Unit - 5

Contents

a) Reduced Instruction set computer (RISC)
   - → CISC characteristics
   - → RISC characteristics

b) Pipeline and Vector processing :-
   - → Parallel Processing
   - → Pipelining
   - → Arithmetic Pipeline
   - → Instruction Pipeline
   - → RISC Pipeline
   - → Vector Processing
   - → Array Processors.

c) Multiprocessors :-
   - → Characteristics of Multiprocessors
   - → Interconnection Structures
   - → Interprocessor Arbitration
   - → Interprocessor communication and Synchronization.
   - → Cache Coherence.

Reduced Instruction Set computer (RISC)

CISC :-

A computer with a large number of instructions is classified as a complex instruction set computer, abbreviated CISC.

→ the major characteristics of CISC architecture are.

1) A large number of instructions - typically from 100 to 250 instructions.
2. Some instructions that perform specialized tasks and are used infrequently.
3. A large variety of addressing modes - typically from 5 to 20 different modes.
4. Variable-length instruction formats.
5. Instructions that manipulate operands in memory.

RISC :- The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer

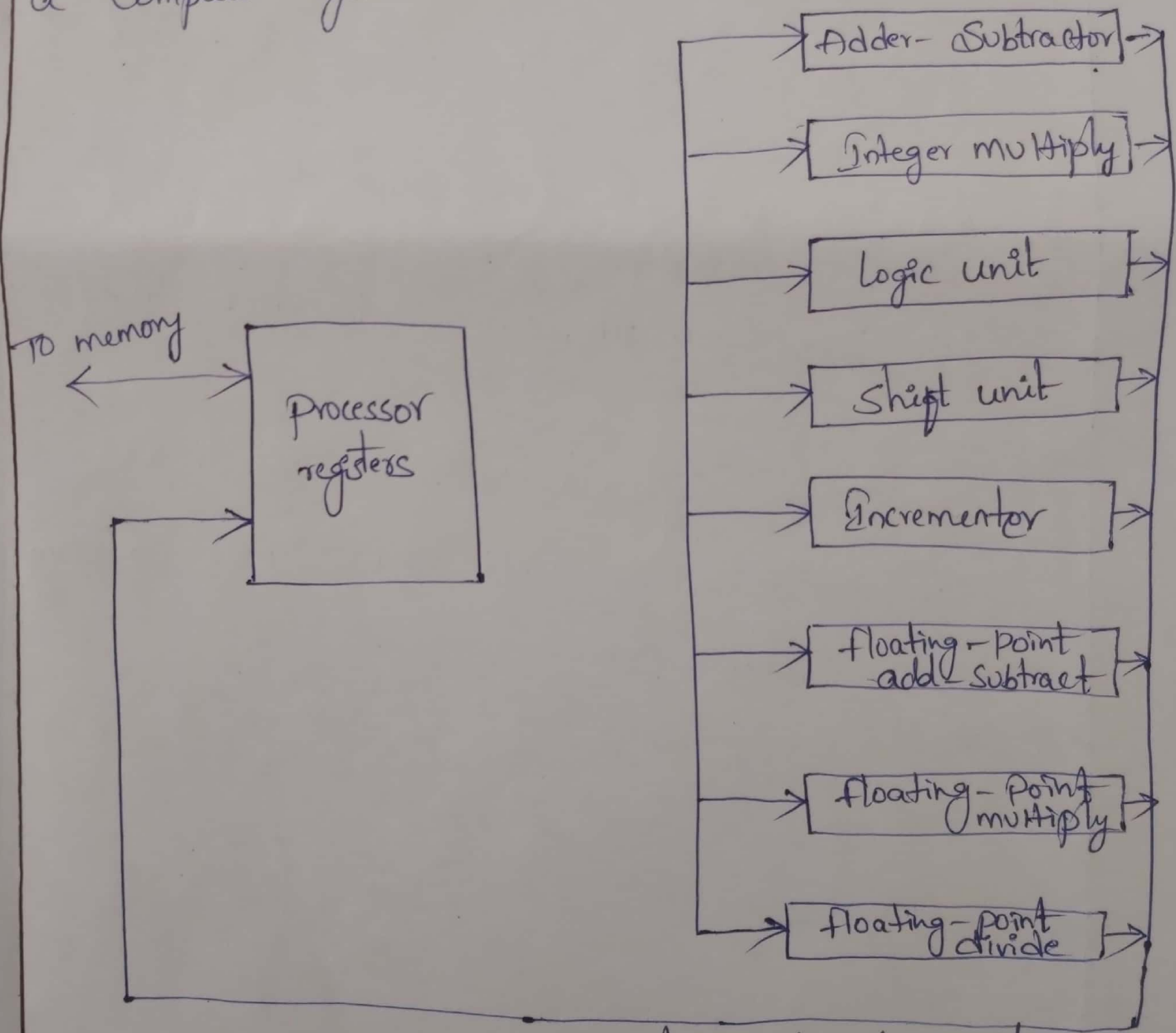
→ The major characteristics of a RISC processor are:

1. Relatively few instructions
2. Relatively few addressing modes.
3. Memory access limited to load and store instructions.
4. All operations done within the registers of the CPU.
5. Fixed length, easily decoded instruction format.
6. Single-Cycle instruction execution.
7. Hardwired rather than microprogrammed control.

Pipeline and Vector Processing :-

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.



Processor with multiple functional units

→ The Processor with multiple functional units shows one possible way of seperating the execution unit into eight functional units operating in parallel. each functional unit works independently and perfor arithmetic, logic and shift operations.

→ There are a variety of ways that parallel processing can be classified. One classification introduced by M.J flynn.

→ Flynn's classification divides computers into four major groups as follows:

a) Single instruction stream single data stream (SISD)

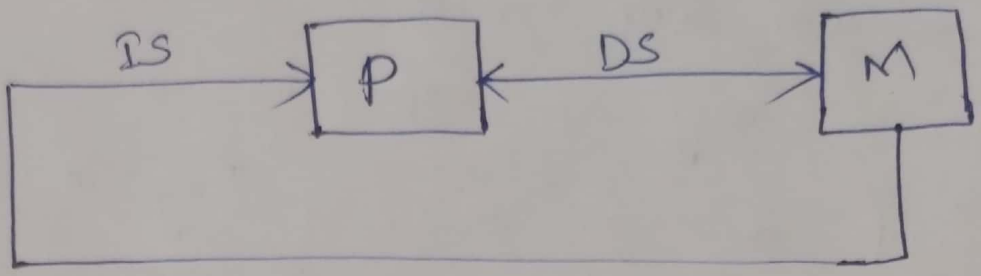
b) Single instruction stream, multiple data stream (SIMD)

c) Multiple instruction stream, Single data stream (MISD)

d) Multiple instruction stream, multiple data stream (MIMD)

SISD :- SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.

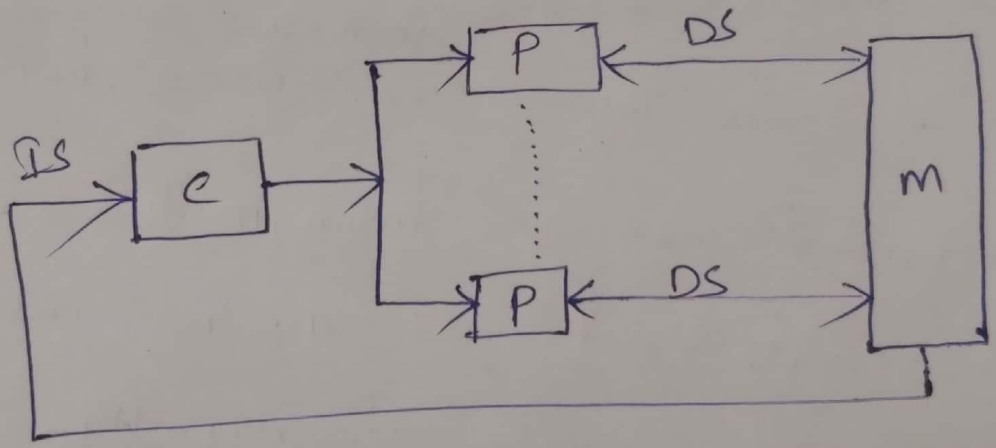
→ Instructions are executed sequentially and the system may & may not have internal parallel processing capabilities.



Schematic diagram of an SISD processor.

→ The processor (P) gets a single data stream and a single instruction stream from the memory (M) and executes the decoded instructions on the corresponding data streams.

SIMD :-



Schematic diagram of an SIMD processor

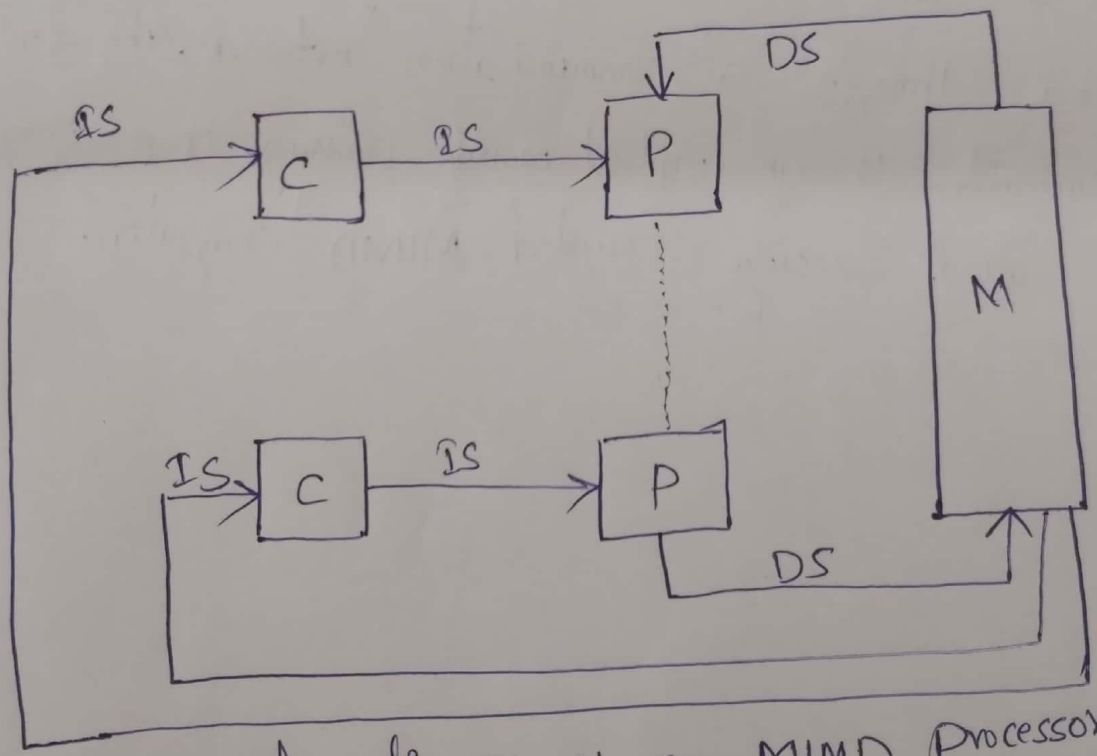
→ A single instruction stream is fetched from memory by the control unit, and is decoded one instructions at a time.

→ The processors execute the same instruction stream on different data stream.

→ popular examples of SIMD computers are vector computers and array computers.

MISD :- MISD structure is only of theretical interest since no practical system has been constructed using this Organization.

MISD :-



Schemantic diagram of an MIMD Processor

→ An MIMD computer in which the processors share a common memory is called a multiprocessor.

→ Communication is achieved by one processor writing to a memory location and another processor reading from the same memory location.

→ The communication time among processors is the same as memory read and write times, therefore, communication is very fast and the programs running in parallel on different processors can communicate frequently. Therefore multiprocessors are called tightly coupled computers.

→ In a distributed computing system, the nodes are connected through a communication network. The processors communicate through explicit message passing. These computers are called loosely coupled MIMD computers.

Pipelining :-

→ Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.

→ 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.

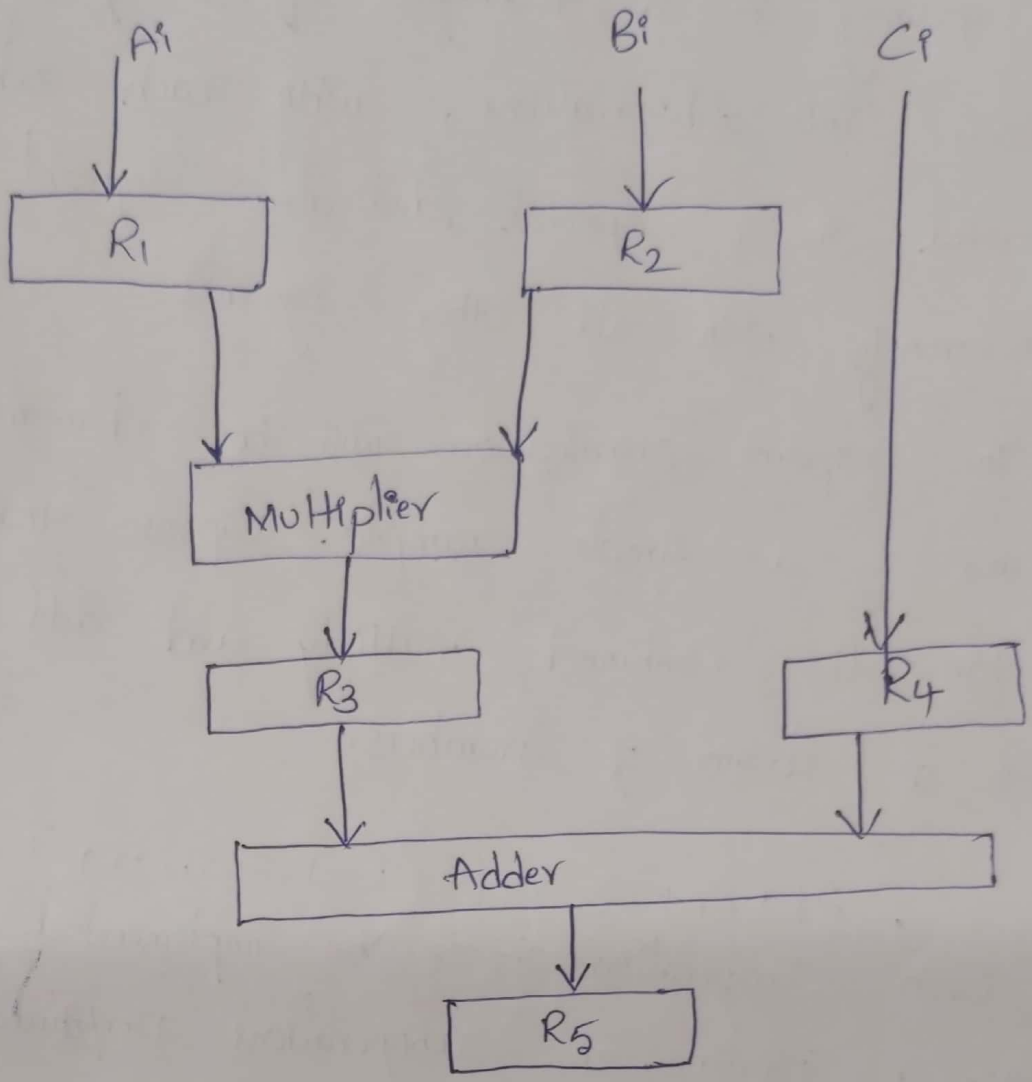
$A_i * B_i + C_i$ for $i = 1, 2, 3, \ldots, 7$.

→ Each suboperation is to be implemented in a segment within a pipeline. The suboperations performed in each segment of the pipeline are as follows.

$R_1 \leftarrow A_i$, $R_2 \leftarrow B_i$ Input A_i and B_i

$R_3 \leftarrow R_1 * R_2$, $R_4 \leftarrow C_i$ Multiply and input C_i

$R_5 \leftarrow R_3 + R_4$ Add C_i to product.



Example of Pipeline processing

→ R_1 through R_5 are registers that receive new data with every clock pulse. The multiplier and adder are combinational circuits.

Content of Registers in pipeline example

clock pulse number	Segment 1		Segment 2		Segment 3
	R_1	R_2	R_3	R_4	R_5
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_8~~	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

→ the first clock pulse transfers A_1 and B_1 into R_1 and R_2. the second clock pulse transfers the product of R_1 and R_2 into R_3 and C_1 into R_4.

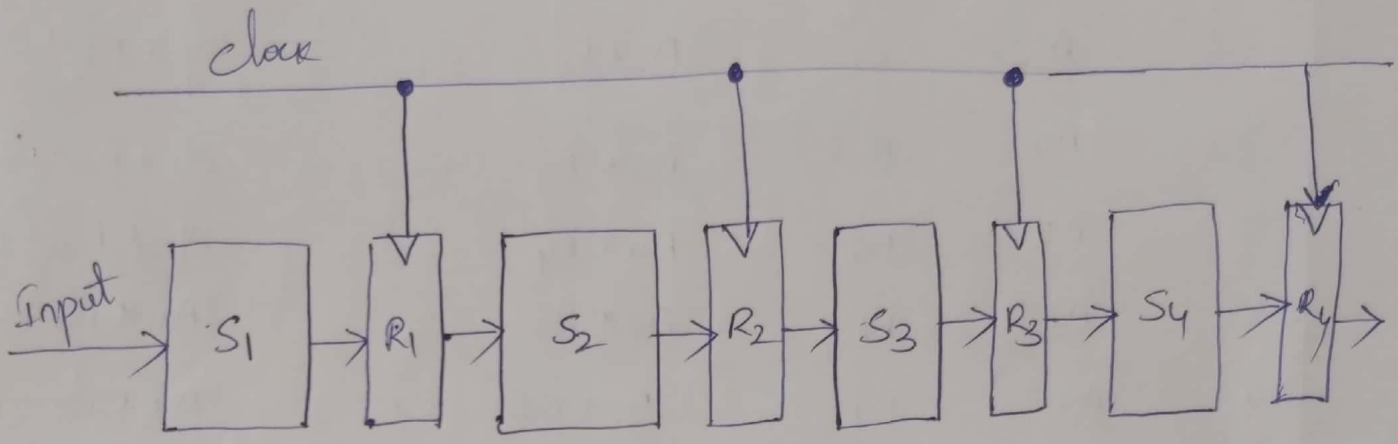
→ The same clock pulse transfers A_2 and B_2 into R_1 and R_2.

→ the third clock pulse operates on all three segments simultaneously. It places A_3 and B_3 into R_1 and R_2, transfers the product of R_1 and R_2 into R_3, transfers C_2 into R_4, and places the sum of R_3 and R_4 into R_5.

→ It takes three clock pulse to fill up the pipe and retrieve the first output from R_5.

→ From there on, each clock produces a new output and moves the data one step down the pipeline.

General considerations :-



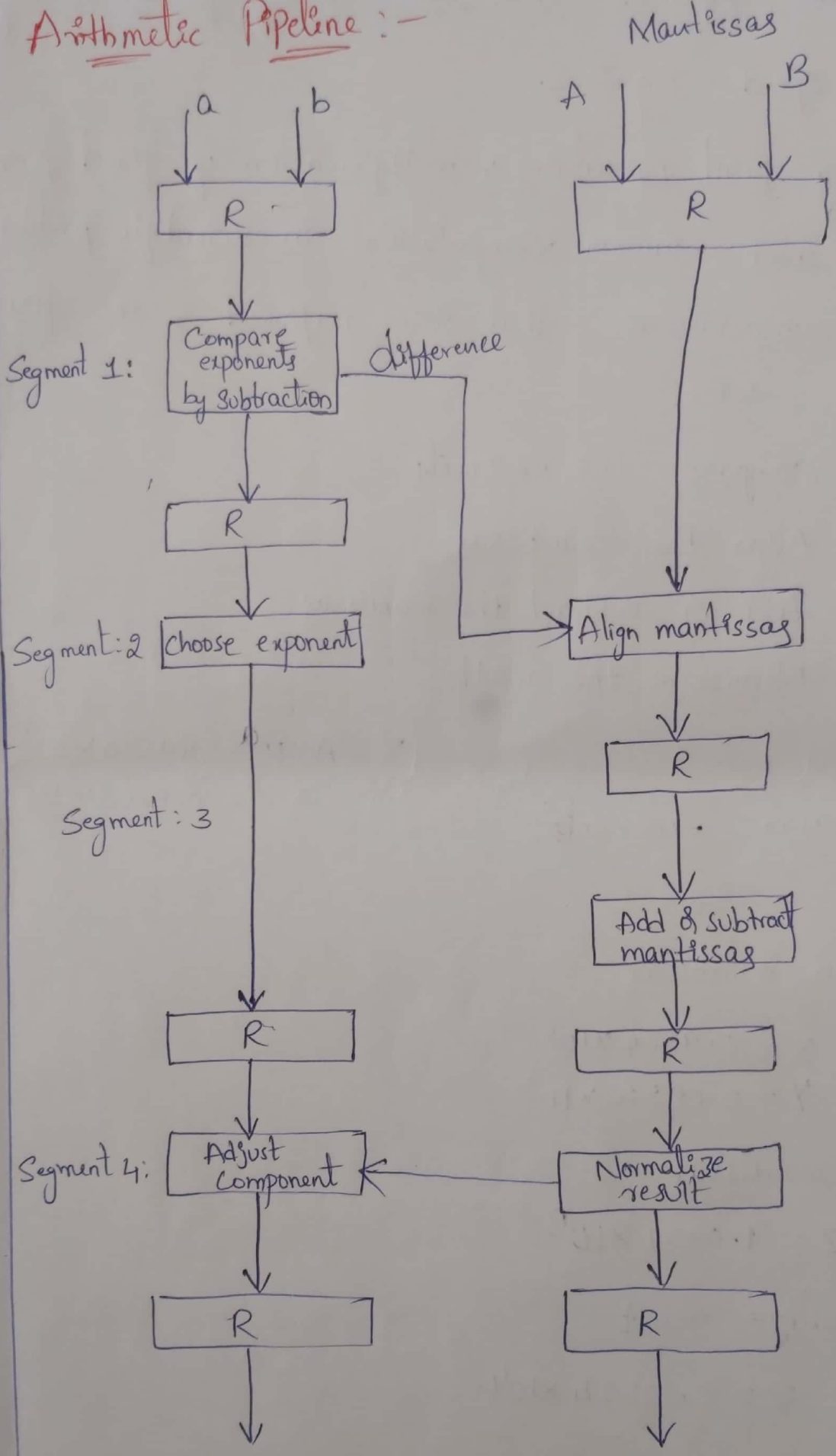
Four-Segment pipeline

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

→ Each segment consists of a combinational circuit S_i that performs a suboperations over the data stream flowing through the pipe.

→ The segments are seperated by registers R_i that holds the intermediate results between the stages.

Arithmetic Pipeline :-



Pipeline for floating-point addition and subtraction

→ Arithmetic pipeline units are usually found in very high speed computers.

→ floating point operations; multiplication of fixed point numbers, and similar computations in scientific problem.

→ The suboperations 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.

Example:-

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

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

→ Compare exponents:

$3 - 2 = 1$

→ align mantissas

$X = 0.9504 \times 10^3$

$Y = 0.08200 \times 10^3$

→ add mantissas

$Z = 1.0324 \times 10^3$

→ Normalize result

$Z = 0.10324 \times 10^4$.

Instruction Pipeline :-

An instruction pipeline reads consecutive instructions are being executed in other segments.

→ Pipelining processing can occur not only in the data stream but in the instruction stream.

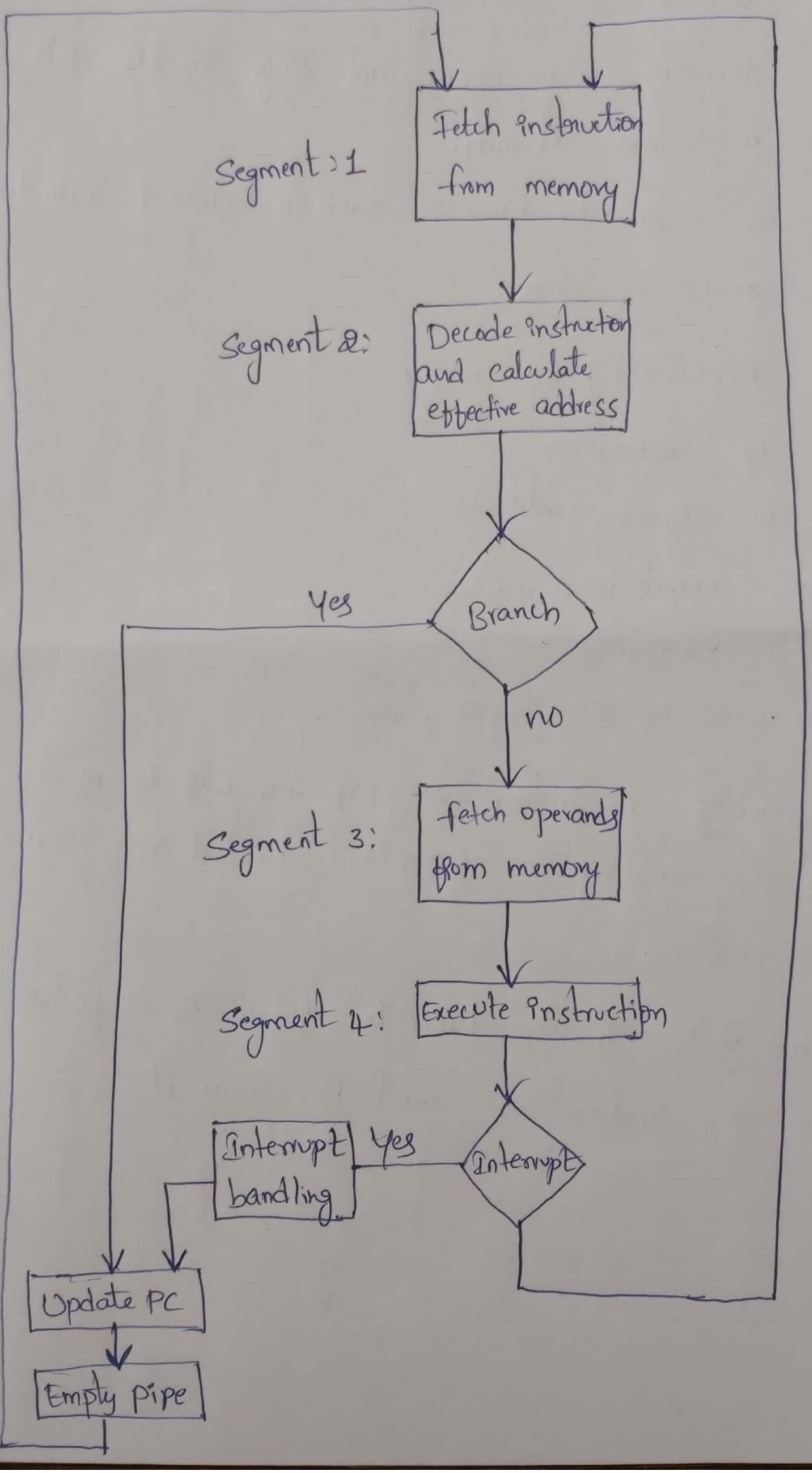
→ The computer needs to process each instruction with the following sequence of steps.

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



→ Four-Segment CPU pipeline. shows how the instruction cycle in the CPU can be processed with a four-segment pipeline.

→ While an instruction cycle in the CPU can be processed with a four-segments.

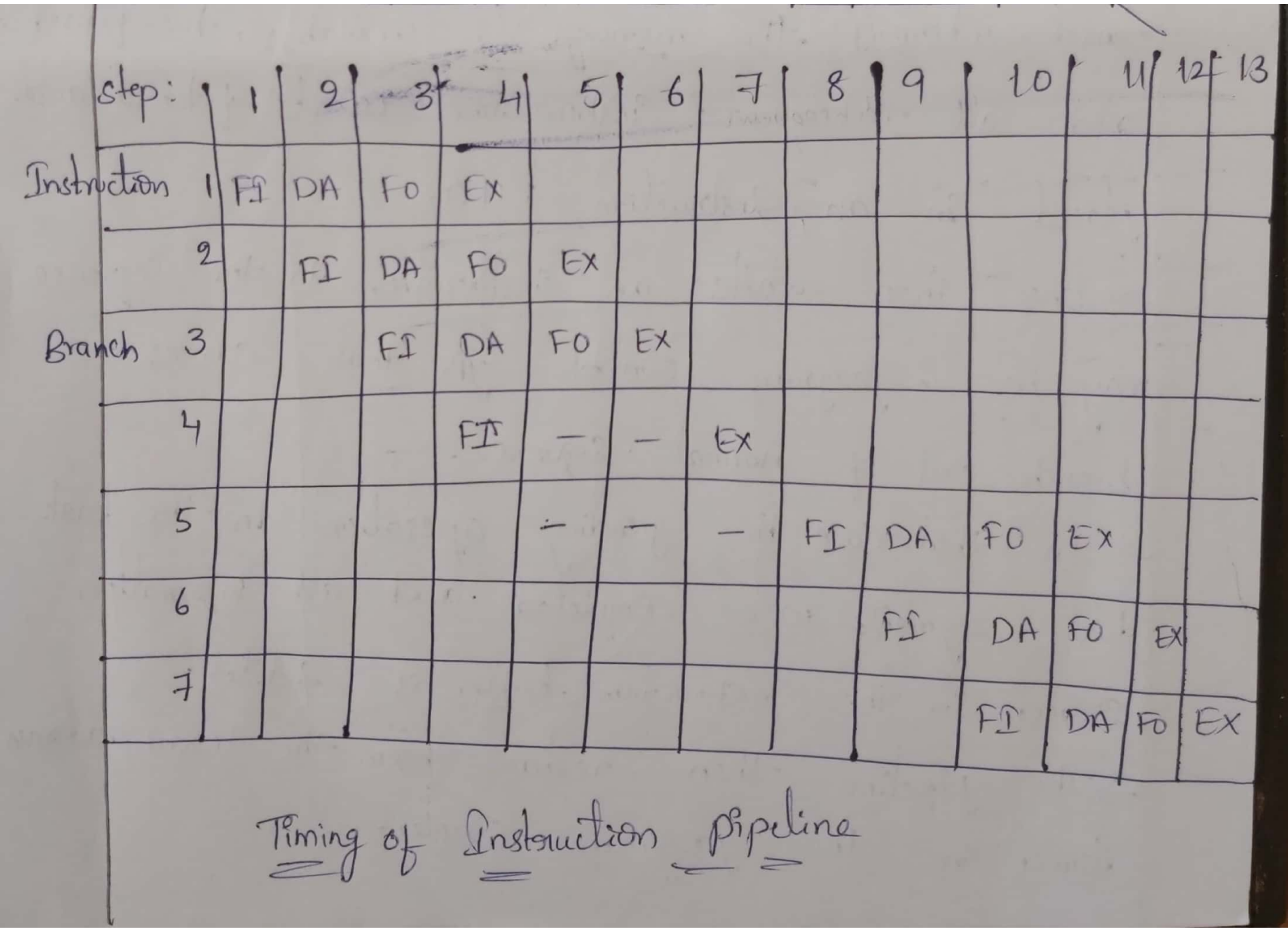
→ While an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3.

→ The effective address may be calculated in a seperate 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.

→ Once in a while, an instruction in the sequence may be a program control type that causes a branch out of normal sequence.

→ In that case the pending operations in the last two segments are completed and all information stored in the instruction buffer is deleted.

→ The pipeline then restarts from the new address stored in the program counter.



Timing of Instruction pipeline

→ Timing of instruction pipeline shows the operation of the instruction pipeline.

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

→ The four segments are represented in the diagram with an abbreviated symbol.

1). FI is the segment that fetches an instruction

2) DA is the segment that decodes the instruction and calculates the effective address.

3) FO is the segment that fetches the operand

4) EX is the segment that executes the instruction.

RISC Pipeline :-

→ RISC stands for Reduced Instruction Set Computer.

→ The major characteristics of RISC is its ability to use an efficient instruction pipeline.

→ The simplicity of the instruction set can be utilized to implement an instruction pipeline using a small number of suboperations, with each being executed in one clock cycle.

→ The data transfer instructions in RISC were limited to load and store instructions

→ Another characteristic of RISC is the support given by the compiler that translates the high-level language program into machine language program.

Example :- Three-Segment instruction pipeline

1. Instruction fetch : I
2. ALU operation : A
3. Execute instruction : A

→ The I Segment fetches the instruction from program memory.

→ The instruction is decoded and an ALU operation is performed in the A segment.

→ The ALU is used for three different functions, depending on the decoded instruction.

→ It performs an operation for a data manipulation instruction; it evaluates the effective address for a load & store instruction, & it calculates the branch address for a program control instruction.

→ The E segment directs the output of the ALU to one of three destinations, depending on the decoded instruction.

→ It transfers the result of the ALU operation into a destination register in the register file.

Delayed Load :-

1. LOAD : $R_1 \leftarrow M[\text{address } 1]$
2. LOAD : $R_2 \leftarrow M[\text{address } 2]$
3. ADD : $R_3 \leftarrow R_1 + R_2$
4. STORE : $M[\text{address } 3] \leftarrow R_3$

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

→ If the three-segment pipeline proceeds without interruptions, there will be a data conflict in instruction 3 because the operand in R2 is not yet available in the A segment.

→ The above timing diagram shows the E segment in a clock cycle 4 is in a process of placing the memory data into R2.

→ The A segment in clock cycle 4 is using the data from R2, but the value in R2 will not be the correct value since it has not yet been transfered from memory.

→ If the compiler cannot find a useful instruction to put after the load, it inserts no-operation (no-op) instruction

→ The concept of delaying the use of the data loaded from memory is refered to as "delayed load".

Clock cycle:	1	2	3	4	5	6	7
1. Load R_1	I	A	E				
2. Load R_2		I	A	E			
3. No-operation			I	A	E		
4. Add R_1+R_2				I	A	E	
5). Store R_3					I	A	E

(b) Pipeline timing with delayed load.

→ The above timing diagram shows same program with no-operation instruction inserted after the load to R_2 instruction.

→ The data is loaded into R_2 in clock cycle 4. The add instruction uses the value of R_2 in step 5.

→ Thus the no-operation instruction is used to advance one clock cycle in order to compensate for the data conflict in the pipeline.

Delayed Branch :-

A branch instruction delays the pipeline operation until the instruction at the branch address is fetched. this method is refered to as delayed branch.

Example :- An example of delayed branch in shown in below. The program for this example consists of five instructions.

Load from memory to R_1

Increment R_2

Add R_3 to R_4

Subtract R_5 from R_6

Branch to address X

(a) using no-op instructions

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 on X								I	A	E

In the fig (a) the compiler inserts two no-op instructions after the branch.

→ The Branch address X is transferred to PC in clock cycle. 7.

→ The fetching of the instruction at X is delayed by two clock cycles by the no-op instructions.

→ The instruction at X starts the fetch phase at clock cycle 8 after the program counter PC has been updated.

clock cycle	1	2	3	4	5	6	7	8
1. load	I	A	E					
2. Increment		I	A	E				
3. Add			I	A	E			
4. Branch to X				I	A	E		
5. Subtract					I	A	E	
6. Instruction in X						I	A	E

(b) Rearranging the instructions

→ ~~The compiler inserts two no-op~~ :

→ The above timing diagram can shows the arranging by placing the add and subtract instructions after the branch instruction instead

of before as in the original program.

Vector Processing :-

→ There is a class of computational problems that are beyond the capabilities of the conventional computer.
→ These are characterized by the fact that they require vast number of computations that will take a conventional computer days or even weeks to complete.
→ 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.

→ A vector V of length n is represented as row vector by

$$V = [V_1\ V_2\ V_3 \ldots V_n]$$

→ The element V_i of vector V is written as V(I) and the index I refers to a memory address & register where the number is stored.

→ Let us consider the program in assembly language that two vectors A and B of length 100 and put the result in vector C

```
      Initialize I = 0
20    Read A[I]
      Read B(I)
      Store C(I) = A(I) + B(I)
      Increment I = I + 1
      If I ≤ 100 go to 20
      Continue.
```

→ 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 instructions of the form.

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

Operation code	Base address Source 1	Base address Source 2	Base address destination	Vector Length

Matrix multiplication

→ Let us consider the multiplication of two 3*3 matrix A and B.

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

→ The product matrix C is a 3x3 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

$$C_{I1} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31}$$

→ This requires three multiplication and (after initializing C_{11} to 0). three addition.

→ Total number of addition or multiplication required is 3 * 9

→ In general inner product consist of the sum of K product terms of the form:

$$C = A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4 + \cdots A_KB_K$$

→ In typical application value of K may be 100 or even 1000.

→ The inner product calculation on a pipeline vector processor is shown below.

→ Floating point adder and multiplier are assumed to have four segments each.

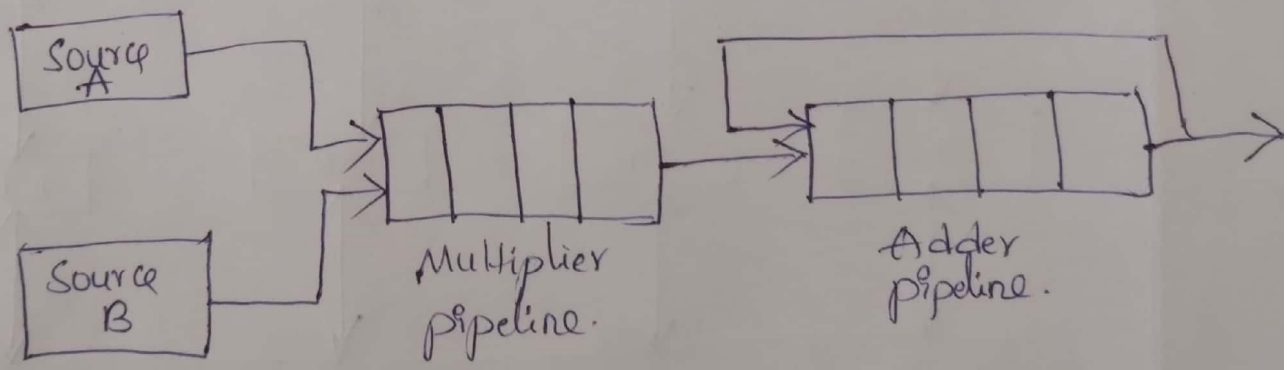


Fig:- Pipeline for calculating an inner product.

$$C = A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \cdots$$
$$+ A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \cdots$$
$$+ A_3B_3 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \cdots$$
$$+ A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16} + \cdots$$

→ The four partial sum were added to form the final sum

Memory Interleaving

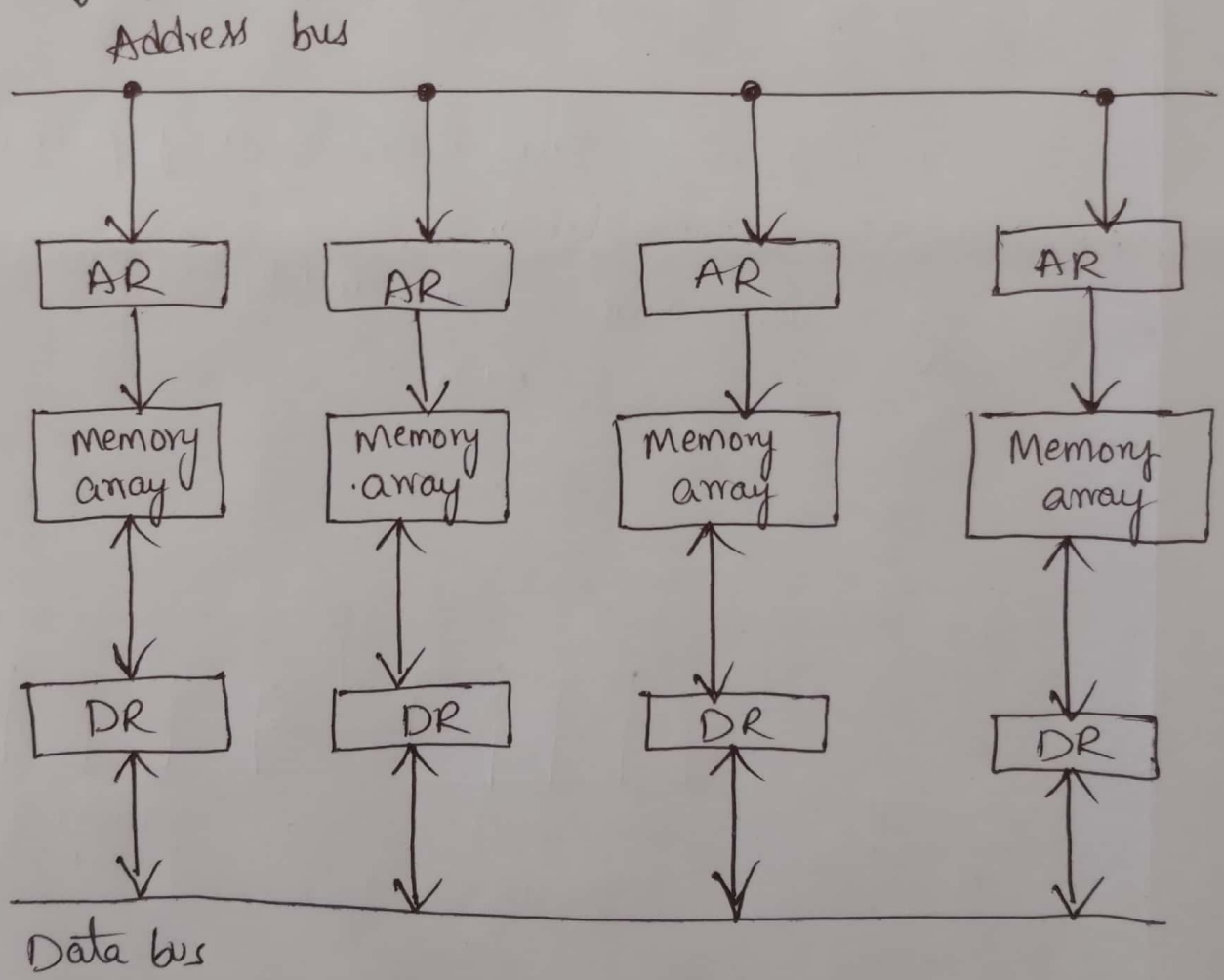


fig: Multiple module memory Organization

→ It is a technique for compensating the relatively slow speed of DRAM (Dynamic RAM). In this technique, the main memory is divided into memory banks which can be accessed individually without any dependency on the other.

→ The advantage of a modular memory is that it allows the use of a technique called interleaving. In an Interleaved memory, different sets of addresses are assigned to different memory modules.

Array Processors :—

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.

a) An attached array processor

b) SIMD Array processor.

Attached array processor:-

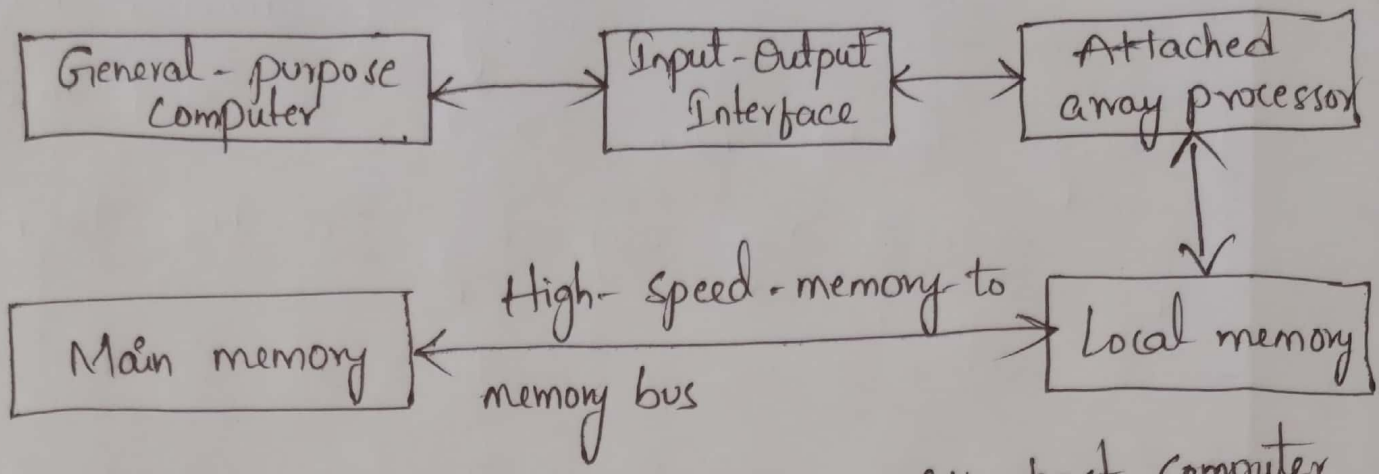


fig: Attached array processor with host computer

→An attached array processor is an auxilary processor attached to a general-purpose computer.

→It is intended to improve the performance of the host-computer in specific numerical computation tasks.

→the above figure shows the interconnection of an attached array processor to a 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 system with the attached processor satisfies the needs for complex arithmetic applications.

SIMD Array Processor :-

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.

→ The processing units are synchronized to perform the same operation under the control of a common control unit.

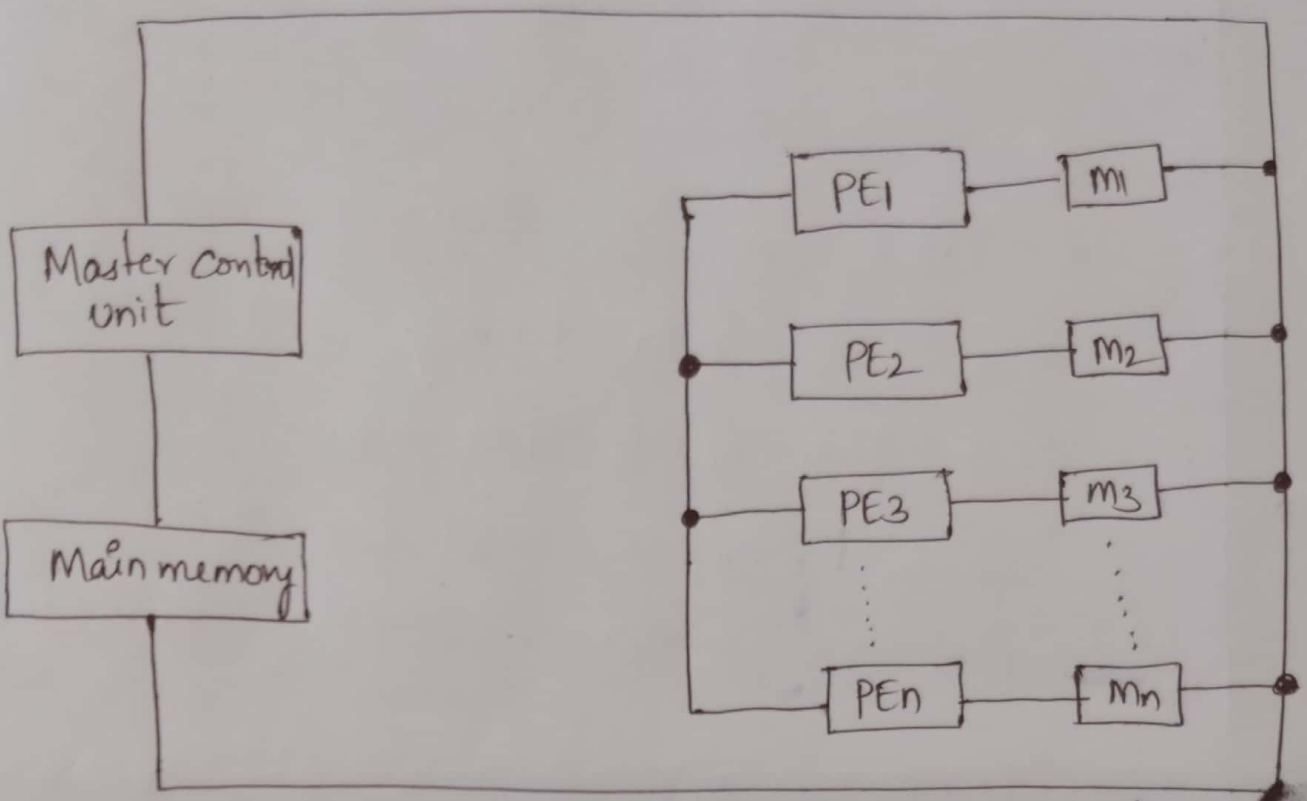


Fig: SIMD array processor Organization

→ The above figure 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.

→ A 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.

Multiprocessors

Characteristics of Multiprocessors :-

→ A multiprocessor is an interconnection of two & more CPU's with memory and input-output equipment.

→ The term processor in multiprocessor system can mean either a central processing unit (CPU) & can input-output processor (IOP)

→ Multiprocessors are classified as multiple instruction stream multiple data stream (MIMD) systems.

→ Multiprocessing improves the reliability of the system.

→ Computation can proceed in parallel in one of two ways

- Multiple independent jobs can be made to operate in parallel
- A single job can be partitioned into multiple parallel tasks.

→ Improved system performance

→ Multiprocessing can improve performance by decomposing a program into parallel executable tasks.

Interconnection structures :-

The components that form a multiprocessor system are CPUs, IOPs connected to input-output devices, and a memory unit.

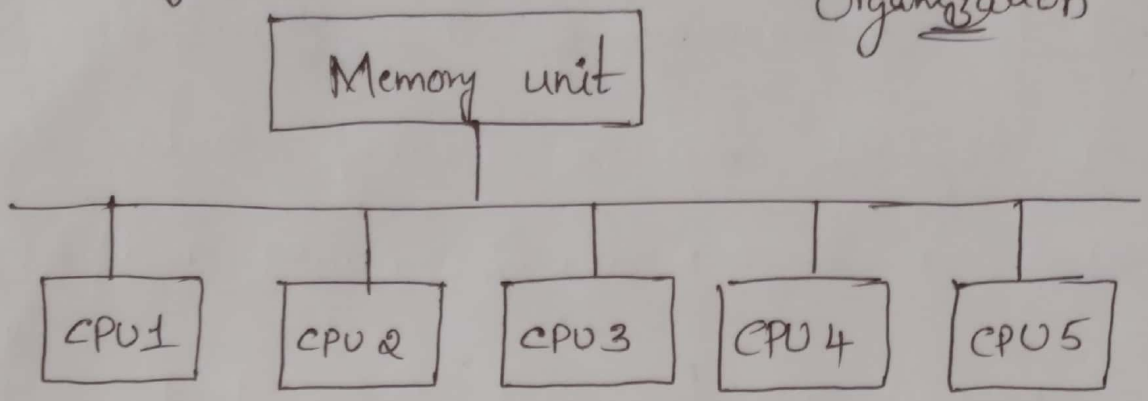
→ There are several physical forms available for establishing an interconnection network.

- Time-shared common bus
- Multiport memory
- Crossbar switch
- Multistage switching network
- Hypercube system.

Time shared common bus :-

→ A common bus multiprocessor system consist of a number of processors connected through a common path to a memory unit.

Fig: Time shared common bus Organization



→ But the main disadvantage of these Organization is only one processor can Communicate with the memory & another processor at any given time

→ A more economical implementation of a dual bus structure is depicted.

Multiport memory :-

→ A multiport memory system employs seperate buses between each memory module and each CPU.

→ Memory access conflicts are resolved by assigning fixed priorities to each memory port.

→ The advantage of these Organization is high transfer rate. Can be achieved because of the multiple paths.

→ The disadvantage of these organization is, it requires expensive memory control logic and a large number of cables and connections.

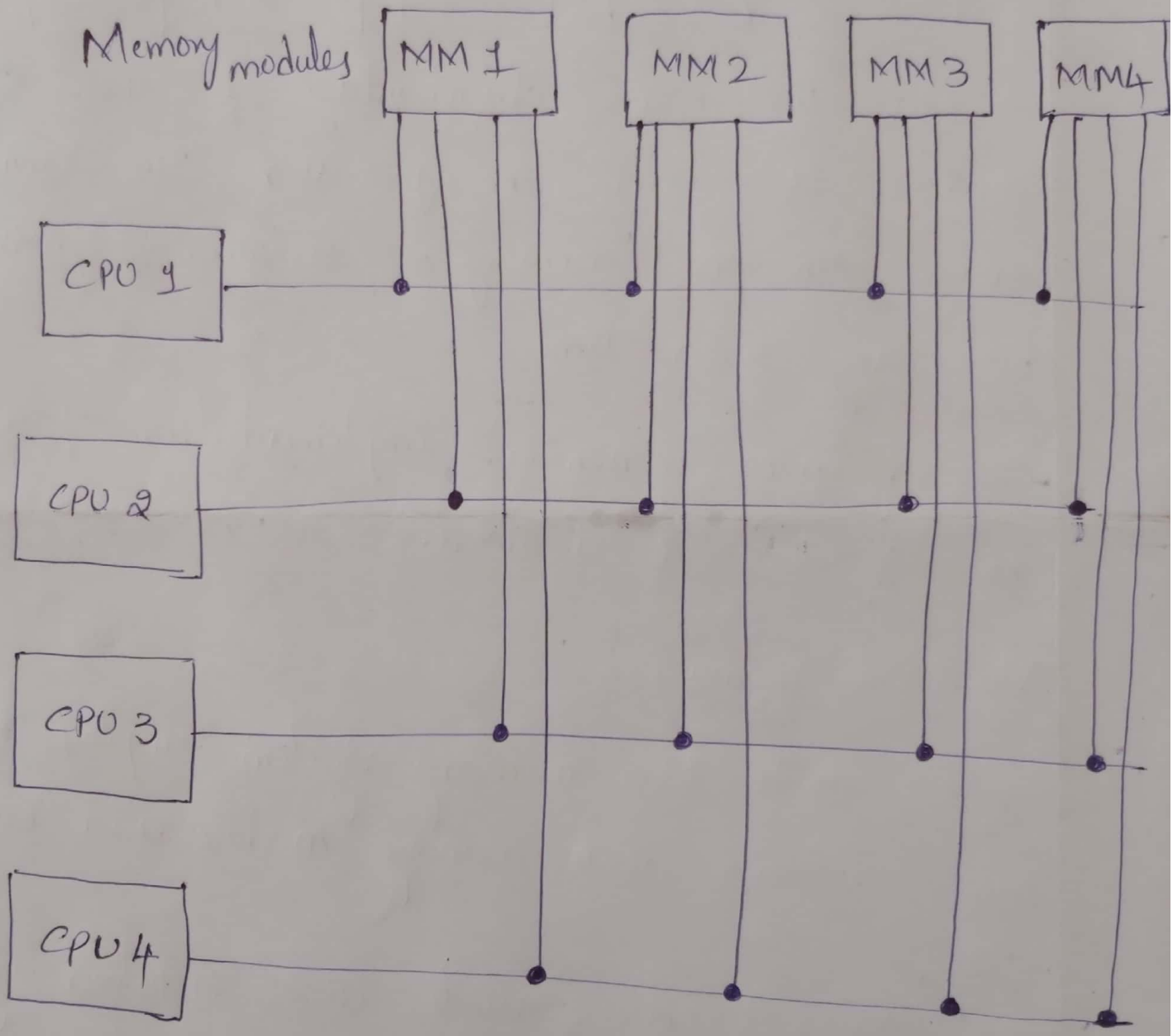


Fig: Multiport Memory Organization

Crossbar Switch :-

→ It consist of a number of crosspoints that are placed at intersections between processor buses and memory module paths.

→ The small square in each crosspoint is a switch that determines the path from a processor to a memory module.

→ The main advantage of these organization is support simultaneous transfers from all memory modules.

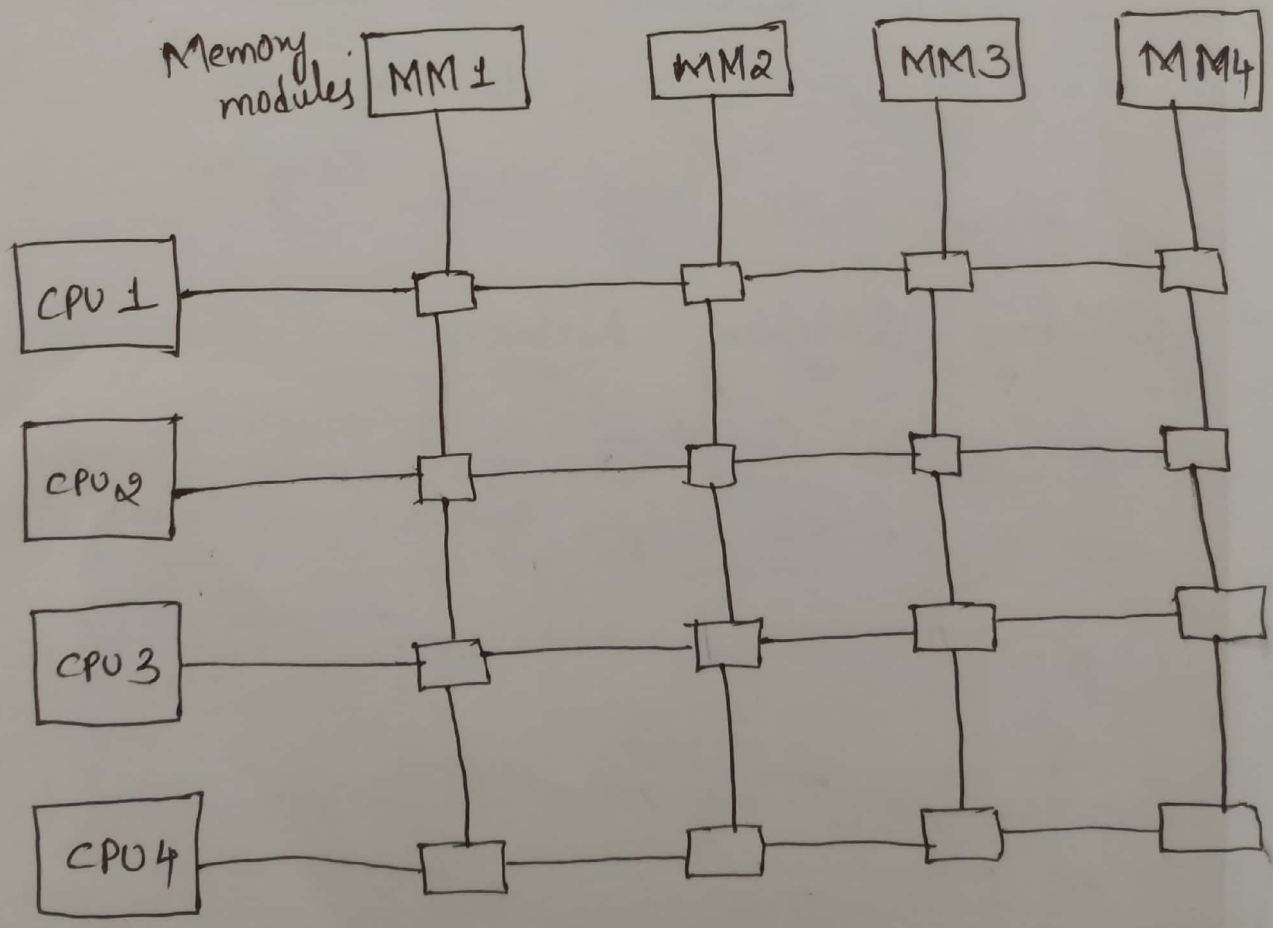


Fig: Crossbar Switch

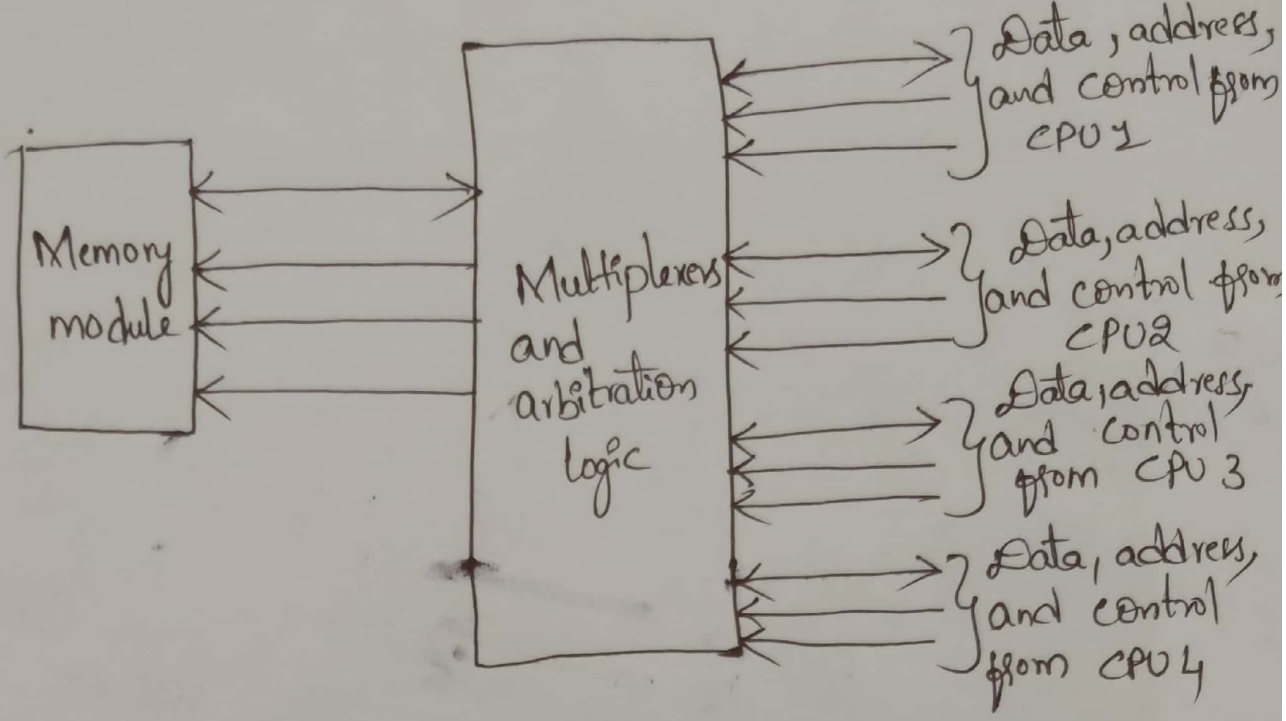
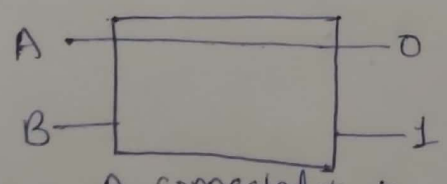


Fig: Block diagram of Crossbar Switch

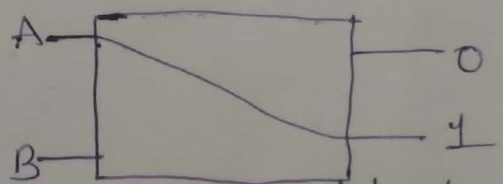
→ The main disadvantage of these Organization is, the hardware required to implement the switch can become quite large and complex.

Multistage Switching Network :-

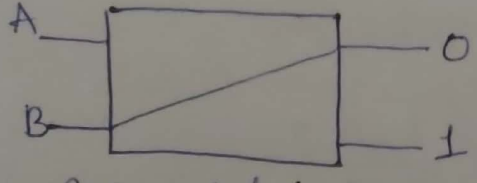
→ The basic component of a multistage network is a two-input, two-output interchange switch.



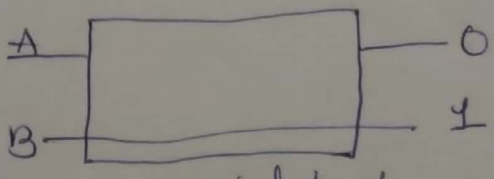
A connected to 0



A connected to 1



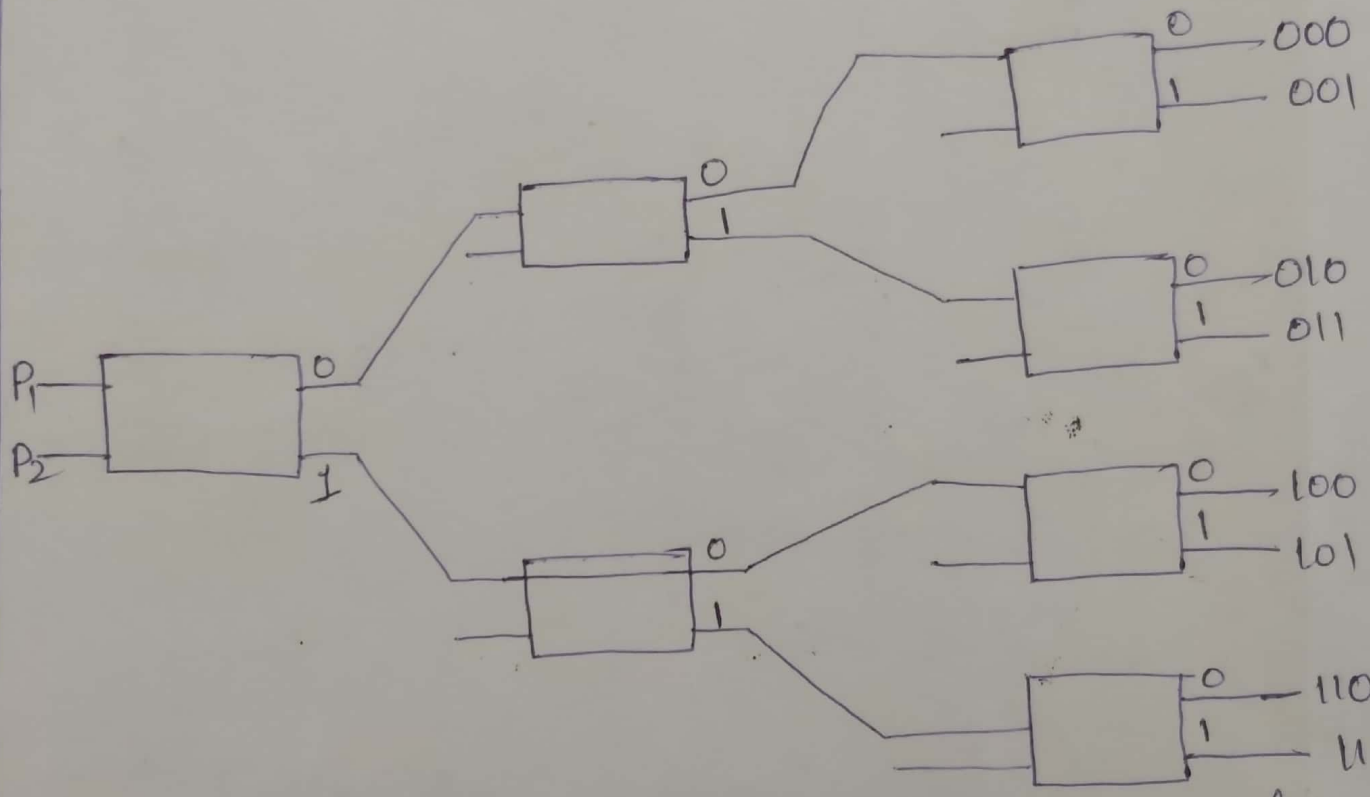
B connected to 0



B connected to 1

→ Using the 2×2 switch as a building block, it is possible to build a multistage network to control the communication between a number of source and destinations.

→ Certain request patterns cannot be satisfied simultaneously i.e. if P1 □ 000 ~ 011, then P2 □ 100 ~ 111



→ One such topology is the Omega switching network

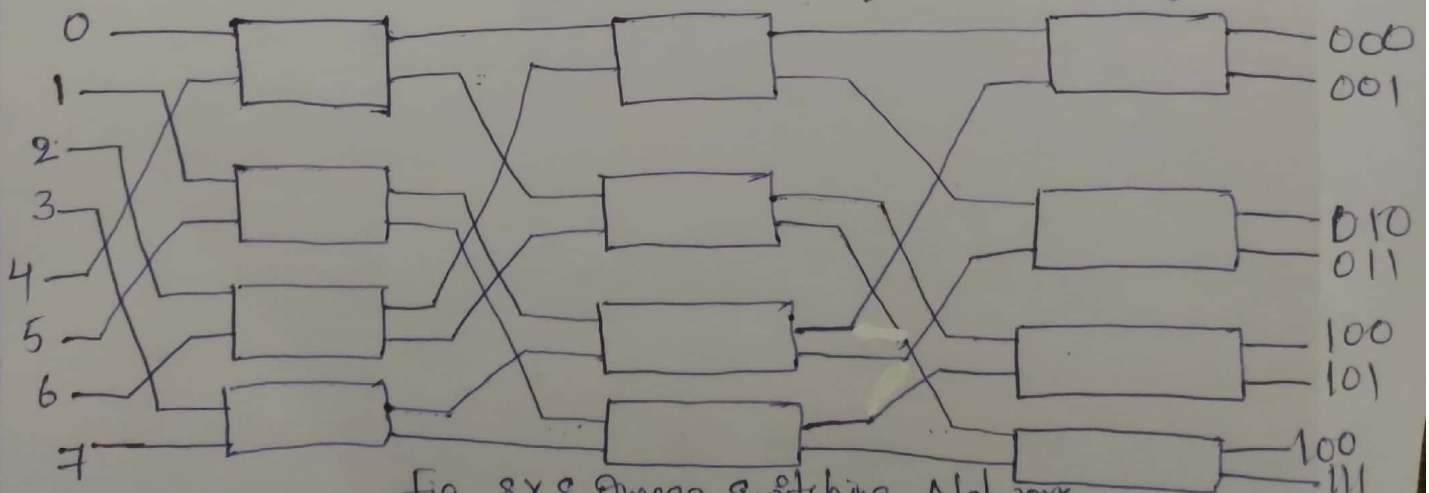
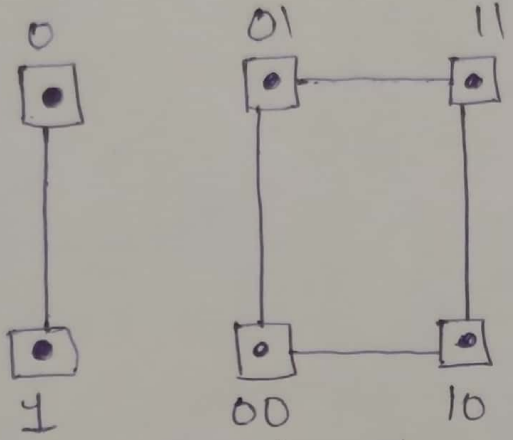


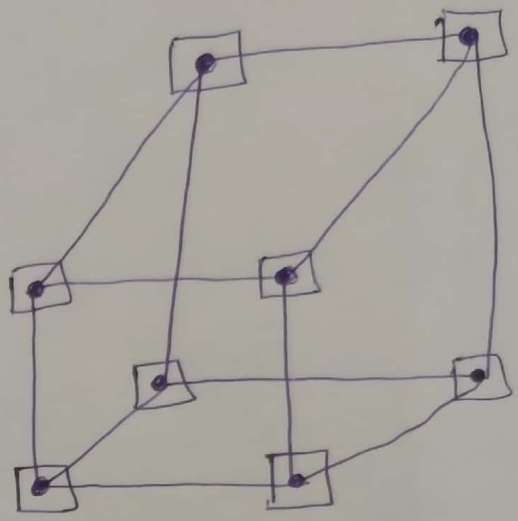
Fig 8×8 Omega Switching Network

Hypercube System :-

1. The hypercube is a binary n-cube multiprocessor structure is a loosely coupled system composed of N = 2^n processors interconnected in an n-dimensional binary cube.



One cube two cube



Three-code

Source node 010

Destination node 001

EX-OR 011

Fig:- Hypercube structures for n = 1, 2, 3.

Interprocessor Arbitration :-

→ Arbitration logic ressolves bus conflict. It would be the part of system bus controller.

→ In multiprocessor system it have number of processors. Simultaneously number of processors want to access the memory. Interprocessor arbitration decides which processor has to be accessed memory first.

→ there are 2 types of arbitrations algorithms

a) static arbitration algorithms

b) Dynamic arbitration algorithms.

a) static arbitration algorithm :-

In static arbitration algorithm the priority of each and every CPU is fixed. It will be achieved by using 2 procedures.

(i) Serial arbitration procedure

(ii) Dynamic arbitration procedure

(i) Serial arbitration algorithm :- The serial priority resolving technique is obtained from a daisy-chain connection of bus arbitration circuits

similar to the priority interrupt logic.

→ The processors connected to the system bus are assigned priority according to their position along the priority control line.

→ The device closest to the priority line is assigned to the highest priority. When multiple devices concurrently request the use of the bus, the device with the highest priority is granted access to it.

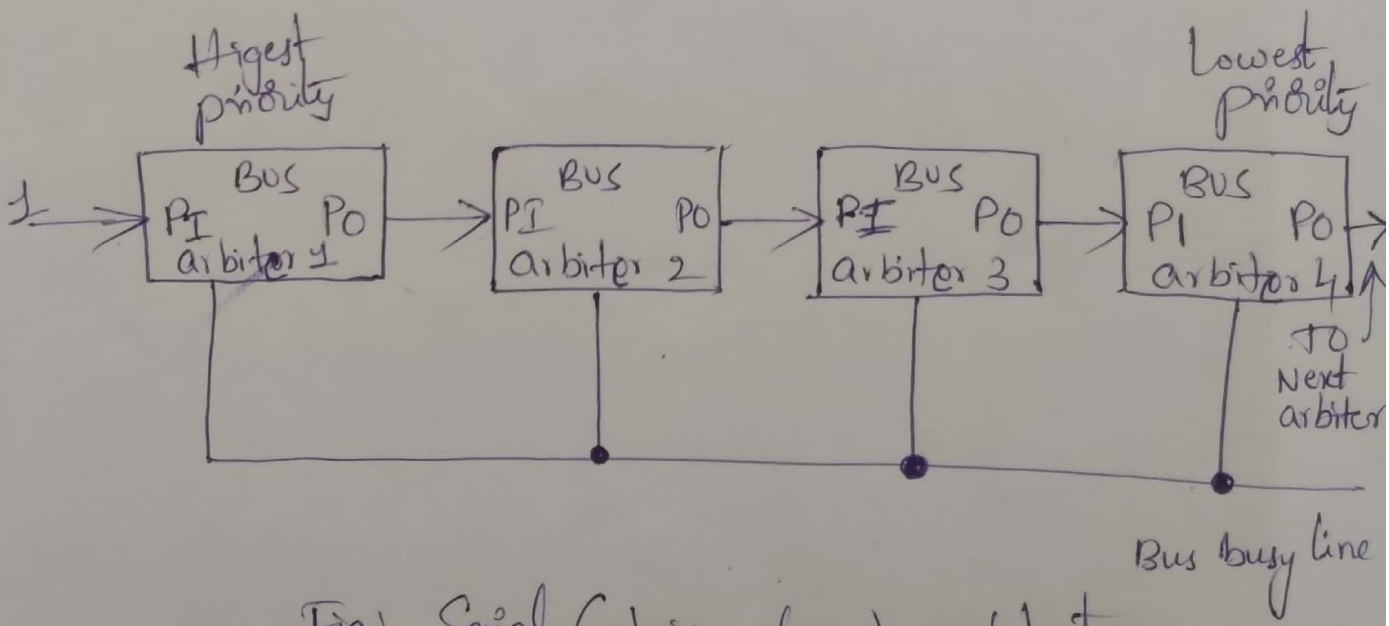


Fig: Serial (daisy-chain) arbitration

→ The above figure shows the daisy-chain connection of four arbiters. It is assumed that each processor has its own bus arbiter logic with priority-in and priority-out lines.

→ The priority out (PO) of each arbiter is connected to the priority in (PI) of the next-lower-priority arbiter.

→ The PI of the highest-priority unit is maintained at a logic 1 value.

→ The highest priority unit in the system will always receive access to the system bus when it requests it.

→ The PO output for a particular arbiter is equal to 1 if its PI input is equal to 1 and processor associated with the arbiter logic is not requesting control of the bus.

→ This is the way that priority is passed to the next unit in the chain.

→ If the processor requests control of the bus and the corresponding arbiter finds its PI input equal to 1, it sets its PO output to 0.

→ Lower priority arbiters receive a 0 in PI and generate a 0 in PO. Thus the processor whose arbiter has a PI = 1 and PO = 0 is the one that given control of the system bus.

→ The main disadvantage of these arbitration is the value of priority assigned to a device is depends on the position of bus.

→ Propagation delay is arises in this method.

(ii) Parallel Arbitration logic :-

→ The parallel bus arbitration technique uses an external priority encoder and a decoder.

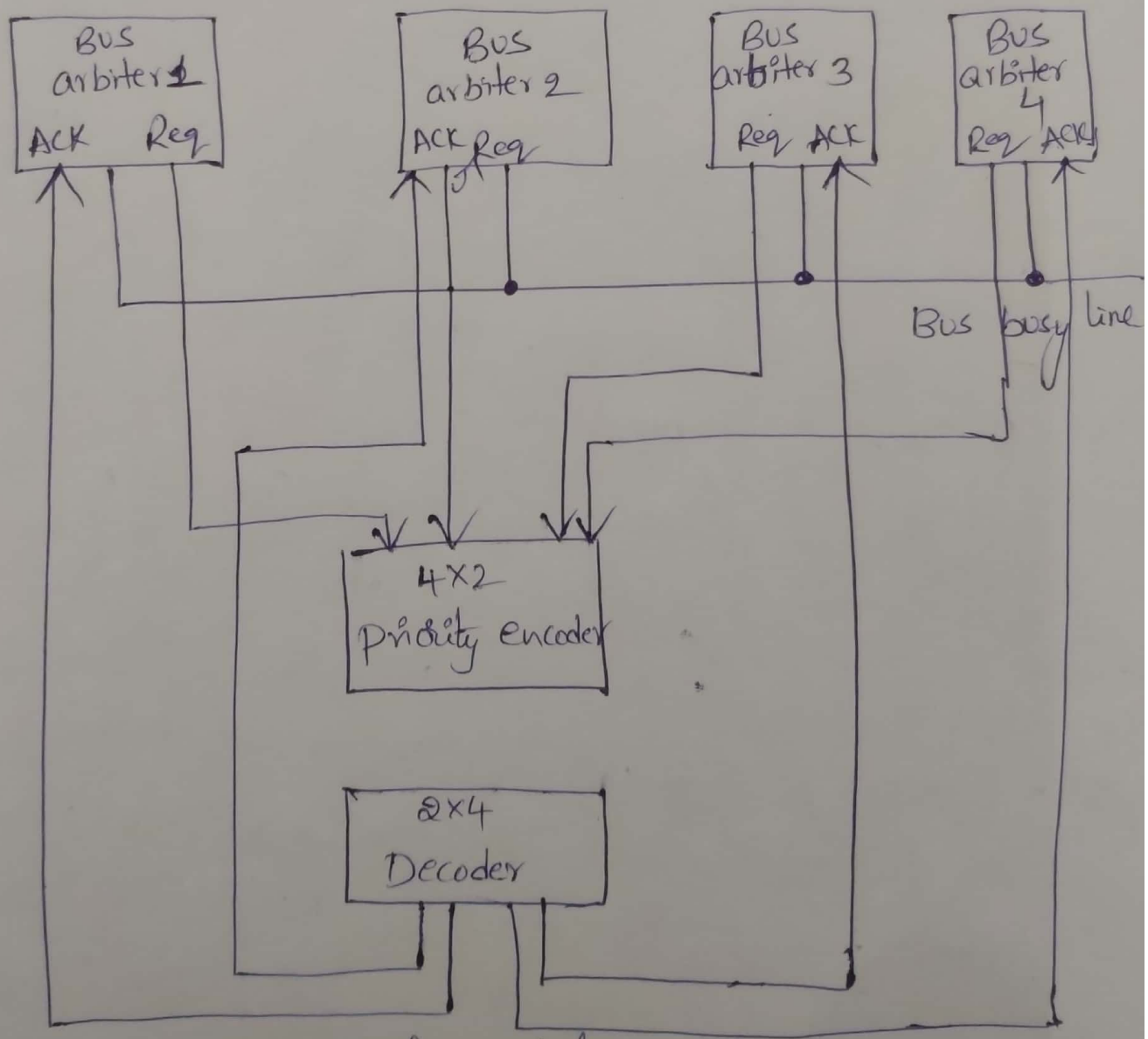


Fig: Parallel arbitration

→ In parallel arbitration each bus arbiter in the parallel scheme has a bus request output line and a bus acknowledge input line.

→ each arbiter enables the request line when its processor is requesting access to the system bus.

→ The processor takes control of the bus if its acknowledge input line is enabled.

→ The bus busy line provides an orderly transfer of control, as in the daisy-chaining case.

→ The parallel arbitration shows the request lines from four arbiters going into a 4x2 priority encoder.

→ The output of the encoder generates a 2-bit code which represents the highest priority unit among those requesting the bus.

→ The 2-bit code from the encoder output drives a 2x4 decoder which enables the proper acknowledge line to grant bus access to the highest-priority unit.

(b) Dynamic Arbitration algorithms :-

→ A Dynamic priority algorithm gives the system the capability for changing the priority of the devices. While the system is in operation

→ few arbitration procedures that use dynamic priority algorithms.

- Time slice
- Pooling
- LRU
- FIFO
- Rotating daisy-chain

Time slice :- the time slice algorithm allocates a fixed-length time slice of bus time that is offered sequentially to each processor, in round-robin fashion.

Pooling :- In a bus system that uses pooling.

LRU :- The least recently used (LRU) algorithm gives the higest priority to the requesting device that has not used the bus for the longest interval.

FIFO :- In the first-come, first-serve scheme, requests were served in the order received.

rotating daisy-chain :- The rotating daisy-chain procedure is a dynamic extension of the daisy-chain algorithm.

Interprocess Communication and Synchronization

Interprocess communication is a mechanism which allows processes to communicate with each other and synchronize their actions.

→ The communication between these processes can be seen as a method of co-operation between them.

→ Processes can communicate with each other through both

- Shared memory
- Message passing.

→ In shared memory, a multiprocessor system may have other shared resources.

For example, a magnetic disk storage unit connected to an IOP may be available to all CPUs. this provides a facility for sharing of system programs stored in the disk.

→ A communication path between two CPUs can be established through a link allows each CPU to treat the other as an I/O device so that messages can be transfered through the I/O path.

→ To prevent conflicting use of ~~sharing~~ shared resources by several processors there must be a provision for assigning resources to processors.

→ This task is given to the operating system.

→ In message passing system the sender processor structures a request, a message, & a procedure, and places it in the memory mailbox.

→ status bits residing in common memory are generally used to indicate the condition of the mailbox, whether it has meaningful information, and for which processor it is intended.

Interprocessor Synchronization

Message passing may be either blocking & non-blocking

→ The instruction set of a multiprocessor contains basic instructions that are used to implement communication and synchronization between cooperating processes.

→ Synchronization is needed to enforce the correct sequence of processes and to ensure mutually exclusive access to shared writable data.

→ Multiprocessor systems usually include various mechanisms to deal with the synchronization of resources.

→ Low-level primitives are implemented directly by the hardware.

→ These primitives are the basic mechanisms that enforce mutual exclusion for more complex mechanisms implemented in software.

→ One of the most popular methods is through the use of a binary semaphore.

Mutual exclusion with a Semaphore

→ A properly functioning multiprocessor system must provide a mechanism that will guarantee orderly access to shared memory and other shared resources.

→ This is necessary to protect data from being changed simultaneously by two or more processors.

→ The mechanism has been termed "mutual exclusion"

→ mutual exclusion must be provided in a multiprocessor system to enable one processor to exclude & lock out access to a shared resource by other processors when it is in a "critical section"

→ A critical section is a program sequence that, once begun, must complete execution before another processor accesses the same shared resource

→ A binary variable called a "Semaphore" is often used to indicate whether or not a processor is executing a critical section.

→ A semaphore is a software-controlled flag that is stored in a memory location that all processors can access.

→ when the semaphore is equal to 1, it means that a processor is executing a critical program, so that shared memory is not available to other processors.

→ when the semaphore is equal to 0, the shared memory is available to any requesting processor.

→ A semaphore can be initialized by means of a test and set instruction in conjunction with a hardware lock mechanism.

→ The test-and-set instruction tests and sets a semaphore and activates the lock mechanism during the time that the instruction is being executed.

TSL SEM

Let the mnemonic TSL designates the "test and set while locked" operation.

→ The instruction will be executed in two memory cycles (the first to read and the second to write) without interference as follows.

R ← M[SEM] ; Test semaphore

M[SEM] ← 1 ; set semaphore.

If the processor finds that R=1, it knows that the semaphore was originally set.

→ If R=0, it means that the common memory is available.

Cache Coherence :-

→ the primary advatage of cache is its ability to reduce the average access time in uniprocessors.

→ when a processor finds a word in cache during a read operation, the main memory is not involved in the transfer.

→ If the operation is to write, there are two commonly used procedures to update memory.

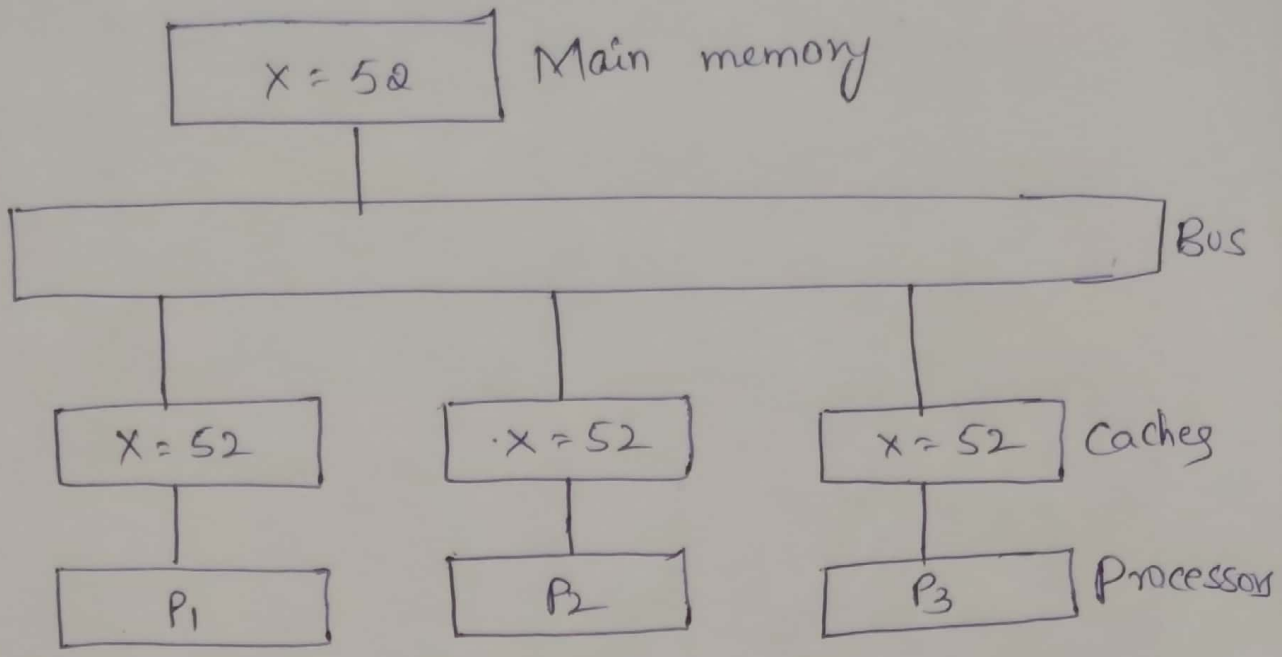
→ In the write through policy, both cache and main memory are updated with every write operation.

→ In the write-back policy, only the cache is updated and the location is marked so that it can be copied later into main memory.

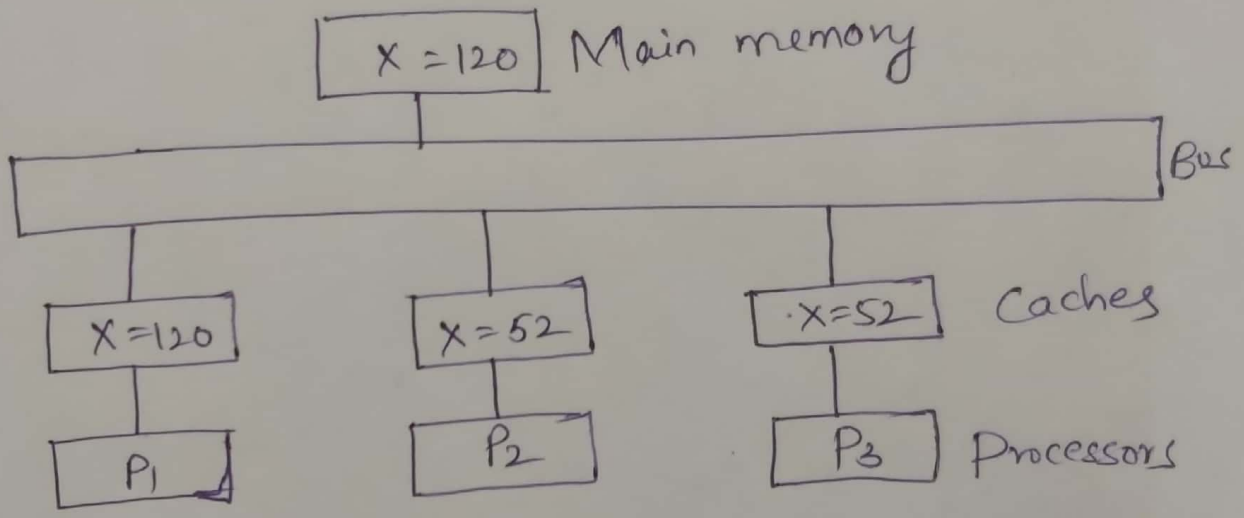
Cache Write Policies :-

There are two main cache write policies.

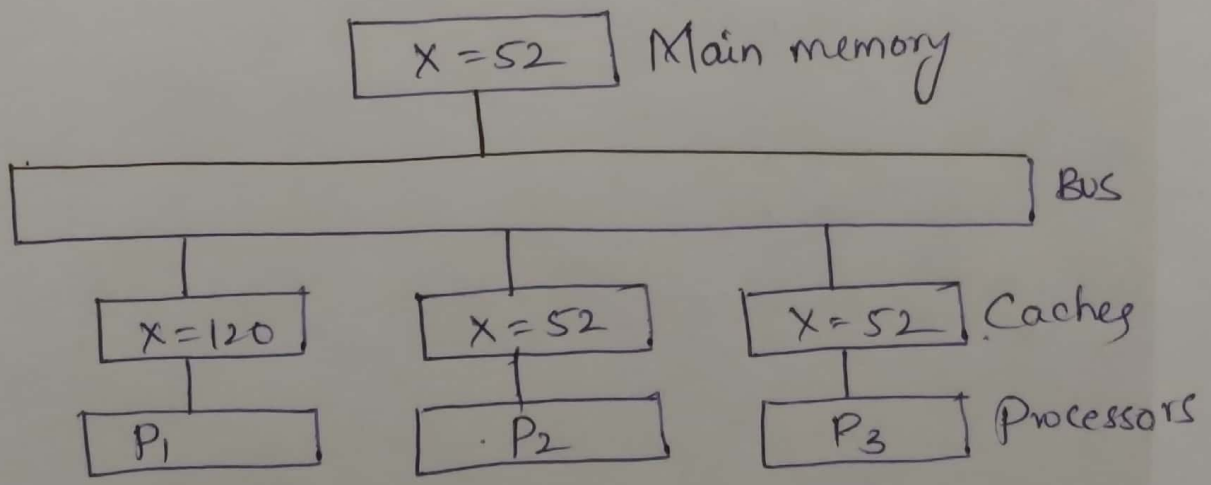
→ write back
→ write through.



Cache configuration after a load on X



(a) With write-through cache policy.



(b) With write-back cache policy.

Write back :- Write operations are usually made only to the cache. Main memory is only updated when the corresponding cache line is flushed from the cache.

Write through :- All write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.

* Unit - 5 completed *