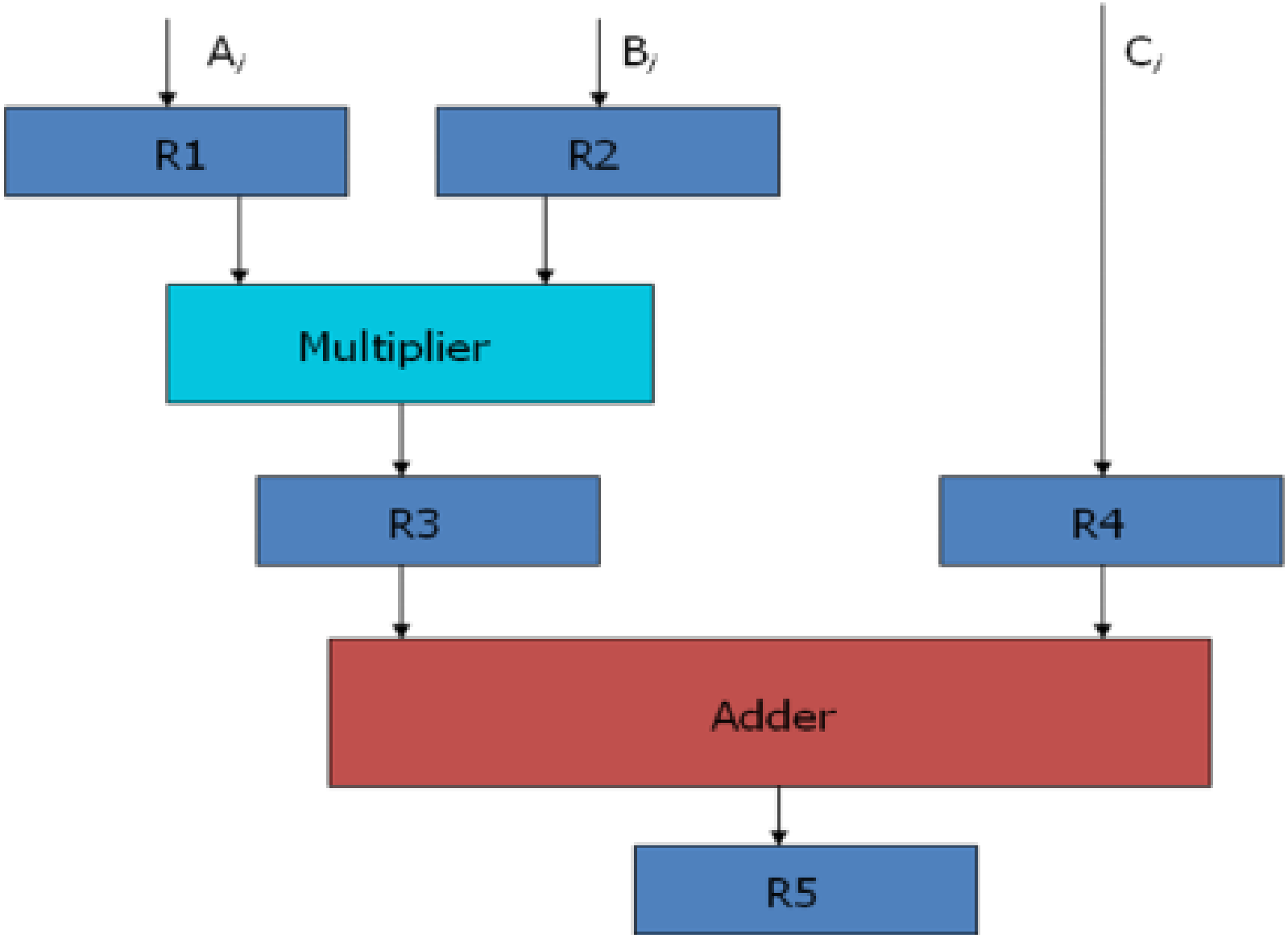


Fig 4-1: Example of pipeline processing

- Each sub operation is to be implemented in a segment within a pipeline.

$R1 \leftarrow A_i,\ R2 \leftarrow B_i$ Input A_i and B_i
$R3 \leftarrow R1 * R2,\ R4 \leftarrow C_i$ Multiply and input C_i
$R5 \leftarrow R3 + R4$ Add C_i to product

- The five registers are loaded with new data every clock pulse. The effect of each clock is shown in Table.

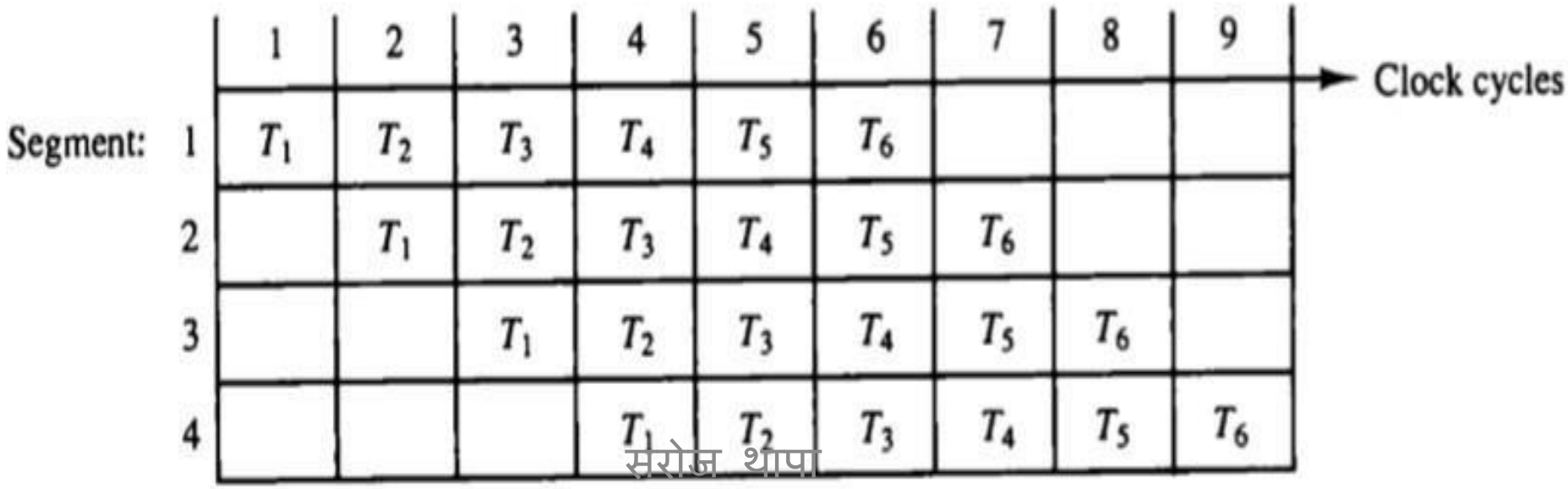
Content of Registers in Pipeline Example

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	--	--	--	--	$A_7*B_7+C_7$

Table 4-1: Content of Registers in Pipeline Example

- A **task** is defined as the total operation performed going through all the segments in the pipeline. The behavior of a pipeline can be illustrated with a **space-time** diagram. It shows the segment utilization as a function of time. The space-time diagram of a 4 segment pipeline is given below:
- Figure: Space time diagram of 4 segment and 6 tasks

Figure 9-4 Space-time diagram for pipeline.



Speedup equation

Consider a **'k'** segment pipeline with clock cycle time as **'Tp'.** Let there be 'n' tasks to be completed in the pipelined processor. Now, the first instruction is going to take 'k' cycles to come out of the pipeline but the other 'n – 1' instructions will take only '1' cycle each, So, time taken to execute 'n' instructions in a pipelined processor: $\mathbf{ET_{pipeline} = k + n - 1}$ **cycles = (k + n – 1) Tp**

In the same case, for a non-pipelined processor, execution time of 'n' instructions will be:

$\mathbf{ET_{non\text{-}pipeline} = n * k * Tp}$

So, speedup (S) of the pipelined processor over non-pipelined processor, when 'n' tasks are executed on the same processor is:

S = Performance of pipelined non processor / Performance of pipelined processor

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

Cont..

When the number of tasks ‘n’ are significantly larger than k, that is, n >> k

$$S = \frac{n * k}{n}$$

$$S = k$$

Maximum speed up = no of segment

There are various reasons why the pipeline cannot operate at its maximum theoretical rate.

Different segments may take different times to complete their sub-operation.It is not always correct to assume that a non-pipelined circuit has the same time delay as that of an equivalent pipeline circuit.

Illustration

- Let the time it takes to process a sub-operation in each segment be equal to $t_p = 20$ ns.
- Assume that the pipeline has k = 4 segments and executes n = 100 tasks in sequence.
- The pipeline system will take
  $(k + n - 1)t_p = (4 + 99) \times 20 = 2060$ ns to complete.
- Assuming that $t_{np} = kt_p = 4 \times 20 = 80$ ns, a non pipelined system requires $nkt_p = 100 \times 80 = 8000$ ns to complete the 100 tasks.
- The speedup ratio is equal to 8000/2060 = 3.88.
- As the number of tasks increases, the speedup will approach 4, which is equal to the number of segments in the pipeline.
- If we assume that $t_{np} = 60$ ns, the speedup becomes 60/20 = 3.

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.
- Let's take an example of a pipeline unit for floating-point addition and subtraction. The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.
- $X = A * 10^a = 0.9504 * 10^3$
- $Y = B * 10^b = 0.8200 * 10^2$

Where **A** and **B** are two fractions that represent the mantissa and **a** and **b** are the exponents.

Cont..

The combined operation of floating-point addition and subtraction is divided into four segments. Each segment contains the corresponding suboperation to be performed in the given pipeline. The suboperations that are shown in the four segments are:

1. Compare the exponents by subtraction.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

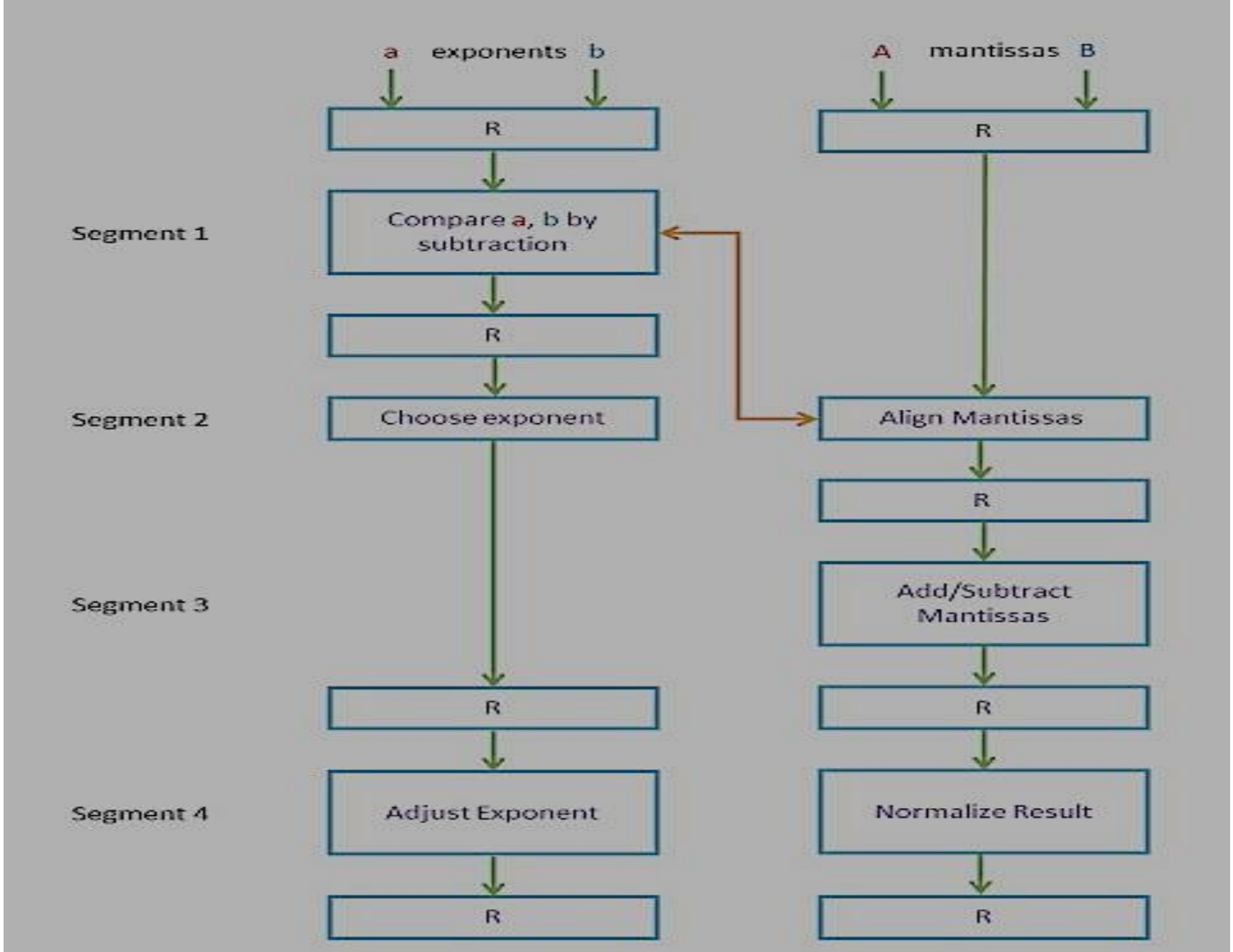


Figure | Arithmetic pipeline for floating point add/subtract operation

Procedure:

1. Compare exponents by subtraction:

The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.

The difference of the exponents, i.e., **3 - 2** = **1** determines how many times the mantissa associated with the smaller exponent must be shifted to the right.

2. Align the mantissas:

The mantissa associated with the smaller exponent is shifted according to the difference of exponents determined in segment one.

$X = 0.9504 * 10^3 \quad Y = 0.08200 * 10^3$

Cont..

3. Add mantissas:

The two mantissas are added in segment three.

$Z = X + Y = 1.0324 * 10^3$

4. Normalize the result:

After normalization, the result is written as:

$Z = 0.10324 * 10^4$

Instruction Pipeline

- Pipeline processing can occur not only in the data stream but in the instruction stream as well. An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- This causes the instruction fetch and execute phases to overlap and perform simultaneous operations.
- Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely..

Cont..

- In the most general case, the computer needs to process each instruction with the following sequence of steps:

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.

- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration. The time that each step takes to fulfill its function depends on the instruction and the way it is executed

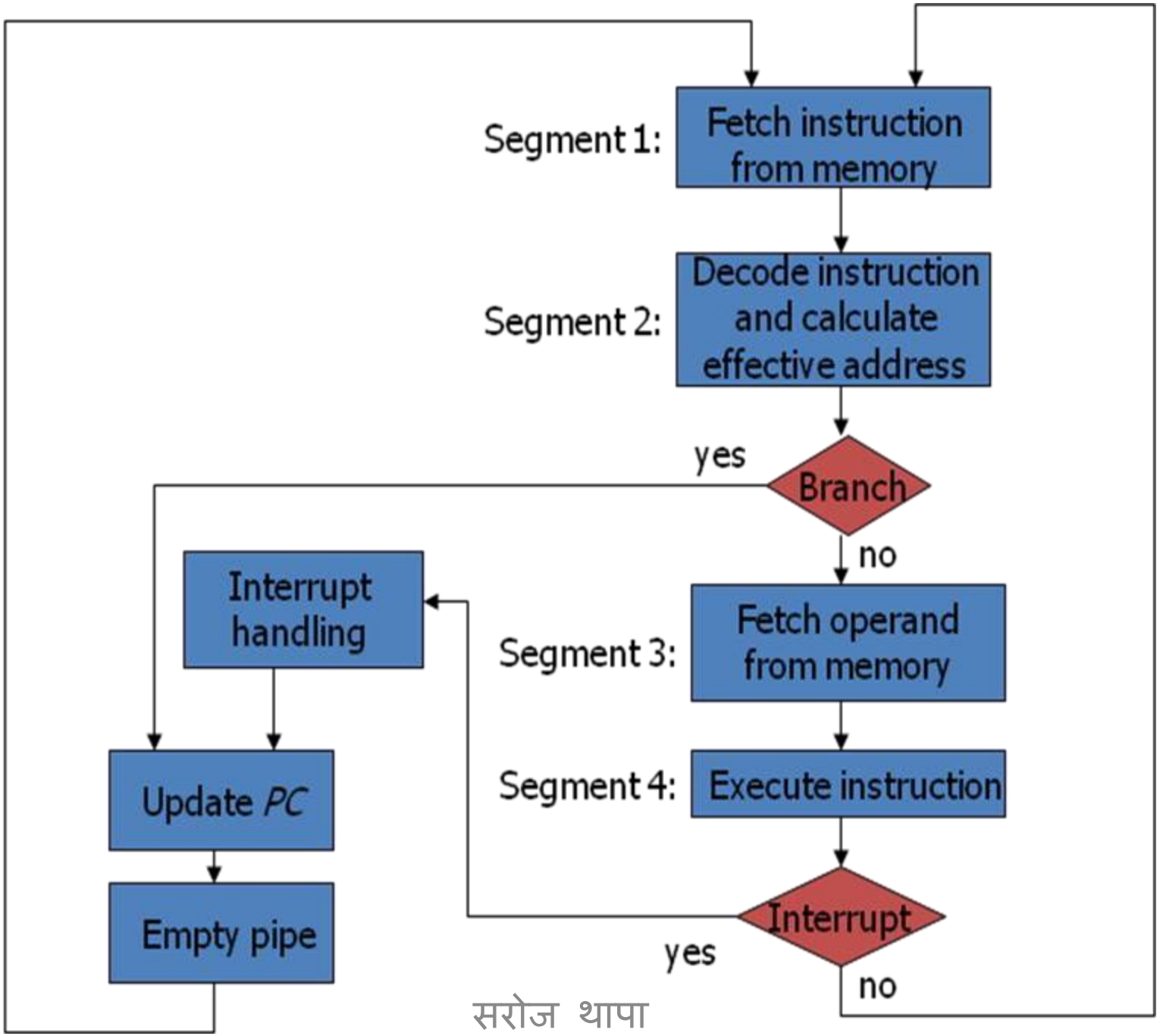
Example: Four-Segment Instruction Pipeline

The above figure shows operation of 4-segment instruction pipeline. The four segments are represented as:

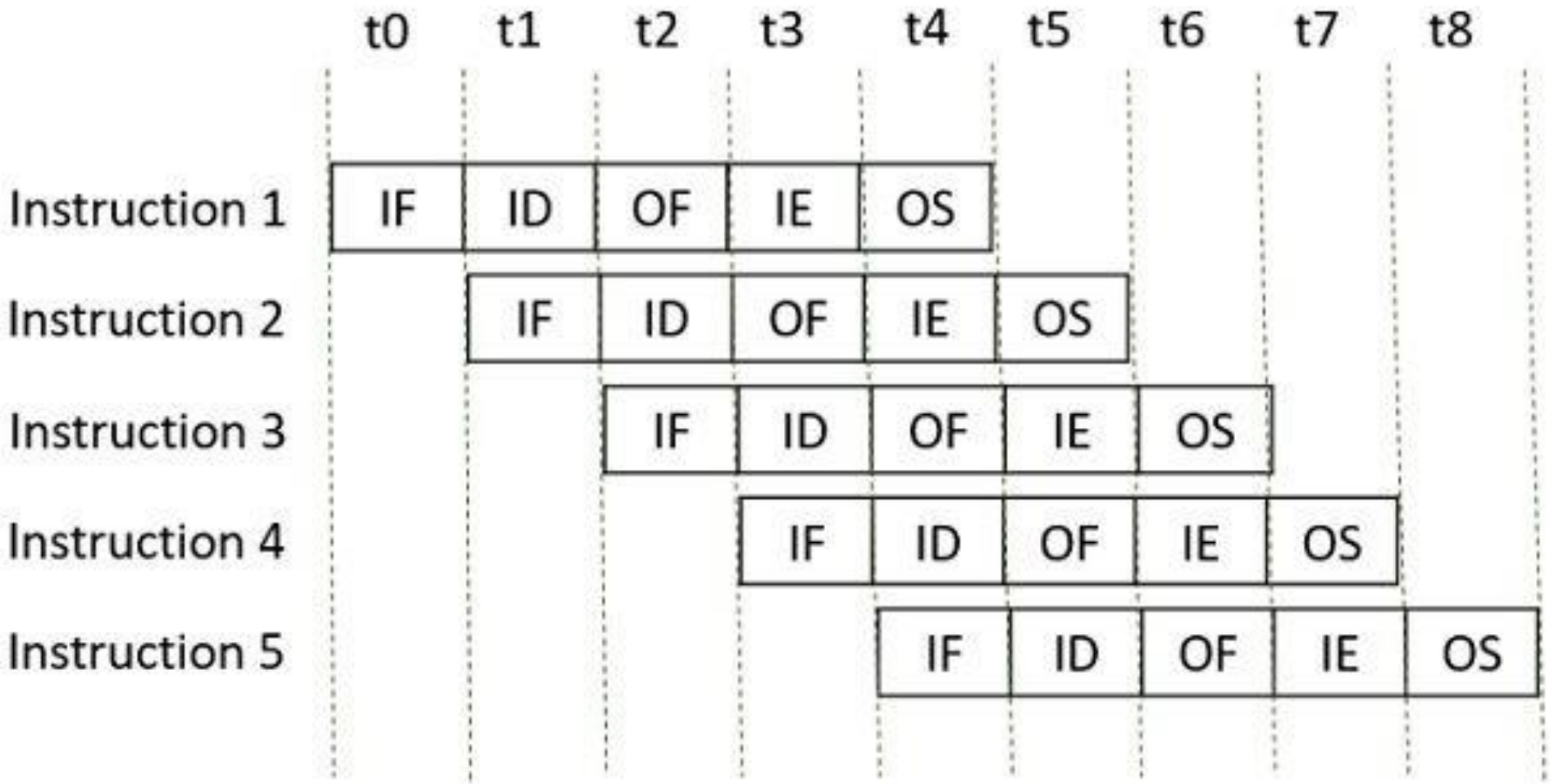
1. FI: segment 1 that fetches the instruction.
2. DA: segment 2 that decodes the instruction and calculates the effective address.
3. FO: segment 3 that fetches the operands.
4. EX: segment 4 that executes the instruction. The space time diagram for the 4-segment instruction pipeline is given below:

Step	1	2	3	4	5	6	7	8	9
1	FI	DA	FO	EX					
2		FI	DA	FO	EX				
3			FI	DA	FO	EX			
4				FI	DA	FO	EX		
5					FI	DA	FO	EX	
6						FI	DA	FO	EX

Fig: timing diagram for 4-segment instruction pipeline


Segment 1:
Fetch instruction from memory
Segment 2:
Decode instruction and calculate effective address
yes
Branch
no
Interrupt handling
Segment 3:
Fetch operand from memory
Update PC
Segment 4:
Execute instruction
Empty pipe
Interrupt
yes
no

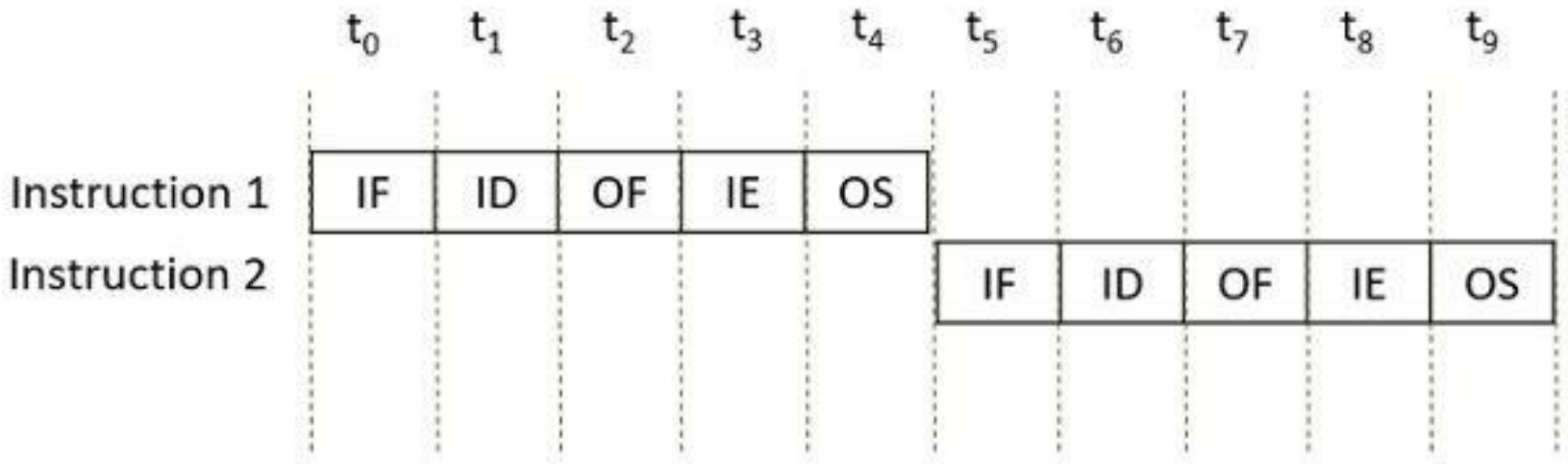
सरोज थापा



Instruction Pipelining of Five Instructions

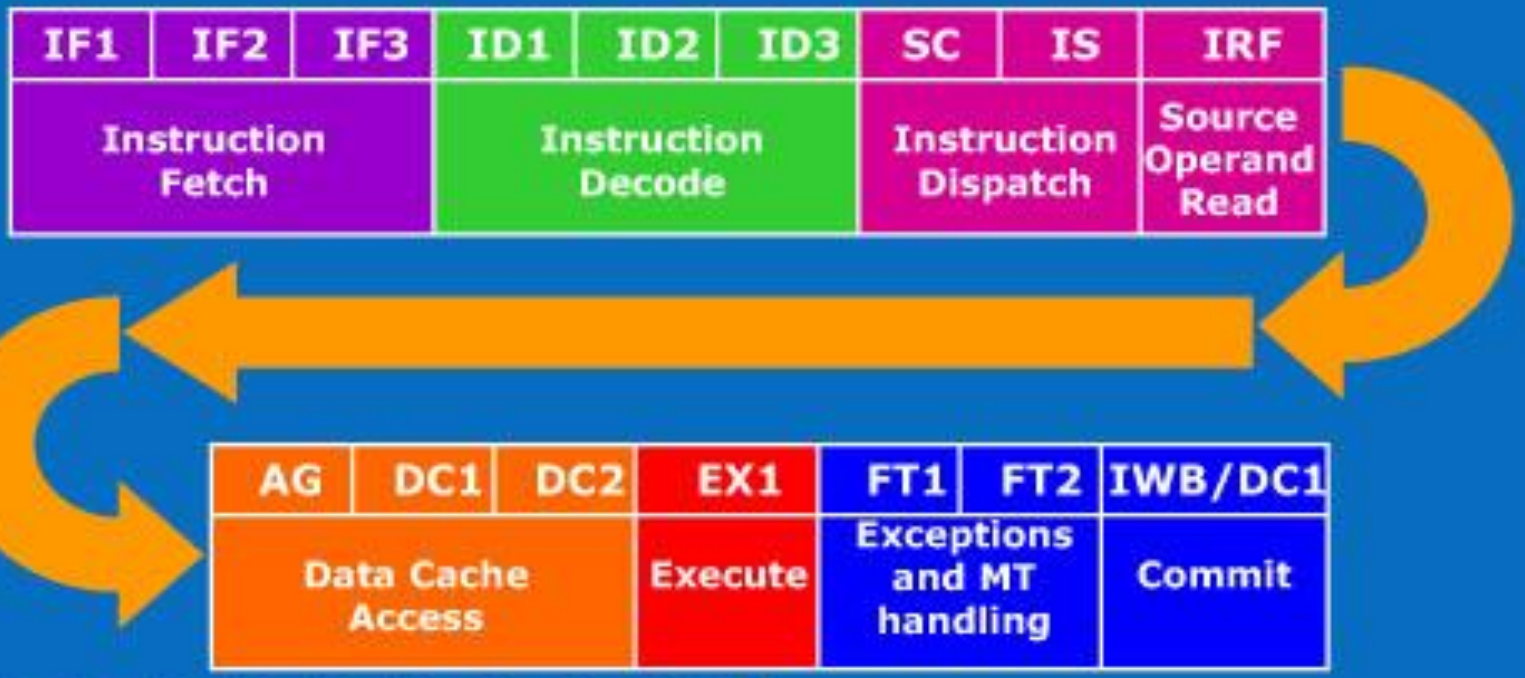
सरोज थापा

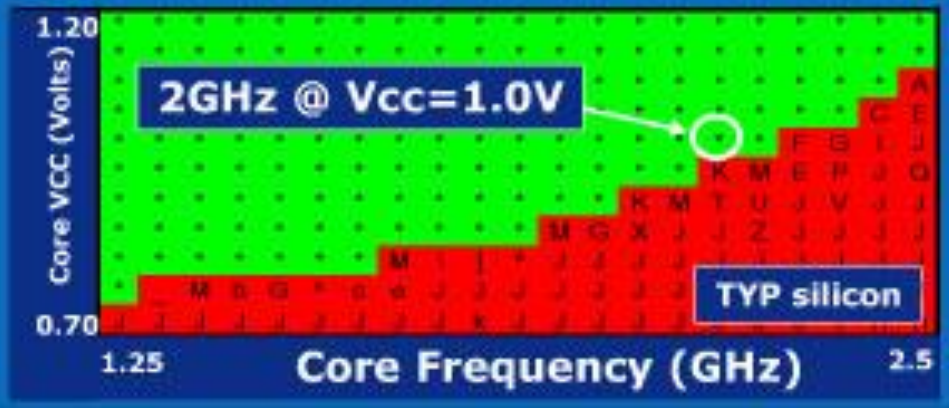
Else,,,,



Sequential Execution of Instructions

16 STAGE PROCESSOR PIPELINE





Silicon is capable of high core frequencies.

Under Embargo until April 1, 8:30pm US pacific time

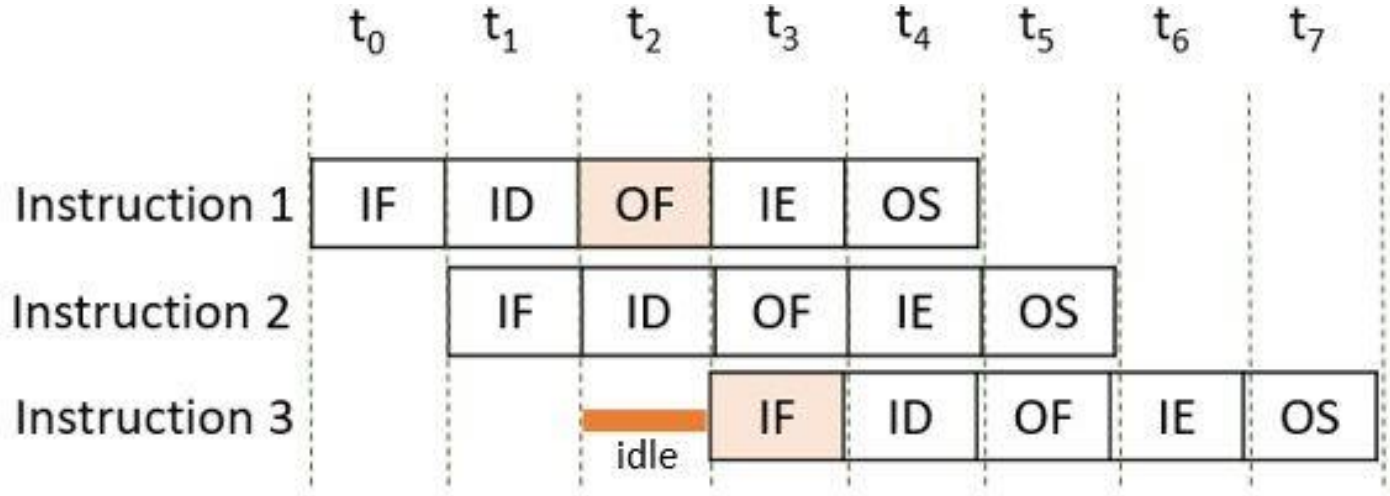
Pipeline Conflicts (Hazards)

- A pipeline hazard occurs when the instruction pipeline deviates at some phases, some operational conditions that do not permit the continued execution. In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. **Resource conflicts** caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
2. **Data dependency conflicts** arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
3. **Branch difficulties** arise from branch and other instructions that change the value of PC.

Resource Conflicts

- When two pipelined instructions or even more, want to access the same resource it results in resource hazards. It is also termed *structural hazards*.
- A solution to this hazard is that these instructions must be executed serially up to some portion of the pipeline.



Resource Hazards

Illustration

Let us understand this with the help of an example. Consider the main memory you have has a single port that restricts the processor to perform instruction fetch, read data and write data one at a time. This means you cannot perform the operand read & write operation, from memory in parallel with instruction fetch.

Now consider that there are three instructions in the pipeline. So in the normal conditions, the operand fetch of instruction 1 must-have overlapped the instruction fetch of instruction 3. But there is a case that source operand of instruction 1 is present in main memory instead of register and the rest of all the operands are in register. So we would halt the instruction fetch of instruction 3 for one clock cycle. Because instruction 1 will fetch its source operand from memory so in the same clock cycle instruction 3 cannot perform instruction fetch in parallel to operand fetch of instruction 1.

Data Dependency

- It arises when instructions depend on the result of previous instruction but the previous instruction is not available yet.
- For example an instruction in segment may need to fetch an operand that is being generated at same time by the previous instruction in the segment.
- The most common techniques used to resolve data hazard are:

(a) Hardware interlock - a hardware interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in the pipeline. It then inserts enough number of clock cycles to delays the execution of such instructions.

Cont..

(b) Operand forwarding - This method uses a special hardware to detect conflicts in instruction execution and then avoid it by routing the data through special path between pipeline segments. For example, instead of transferring an ALU result into a destination register, the hardware checks the destination operand, and if it is needed in next instruction, it passes the result directly into ALU input, bypassing the register.

(c) Delayed load - It is software solutions where the compiler is designed in such a way that it can detect the conflicts; re-order the instructions to delay the loading of conflicting data by inserting no operation instruction.

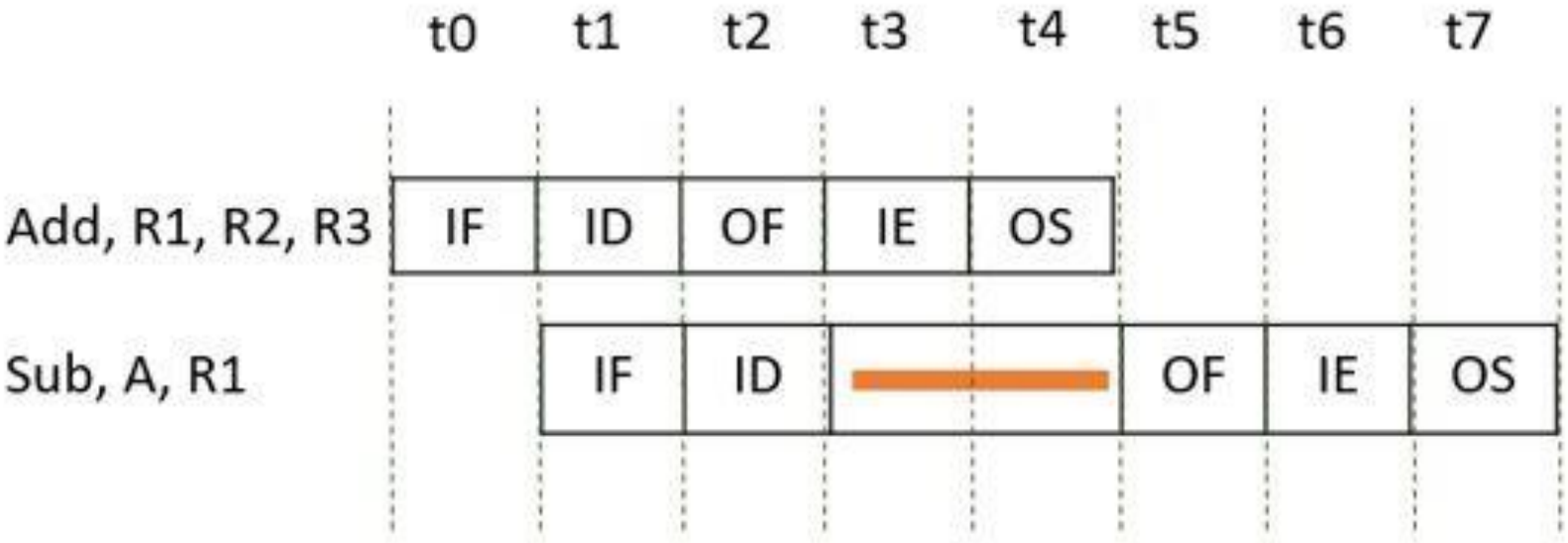
Example

Consider that you have two instructions:

ADD R1, R2, R3

SUB A, R1

Observe the instructions above the result of add instruction is stored in register R1 after the execution. This R1 act as an operand for subtract instruction. Now in the figure below notice that the register R1 is updated with the add result in clock cycle t_4. But the subtract instruction need its operand R1 at the t_3 clock cycle. But if subtract instruction fetches the operand R1 in the t_3 clock cycle then it will generate an incorrect result.



Data Hazards

Handling of Branch Instructions

- Branch hazard arises from branch and other instruction that change the value of program counter (PC). The conditional branch provides plenty of instruction branch line and it is difficult to determine which branches will be taken or not taken. A variety of approaches have been used to deal with branch hazard and they are described below.

(a) Multiple streaming - It is a brute-force approach which replicates the initial portions of the pipeline and allows the pipeline to fetch both instructions, making use of two streams (branches).

(b) Prefetch branch target - When a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch. This target is then saved until the branch instruction is executed. If the branch is taken, the target has already been prefetched.

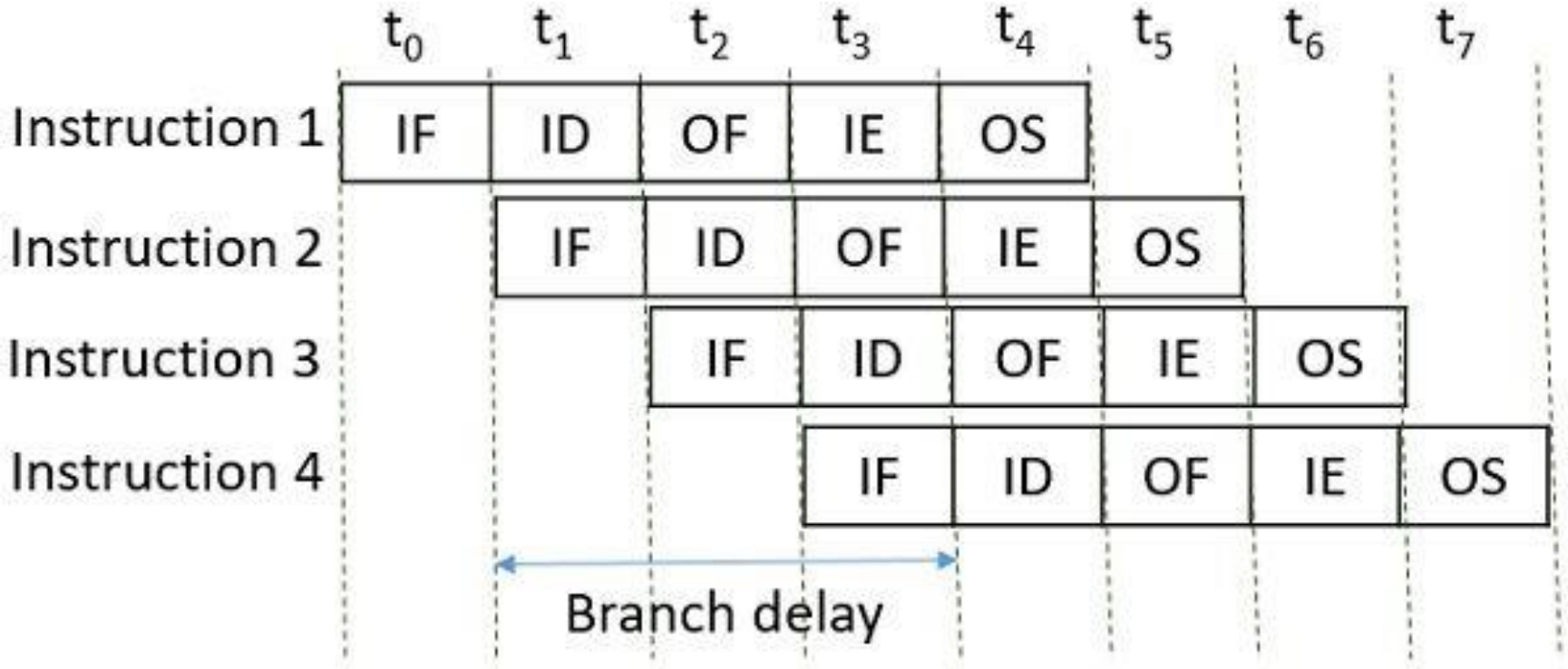
Cont..

(c) Branch prediction - uses additional logic to prediction the outcomes of a (conditional) branch before it is executed. The popular approaches are - predict never taken, predict always taken, predict by opcode, taken/not taken switch and using branch history table.

Cont..

d)Loop buffer - A loop buffer is a small, very-high-speed memory maintained by the instruction fetch stage of the pipeline and containing the most recently fetched instructions, in sequence. If a branch is to be taken, the hardware first checks whether the branch target is within the buffer. If so, the next instruction is fetched from the buffer.

e)Delayed branch - This technique is employed in most RISC processors. In this technique, compiler detects the branch instructions and re-arranges the instructions by inserting useful instructions to avoid pipeline hazards.



Control Hazards

Illustration

- Consider the first instruction I_1 in the pipeline is a branch instruction that targets instruction I_6.
- The instruction I_1 is fetched in cycle t_0, decoded in cycle t_1, fetch operands in cycle t_2 and perform execution in cycle t_3 where the target address is computed.
- But till then three instructions I_2, I_3 and I_4 are pipelined in cycle t_1, t_2 and t_3 which must be discarded as instruction I_1 is branch instruction which will compel the processor to execute the instruction I_6 after instruction I_1.
- This is how control hazards lead to delay of three cycles t_1, t_2 and t_3 between I_1 and I_6 which is also termed as *branch delay*.

4.5 RISC Pipeline

- To use an efficient instruction pipeline
- To implement an instruction pipeline using a small number of sub operations, with each being executed in one clock cycle.
- Because of the fixed-length instruction format, the decoding of the operation can occur at the same time as the register selection.
- Therefore, the instruction pipeline can be implemented with two or three segments.
- One segment fetches the instruction from program memory
- The other segment executes the instruction in the ALU
- Third segment may be used to store the result of the ALU operation in a destination register

Cont

- The data transfer instructions in RISC are limited to load and store instructions.
- These instructions use register indirect addressing. They usually need three or four stages in the pipeline.
- To prevent conflicts between a memory access to fetch an instruction and to load or store an operand, most RISC machines use two separate buses with two memories.
- Cache memory: operate at the same speed as the CPU clock
- One of the major advantages of RISC is its ability to execute instructions at the rate of one per clock cycle.
- In effect, it is to start each instruction with each clock cycle and to pipeline the processor to achieve the goal of single-cycle instruction execution.

Cont…

- *Compiler* supported that translates the high-level language program into machine language program.
- Instead of designing hardware to handle the difficulties associated with data conflicts and branch penalties.
- RISC processors rely on the efficiency of the compiler to detect and minimize the delays encountered with these problems.

Vector Processing

- Vector processing is a procedure for speeding the processing of information by a computer, in which pipelined units perform arithmetic operations on uniform, linear arrays of data values, and a single instruction involves the execution of the same operation on every element of the array.
- 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.
- To achieve the required level of high performance it is necessary to utilize the fastest and most reliable hardware and apply innovative procedures from vector and parallel processing techniques

Application Areas of Vector Processing

- Computers with vector processing capabilities are in demand in specialized applications. The following are representative application areas where vector processing is of the utmost importance.

- Long-range weather forecasting
- Petroleum explorations
- Seismic data analysis
- Medical diagnosis
- Aerodynamics and space flight simulations
- Artificial intelligence and expert systems
- Mapping the human genome
- Image processing

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 order set of one dimensional array of data items. A vector V of length 'n'is represented as a row vector by $V = [V_1, V_2, V_3, \ldots\ldots\ldots\ldots, V_n]$
- 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 Vi of vector V is written as V(I) and the index I refers to a memory address or register where the number is stored.
- To examine the difference between a conventional scalar processor and a vector processor, consider the following Fortran DO loop

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

Conventional computer

```
      Initialize I = 0
20    Read A(I)
      Read B(I)
      Store C(I) = A(I) + B(I)
      Increment I = i + 1
      If I ≤ 100 goto 20
```

This is a program for adding two vectors A and B of length 100 to produce a vector C.

- A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop. It allows operations to be specified with a single vector instruction of the form

$$\mathbf{C(1 : 100) = A(1 : 100) + B(1: 100)}$$

- The vector instruction includes the initial address of the operands, the length of the vectors, and the operation to be performed, all in one composite instruction.
- It is also possible to design the processor with a large number of *registers* and store all operands in registers prior to the addition operation.
  - The base address and length in the vector instruction specify a group of CPU registers.

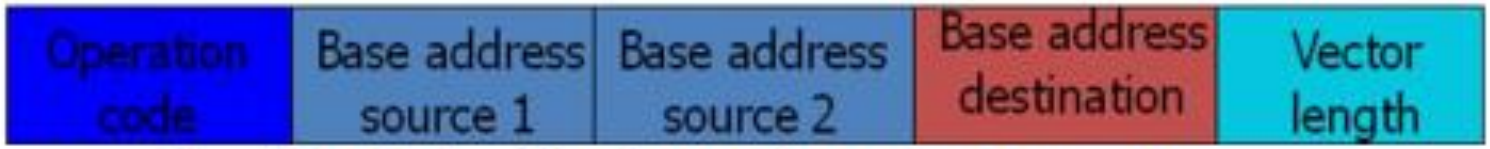


Fig 4-11: Instruction format for vector processor

Matrix Multiplication

- Matrix multiplication is one of the most computational intensive operations performed in computers with vector processors. An n x 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 x 3 matrices A and B.

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

The product matrix C is a 3×3 matrix whose elements are related to the elements of A and B by the inner product:

$$c_{ij} = \sum_{k=1}^{3} a_{ik} \times b_{kj}$$

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

$$c_{11} = a_{11}\, b_{11} + a_{12}\, b_{21} + a_{13}\, b_{31}$$

Inner Product

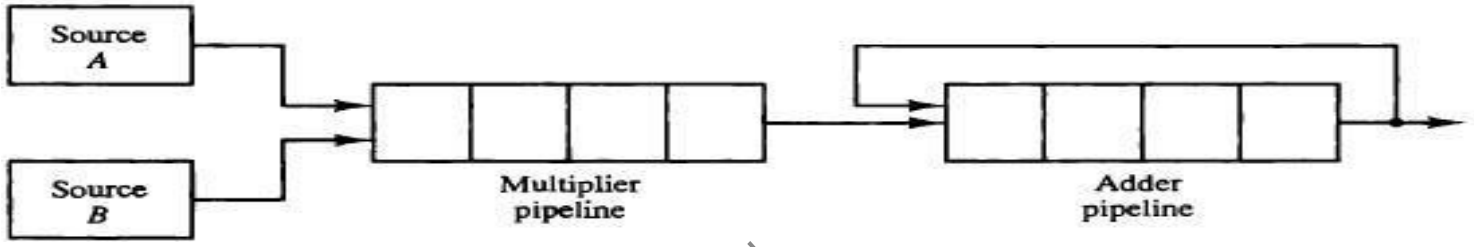
- In general, the inner product consists of the sum of k product terms of the form

$$C = A_1 B_1 + A_2 B_2 + A_3 B_3 + A_4 B_4 + \cdots + A_k B_k$$

In a typical application k may be equal to 100 or even 1000. The inner product calculation on a pipeline vector processor is shown below:

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

Figure 9-12 Pipeline for calculating an inner product.



Array Processing

- An array processor is a processor that performs computation on large array of data. An **attached array processor** is an auxiliary processor attached to a general purpose computer. **SIMD array processor** is a processor that has a single instruction multiple data organization. It manipulates vector instruction by means of multiple functional unit responding through a common instructions.

1. Attached Array Processor:

It is designed as a peripheral of conventional host computer and its purpose is to enhance the performance of the computer by providing vector processing for complex scientific application. It achieves high performance by means of parallel processing with multiple functional units.

2. SIMD Array Processor:

It is a computer with multiple processing unit operation in parallel. The processing unit are synchronized to perform the same operations under the control of a common bus unit, providing SIMD organisation.

Attached Array processor

- Its purpose is to enhance the performance of the computer by providing vector processing for complex scientific applications.
- Parallel processing with multiple functional units
- Fig. 4-14 shows the interconnection of an attached array processor to a host computer.
- For example, when attached to a VAX 11 computer, the FSP-164/MAX from Floating-Point Systems increases the computing power of the VAX to 100megaflops.
- The objective of the attached array processor is to provide *vector manipulation capabilities* to a conventional computer at a fraction of the cost of supercomputer

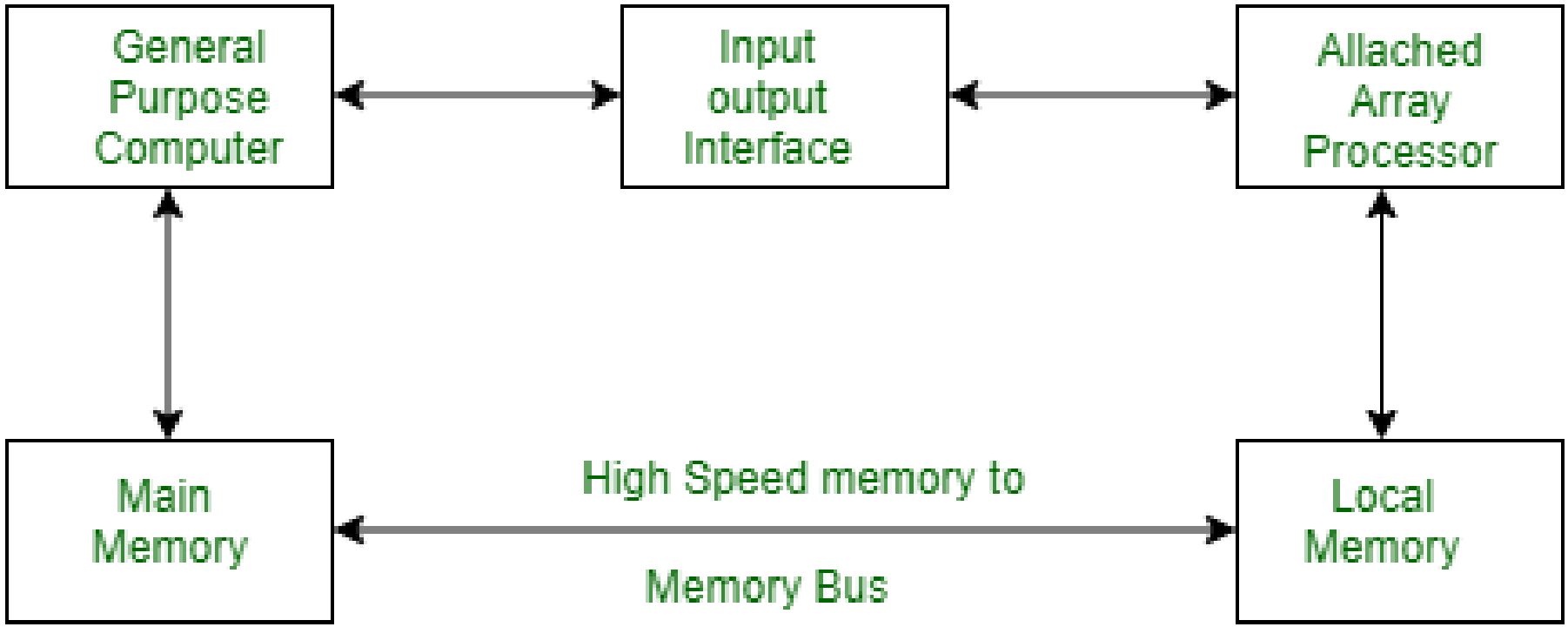
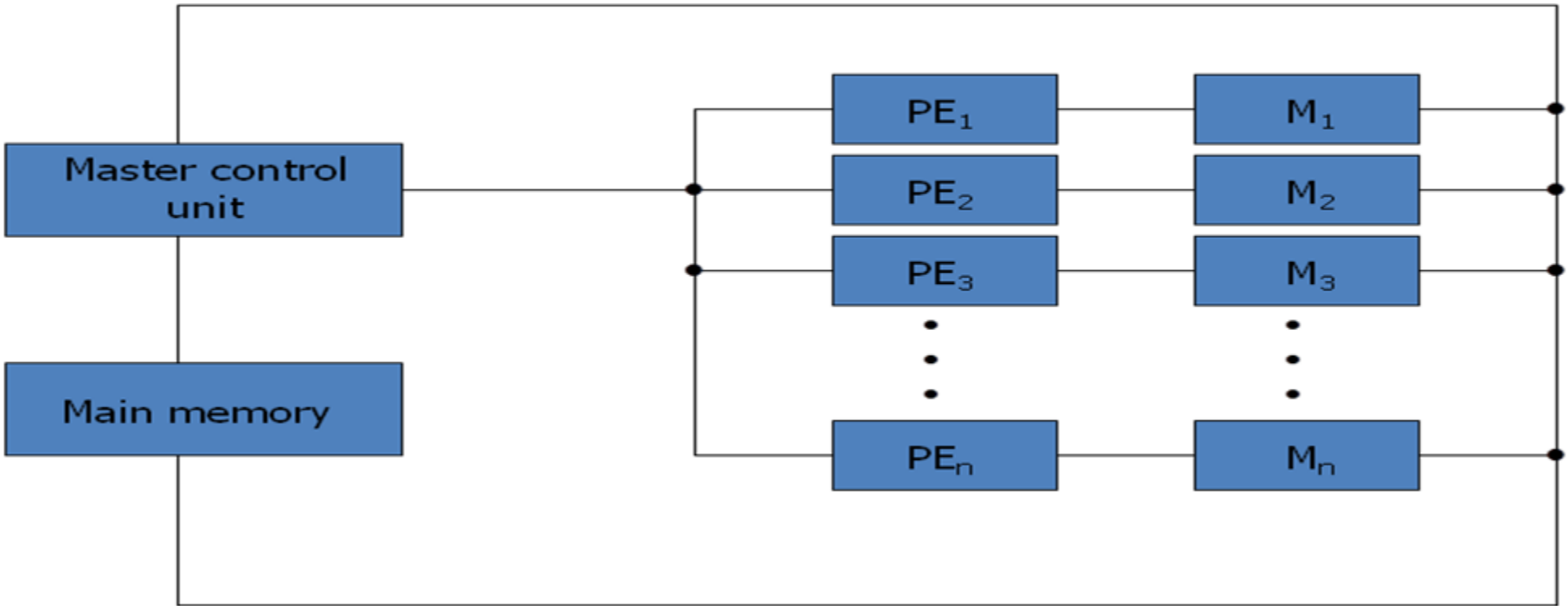


Figure - Interconnection of an attached array Processor to a host computer

SIMD Array Processor

- An SIMD array processor is a computer with multiple processing units operating in parallel.
- It contains a set of identical processing elements (PEs), each having a local memory *M*.
- Each PE includes an ALU, a floating-point arithmetic unit, and working registers.
- Vector instructions are broadcast to all PEs simultaneously.
- 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.
- For example, the ILLIAC IV computer developed at the University of Illinois and manufactured by the Burroughs Corp.
- Are highly specialized computers.
- They are suited primarily for numerical problems that can be expressed in vector or matrix form.


Master control unit
Main memory
PE1
PE2
PE3
PEn
M1
M2
M3
Mn

END OF UNIT 5

सरोज थापा