# Pipelining:

Pipelining offers an economical way to realize temporal parallelism in digital computers. To achieve pipelining, one must divide the input task into sequence of subtasks, each of which can be executed by a specialized hardwire stage that operates concurrently with other stages in the pipeline. Successive tasks are streamed into the pipe & get executed overlapped fashion at the subtask level.

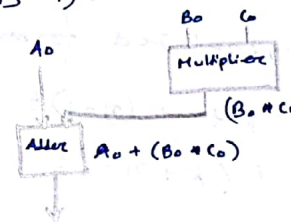
Pipelining is of two types — Linear
                           — Non Linear

- **Linear Pipeline:**

In a uniform-delay pipeline, all tasks have equal processing time at all ~~stages~~ station facilities. However, in reality, successive stations have unequal delays. The stages are seperated latches which holds intermediate results between the stage.

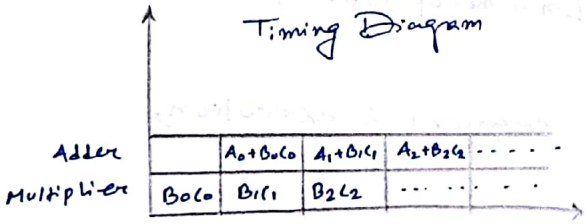
Example: $D[i] = A[i] + B[j] * C[i]$

Here are two subunits i) adder & ii) Multiplier; which are both independent.

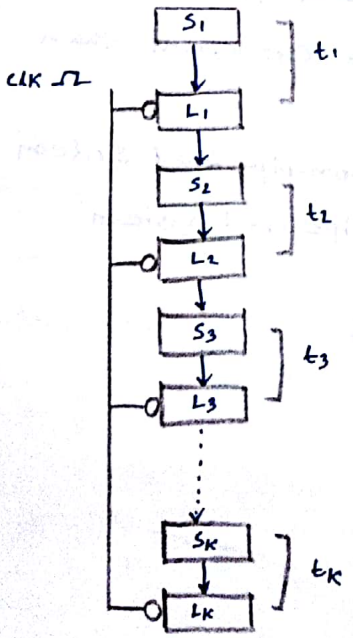



- when the adder is adding $A_0 + B_0 * C_0$, the multiplier will be busy multiplying $B_1 * C_1$

Timing Diagram




## K-Stage Pipeline Example:-

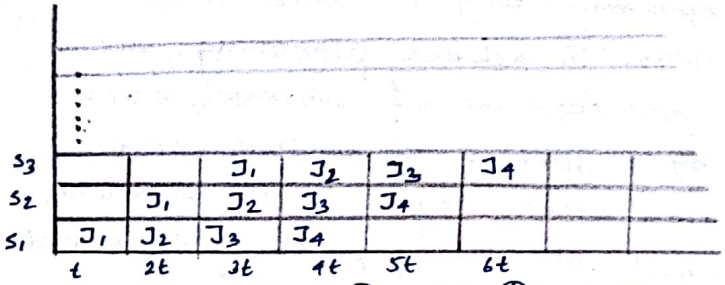



The time taken between two stages in latch = $t_L$

$t = t_L + \max(t_1, t_2, t_3 \ldots\ldots t_K)$

↑
Max time ~~between~~ taken in a typical pipeline considering buffering & operation.

## Calculating time to execute K stage pipelined & non pipelined processor:-




Space-time Diagram Depicting the overlapped operations.

At time Kt job $J_1$ will be processed.

In Above example,

$K = 3$ (No. of stages)

$J_1$ will be completed at $3t$

$J_2$ will be completed at $4t$ $[3t + t]$

$J_3$ will be completed at $5t$ $[3t + 2t]$

$J_4$ will be completed at $6t$ $[3t + 3t]$

$J_n$ will be completed at $(3t + (n-1)t)$

So, the generalized formula is,

$$t_p = Kt + (n-1)t \quad [K = \text{no. of stages}, n = \text{no. of jobs}]$$
$$= t(K + n - 1)t$$

- For Strictly non-pipelined sequential execution,

$$\text{time} = n \times K \times t.$$

## ★ Speedup Ratio:-

Speedup ratio is important to determine how many time a pipelined system is more efficient than a non-pipelined system.

$$\text{Speedup Ratio} = \frac{\text{Time taken by a non-pipelined system}}{\text{Time taken by a pipelined system}}$$

$$= \frac{nKt}{(n+K-1)t} = \frac{nK}{n+K-1}$$

| K \ n | 4 | 5 | 7 | | |
|---|---|---|---|---|---|
| 1000 | 1003 | 1004 | 1006 | | |
| 10,000 | 10003 | 10004 | 10006 | | |
| ⋮ | ⋮ | ⋮ | ⋮ | | |
| $\alpha$ | 0 | 0 | 0 | | |

Proportion - $\frac{3}{1003}$ ↓ $\frac{3}{10003}$ ↓ $\frac{3}{100003}$ ⋮ ↓ $\frac{3}{\alpha} \to 0$

∴ the value of K tends towards total no. of instructions

| n \ K | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 10 | 20/11 1.8 | 30/12 2.5 | 40/13 3.07 | 50/14 3.57 |
| 100 | 200/101 1.9 | 300/102 2.94 | 400/103 3.88 | 500/104 4.807 |
| 1,000 | 2000/1001 | 3000/1002 2.99 | 4000/1003 3.988 | 5000/1004 4.980 |
| 10,000 | 20000/10001 1.998 | 30000/10002 2.999 | 40000/10003 3.998 | 50000/10004 4.998 |
| 100,000 | 200000/100001 1.9998 | 300000/100002 2.9999 | 400000/100003 3.9998 | 500000/100004 4.9998 |

↳ Speedup increases as the value of K increases; but the extent of increase is gradually decreasing.

$$\lim_{n \to \alpha} S_K = \lim_{n \to \alpha} \frac{nK}{n+K-1}$$

$$= \lim_{n \to \alpha} \frac{nK}{n} \quad [\because \alpha + K - 1 = \alpha \text{ as } n \to \alpha]$$

$$= K$$

∴ $S_K$ will tend towards K as $n \to \alpha$

$S_K \to K$ as $n \to \alpha$

## Classification of Pipelined processors:

- Arithmatic Pipelining: The arithmatic logic units of a computer can be segmentized for pipeline operations in various data formats. Ex:- TI ASC, STAR-100, cyber-205 etc.

- Instruction Pipelining: The execution of a stream of instructions can be pipelined by overlapping the execution of current instruction with the fetch, decode, operand fetch, execute, memory write etc. Almost all high. performance computers are now equipped with instruction-execution pipelines.

- Processor pipelining:

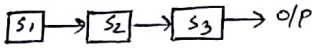
This refers to the pipeline processing of the same data stream ~~of instructions~~ by cascade of processors. Each of which processes a specific task.

The data stream passes the first processor with results stored in a memory block which is also accesible by the second processor. The second processor then passes the refined result & so on. The pipelining of multiple processors is not yet well accepted as a common practice.

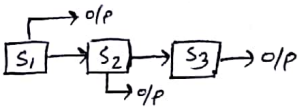
- Unifunction vs. multifunctional pipelines:

☑ A pipeline with a fixed & dedicated function is called unifunctional pipeline.

Example: floating point adder.




☑ A multifunction pipe may perform different functions either at different times or at same time; by interconnecting different steges at different times.




- Static vs. dynamic pipelines:

☑ A static pipeline may assume only one functional configuration at a time. It can be unifunctional or multifunctional.

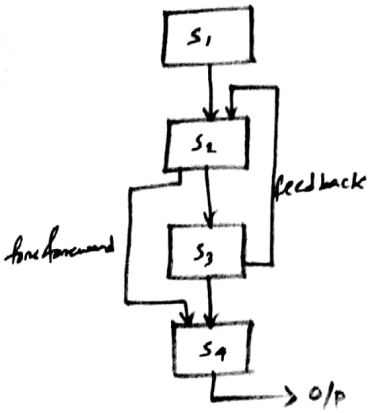
☑ A dynamic pipeline processor permits several functional configurations to exists simultaniously. In this case, a dynamic pipeline must be multifunctional.

- Scalar vs. Vector pipelines:

☑ A scalar pipeline processor processes a sequence of scalar operands under control of DO loop. Instructions in a small DO loop are often prefetched into the instruction buffer. The required scalar operands for repeated scalar instructions are moved into a data cache in order to continuously supply the pipeline with operands.

☑ Vector pipelines are specially designed to handle vector instructions over vector operands. The design of vector pipeline is extended form of a scalar pipeline. The handling of vector operands in vector pipelines is under firmware.

## Non-Linear Pipeline



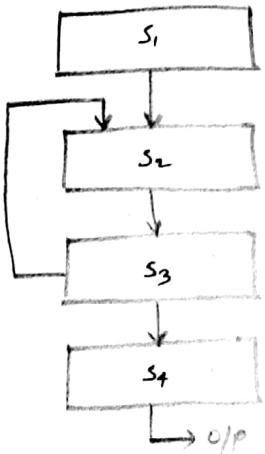

Nonlinear Pipeline can be static or Dynamic
It must have feedback & fore forward loops.

$J_1 = S_1, S_2, S_3, S_4$

$J_2 = S_1, S_2, S_3, S_2, S_3, S_4$

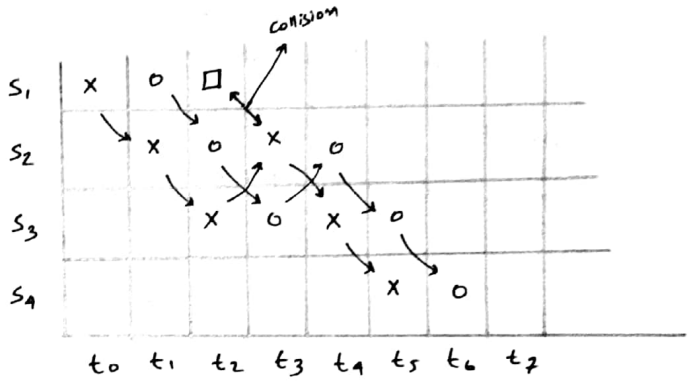
$J_3 = S_1, S_2, S_4$

## Reservation Table:




Static Non-linear pipeline.

Job sequence : $S_1, S_2, S_3, S_2, S_3, S_4$



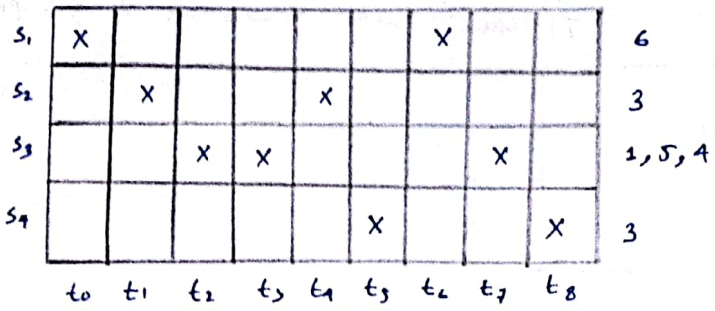

One complete time cycle for one job.

- **Delay:**

Delay is the time interval between two successive jobs and also known as latency.

A forbidden latency is a latency which cannot be assigned for a job.

# Calculation of forbidden latency, collision vector.




∴ forbidden latency $F = (1, 3, 4, 5, 6)$

We choose the minimum possible latency in this case 2. (2 is only applicable for $J_1$ & $J_2$)

To calculate forbidden latency for all jobs, we declare 'c' known as 'Collision vector'.

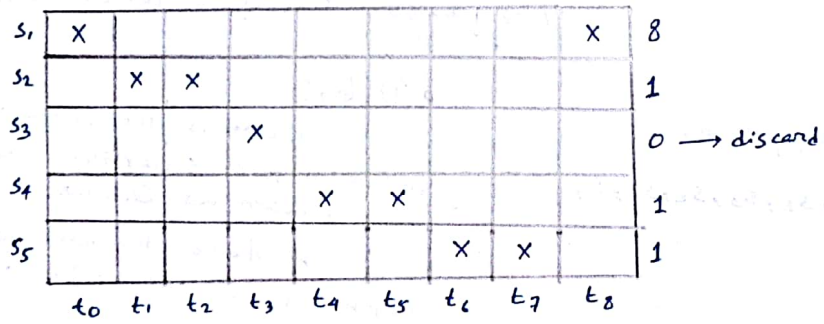
The 'Collision Vector' will be of maximum value in forbidden latency,

For upper example,

$$c = (c_6\ c_5\ c_4\ c_3\ c_2\ c_1)$$
$$= (1\ 1\ 1\ 1\ 0\ 1)$$

$\left[ c_i = 1 \text{ if } i \in F \text{ otherwise } 0 \right]$

Ex: 2




$F = (1, 8)$

$$c = (c_8\ c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1)$$
$$= 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1$$

# Reservation Table for Dynamic Pipeline:

Eg:

| | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| $S_1$ | X | | | | | | | X | X | 7, 8, 1 |
| $S_2$ | | X | X | | | | | | | 1 |
| $S_3$ | | | | X | | X | | | | 2 |
| $S_4$ | | | | | X | X | | | | 1 |
| $S_5$ | | | | | | | X | X | | 1 |

$F = (1, 2, 7, 8)$

$C = (C_8\ C_7\ C_6\ C_5\ C_4\ C_3\ C_2\ C_1)$

$= (1\ 1\ 0\ 0\ 0\ 0\ 1\ 1)$

## Defⁿ: (Greedy Cycle)

We choose minimum cost of edge from starting vertex & try to find simple circuit. And continue for all vertices.

## Calculation of Minimal Avg Latency:

| | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| $S_1$ | X | | | | | | | | X | 8 |
| $S_2$ | | X | X | | | | | X | | 6, 5, 1 |
| $S_3$ | | | | X | | | | | | |
| $S_4$ | | | | | X | X | | | | 1 |
| $S_5$ | | | | | | | X | X | | 1 |

$f = (1, 5, 6, 8)$

$C = (C_8\ C_7\ C_6\ C_5\ C_4\ C_3\ C_2\ C_1)$

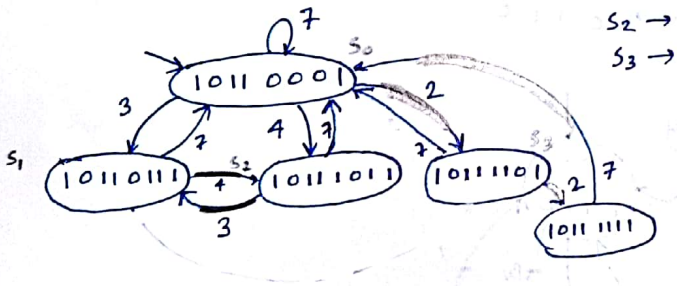
$= (1\ 0\ 1\ 1\ 0\ 0\ 0\ 1)$

$S_0 \to 2, 3, 4, 7$

$S_1 \to 2, 7$

$S_2 \to 3, 7$

$S_3 \to 2, 7$




We found two greedy cycles

i) (2, 2, 7) Avg = $\frac{11}{3} = 3.6$

ii) (3, 4) Avg = $\frac{7}{2} = 3.5$

∴ Minimal Avg latency = 3.5

## Rough

00101100 ↦ 2
10110001
10111101

00010110 ↦ 3
10110001
10110111

0000 0001 ↦ 4
1011 0001
1011 1011

00000001 ↦ 7
1011 0001
1011 0001

0000 1011
00[illegible]01 ↦ 2 (S1)
10110001
[illegible]

0001011[illegible] ↦ 3 (S2)
10110001
10110111

00101111 ↦ 2 (S3)
10110001
10111111

## For Multifunctional Pipelines with various o/ps:-

$M_A = \begin{pmatrix} V_{AA} \\ V_{BA} \end{pmatrix}$ $M_B = \begin{pmatrix} V_{AB} \\ V_{BB} \end{pmatrix}$

If n = no. of o/p matrix will be of $(n \times n^2)$

Ex:

| | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ |
|---|---|---|---|---|---|
| $S_1$ | A | B | | A | B |
| $S_2$ | | A | | B | |
| $S_3$ | B | | AB | | A |

| $V_{AA}$ | $V_{AB}$ | $V_{BA}$ | $V_{BB}$ |
|---|---|---|---|
| 3 | 4 | 2 | 3 |
| 3 | 2 | 4 | |
| 2 | | 2,4 | 2 |

$F_{(V_{AA})} = 3,2$
$= (1\,1\,0) \rightarrow 0\,1\,1\,0$

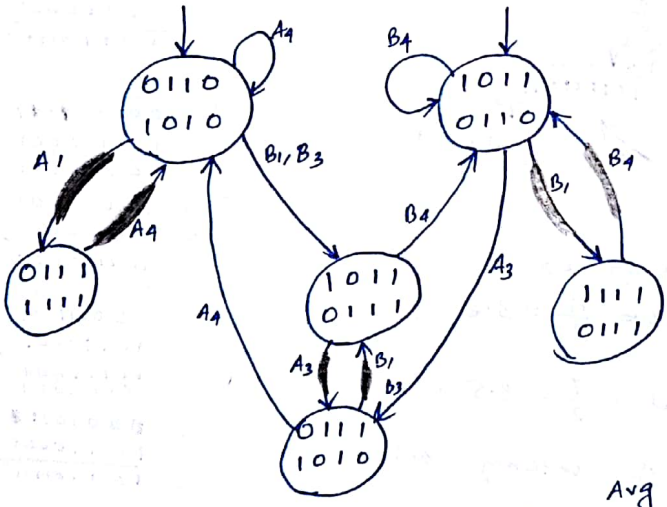
$F_{(V_{AB})} = (4,2,4)$
$= (1\,0\,1\,0) \rightarrow 1\,0\,1\,1$

$F_{(V_{BA})} = (1,2,4)$
$= 1\,0\,1\,0 \rightarrow 1\,0\,1\,0$

$F_{(V_{BB})} = (3,2)$
$= (1\,1\,0) \rightarrow 0\,1\,1\,0$

$M_A = \begin{pmatrix} 0110 \\ 1010 \end{pmatrix}$ — A, B

$M_B = \begin{pmatrix} 1011 \\ 0110 \end{pmatrix}$ — A, B




Greedy Cycle → A (A1, A4)
(A3, B4)
B (B1, B4)

Avg latency
2.5
2
2.5

∴ min Avg latency = 2

# 5 stage instruction pipeline:-

i) fetch
ii) Decode
iii) Data operands
iv) Execution
v) Write result in main memory

Example:

$c = a+b$, $d = a-b$, $e = a*b$, $f = a/b$, $g = a \wedge b$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $S_5$ | | | | | write to c | write to d | write to e | | |
| $S_4$ | | | | a+b | a-b | a*b | | | |
| $S_3$ | | | fetch a/b | fetch a/b | fetch a/b | | | | |
| $S_2$ | | $I_0$ | $I_1$ | $I_2$ | | | | | |
| $S_1$ | $I_0$ | $I_1$ | $I_2$ | | | | | | |
| | $T_0$ | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | | | |

## Hazards:

Pipeline hazards are caused by resource-usage conflicts among various instructions in the pipeline. Such hazards are triggered by interinstruction dependencies.

When successive instructions overlap their fetch, decode and execution through a pipeline processor, interinstructions dependencies may arise to prevent the sequential data flow in the pipeline. Until the completion of the previous instances, ~~two stages of a~~ the present instruction cannot be instantiated into the pipeline. In other instances, two stages of a pipeline may need to update the same memory location.

— There are three classes of Data Hazards

i) WAR (Write After Read)
ii) RAW (Read After Write)
iii) WAW (Write After Write

★ Bernstain's Condition:-

Using Bernstain's Condition, we can detect hazards.

R → Set of all those values of which will be required to execution

W → Value of which will undergo a change during execution

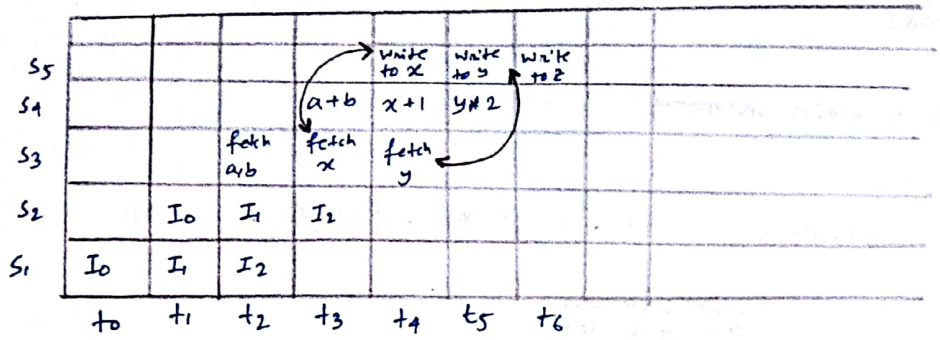
i) $R(S_i) \cap W(S_j) = \phi$
ii) $W(S_i) \cap R(S_j) = \phi$
iii) $W(S_i) \cap W(S_j) = \phi$

**Example: 1)**

$i_0$ $x = a + b$

$i_1$ $y = x + 1$

$i_2$ $z = y * 2$




Considering $i_0$ & $i_1$,

$R(i_0) = \{a, b\}$

$W(i_0) = \{x\}$

$R(i_1) = \{x\}$

$W(i_1) = \{y\}$

i) $R(I_0) \cap W(I_1) = \phi$ ✓

ii) $W(I_0) \cap R(I_1) \neq \phi$ ✗

Hence $i_0$ & $i_1$ can't be executed concurrently.

**Example: 2)**

$S_i$ : $x = a + b$

$S_j$ : $a = y * 5$

$R(S_i) = \{a, b\}$
$R(S_j) = \{y\}$
$W(S_i) = \{x\}$
$W(S_j) = \{a\}$

$R(S_i) \cap W(S_j) \neq \phi$ Hence data hazard occured.

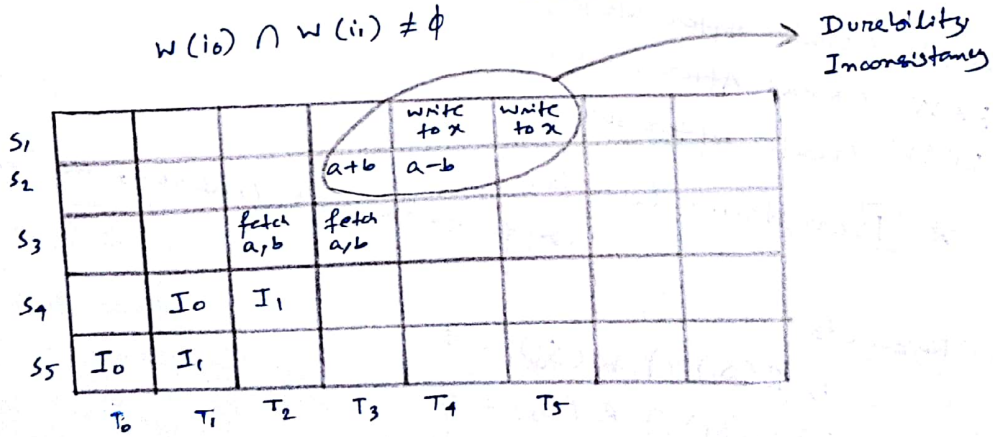
**Example: 3)**

$i_0$ : $x = a + b$

$i_1$ : $x = a - b$

$R(i_0) = \{a, b\}$
$W(i_0) = \{x\}$
$R(i_1) = \{a, b\}$
$W(i_1) = \{x\}$

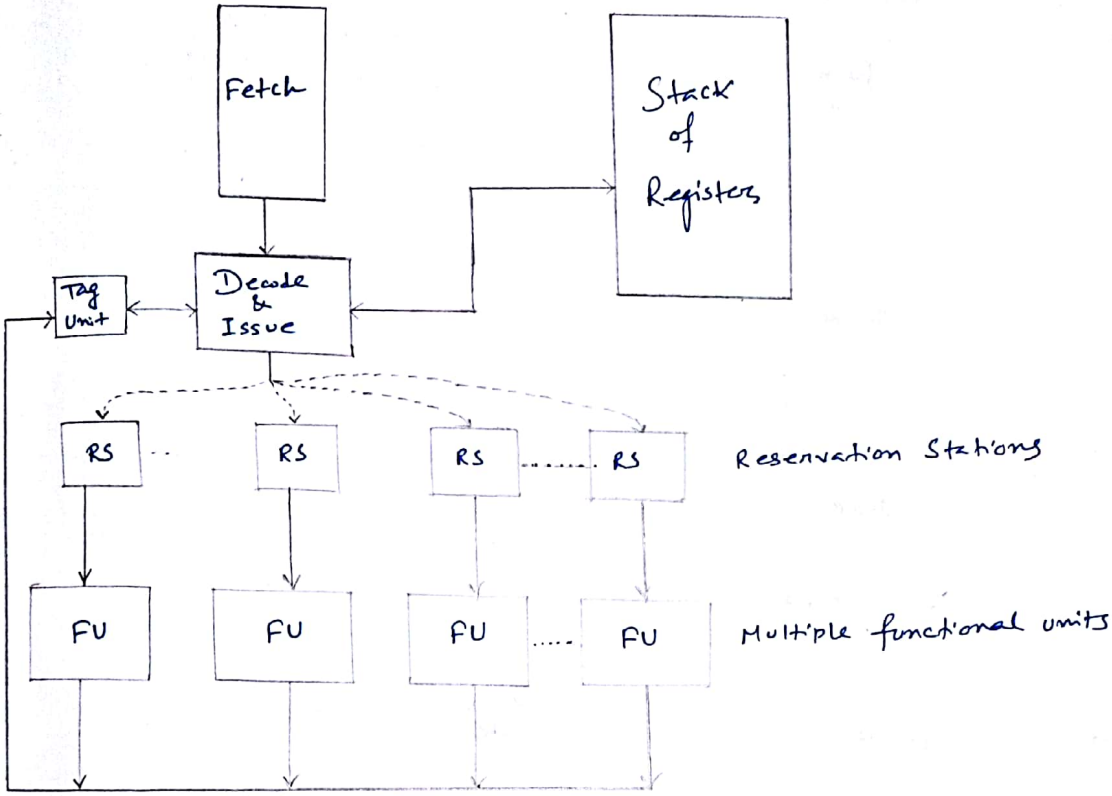
$W(i_0) \cap W(i_1) \neq \phi$



To Resolve data depencies two dynamic scheduling algorithms are there

1) Scoreboarding

2) Tomasulo's Algorithm

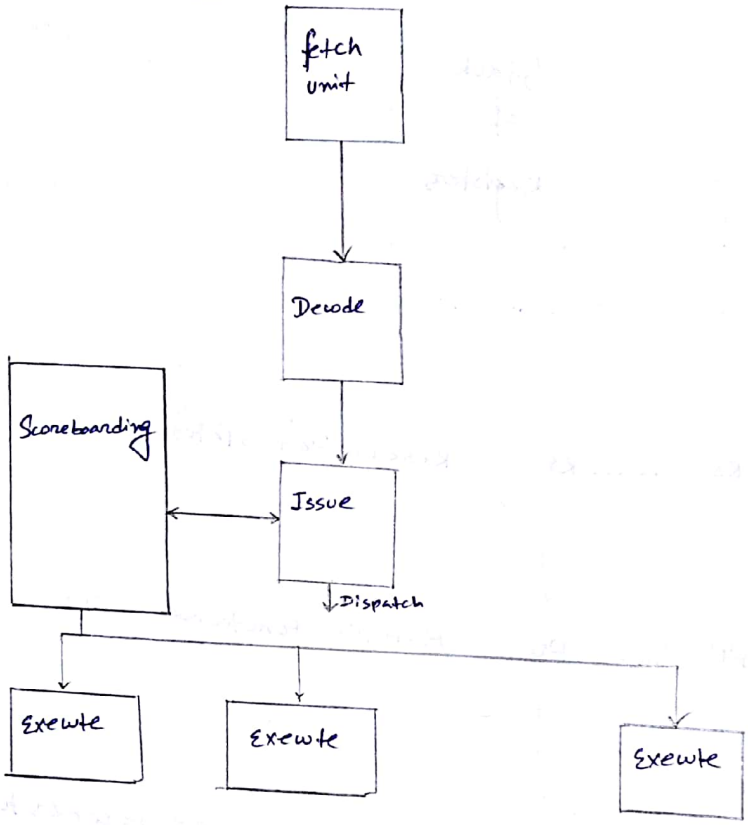



Sometimes in certain pipeline stage becomes the bottleneck. This bottleneck problem can be resolved by using multiple functional units. In order to resolve data or resource dependencies among the successive instructions entering the pipeline, the reservation stations (RS) are used with each functional units. Operands can wait in the RS until its data dependency have been resolved. Each RS is monitored by a tag unit. The tag unit keeps checking the tags from all currently used registers or variable. This register tagging technique allows the H/w to resolve conflicts b/w source and destination registers assigned for multiple instructions. Besides resolving conflicts, the RSs also serve as buffer to interface the pipelined functional units with the decode & issue unit.

The multiple functional units are supposed to operate parallel once the dependency are resolved.

An issued instruction ~~for~~ whose operands are not available is forwarded to an RS associated with the functional unit it will use. It waits until its data dependency is resolved. The dependence is resolved by monitoring the result bus. When all operands for an instruction are available, it is dispatched to the functional unit for execution. All working registers are tagged. The tag can signal the availability of the register.

Scoreboarding :— (A generalized Approach)




The scoreboarding technique assumes the presence of multiple functional units that allow instructions to complete out of the original program order. Instructions are issued to available functional units. To control the correct routing of data b/w execution units & registers.

A centralized control unit known as the scoreboard is used. This unit keeps track of the registers needed by instructions waiting for various functional units. When all registers have valid data, the scoreboard enables the instruction execution.

Similarly, when a functional unit is finished executing, it signals the scoreboard to release the resources. The scoreboard is a centralized control logic which keep tracks of the status of registers and multiple functional units. When functional units generate new results, some data dependency can be resolved and thus a higher degree of parallelism can be explored.

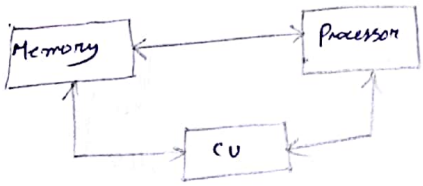
# Parallel Architecture

## Flynn's Classification:-

Flynn's classification of computers was based on Instruction stream & Data stream for parallel computing.

i) SISD
ii) SIMD
iii) MISD
iv) MIMD

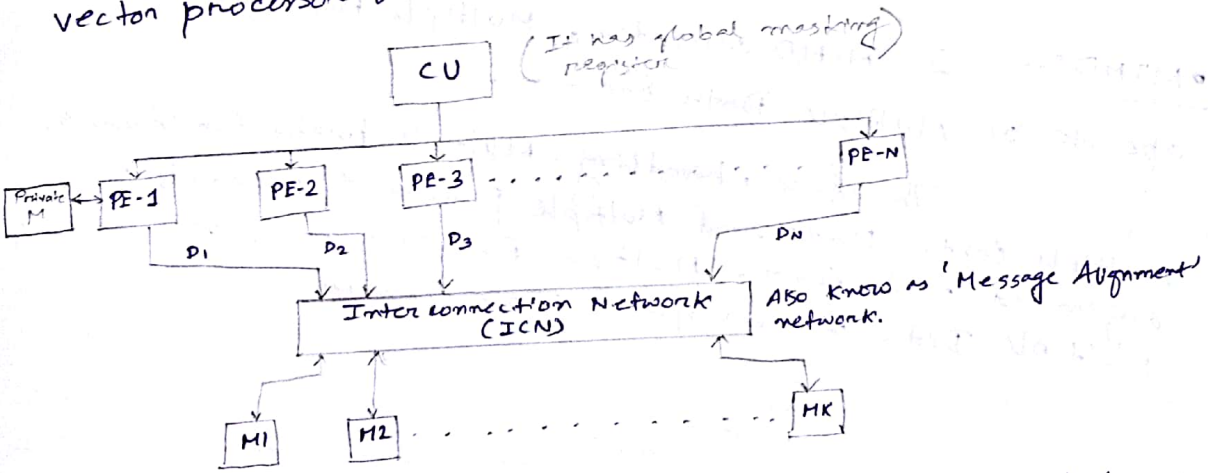
- **SISD:** One processing unit working on one instruction which operates on one data.




Example: CDC - 6600
CDC - 7600

- **SIMD:** In SIMD computers, one instruction is executed upon multiple Data items using multiple 'Processing elements' controlled by one single control unit.

This kind of computer is also known as array and vector processor & hence its popular in Image Processing.




Since multiple processing elements don't have control units hence they are called processing elements not a complete processor.

A processing element inside SIMD can have private cache & Main memory of its own.

Interconnection network also let sharing data between two processing elements.

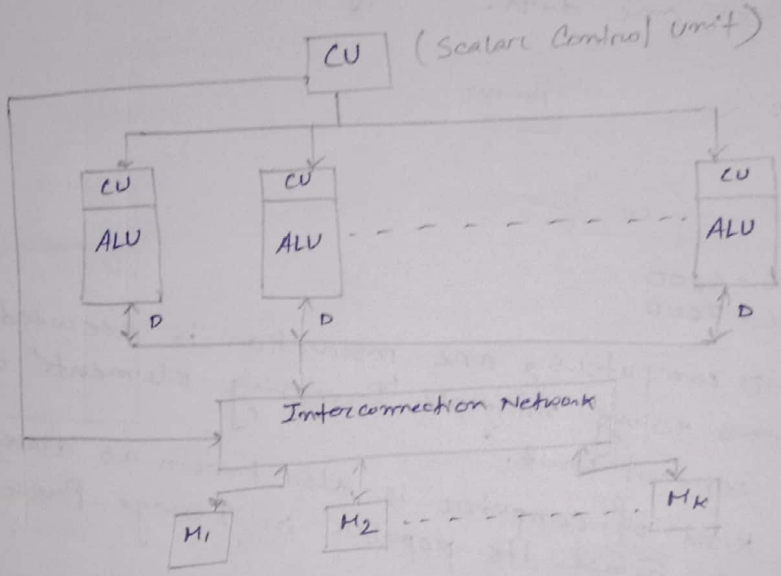
Example: ILLIAC - 4

• **MISD:** In MISD, Multiple Processing units are present instead of processing elements since multiple processing elements have their own control units.

Each Processing Units handle one instruction stream and can process only a single Data Stream at a time.

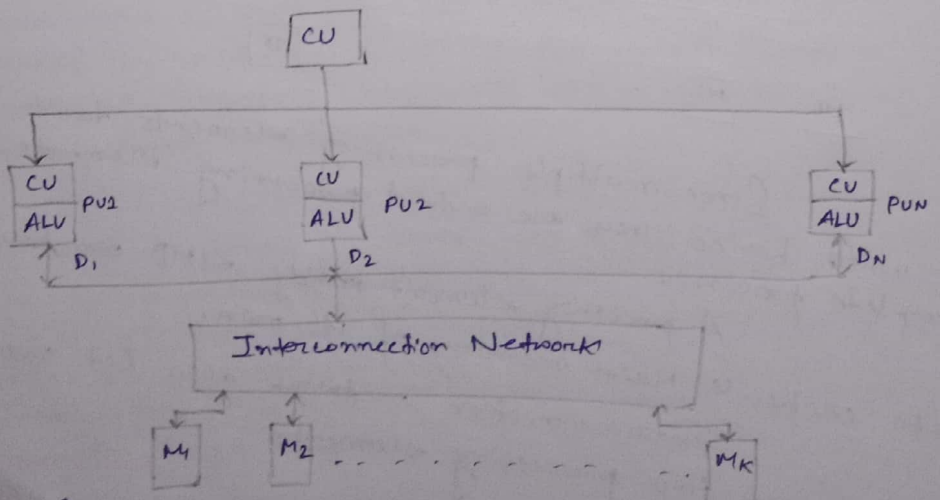
Various processing units can perform different types of instructions under different data. (Eg:- If PE1 is performing Add, PE2 might perform SUB etc.)

Thus, as the name implies, it processes Single datastream using multiple processing units.




• **MIMD:-** In MIMD computer, Multiple Instruction Streams operate on Multiple Data Streams.

Therefore, handling Multiple Instruction Streams, Multiple Control Units and Multiple Processing Elements are organized such that, Multiple Processing units are handling Multiple Data stream from Main Memory.




This is the most generalized approach of all other classifications.

## SIMD Architecture :-

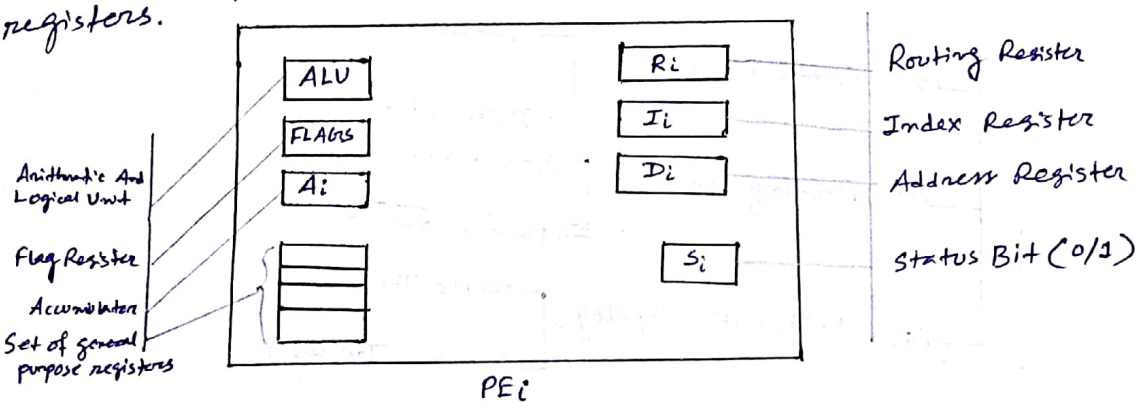
The two most important components of SIMD computers are

i) Processing Elements (PE)

ii) Interconnection Network (ICN)

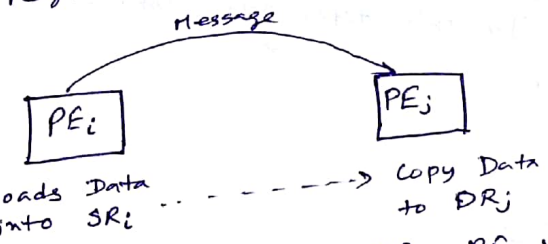
### • Processing Element :-

A SIMD computer can have n numbers of PE's ($PE_0, PE_1, PE_2 \ldots PE_K \ldots PE_n$). Each of which contains its own separate ALU, Flags, General purpose & special purpose registers.




**Ri :** Ri is called Routing Register that have Source & Destination register. Routing Register is useful to transmit data between different PEs.

Ex:




Hence, By SRi & DRj, PE works Both as transmitter & reciever.

**Ii :** Ii is called Index Register which holds address of local memory generated by control unit, during any operation.

• Hence, it acts as a memory Address register.

**Di :** Di is called address register. Each PE has logn size of address for n pes. Di holds address of its own & respective PEs.

• If a message is passed from $PE_i \longrightarrow PE_j$, it may first go to $PE_K$ then by seeing Di, PEi redirects data to $PE_j$.

**Si :** Si is special bit used to determine whether the corresponding Pi is active or not in a given instance of time.

## Global Masking Register:

In SIMD computers, the control unit contains a register called global masking register which indicates that, at any given instance of time which $P_i$ should be active & others to be inactive.

Ex: $P_1$ $P_2$ ------ $P_7$ [∵ $P_1$ & $P_2$ are active]
1 1 0

## Interconnection Network:

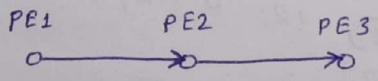
- (1) Operation Mode
  - Synchronous
  - Asynchronous
- (2) Control Strategy
  - Centralized
  - Distributed
- (3) Routing Strategy
  - Circuit switch
  - Packet Switch
- (4) Network Topology
  - Static Topology
  - Dynamic Topology

## Static Topology:

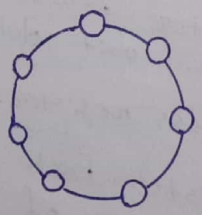
In this topology, connection is predetermined for transmission.

Example:

- Linear:

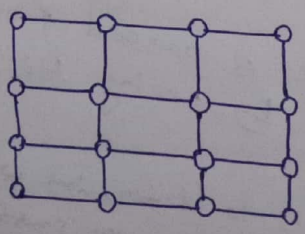



- 2D:



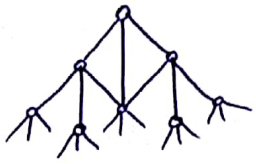
Ring

- Near Neighbour Mesh:



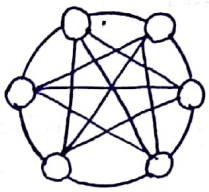
## Systolic Array:

Here, Any given node is connected to 3 other nodes.

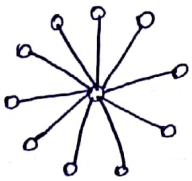


• 3D Network: If graph representation of a network is planner then its 2D else 3D.

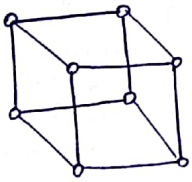
## Chordal Ring: (3D)



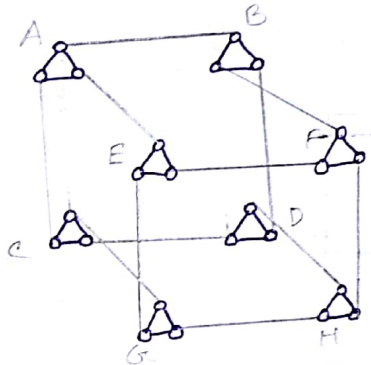
## Star:



## 3 Qube network:



## 3-cycle cube network:


This is also a recirculating network.

If A wants to send data to G, then the data must reenter in the network to C before finally reaching to G.

This can also be implemented Dynamically.

---

## Dynamic Topology:-

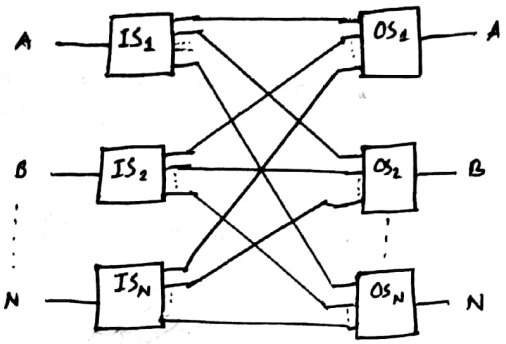
In this topology, connection is changed dynamically for each transmission.

- Dynamic Topology
  - Single stage Dynamic Network
  - Multistage Dynamic Network
    - Type Blocking
    - Non-Type-Blocking

Two main aspects of single stage network

- Input Selectors (1 to N Demux)
- Output Selectors (M x 1 MUX)

## Single Stage Recirculating network




## Switch in Multistage Recirculating Network

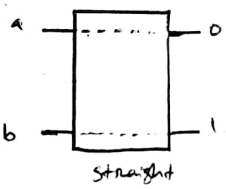
Classification of switches

- Straight
- Exchange
- Lower Broadcast
- Upper Broadcast

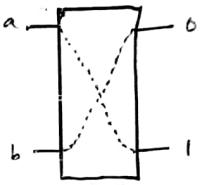
- Two levels of switches
  - Level 1 (MUX)
  - Level 2 (DEMUX)

**Blocking & Non-Blocking:**

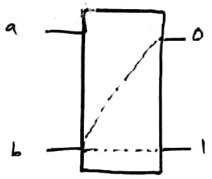
There may or may not be buffer or register between switches. If there is Buffer, the multistage networking is non blocking otherwise its called blocking.



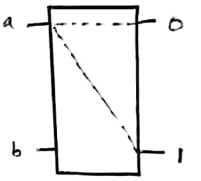
Straight



Exchange

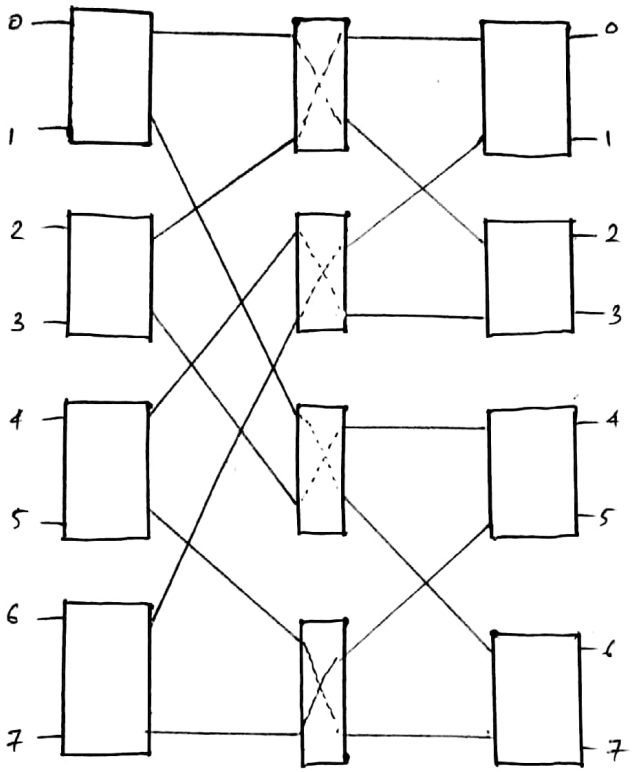


Lower Broadcast



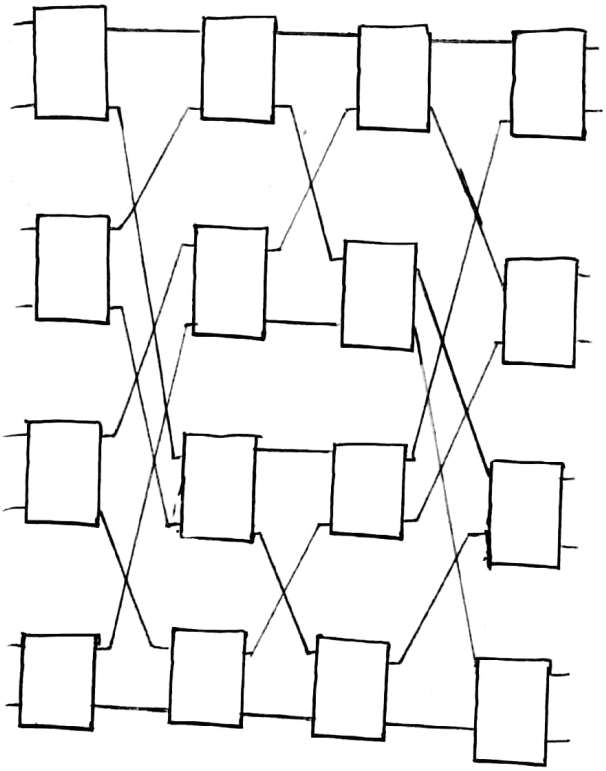
Upper Broadcast

## 8×8 Baseline Network:- (8 I/P, 8/OP)

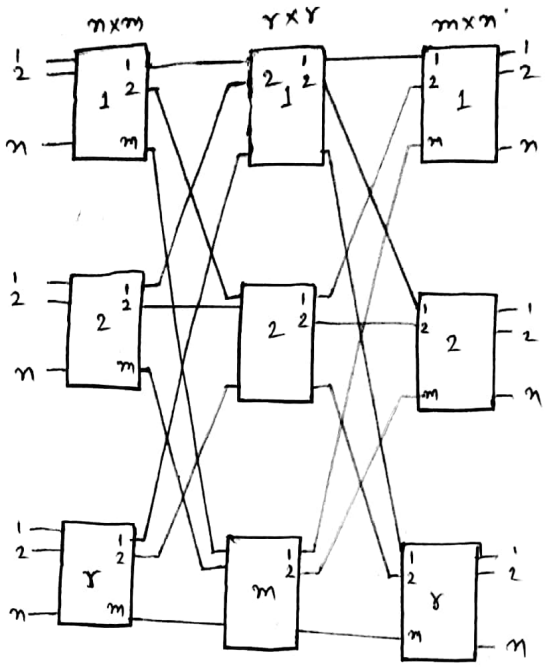



- Unlike recirculating network, one circulation of data is enough.

## 8×8 Benes Network:



# Clos Network:




---

## Single stage recirculating network
## OR
## Mesh Connected Illiac Network

For $n$ number of PE's,

$n = 2^r$ [$r$ must be a perfect square]

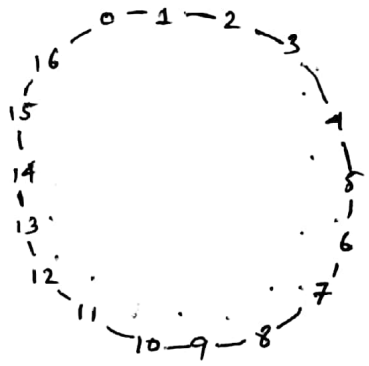
- Routing address formula

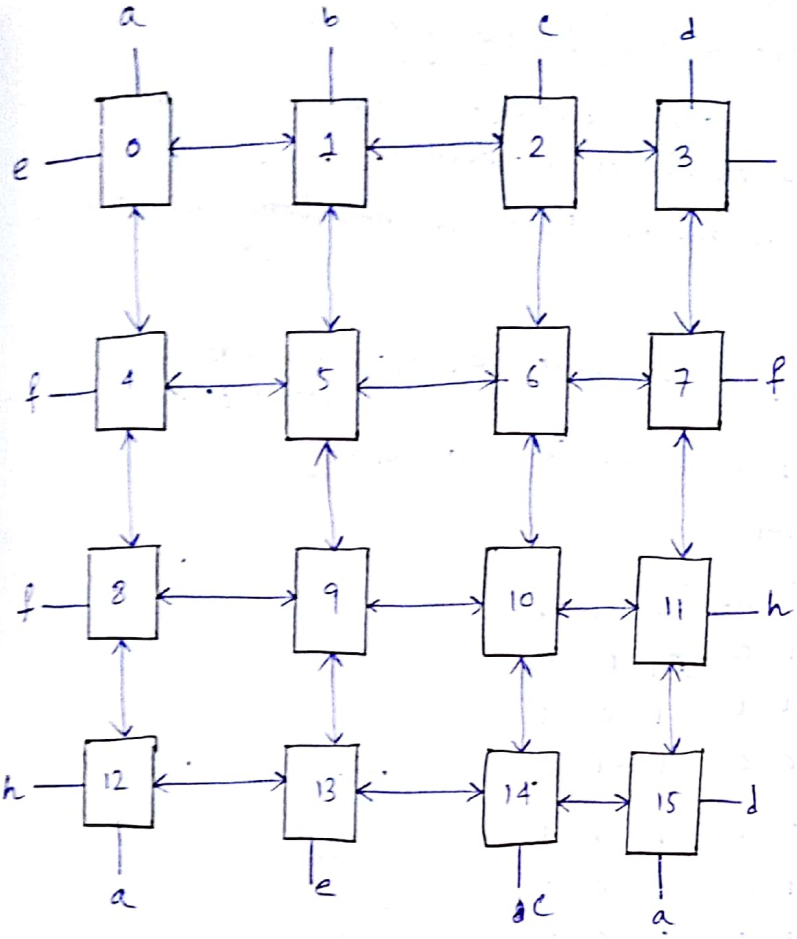
$$R_{+1}(i) = (i+1) \bmod n$$
$$R_{-1}(i) = (i-1) \bmod n$$
$$R_{+r}(i) = (i+r) \bmod n$$
$$R_{-r}(i) = (i-r) \bmod n$$

For $n = 16$ PE, $r = 4$ [$\because 2^4 = 16$]

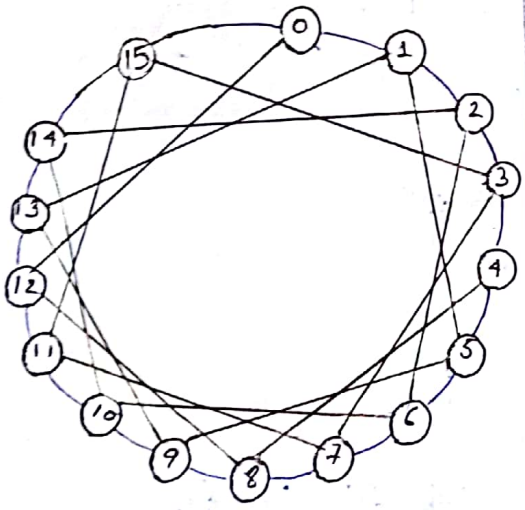
| i | +1 | -1 | +4 | -4 |
|---|---|---|---|---|
| 0 | 1 | 15 | 4 | 12 |
| 1 | 2 | 0 | 5 | 13 |
| 2 | 3 | 1 | 6 | 14 |
| 3 | 4 | 2 | 7 | 15 |
| 4 | 5 | 3 | 8 | 0 |
| 5 | 6 | 4 | 9 | 1 |
| 6 | 7 | 5 | 10 | 2 |
| 7 | 8 | 6 | 11 | 3 |
| 8 | 9 | 7 | 12 | 4 |
| 9 | 10 | 8 | 13 | 5 |
| 10 | 11 | 9 | 14 | 6 |
| 11 | 12 | 10 | 15 | 7 |
| 12 | 13 | 11 | 0 | 8 |
| 13 | 14 | 12 | 1 | 9 |
| 14 | 15 | 13 | 2 | 10 |
| 15 | 0 | 14 | 3 | 11 |

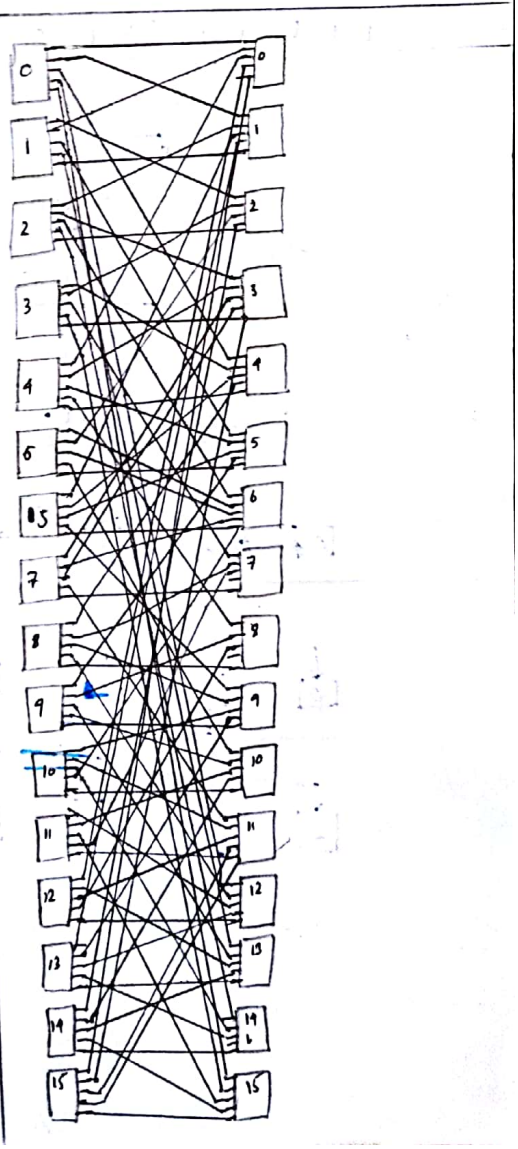
For 16 PE's






The above can also be represented using circular ring



## Cube interconnection Network:-

In a cube interconnection network, maximum $\sqrt{N} - 1$ recirculation is needed.

For n no. of PE's it has $\log_2 n$ address bits.

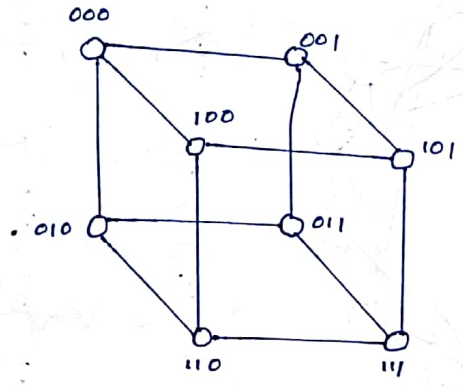
### Routing function:-

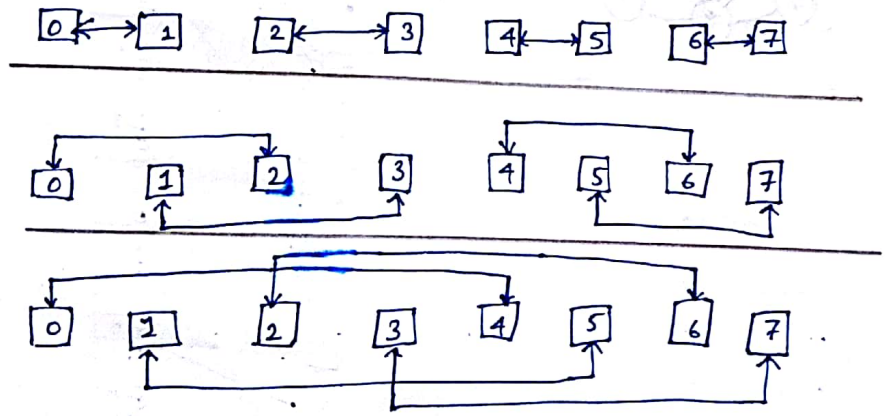
For $C(a_{n-1}\ a_{n-2} \ldots\ldots\ldots a_2\ a_1\ a_0)$ address of any given pe,

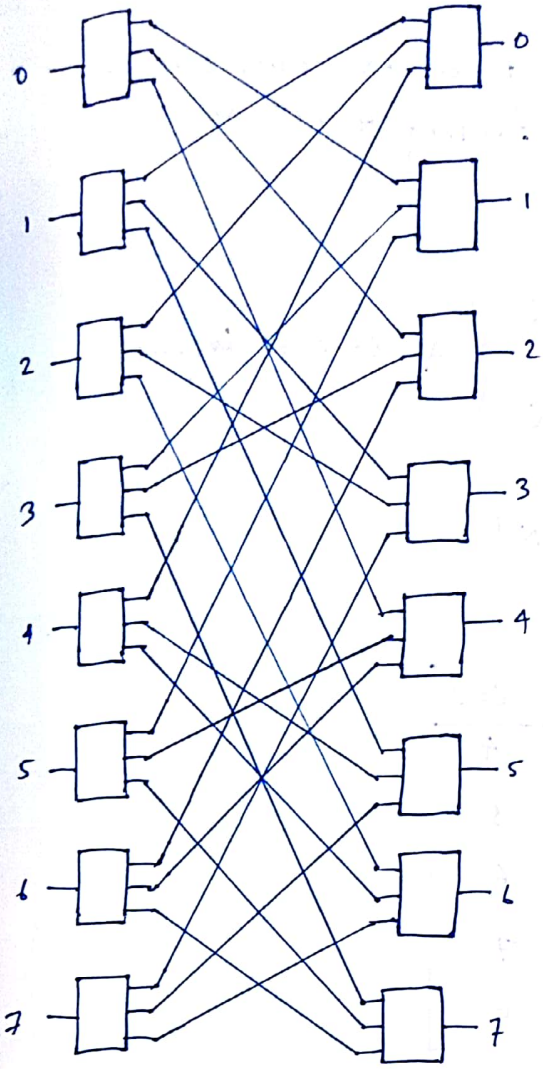
$= (a_{n-1}\ a_{n-2} \ldots\ldots \bar{a}_i \ldots a_2\ a_1\ a_0)$

$\forall\ i = 0 \rightarrow (n-1)$

| | $a_2\ a_1\ \bar{a}_0$ | $a_2\ \bar{a}_1\ a_0$ | $\bar{a}_2\ a_1\ a_0$ |
|---|---|---|---|
| 000 | 001 | 010 | 100 |
| 001 | 000 | 011 | 101 |
| 010 | 011 | 000 | 110 |
| 011 | 010 | 001 | 111 |
| 100 | 101 | 110 | 000 |
| 101 | 100 | 111 | 001 |
| 110 | 111 | 100 | 010 |
| 111 | 110 | 101 | 011 |

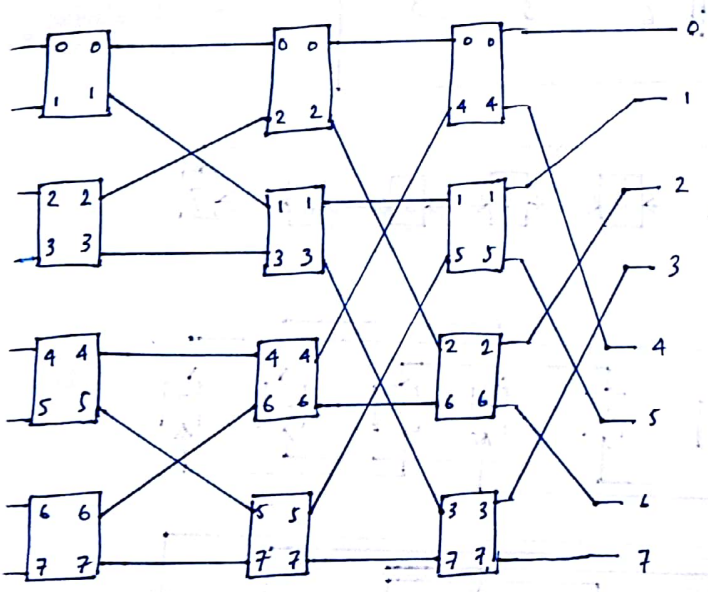





## Multistage Network:-

$n = 2^{\pi}$ ($\pi$ = no. of stages)




| Stage | i=0 | i=1 | i=2 |
|---|---|---|---|
| Distance | $2^0=1$ | $2^1=2$ | $2^2=4$ |

# Barrel Shifter:-

This network is know as Plus-Minus $2^i$ network and based on following Routing functions

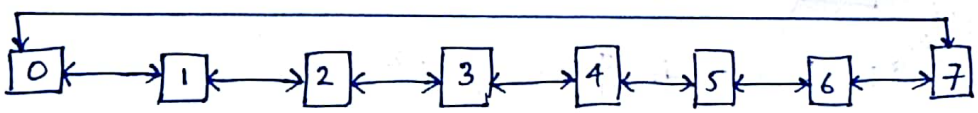
Routing functions—

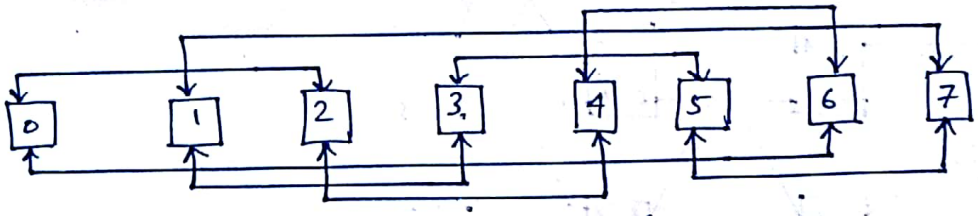
$$\left.\begin{aligned} B_{+i}(j) &= (j + 2^i) \bmod N \\ B_{-i}(j) &= (j - 2^i) \bmod N \end{aligned}\right\} \forall\ i = 0 \text{ to } \left(\log_2 n - 1\right)$$

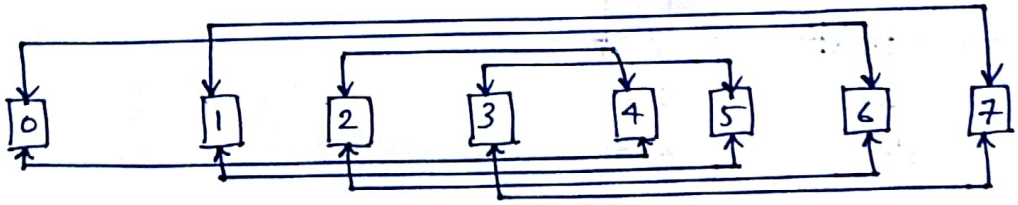
Eg: If $N = 8$ then, $2^3 = 8$ $[2^n = N]$

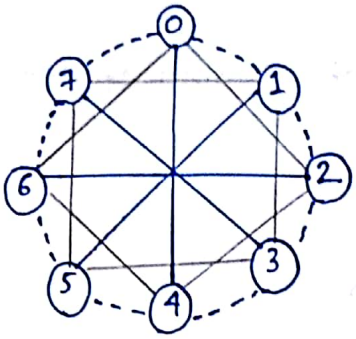
$\therefore\ i = 0, 1, 2$

| | $B_{+i}$ | | | $B_{-i}$ | | |
|---|---|---|---|---|---|---|
| | $i=0$ | $i=1$ | $i=2$ | $i=0$ | $i=1$ | $i=2$ |
| 0 | 1 | 2 | 4 | 7 | 6 | 4 |
| 1 | 2 | 3 | 5 | 0 | 7 | 5 |
| 2 | 3 | 4 | 6 | 1 | 0 | 6 |
| 3 | 4 | 5 | 7 | 2 | 1 | 7 |
| 4 | 5 | 6 | 0 | 3 | 2 | 6 |
| 5 | 6 | 7 | 1 | 4 | 3 | 1 |
| 6 | 7 | 0 | 2 | 5 | 4 | 2 |
| 7 | 0 | 1 | 3 | 6 | 5 | 3 |






----- $2^0$
—— $2^1$
—— $2^2$

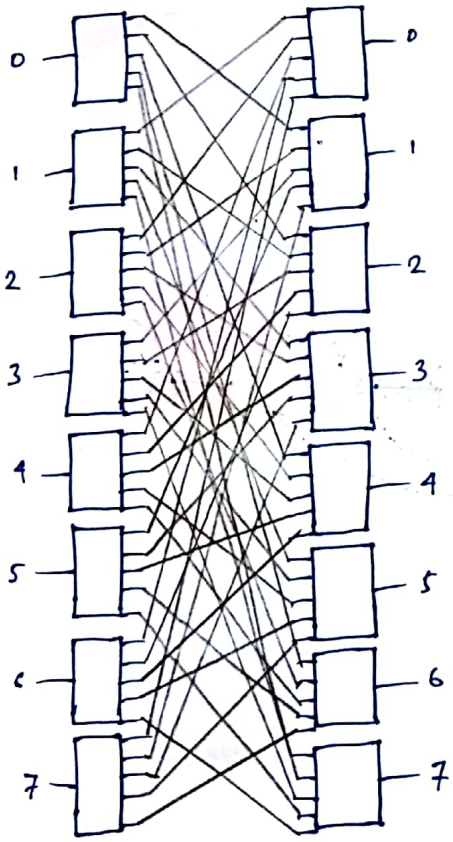
The illiac routing functions are a subset of the barrel shifting function.

Instead of having just 4 nearest neighbours as in illiac network, each PE in a barrel shifter is directly connected to $(2n-1)$ PE's. Therefore connectivity in barrel shifter network is increased compared to the Illiac network.

The barrel shifter can be implemeted as either a single stage recirculating network or as a multistage network. The multistage barrel shifter is know as "A DATA MANIPULATOR

* To implement mutistage version of this network, For n PE's we need 1 to $(2\log_2 n - 1)$ demux.

For $n=8$, we need 1 to 5 demux.



# Shuffle Exchange Network:-

Shuffle exchange network consists of two routing functions

- shuffle function
- Exchange function

Shuffle Routing function:-

$$S(a_{n-1}\ a_{n-2} \ldots\ldots a_2\ a_1\ a_0) = (a_{n-2}\ a_{n-3} \ldots\ldots a_1\ a_0\ a_{n-1})$$

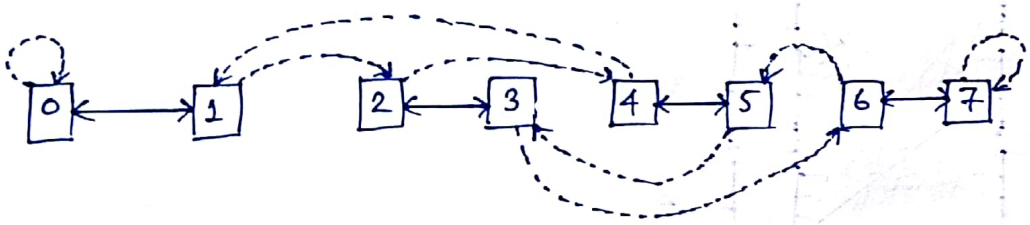
[Circular Left shift operation

Exchang Routing function:-

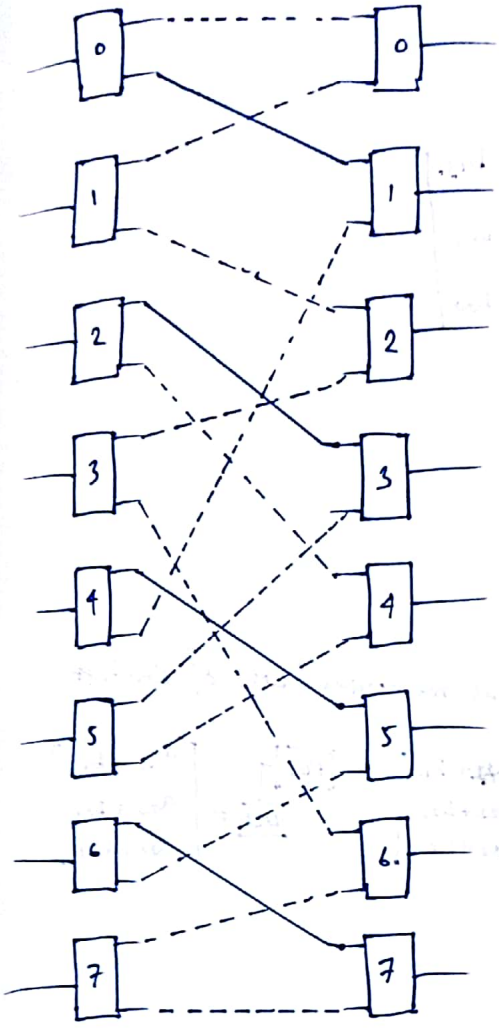
$$E(a_{n-1}\ a_{n-2} \ldots\ldots a_2\ a_1\ a_0) = (a_{n-2}\ a_{n-3} \ldots\ldots a_2\ a_1\ \bar{a}_0)$$

[Complement of LSB

| | E(i) | S(i) |
|---|---|---|
| 000 | 001 | 000 |
| 001 | 000 | 010 |
| 010 | 011 | 100 |
| 011 | 010 | 110 |
| 100 | 101 | 001 |
| 101 | 100 | 011 |
| 110 | 111 | 101 |
| 111 | 110 | 111 |

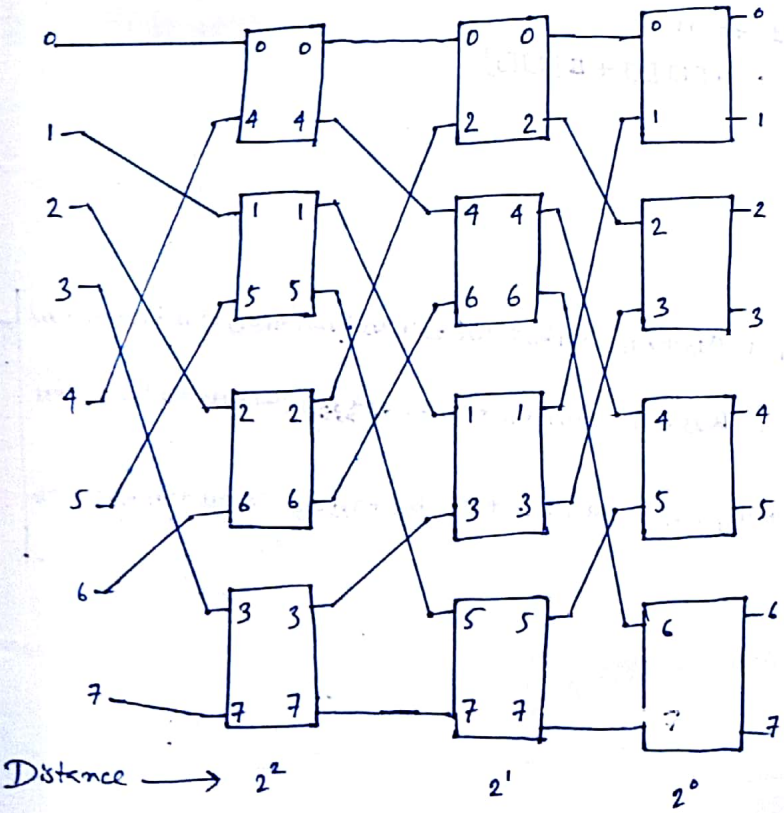


## Single stage implementation:-




OMEGA NETWORK

## Multistage implementation of this network:-




END OF SIMD

# Parallel Programming:-

## Matrix Addition:-

$$A \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} + B \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

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

Parallel operations happen inside local memories [LM] of each PE's

$PE_1$ — $LM1 = \begin{bmatrix} a_{11}+b_{11} \\ a_{21}+b_{21} \\ a_{31}+b_{31} \end{bmatrix}$

$PE_2$ — $LM2 = \begin{bmatrix} a_{11}+b_{12} \\ a_{22}+b_{22} \\ a_{32}+b_{32} \end{bmatrix}$

$PE_3$ — $LM3 = \begin{bmatrix} a_{13}+b_{13} \\ a_{23}+b_{23} \\ a_{33}+b_{33} \end{bmatrix}$

## Program:- (Addition)

```
Begin
    for i = 1 to n
        par for j = 1 to n
            c[i][j] = a[i][j] + b[i][j]
        end par for
    End for
End
```

For multiplication,

$$A * B = \begin{bmatrix} (a_{11}b_{11} + a_{12}b_{21} + a_{13}+b_{31}) & (a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}) & (a_{11}b_{13} + a_{12}b_{23} + a_{33}b_{33}) \\ (a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31}) & (a_{21}b_{12} + a_{22}b_{22} + a_{33}b_{32}) & (a_{21}b_{13} + a_{22}b_{23} + a_{33}b_{33}) \\ (a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31}) & (a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}) & (a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33}) \end{bmatrix}$$

program:- (Multiplication)

```
for i=1 to N
    par for K=1 to N
        C[i][K]=0
    End par for
    for j=1 to N
        par for K=1 to N
            C[i][K] = C[i][K] + a[i][j] * b[j][K]
        End par for
    End
End
```

## MIMD Computers or Multi Computers:-

Two types:-

- Tightly Coupled System - TCS
- Losely Coupled System - LCS (Also known as Distributed system)

### Diagram of Losely Coupled System:-

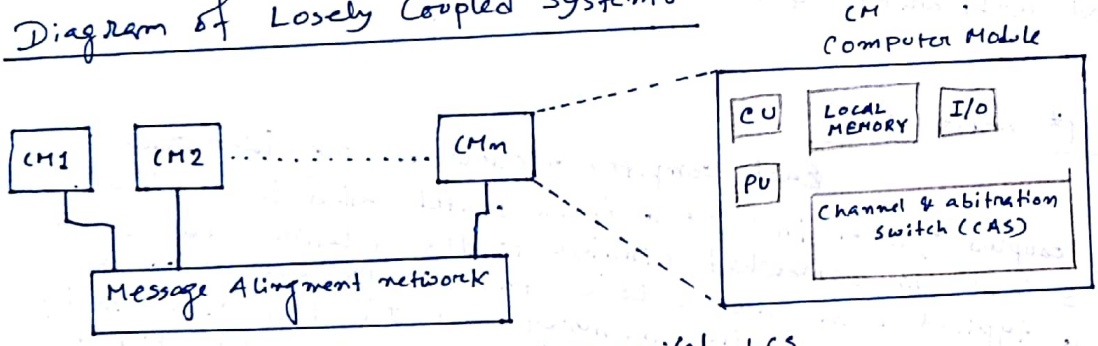



Fig: Non higrarcical LCS

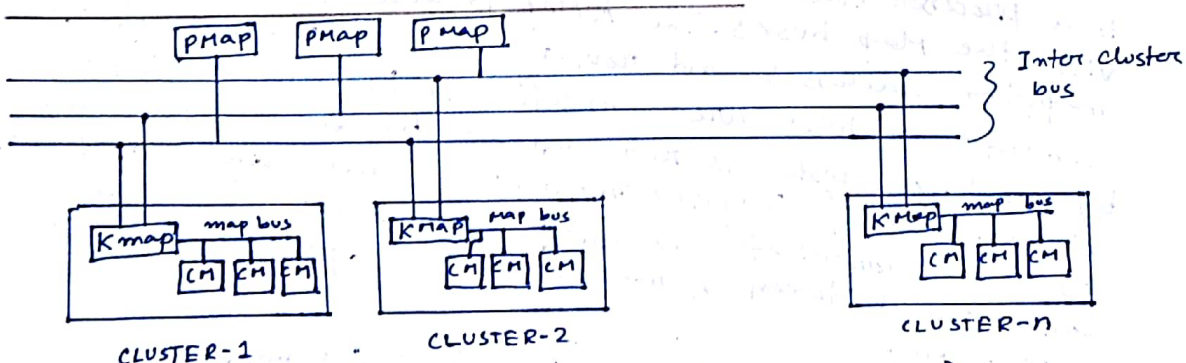



Fig: higherarcical LCS ($CM^*$ arcitecture)

- cluster contains Kmap, computer modules & a map bus all together
- K-map is responsible for communication b/w CMi's & also b/w different clusters.
- To control administration of inter cluster bus, the data is loaded to the queue of PMap. The message will be ~~loaded~~ transfered according to the availibility of the clusters.

Two different sets of architectural model for a multiprocessor has been described

i) Tightly Coupled multiprocessor

ii) Loosely Coupled multiprocessor

i) TCS :-

Tightly coupled Multiprocessors communicate through a shared main memory hence the rate at which the data communicate from one processor to another is limited by the bandwidth of the memory. The degree of conflicts in a tightly coupled system is relatively high.

ii) LCS :

In a Losely coupled Multiprocessors, each processor has a set of I/O devices and a large local memory from where it accesses most of the instructions and each data processes which executes on different computer module communicate by exchanging messages through a message transfer system. The degree of coupling in such system is very low. Each computer module contains a channel and arbitrer switch (CAS) The CAS is needed for buffering block transfer of data and arbitaring requests to the message transfer system.

* CM* Architecture :-

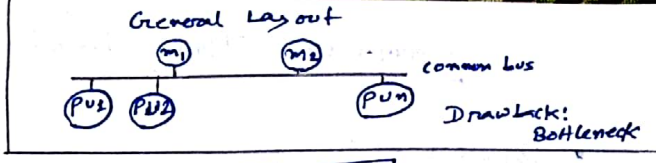
Losely coupled System includes a local switch called the S-local. The S-local is somewhat similar to the channel arbiter switch of a typical LCS. The S-Local intercepts and routes the processor's request to the memory and I/O devices. The K-Map is a processor that connects a number of computer modules via the Map buses. The KMap is also responsible from mapping addresses and routing data between S-locals. The computer modules are connected in hierarcical clusters by buses. A cluster is regarded as the lowest level and is made up of computer modules a kmap and a map bus. Cluster communicate via intercluster buses which are connected between K-maps. The three processors in K-Map are —

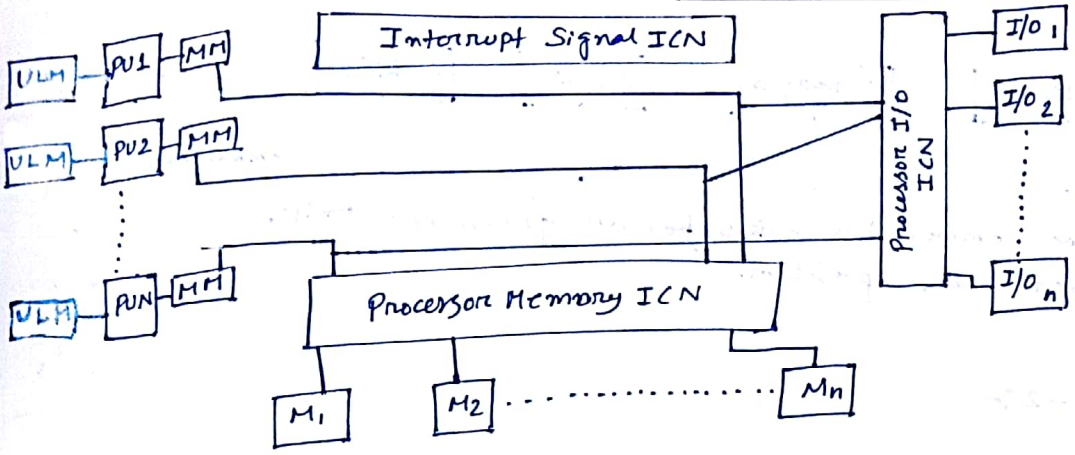
i) The K bus ii) The line iii) The P-map.

The K-bus is the bus controller which arbitrates requests to the map bus. The line manages communication between the Kmap & other Kmaps.

The P-map is the mapping processor which responds to requests from the K-bus and line.

## Tightly Coupled System:-



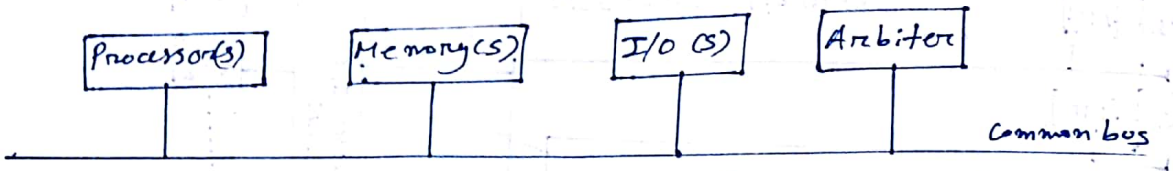




Tightly coupled multiprocessors communicate through shared main memory. The processing units, memory modules and I/O devices are connected through a set of three interconnection networks namely the processor Memory Interconnection network (PMIN), the I/O processor ICN and the Interrupt signal ICN.

The processor memory ICN is a switch which can connect every processor to every memory module to avoid excessive memory conflicts every processor is associated with a reserved storage called unmapped local memory. There is a module attached to each processor that directs memory reference to the ULM or shared memory. This module is called the memory map. The Interrupt Signal ICN permits each processor to direct an interrupt to any other processor. Synchronization between processor is facilitated by the use of such an interprocessor network. The I/O processor ICN permits the processor to communicate with a channel which is connected to peripheral devices.

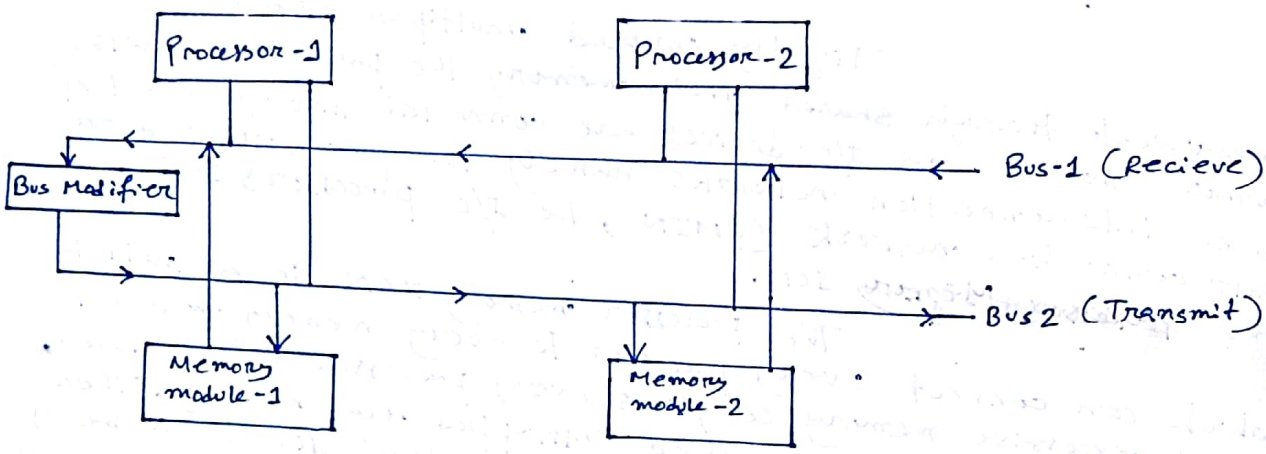
# Common Time Shared bus ICN:-

- Topology-1 :-




* Since one common bus exists for all functional units, it has bottle neck problem.

- Topology-2:-

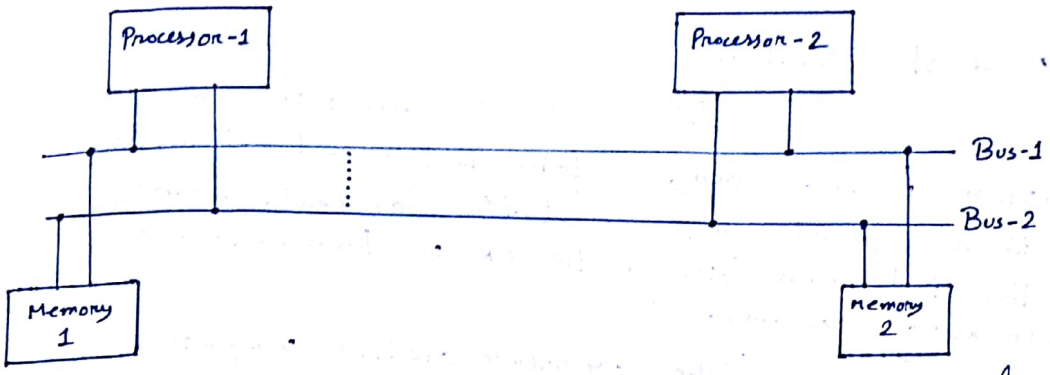



* Processor-1 load data to Bus-1 it will be moved to Bus Modifier & data will move to Memory module-1.

* Since buses are unidirectional, direct data transfer cant happen; hence bus modifier is used.

* In this case also, bus modifier is overloaded since it is being used by all modules. The problem of bottleneck still exist.

• Topology 3:- Multiple no. of bidirection buses are introduced.




If no. of buses are increased, performance can be increased.

* Multiport:- At any given instance of time it is possible to access multiple words.

In uniport, if $P_1$ & $P_2$ wants to load data into memory using bus 1 & bus 2; even if bus collision wouldn't occur, the data can't be written since it is uniport. (Bottleneck in memory)

It can be possible if the memory is multiport for parallel access.

* The bus can be increased upto a certain extent otherwise the control unit will be overloaded to control a large no. of buses.

So, the optimal solution is,

If there are m no. of memory modules then each processor will contain m no. of buses coming out of it.

Timeshared or common bus architecture:- The simplest interconnection system for multiprocessor is a common communication path connecting all of the functional units. The common path is often called a time shared or common bus. This organization is the least complex and the easiest to reconfigure. Since the bus is a shared resource, a mechanism must be provided to resolve contention. Conflict resolution methods such as independent requests, polling, daisy chaining may be used. Although the signal bus organization is quite reliable and relatively inexpensive, it does introduce a single critical component in the system that can cause complete system failure as a result of a malfunction in any of the bus interface circuits. Moreover, system expansion by adding more processor or memory increases the bus connection which degrades system throughput and arbitration logic.

# Mutistage mind ICN's—

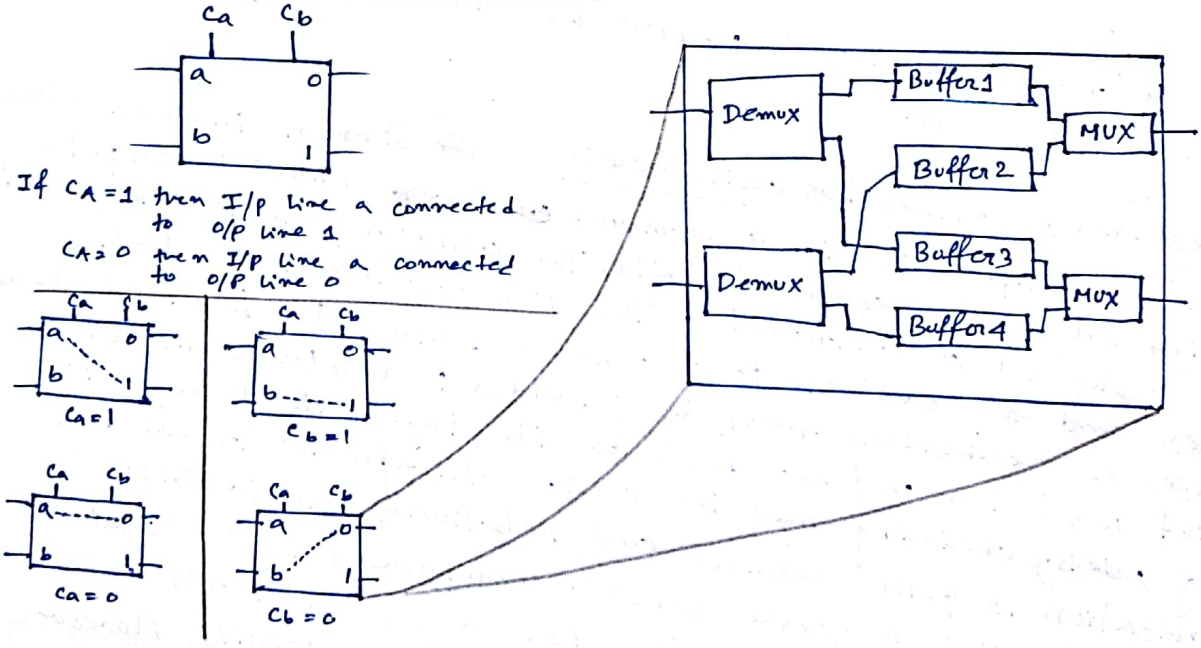
- Crossbar Switch:—

If a no. of buses in time shared bus system is increased, a point is reached at which there is a separate path available for each memory unit. The interconnection network is then called a non blocking crossbar.

The crossbar switch complete connectivity with respect to the memory module because there is a seperate bus associated with each memory module. The conflict occurs in a crossbar switch if two or more concurrent requests are made to the same destination device.

The switch consists of arbitration and multiplexer modules. Each processor generates a memory module request signal to the arbitation unit which selects the processor with highest priority. The arbitration module returns an acknowlegement signal to the selected processor. The multiplexer modules multiplexes data addresses of words within the module and control signals from the processor to the memory module.

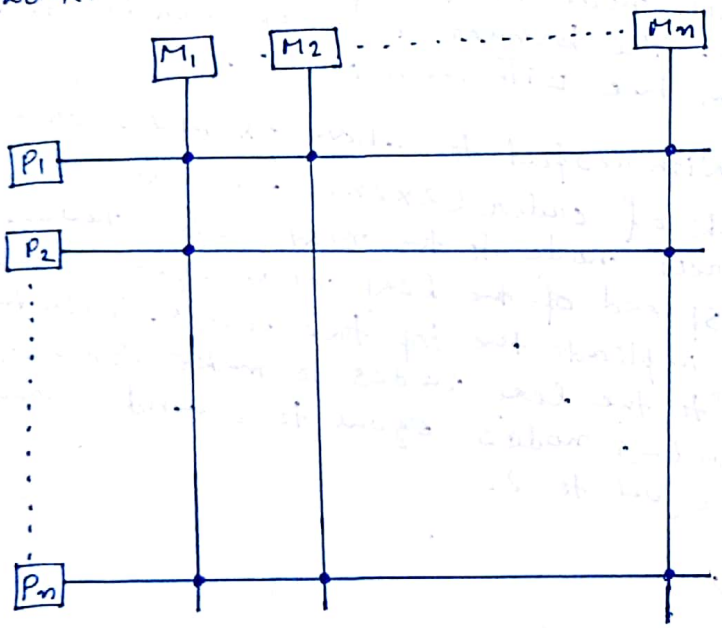
## 2x2 Crossbar Switch:—



Inorder to design multistage networks, we need to construct 2×2 crossbar switches. The 2×2 switch has the capability of connecting the input A to either output-0 or o/p-1 depending on the value of some control bit $C_A$ of the input A. If $C_A = 0$ the i/p is connected to the upper output (0) and if $C_A = 1$ the connection is made to the lower output (1). Terminal B ~~require the same~~ of the switch behaves similarly. with the control bit $C_B$.

If both A and B require the same o/p terminal, then the data will be transmitted first based on the priority based bus arbitration logic.

To improve the performance, buffers can be inserted within the switch. This kind of switch is very effective for packet switching when used in a multistage network.




Crossbar memory organization.

## Banyan Network:-

A Banyan network can be roughly described as partially ordered graph divided into distinct levels. Nodes with no arcs fanning out of them are called base nodes and those with no arcs fanning into them are called apex nodes. The fan out 'f' of a node is the no. of arcs fanning out from the node. The Spread 's' of a node is the no. of arcs fanning into it.

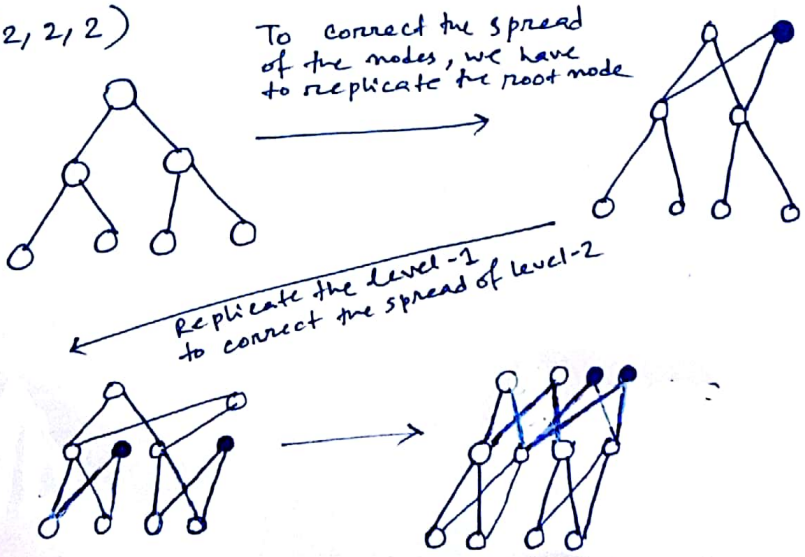
A (f, s, l) banyan network can thus be described as a partially ordered graph with 'l' level in which there is exactly one path from every base to every apex nodes. The fan out of each non base node is 'f' and the spread of each non apex node is 's'. Each node of the graph is an sxf crossbar switch. The banyan network can be derived from a uniform tree with fan out 'f'.

With respect to above example, for a banyan network of order (2x2x2), we start by replicating the root node to the next level nodes. To make the spread of the leaf node equal to ~~the leaf nodes~~ 2 replicate the top two levels. Join the level 1 nodes to the leaf nodes to make the fan out of the level-1 nodes equal to 2 and spread of leaf nodes equal to 2.

Example: B (f, s, l)
- f → fan out
- s → spread
- l → levels

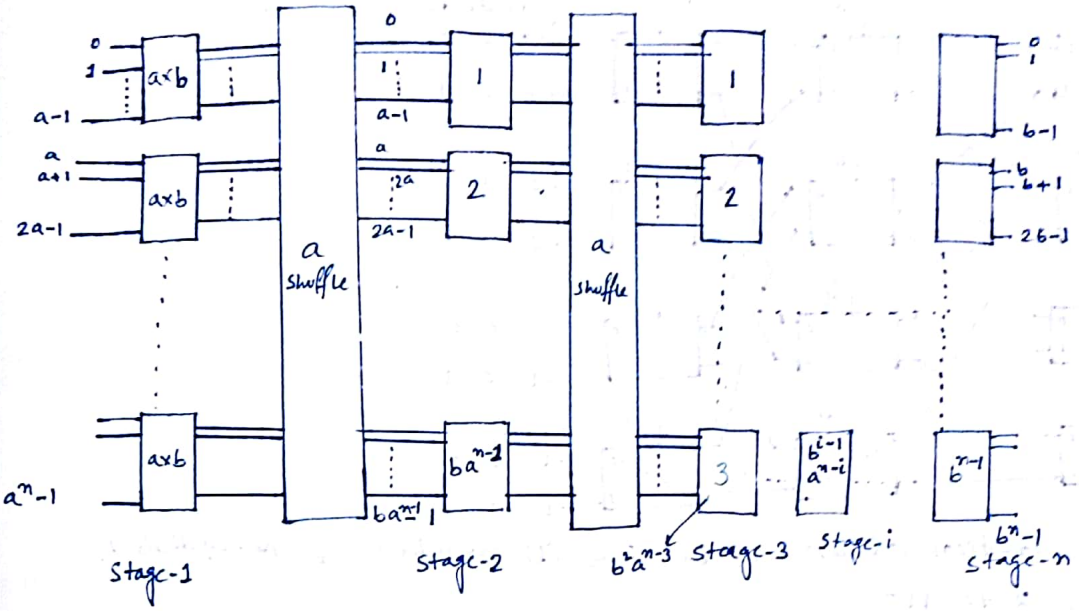
* Switches at the topmost level is called apex node
  Switches at the buttom level is called leaf node

Eg: B(2, 2, 2)



## $a^n \times b^n$ (delta network) :-

It uses axb crossbar switches and a no. of control signals
n is the no. of stages used.

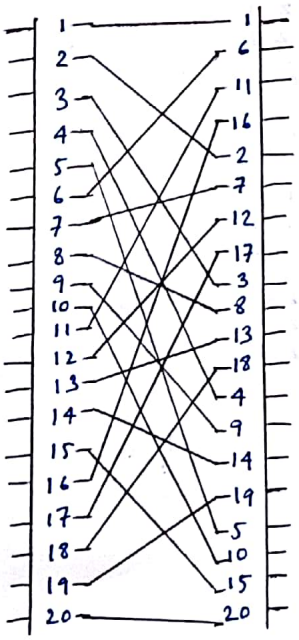



- No. of I/p's = $a^n$
  for each, it will be $\frac{a^n}{a} = a^{n-1}$
- No. of O/p's = $b \times a^{n-1}$ since each box has b o/p's
- stage 2 will have $(ba^{n-1})$ inputs
- Total switching in stage-2 needed having a input is $\frac{ba^{n-1}}{a} = ba^{n-2}$
- Total stage ~~###~~ o/p's in stage-2 is $b^2 a^{n-2}$ with b-output each.
- Last stage will have $b^n$ o/p.

### Q - Shuffle

In a shuffle, no of piles = a
Total cards = $ba^{n-1}$ (Total no. of o/p's)
∴ Each pile has $\frac{ba^{n-1}}{a} = ba^{n-2}$

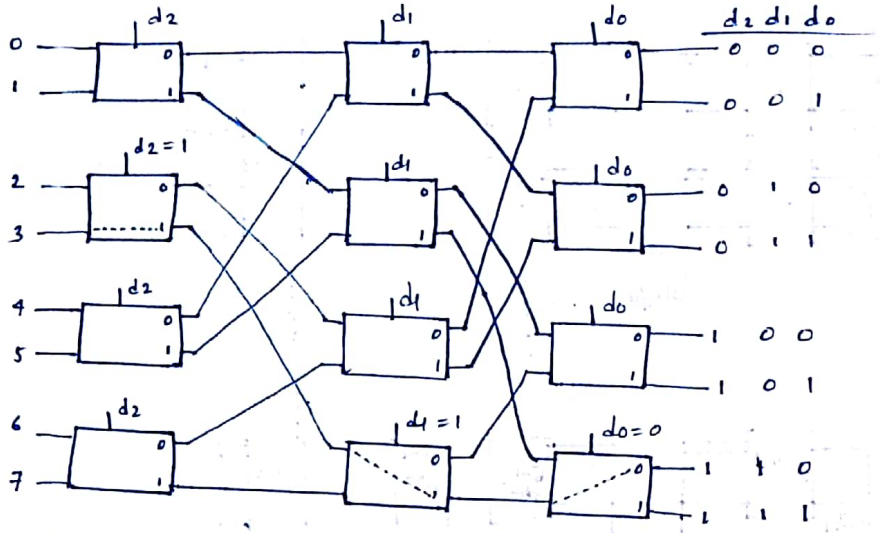
Eg: Tota no. of cards = 20
a = 4 (4 piles)



## $2^3 \times 2^3$ delta network:-

$a = 2 \quad b = 2 \quad n = 3$

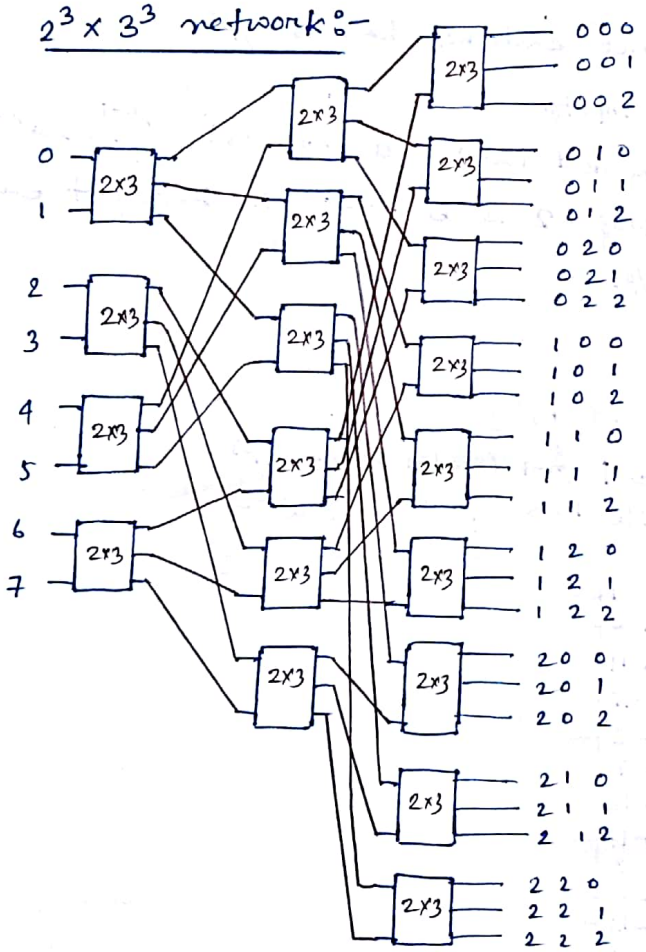
We have to use 2-shuffle




The destination address will be base-'b' number of no. of digits = n

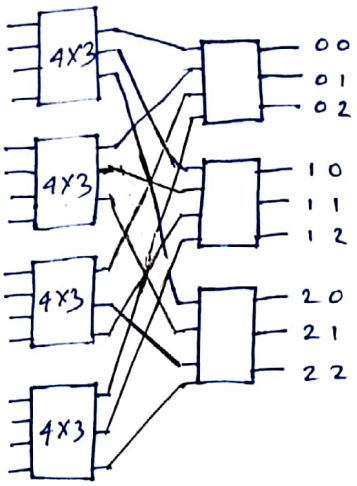
eg: 3 to 110

## $2^3 \times 3^3$ network:-



## $4^2 \times 3^2$ network:-

$a = 4 \quad b = 3 \quad n = 2$



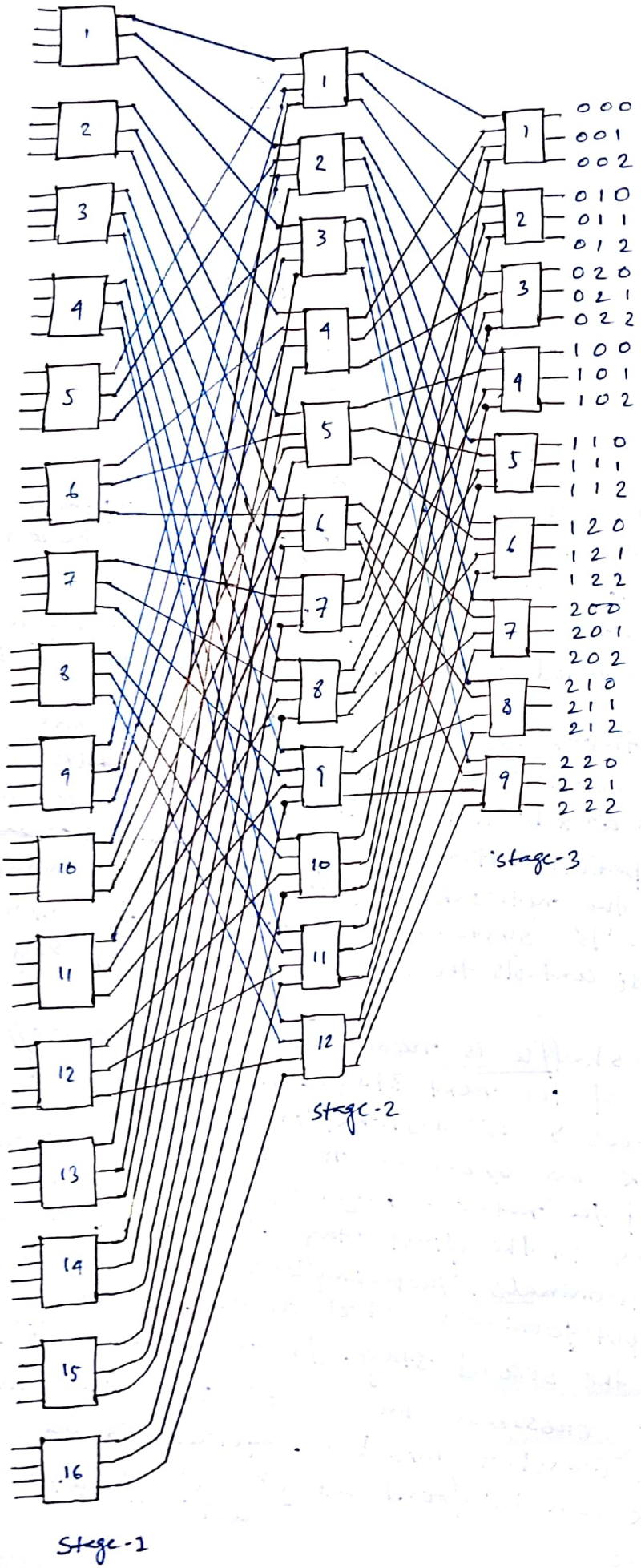

Let an axb crossbar module have the capibility to connect any of its 'a' input to any of its 'b' outputs where the outputs are labelled $0, 1 \ldots\ldots (b-1)$.

An input terminal is connected to the output labelled 'd' if the control digits supplied by the input is 'd' where 'd' is a 'b' digit.

A delta network is defined as $a^n \times b^n$ switching network with n-stages consisting of axb crossbar modules. In order to construct an $a^n \times b^n$ delta network, we have to use 'a'-shuffle as the link pattern between every two ~~delta network~~ consecutive stages ~~as~~ of the network. If the destination ~~network~~ is expressed in a base-'b' system as $(d_{n-1}, d_{n-2}, \ldots\ldots d_1, d_0)$ then the base-b digit $d_i$ controls the crossbar modules of stage $(n-i)$.

The 'a'-shuffle is needed to convert the O/P's of a stage to the I/P of the next stage. An $a^n \times b^n$ delta network has $a^n$ sources & $b^n$ destinations. Numbering the stages of the network as $1, 2 \ldots\ldots n$ through 'n' starting at the source side of the network requires that there be $a^{n-1}$ crossbar modules in the first stage. The first stage then has $b a^{n-1}$ O/p terminals. This implies that stage-2 must have $b a^{n-1}$ input terminals which requires $b a^{n-2}$ crossbar modules in the second stage. In general, the 'i'-th stage has $a^{n-i} b^{i-1}$ crossbar modules of size axb thus the total no. of axb crossbar modules required in an $a^n \times b^n$ delta network can be found as $\frac{a^n - b^n}{a-b}$ (for $a \neq b$) and $n b^{n-1}$ (for $a = b$)

# $4^3 \times 3^3$ network:-



# Memory Organization :-

## Memory Interleaving :-

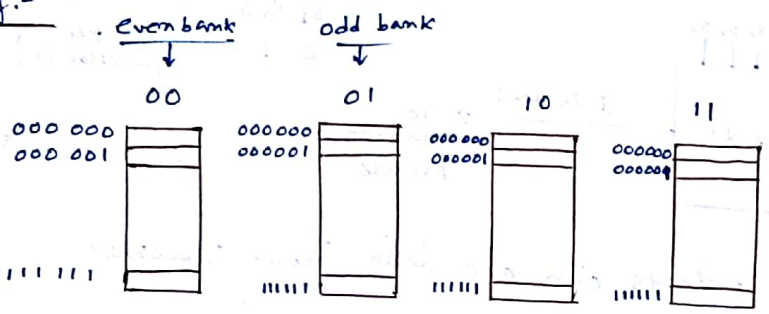
- High order interleaving
- Low order interleaving

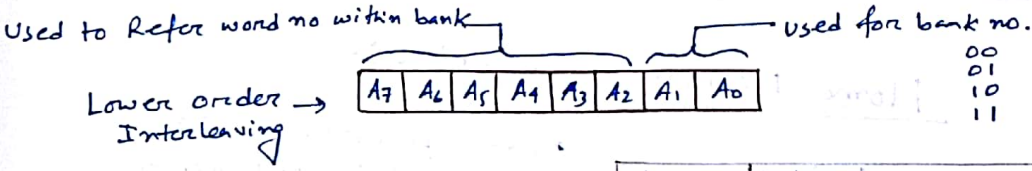
— In high order interleaving, the msb is used to access the memory bank and remaining 7 bits used to locate a word in the bank.

— In low order interleaving, the consecutive array location would split into two memory banks. This problem is not faced by high order.

Eg: 8086, 8088 and all μp's use low order interleaving ; and it has even bank and odd bank.

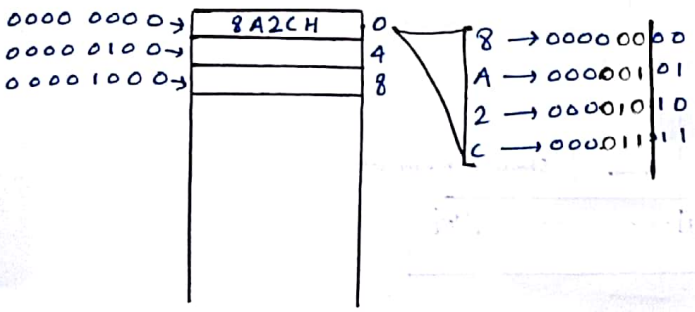
Eg:- For 32 bit data.



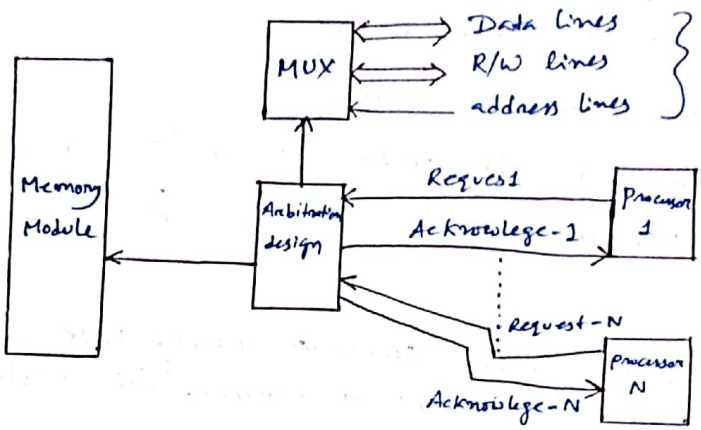

choose either :
- LSB 2 bits for bank-no (Lower order)
- MSB 2 bits for bank-no (Higher order)

total 8 bit address



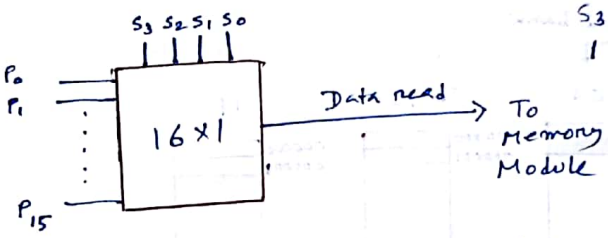

* Advantage :- 4 parallel memories are assembled, loaded into databus and transferred.

# Implementation of Selection Mechanism:-




- Data lines will write to memory module
- All processors will have different address lines.

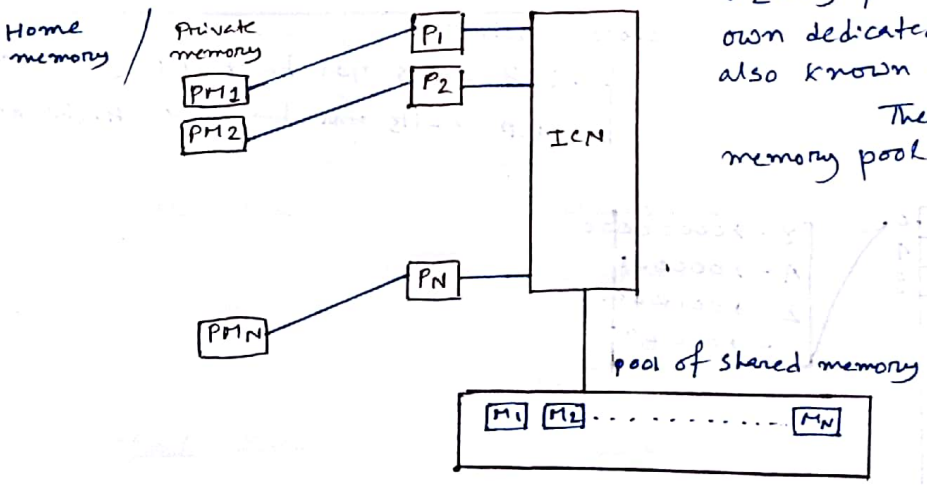
— If N=16, 4 bits will be used for data size.
if processor-10 is selected by arbitration logic for data read then,




$S_3$ $S_2$ $S_1$ $S_0$
1 0 1 0 (To select processor 10)

Mux selects only one data from processor and forwards into memory.

# Home Memory organization:-




- Every processor has its own dedicated private memory also known as home memory

There is also common memory pool for all processors.

## Interleave Memory:-

The low order interleaving of memory module is advantageous in multiprocessing when the address spaces of active processors are shared intensively. If there is very little sharing, low order interleaving may cause undesirable ~~problems~~ conflicts.

Concentrating a no. of pages of a single process in a given memory module of a high-order interleaved main memory is sometimes effective in reducing memory interference.

## Home Memory:-

In this case, a specific memory module 'Mi' may be assigned to place most of the pages belonging to a process executing on processor 'i'. Such a memory module Mi is called the 'Home Memory' of processor 'i'.

If the entire set of active pages of a process being executed on processor i contained in memory 'Mi' and if memory 'Mi' contains no. of pages belonging to the processes running on other processors; then processor i encounters no memory conflicts.

The concept of home memory can be extended so that a set of modules are assigned as the home memory of processor i. This assumes that there are more memory modules than processors. The processor-memory interconnection network (PMICN) of a mutiprocessor system may be used by processor Pi to access memory module mj; where mj is not home memory of Pi. Each memory module must have two port one of which connects to the PMICN and other connects directly to home processor.

Classification of memory access

i) Uniform memory access :- (UMA)

All the processors in UMA model share the physical memory uniformly. Here, access time to a memory location is independent of which processor makes the request or which memory chip contains the transfered data.

Eg: Crossbar Memory Organization.

ii) Non-Uniform memory access :- (NUMA)

Here, the memory access time depends on the memory location relative to the processor. Under NUMA, a processor can access its own local memory faster than non-local memory.

Eg: Home memory organization

iii) Cache only memory access :- (COMA)

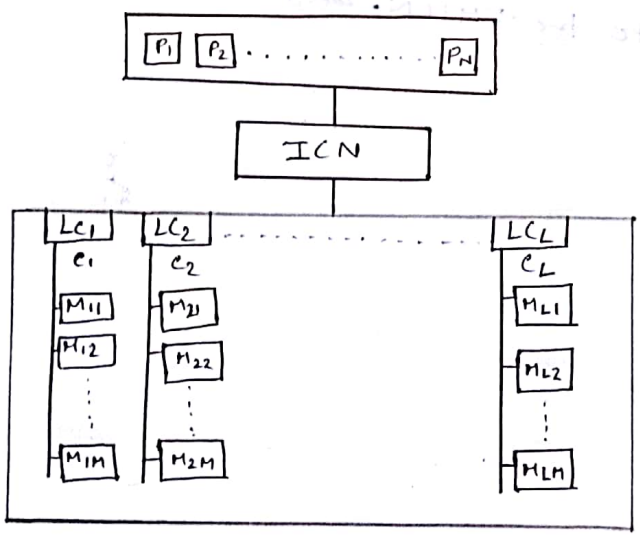
Instead of having local memory, each processor have its own cache memory.

---

LM memory Organization :-

A 2-dimensional memory organization called LM-memory organization arranged as L lines each of which contains M modules provides more flexibility.

Here total no. of memory in the system is L*M hence the name LM-memory organization.




Line Controller (LC) :-

- Line controller keeps track which data are stored in local cache.
- It decides which data should be retrieved from which memory module.
- It also keeps track whether the data is for Read or Write operation.
- It also has blocking mechanism if there is a write operation.

# Cache Coherence problem → Static / Dynamic

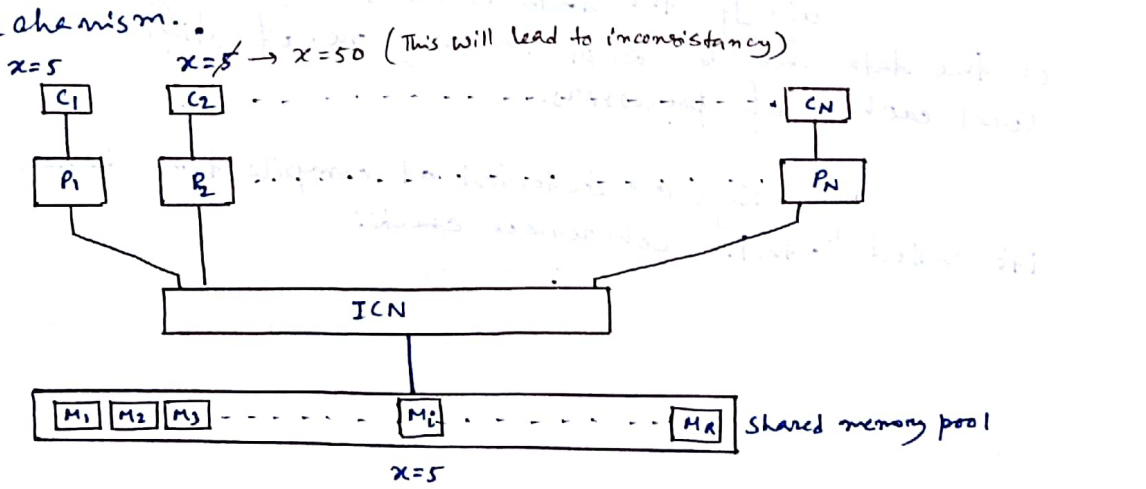
The presence of privat caches in a multiprocessor necessarily introduces problems of cache coherence which may result data inconsistancy ie: several copies of same data exists in different caches at any given time.

This is a potential problem especially in asynchronous parallel algorithm; which donot possess explicit synchronization stages of the computation.

For example, Process A which runs on processor 'i' produces data 'x' which is to be consumed by process 'B' which runs on processor $j \neq i$ asynchronously. Process 'A' writes a new 'x' into its cache while process 'B' holds old value of 'x' in its cache because it is not aware of the new 'x'.

A system of caches is coherent if and only if a 'Read' performed by any processor 'i' of a main memory location 'x' (which may be cached by other processors) always delivers most recent value with same address x.

The cache coherence problem exists only when caches are associated with processor. This problem can't be solved by a 'write-through' policy. If a write-through policy is used, the main memory location is updated but the possible copy of the variable in other caches are not automatically updated by the write through mechanism.




- In case of uniprocessor system, write-back cache work. For multiprocessor system, it wont work.

Static coherence check:-

Every piece of data is associated with two tags i) modifyable tag (R/W)
ii) Non modifyable tag (Read only)

A static cache coherence check avoids multiple copies by implementing different paths for shared writable (non cachable) and private (cachable) data. The shared data structures which are modifyable reside in the main memory. They are never placed in the cache; ie they are non cachable. A reference to this shared data is made directly to main memory. Conversely, a read only segment of data which is shared by several processors need not be non-cachable.

During the time of compilation, it is known which variable is read only or which one is R/W only. If the variable has entry of itself in atleast one write set, then its writable.

Rule:-

i) Any piece of ~~data~~ which is modifiable ~~at~~ can't be kept in local cache of any processor.

ii) Any modifications that has to be done on a modifyable data has to be done in a shared memory pool. This kind of data is called non-cachable.

iii) If the data is cachable, then the copy of the data can be kept in any no. of different local caches of processors.

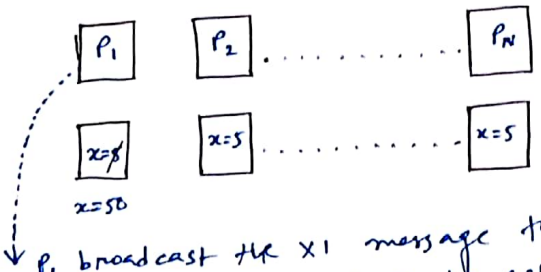
— Tags are detected at compile time hence it's called 'Static Coherence check'

## Dynamic Cache Coherence check:

Dynamic cache coherence check for solving cache coherence is more flexible than the static coherence check but also more complex and more costly. In this scheme, multiple copies of data are allowed; however, when a processor modifies a location x in cache block, it must check the other caches to invalidate possible copies. This operation is reffered to as 'Cross Interrogate' (XI). In the most basic implementation of this method, the caches are tied on a high speed bus.

When a local processor sends a signal to all the remote caches to indicate that the data at memory address X is modified. Note that to ensure correctness of execution, a processor which requests on (XI) must wait for an acknowlegement signal from all other processors before it can complete the write operation. The (XI) invalidates the remote cache location corresponding to x if it exists in that cache. When the remote processor references this invalid cache location, it results in a cache miss which is serviced to retrieve the block containing the updated information. For each write operation, (n-1) (XI)'s result where n is the no. of processors. When n increases the traffic on the high speed bus becomes a bottle neck.

XI → Cross interrogate message




$P_1$ broadcast the XI message that the variable x is changed and other P's there send acknowlegement to $P_1$ after recieving the message.

— Since the cache coherence check occures at runtime, hence the term 'Dynamic Cache Coherence Check'.