

MODULE 4

Message Passing Mechanisms-Message Routing schemes, Flow control Strategies, Multicast Routing Algorithms.

Pipelining and Superscalar techniques – Linear Pipeline processors and Nonlinear pipeline processors

4.1.MESSAGE PASSING MECHANISMS

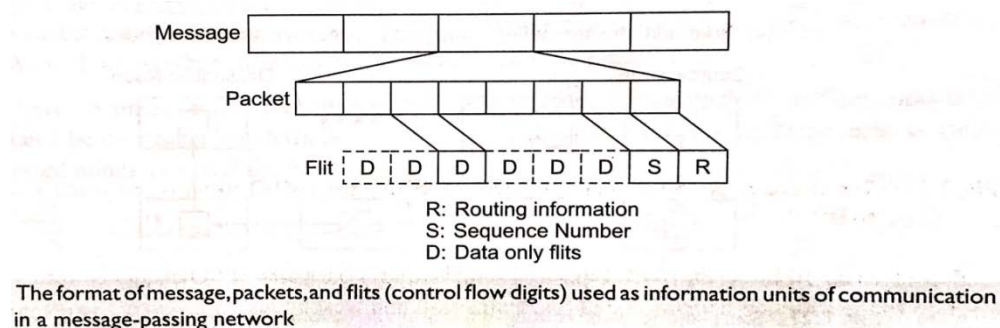
Message passing in multicomputer network demands special hardware and software support

4.1.1Message-Routing Schemes

Qn:Draw the message format used in message routing schemes?

Message Formats

- Information units in message routing is specified in fig below.



- Message** is the logical unit for inter node communication.
- A **Packet** is the basic unit containing the destination address for routing purpose.
- Since different number of packets may arrive at the destination asynchronously, a **sequence no** is needed in each packet to allow **reassembly of message** transmitted.
- A packet can be further divided into a number of fixed-length **flits(flow control digits)**. **Routing information and sequence number** occupy the header flits. The remaining flits are the data elements of a packet.
- The packet length is determined by the routing scheme and network implementation. Typical packet length ranges from 64 to 512 bits.
- Factors affecting **packet and flit size** include channel bandwidth, router design, network traffic intensity etc.

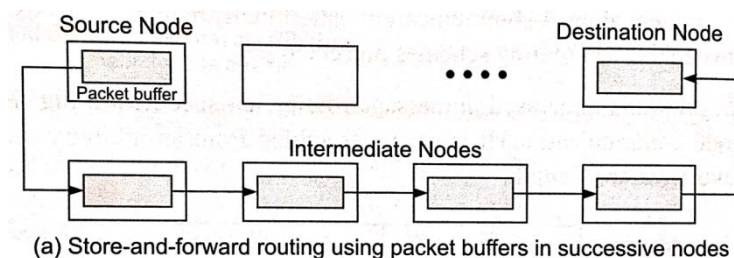
Qn: Describe two message routing Mechanisms?

Qn: Illustrate and explain two message passing format (store fwd & wormhole)

Two message routing mechanisms are

1. **Store and Forward Routing** (Packets are the smallest unit of information transmission)
2. **Wormhole routing.** (Packets are again subdivided into flits)

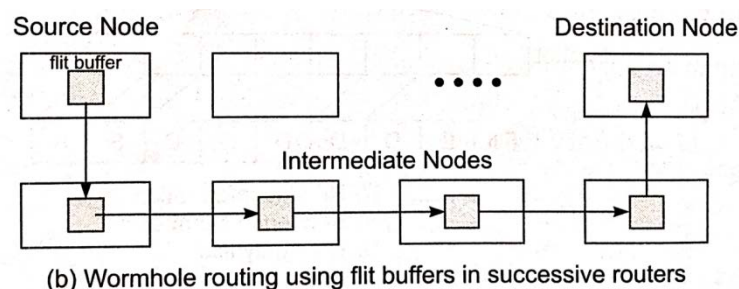
Store and Forward Routing



- Packets are the smallest unit of information transmission
- Each node has a packet buffer to store packets before forwarding to next node.
- A packet is transmitted from a source to destination node through a sequence of intermediate nodes.
- When a packet reaches an intermediate node, it is first stored in the buffer and then it is forwarded to the next node if the desired output channel and a packet buffer in the receiving node are both available.
- Latency in store and forward is directly proportional to distance (no of hops between source and destination)
- This scheme was implemented in the first generation of multicomputers.

Wormhole Routing

Qn: Describe wormhole routing?



wormhole routing

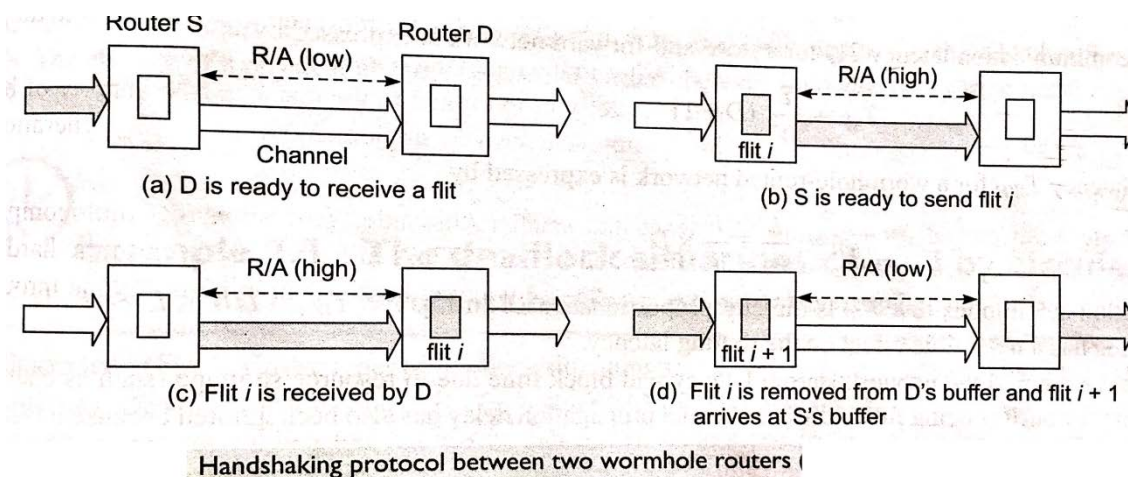
- Packets are further divided into flits

- Flit buffers are used in hardware routers attached to nodes.
- The transmission from the source node to the destination node is done through a sequence of routers.
- All flits in same packet are transmitted in pipeline fashion. Header flit knows the destination and all other flits(data flits) follow it.
- Latency in wormhole routing is independent of distance between source and destination.

Qn: Illustrate asynchronous pipelining in wormhole routing

Asynchronous Pipelining in Wormhole Routing (read lines under figs- low signals means ready to receive flit, high signal indicates ready to send flit)

- The pipelining of successive flits in a packets is done asynchronously **using a handshaking protocol**. Along the path , a **1-bit ready/request(R/A) line** is used between adjacent routers.
- When the receiving router (D) is ready to receive a flit (flit buffer is available) it pulls the R/A line low(fig.a). When the sending router (S) is ready(fig.b) it raises the line high and transmits flit i through the channel.
- While the flit is being received by D (fig.c) the R/A line is kept high. After flit i is removed from D's buffer (ie transmitted to the next node (fig d) , the cycle repeats itself for the transmission of the next flit $i+1$ until the entire packet is transmitted.

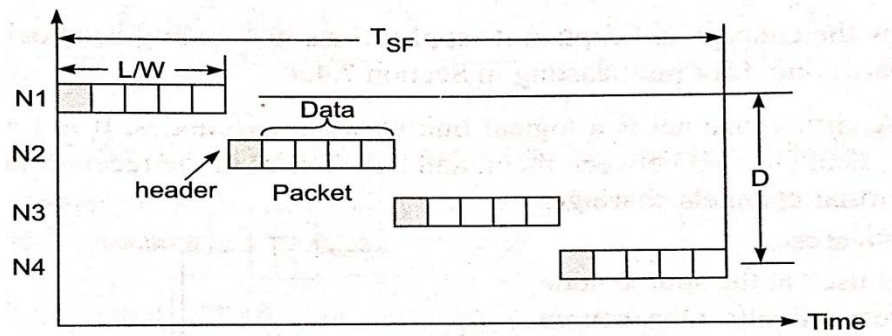


- Asynchronous pipeline can be very efficient. But the pipeline can be stalled if flit buffers or successive channels along the path are not available during certain cycles

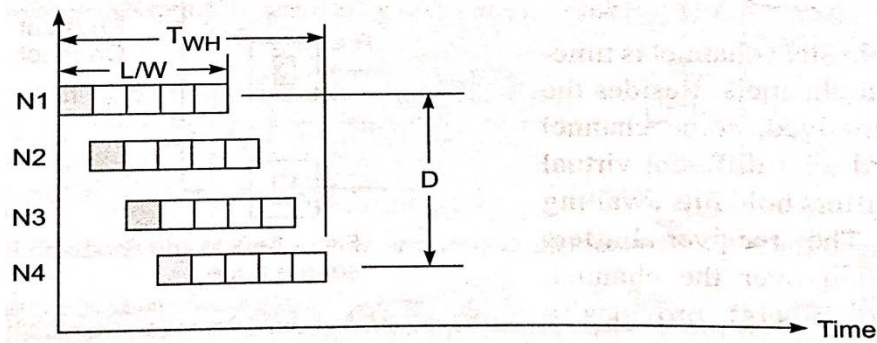
Qn:compare latencies of store-and-forward and wormhole routed networks?

Latency Analysis – A time comparison between **store-and-forward** and **wormhole routed networks**.

- **L**- packet length in bits, **W** channel bandwidth in bits/s, **D** is the distance (no of nodes traversed minus 1) and **F** is the flit length in bits



(a) Store-and-forward routing



(a) Wormhole routing

Time comparison between the two routing techniques

The communication latency T_{SF} for a store-and-forward network is expressed by

$$T_{SF} = \frac{L}{W} (D + 1) \quad (7.5)$$

The latency T_{WH} for a wormhole-routed network is expressed by

$$T_{WH} = \frac{L}{W} + \frac{F}{W} \times D \quad (7.6)$$

Equation 7.5 implies that T_{SF} is directly proportional to D . In Eq. 7.6, $T_{WH} = L/W$ if $L \gg F$. Thus the distance D has a negligible effect on the routing latency.

4.1.2 Deadlock and Virtual Channels

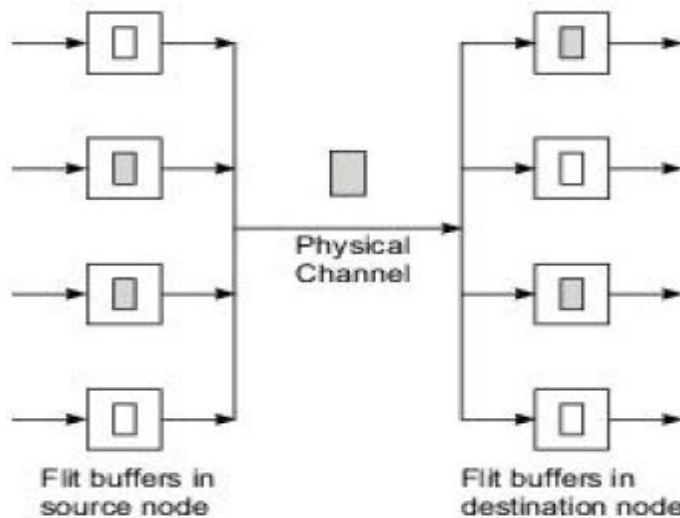
Qn: Illustrate and explain the problem deadlock and suggest solution?

Qn: Illustrate and explain virtual channel?

- The communication channels between nodes in wormhole-routed multicomputer network are shared by many source and destination pairs. The sharing of a physical channel leads to concept of **virtual channels**
- Virtual Channel** is a logical link between two nodes. It is formed by a **flit buffer in the source node**, a **physical channel between them**, and a **flit buffer in the receiver node**.
- Fig below shows the concept of four virtual channels sharing a single physical channel. Four flit

buffers used at the source node and receiver node respectively. One source buffer is paired with one receiver buffer to form a virtual channel when the physical channel is allocated for the pair.

- If Physical channel is time shared by virtual channel. The source buffer hold flits awaiting use of channel. The receiver buffer hold flits just transmitted over the channel. The channel(wires or fibers) provides a communication medium between them.

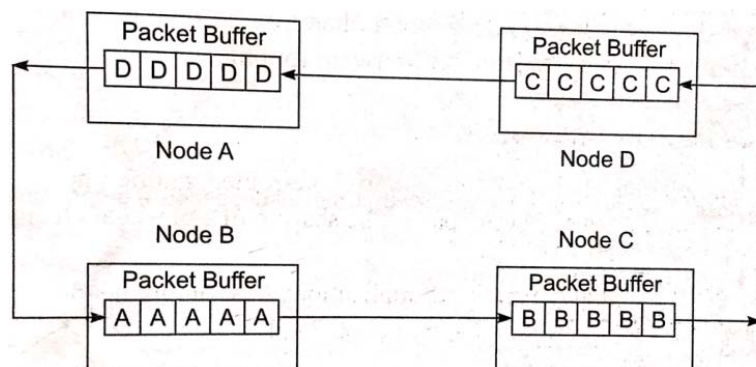


30 Four virtual channels sharing a physical channel with time multiplexing on a flit-by-flit basis

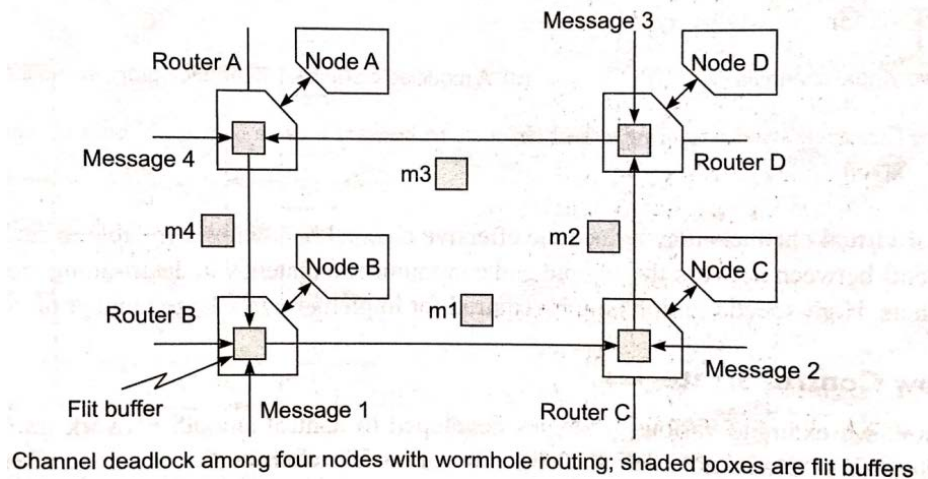
Deadlocks

Deadlocks can occur when there is circular wait at buffers or at channels. Fig below shows deadlock situation in store-and-forward and worm-hole routing.

- A **buffer deadlock** is shown in Fig a for a store and forward network. A circular wait situation results from **four packet occupying four buffers in four nodes**. Unless one packet is discarded or misrouted, the deadlock cannot be broken.
- A **channel deadlock** results from four messages being simultaneously transmitted along four channels in a mesh-connected network using wormhole routing. Four flits from four messages occupy the four channels simultaneously. If none of the channels in the cycle is freed, the deadlock situation will continue



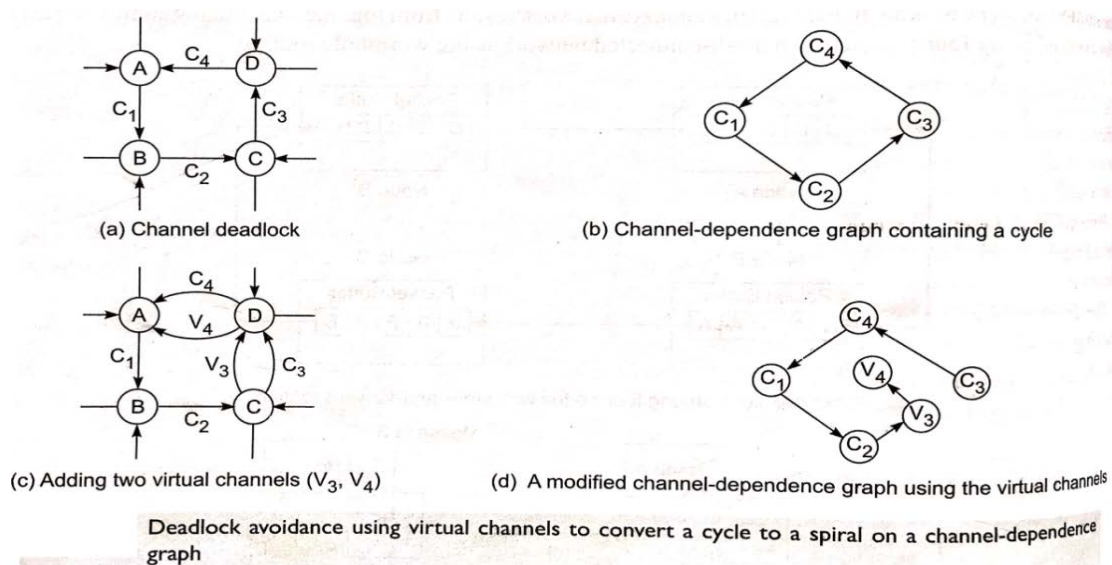
(a) Buffer deadlock among four nodes with store-and-forward routing



- circular waits also illustrated using **Channel dependence graph** in the figure below:. The channels involved are represented by nodes, and directed arrows are used to show the dependence relations among them.

Qn:describe deadlock avoidance mechanisms?

- Deadlocks can be avoided** by adding unidirectional or bidirectional Virtual Channels. Fig below shows it.



- By adding two virtual channels, V3 and V4 in the above fig. C, one can break the deadlock cycle. A modified channel dependence graph is obtained by using the virtual channels V3 and V4, after the use of channel C2 instead of reusing C3 and C4. The cycle in the fig b can be converted to a spiral, thus avoiding a deadlock. Virtual channels can be implemented with either unidirectional channels or bidirectional channels.
- The use of virtual channels may reduce the effective channel bandwidth available to each request.

4.1.3 Flow Control Strategies

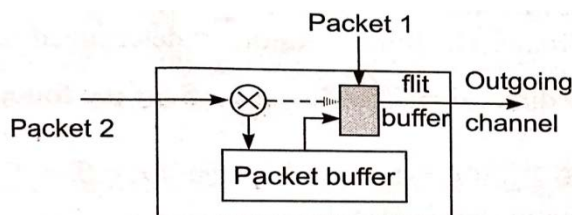
Qn: Explain the 4 packet collision resolution policies/ flow control strategies

- Flow control strategies are used **to control smooth network traffic flow** without causing congestion or deadlock situation.
- When two or more packets collide at a node competing for buffer or channel resources we need policies to resolve the conflict.

1.Packet Collision Resolution

- When two packets reach the same node, they may request the same receiver buffer or the same outgoing channel. Two arbitration decision must be made.(1) which packet will be allocated the channel? And (2) What will be done with the packet being denied the channel?
- 4 policies are illustrated below** , for resolving the conflict between two packets competing for the use of the same outgoing channel at an intermediate node.

a) Buffering in Virtual circuit routing



Buffering in virtual cut-through routing

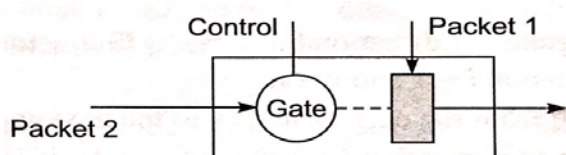
- Packet 1 is allocated the channel, packet 2 is temporarily stored in packet buffer and will be transmitted when channel becomes available.(ie combines store and forward and wormhole routing schemes)

Adv: Not wasting the resources already allocated.

Disadv: Requires the use of a large buffer to hold the entire packet.

Packet buffer may cause significant storage delay.

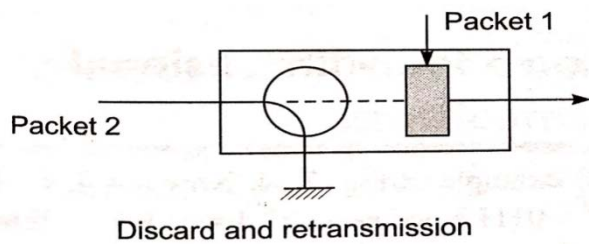
b) Blocking flow control



Blocking flow control

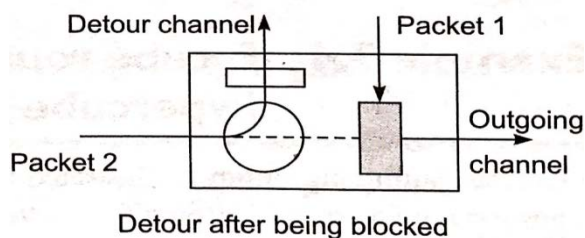
- Wormhole routing uses a blocking policy in case of packet collision .Second packet is blocked but not abandoned

c) Discard and retransmit



- Discard policy drops the second packet. The discard policy may result in severe wastage of resources, and it demands packet retransmission and acknowledgement. This scheme is rarely used now. BBN Butterfly network used this policy.

d) Detour after being blocked



- Blocked packet is routed to a **detour channel**. It is economical to implement but may result in the idling of resources allocated to the blocked packet. This scheme offers more flexibility in packet routing. But this scheme may waste more channel resources than necessary to reach at the destination. Furthermore a re-routed packet may enter a cycle of *livelock*, which wastes network resources.

Qn: Explain any 2 dimension order routing mechanisms (deterministic and adaptive, also comparison between deterministic and adaptive) ?

Qn: Illustrate and explain any 2 deterministic routing algorithms/mechanisms (E cube routing on Hypercube and X-Y routing on 2D Mesh)?

Dimension-Order Routing

- Packet routing can be done **deterministically or adaptively**. Both are deadlock free routing schemes.
 - In **deterministic routing** communication path is completely determined by the source and destination address and is independent of network condition.
 - **Adaptive routing** depends on network conditions and alternate paths are possible.
- In both type of routing, deadlock – free algorithms are desired.

- Two such **deterministic algorithms** are
 - *X-Y routing* and
 - *E-cube routing* which are based on the concept called **dimension order routing**.
- **Dimension order routing requires** the selection of successive channels to follow a specific order based on the dimensions of a multidimensional network.
 - In the case of a **two dimensional mesh network** , the scheme is called **X-Y routing** because a routing path along the X-dimension is decided first before choosing a path along the Y dimension.
 - For **hypercube (n cube) networks**, the scheme is called **E-cube routing** (proposed by Sullivan and Bashkow in 1977).

2 Deterministic algorithms are explained below

Qn: Explain with example E-cube Routing on Hypercube

1. E-cube Routing on Hypercube

Just for reference

Hypercube Routing Functions A three-dimensional binary cube network is shown in Fig. 2.15. Three routing functions are defined by three bits in the node address. For example, one can exchange the data between adjacent nodes which differ in the least significant bit C_0 , as shown in Fig. 2.15b.

Similarly, two other routing patterns can be obtained by checking the middle bit C_1 (Fig. 2.15c) and the most significant bit C_2 (Fig. 2.15d), respectively. In general, an n -dimensional hypercube has n routing functions, defined by each bit of the n -bit address. These data exchange functions can be used in routing messages in a hypercube multicomputer.

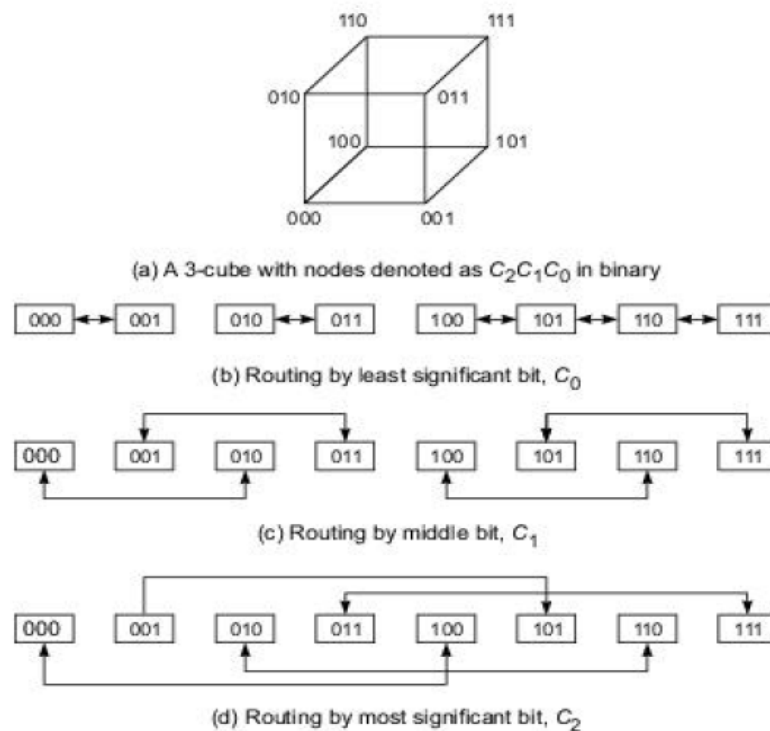


Fig. 2.15 Three routing functions defined by a binary 3-cube

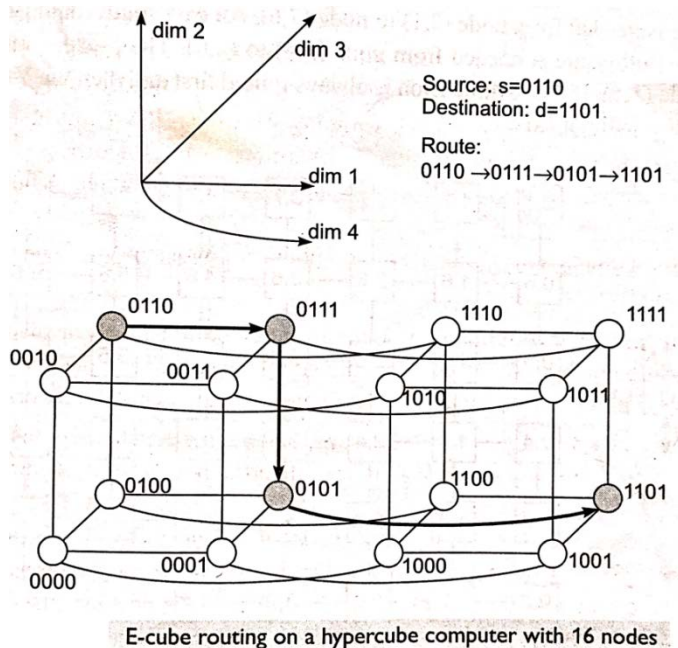
That is in this 3 dimensional binary cube network, (ie $n=3$, ie 8 nodes), figure a shows all the connections resulted by 3 routing functions defined by fig,b,c,and d.

E-cube Routing on Hypercube Consider an n -cube with $N = 2^n$ nodes. Each node b is binary-coded as $b = b_{n-1}b_{n-2} \dots b_1b_0$. Thus the source node is $s = s_{n-1} \dots s_1s_0$ and the destination node is $d = d_{n-1} \dots d_1d_0$. We want to determine a route from s to d with a minimum number of steps.

We denote the n dimensions as $i = 1, 2, \dots, n$, where the i th dimension corresponds to the $(i-1)$ st bit in the node address. Let $v = v_{n-1} \dots v_1v_0$ be any node along the route. The route is uniquely determined as follows:

1. Compute the direction bit $r_i = s_{i-1} \oplus d_{i-1}$ for all n dimensions ($i = 1, \dots, n$). Start the following with dimension $i = 1$ and $v = s$.
2. Route from the current node v to the next node $v \oplus 2^{i-1}$ if $r_i = 1$. Skip this step if $r_i = 0$.
3. Move to dimension $i + 1$ (i.e. $i \leftarrow i + 1$). If $i \leq n$, go to step 2, else done.

The above E-cube routing algorithm is illustrated with the example in Fig. Now $n = 4$, $s = 0110$, and $d = 1101$. Thus $r = r_4r_3r_2r_1 = 1011$. Route from s to $s \oplus 2^0 = 0111$ since $r_1 = 0 \oplus 1 = 1$. Route from $v = 0111$ to $v \oplus 2^1 = 0101$ since $r_2 = 1 \oplus 0 = 1$. Skip dimension $i = 3$ because $r_3 = 1 \oplus 1 = 0$. Route from $v = 0101$ to $v \oplus 2^3 = 1101 = d$ since $r_4 = 1$.



Here in the above example, $n=4$. ie total number of nodes $= 2^4 = 16$. as shown in the figure above.

Additional Problem:

QN: Show E-cube routing on a Three dimensional hypercube?

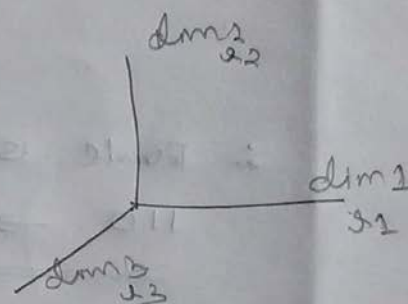
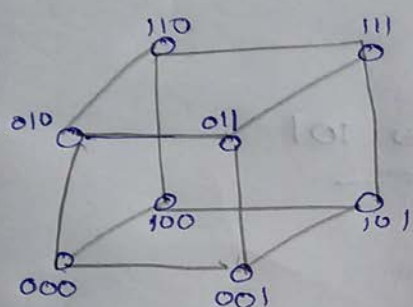
Qn: Explain the routing mechanisms defined by a binary 3 cube using necessary diagrams. (Kerala Univ)

Qn: Show the routing of a message from node (110) to node (101), by E-cube routing on a Three dimensional hypercube (ie no. of nodes $= 2^3 = 8$)

→ E-cube routing on a 3-D hypercube

$$n = 3.$$

$$N = 2^3 = 8 \text{ nodes.}$$



Find routing from $\begin{smallmatrix} S \\ 110 \\ s_2 s_1 s_0 \end{smallmatrix}$ to $\begin{smallmatrix} D \\ 101 \\ d_2 d_1 d_0 \end{smallmatrix}$.

step ① compute direction bit: $s_i = s_{i-1} \oplus d_{i-1}$

$$\therefore s = \begin{array}{r} 110 \\ \oplus 101 \\ \hline 011 \end{array}$$

direction
direction
direction

② Route from s to next node $s \oplus 2^{i-1}$

$$\begin{array}{r} s = 110 \oplus 2^1 \\ = 110 \oplus 10 \\ \hline 111 \end{array} \quad \text{--- ①}$$

$$\begin{array}{r} \text{now } 111 \oplus 2^0 \\ = 111 \oplus 1 \\ \hline 110 \end{array} \quad \text{--- ②}$$

$$\text{Now } 110 \oplus 2^{3-1}$$

No need to consider
as $s_3 \text{ bit} = 0$

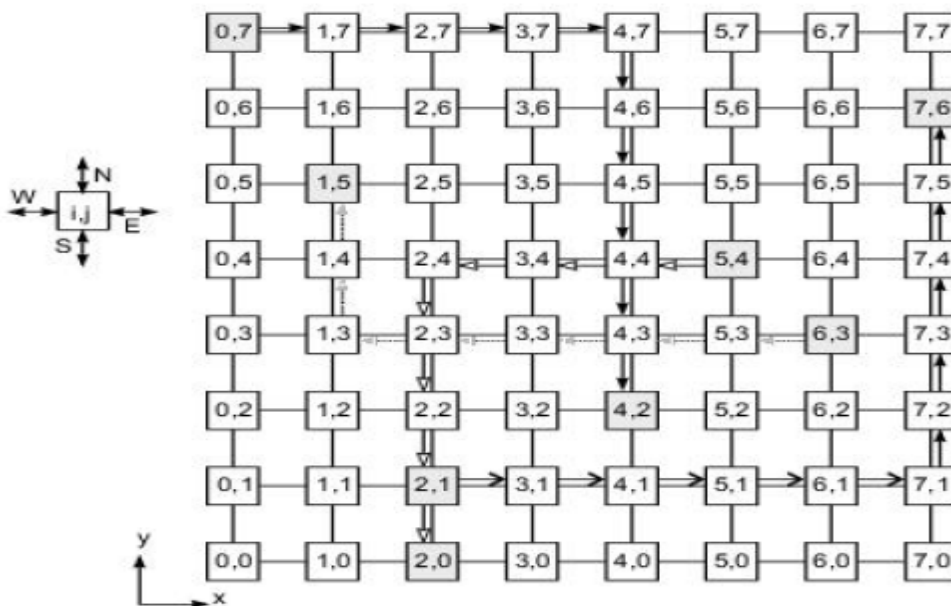
∴ Route is

$$110 \rightarrow 111 \rightarrow 101$$

Qn: Explain X-Y routing on 2-D mesh?

2. X-Y Routing on 2 D Mesh

- From any source node $s=(x_1y_1)$ to any destination node $d=(x_2y_2)$, **route from s along the X-axis first until it reaches the column Y_2 where d is located. Then route to d along the Y-axis.**
- There would be four possible X-Y routing patterns corresponding to **east-north, east-south, west-north and west-south.**
- The below given example illustrate four possible patterns on a two dimensional mesh. That is,
 - an east-North route , From node (2,1) to node (7,6)
 - an east-south route , From node (0,7) to node (4,2)
 - an -west-south route , From node (5,4) to node (2,0)
 - an -west-north route , From node (6,3) to node (1,5)
- If the X—dimension is always routed first and then the Y-dimension, **a deadlock or circular wait situation will not exist**



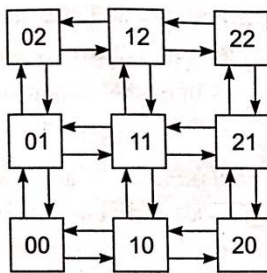
Four (source; destination) pairs: (2,1;7,6) → (0,7;4,2) → (5,4;2,0) → (6,3;1,5) →

Fig. 7.35 X-Y routing on a 2D mesh computer with $8 \times 8 = 64$ nodes

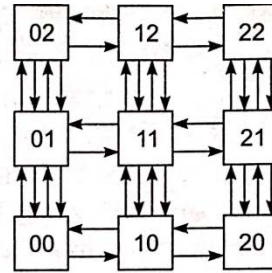
Qn: Illustrate adaptive routing mechanism?

Adaptive Routing

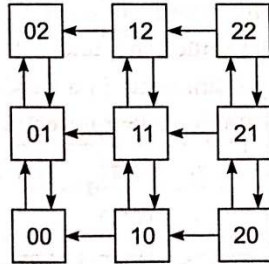
- The main purpose of using adaptive routing is to **achieve efficiency and avoid deadlock.**
- Concept of virtual channels is adopted in adaptive networks making routing more economical and feasible to implement and it avoids deadlocks too
- That is we can have **virtual channels in all connections along the same dimension** of a mesh-connected network as shown in figure below:



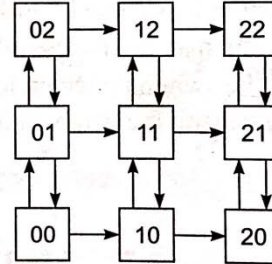
(a) Original mesh without virtual channel



(b) Two pairs of virtual channels in Y-dimension



(c) For a westbound message



(d) For an eastbound message

Adaptive X-Y routing using virtual channels to avoid deadlock; only westbound and eastbound traffic are deadlock-free

This example uses two pairs of virtual channels in the Y-dimension of a mesh using X-Y routing.

For westbound traffic, the *virtual network* in Fig. c can be used to avoid deadlock because all eastbound X-channels are not in use. Similarly, the virtual network in Fig. d supports only eastbound traffic using a different set of virtual Y-channels.

Multicast Routing Algorithms

Qn:exp multicast routing algorithm?

Qn:Describe multicast and broadcast on a hypercube?

Qn: Describe multicast and broadcast on a mesh network?

4 types of communication patterns may appear in **multicomputer networks**.

1. **Unicast** (one source one destination)(This one only discussed in previous sessions)
2. **Multicast** (one to many)
3. **Broadcast** (one to all)
4. **Conference** (many to many)
- 5.

Routing Efficiency parameters are

- **channel bandwidth** :The channel bandwidth at any time instant indicates the effective data transmission rate achieved to deliver the messages.
- **communication latency**. The latency is indicated by the packet transmission delay involved.

A routed network should achieve both **maximum bandwidth and minimum latency** for the communication patterns involved. Depending on the switching technology used, latency is the more important issue in a store-and-forward network, while in general the bandwidth affects efficiency more in a wormhole-routed network.

Multicast and Broadcast on a mesh-connected computer

Multicast routing is implemented on a 3 x 4 mesh is shown below. The source node is identified as S, which transmits a packet to five destinations labeled D_i for $i = 1, 2, \dots, 5$.

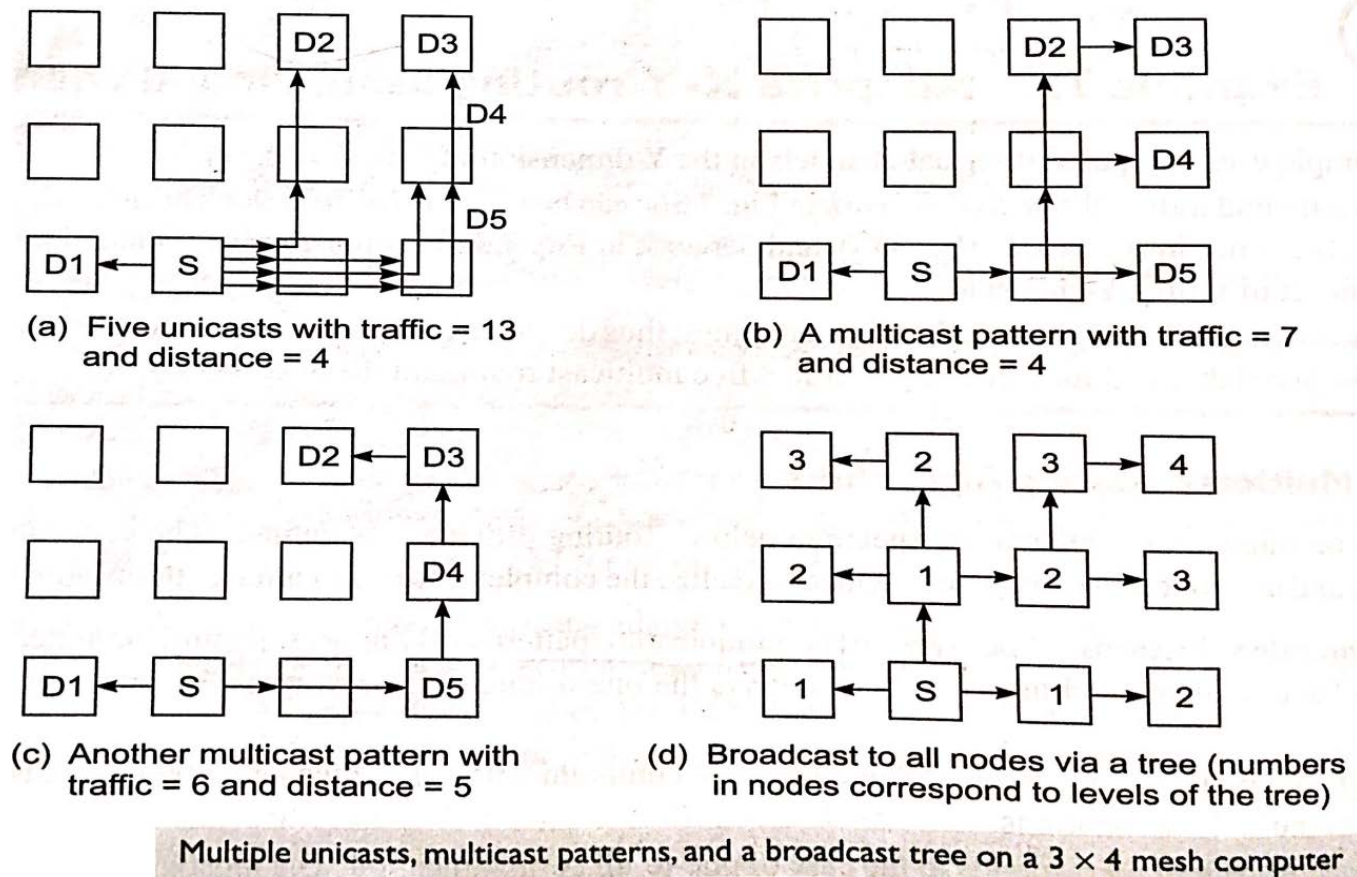


Fig a; This five destination multicast can be implemented by five unicasts, as shown in Fig. a. The X-Y routing traffic requires the use of $1 + 3 + 4 + 3 + 2 = 13$ channels, and the **latency is 4 for the longest path leading to D3.**

- **Traffic** indicates **number of channels** between nodes.

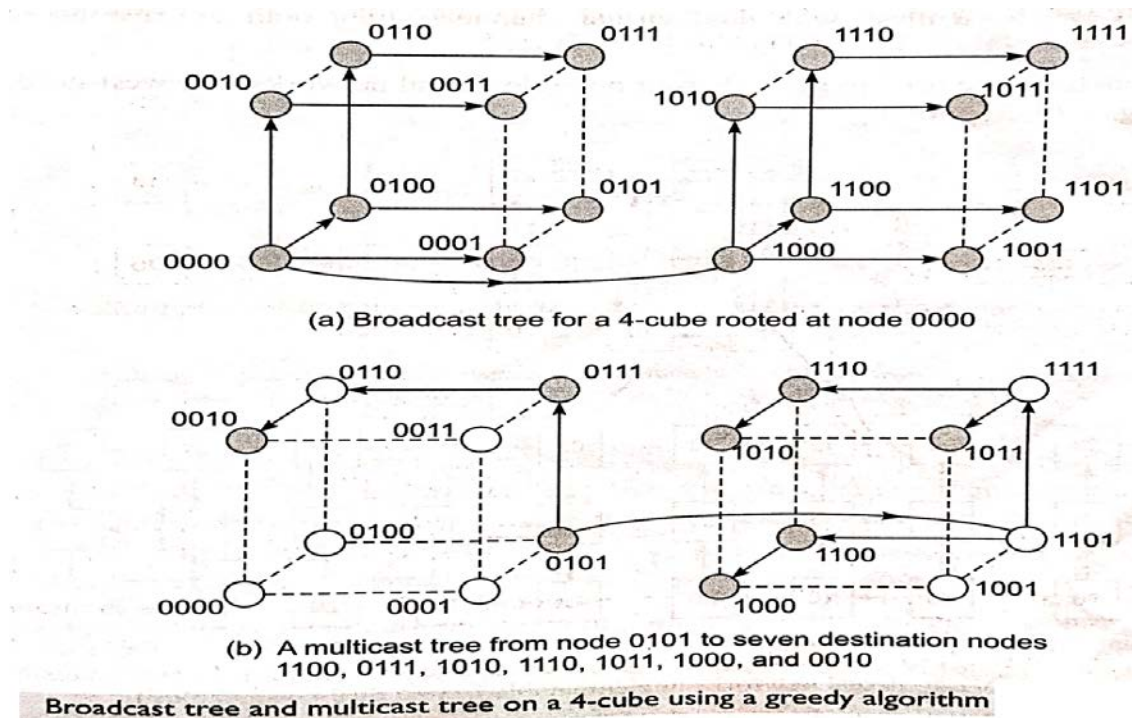
Fig b: is Better choice of multicast for **Store and forward network**. 9(minimum distance from source to destination)

Fig c: is Better choice of multicast for **wormhole routed network (minimum traffic /channels from source to destination)**

In Fig b and c, A multicast can be implemented by replicating the packet at an intermediate node, and multiple copies of the packet reach their destinations with significantly reduced channel traffic.

Fig d: A four-level spanning tree is used from node S to broadcast a packet to all the mesh nodes. Nodes reached at level i of the tree have latency i . This broadcast tree should result in minimum latency as well as in minimum traffic.

Multicast and Broadcast on a hypercube computer



A greedy multicast tree for sending a packet from node 0101 to seven destination nodes. is shown above. The greedy multicast algorithm is based on sending the packet through the dimension(S) which can reach the most number of remaining destinations.

Starting from the source node $S = 0101$, there are

two destinations via dimension 2 and

five destinations via dimension 4.

Therefore, the **first-level channels** used are $0101 \rightarrow 0111$ and $0101 \rightarrow 1101$.

From node 1101, there are

three destinations reachable in dimension 2 and

four destinations via dimension 1.

Thus the **second-level channels** used include $1101 \rightarrow 1111$, $1101 \rightarrow 1100$, and $0111 \rightarrow 0110$.

Similarly, the remaining destinations can be reached with **third-level channels**

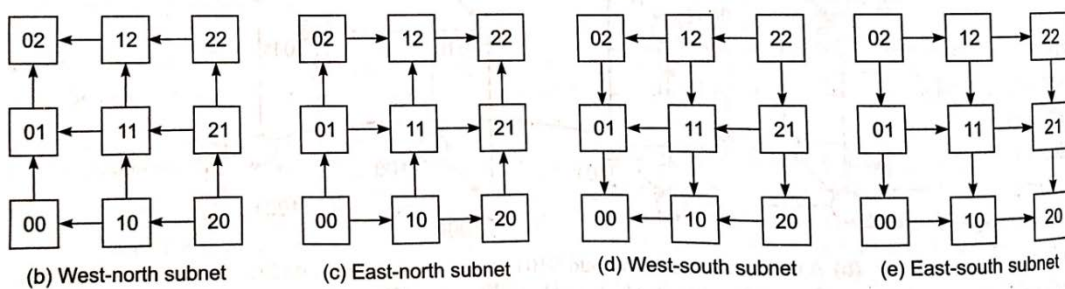
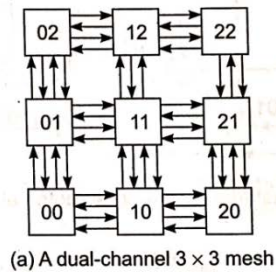
$1111 \rightarrow 1110$, $1111 \rightarrow 1011$, $1100 \rightarrow 1000$, and $0110 \rightarrow 0010$.

Finally **fourth-level channel** $1110 \rightarrow 1010$

Qn: Write note on virtual networks?

Virtual Networks

In fig below a mesh with dual channel along both dimensions are shown, these virtual channels can be used to generate four possible virtual networks as shown below.

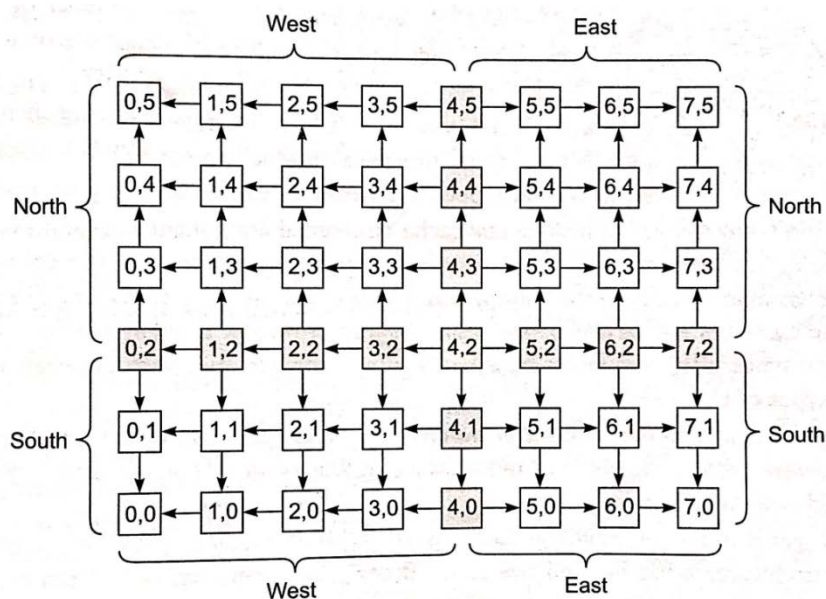


Four virtual networks implementable from a dual-channel mesh

Qn: Write note on network partitioning?

Network partitioning

Concept of virtual networks leads to partitioning of a given physical network into logical subnetworks for multicast communications.



Partitioning of a 6×8 mesh into four subnets for a multicast from source node (4,2). Shaded nodes are along the boundary of adjacent subnets

- Nodes in fifth column and third row are along the boundary between subnets.

Message Passing Mechanisms-Message Routing schemes, Flow control Strategies, Multicast Routing Algorithms.

Pipelining and Superscalar techniques – Linear Pipeline processors and Nonlinear pipeline processors

PIPELINING AND SUPERSCALAR TECHNIQUES

Qn: Compare and contrast linear and non-linear pipelines.?

Qn:Differentiate between synchronous and asynchronous processor pipeline models. Define the parameters a)speedup b)efficiency and c)throughput in the context of a processor pipeline?

Qn: define the following with respect to pipelining: a)Speedup b)Efficiency c)throughput?

4.2 LINEAR PIPELINE PROCESSORS

- Is a cascade of processing stages connected linearly to perform a fixed function over a stream of data flowing from one end to another.
- In modern computers linear pipelines are applied for instruction execution, arithmetic computation and memory-access operations.



- Constructed with k processing stages, external inputs (operands) are fed into the pipeline at first stage S₁
- Processed results are passed from S_i to S_{i+1} for i=1,2,3...k-1.
- Final result emerges from last stage S_k

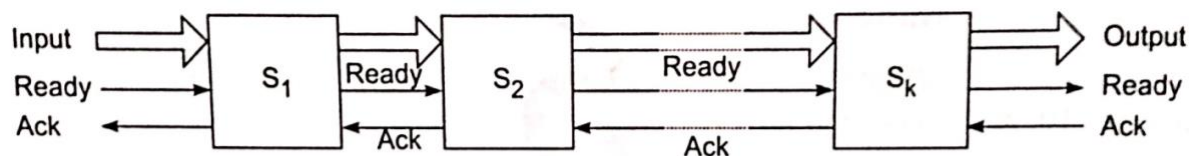
Depending on control flow **linear pipeline divide into 2**

- **Asynchronous**
- **Synchronous**

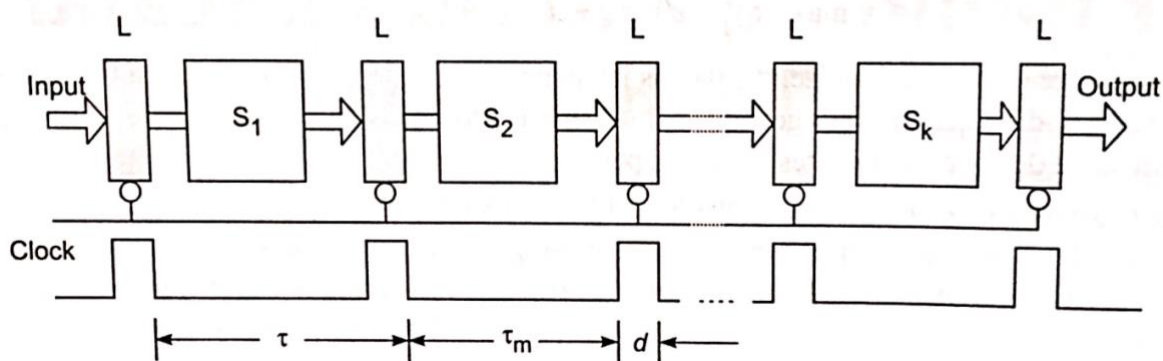
Asynchronous	Synchronous
1. Data flowing btw adjacent stages controlled by Handshaking protocol L	1. Clocked latches used to interface btw stages. Latches made with master-slave flip-flop
2. Figure -below	2. Figure –next below
3. Working – when S _i is ready to transmit, it sends a ready signal to S _{i+1} . after S _{i+1} receives incoming data it returns acknowledgement signal to S _i	3. Working - Upon arrival of clock pulse latches transfer data to next stage simultaneously. [clocked latches are used to interface between stages]
4. Different amount of delay experienced in different stages	4. Have approximately equal delay

5. Asynchronous pipelines are useful in designing communication channels in message passing multicomputer
6. Has variable throughput rate

5. Delays determine clock period and thus speed of pipeline



An asynchronous pipeline model



A synchronous pipeline model

4.2.1 RESERVATION TABLES (static pipeline)

- Reservation tables specifies the utilization pattern of successive stages in synchronous pipeline – which stage is used in which clock cycle.

		Time (clock cycles)			
		1	2	3	4
Stages	S_1	X			
	S_2		X		
	S_3			X	
	S_4				X

Captions:

S_i = stage i

L = Latch

τ = Clock period

τ_m = Maximum stage delay

d = Latch delay

Ack = Acknowledge signal.

Reservation table of a four-stage linear pipeline

- For linear pipeline, the utilization follows diagonal stream line pattern (above fig).
- This table is Space-Time diagram depicting precedence relationship in using pipeline stages.
- For a K -stage linear pipeline, k clock cycles are needed for data to flow through the pipeline.
- Successive tasks or operations are initiated one per cycle to enter the pipeline. Once the pipeline is filled up, one result emerges from the pipeline for each additional cycle.
- This throughput is sustained only if the successive tasks are independent of each other.

4.2.2.CLOCKING & TIMING CONTROL of LINEAR PIPELINE

- To determine the clock cycle(τ_i) of a pipeline-

- Let τ_i be the time delay of circuitry in stage S_i
- d be the time delay of latch
- Let τ_m denote maximum stage delay

Then Clock Cycle (τ) equals

$$\tau = \max_i \{\tau_i\}_1^k + d = \tau_m + d$$

- Usually $\tau_m \gg d$. Thus maximum stage delay (τ_m) dominates clock period of latch..

- To determine Pipeline frequency – it is defined as inverse of clock period :

$$f = \frac{1}{\tau}$$

- If one result is expected to come out of pipeline per cycle, f represents **maximum throughput**.
 - But actual throughput of pipeline is usually lower than f . This is because more than one clock cycle has elapsed between successive task initiation.
- Clock Skewing – Ideally the clock pulses are expected to arrive at all stages (latches) at the same time. But due to a problem known as clock skewing, same clock pulse may arrive at different stages with a time offset of s . It occurs due to some features of medium like resistance, capacitance etc.

4.2.3.SPEEDUP ,EFFICIENCY AND THROUGHPUT

- To determine Speed up of Linear Pipeline v/s non-pipelined processor

- Ideally a linear pipeline of k stages can process n tasks in $k+(n-1)$ clock cycles.
- Where, k cycles are needed to complete execution of very first task.
- Remaining $(n-1)$ tasks require $(n-1)$ cycles.

- Thus total time required $T_k = [k + (n - 1)]\tau$

- Time taken to execute n tasks on non-pipeline processor $T_1 = nk\tau$

- **Speed Up Factor (S_k)** of k stage pipeline over equivalent non-pipelined processor

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{k\tau + (n-1)\tau} = \frac{nk}{k + (n-1)}$$

(T_1 is non-pipelined processor and T_k is pipelined processor with k stages).

The maximum speedup is $S_k \rightarrow k$ as $n \rightarrow \infty$. This maximum speedup is very difficult to achieve because the data dependence between successive tasks (instructions), program branches, interrupts and other factors.

• Optimal Number of Stages

- Most pipelining is staged at functional level $2 \leq k \leq 15$
- Few pipelines **exceed 10 stages**
- The **number of pipeline stages cannot increase indefinitely due to practical constraints on costs**, control complexity, circuit implementation, and packaging limitations.
- In macro-pipelining the optimal choice of number of pipeline stage should be able to maximize **performance/cost ratio (PCR)** for the target processing load.
- Let t be the **total time required for a nonpipelined sequential program** of a given function.
- To execute the same program on a **k -stage pipeline** with an equal flow-through delay t one needs a clock period of $p = t/k + d$, where d is the latch delay.
- Thus the pipeline has maximum throughput of $f = 1/p = 1/(t/k + d)$
- **Total pipeline cost** $= c + kh$, where c -costs for all logic stages and h - cost of latches.

$$PCR = \frac{f}{c + kh} = \frac{1}{(t/k + d)(c + kh)}$$

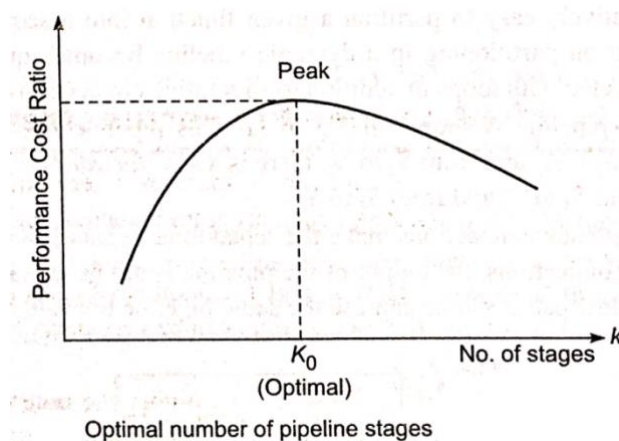


Figure above plots the PCR as a function of k . The peak of the PCR curve(k_0) corresponds to an optimal choice for the number of desired pipeline stages:

$$k_0 = \sqrt{\frac{t \cdot c}{d \cdot h}}$$

where t is the total flow-through delay of the pipeline. Thus the total stage cost c , the latch delay d , and the latch cost h must be considered to achieve the optimal value k_0 .

Problem – Find the optimal number of pipeline stages k_0 using the performance/cost ratio (PCR) ?

Answer :

$PCR = \frac{f}{c + kh}$ where f is clock rate, c is the cost of all logic stages and h represents the cost of each latch on a k – stages pipeline. The optimal pipeline stages – in term of PCR – can be formulated :

$$PCR = \frac{f}{c + kh} \Leftrightarrow PCR(c + kh) = f$$

$$cPCR + khPCR = f$$

$$khPCR = f - cPCR$$

$$k = \frac{f - cPCR}{hPCR} \rightarrow k_0$$

- **To determine Efficiency (E_k)**

- efficiency E_k of a linear K-stage pipeline is

$$E_k = \frac{S_k}{k} = \frac{n}{k + (n - 1)}$$

- **To determine Pipeline Throughput (H_k)**

- Its defined as the no: of tasks(operations) performed per unit time

$$H_k = \frac{n}{[k + (n - 1)]\tau} = \frac{nf}{k + (n - 1)}$$

Problem:1

Consider the execution of a program of 15,000 instructions by a linear pipeline processor with a clock rate of 25 Mhz. Assume that the instruction pipeline has five stages and that one instruction is issued per clock cycle. The penalties due to branch instructions and out-ofsequence executions are ignored.

a. Calculate the speedup factor using this pipeline to execute the program as compared with the use of an equivalent nonpipelined processor with an equal amount of flow-through delay.

b. What are the efficiency and throughput of this pipelined processor ?

Answer :

Information we get are :

☞ $n = 15,000$ instructions or tasks.

☞ $f = 25 \text{ MHz}$.

☞ $k = 5$ stages.

☞ 1 – issued processor.

The Speedup (S_k), Efficiency, (E_k), and Throughput (H_k) factors are :

$$\begin{aligned}
 S_k &= \frac{T_1}{T_k} = \frac{nk\tau}{k\tau + (n-1)\tau} & H_k &= \frac{nf}{k + (n-1)} & E_k &= \frac{S_k}{k} \\
 &= \frac{nk}{k + (n-1)} & &= \frac{(15,000)(25)}{5 + (15,000 - 1)} & &= \frac{4,999}{5} \\
 &= \frac{(15,000)(5)}{5 + (15,000 - 1)} & &= \frac{375,000}{15,004} & &= 0,999 \\
 &= \frac{75,000}{15,004} & &= 24,99 \text{ MIPS} & & \\
 &= 4,999 & & & &
 \end{aligned}$$

PROBLEM -2

Problem 6.15 – A nonpipelined processor X has a clock rate of 25 MHz and an average CPI of 4. Processor Y, an improved successor of X, is designed with a five-stage linear instruction pipeline. However, due to the latch delay and clock skew effects, the clock rate of Y is only 20 MHz.

- If a program containing 100 instructions is executed on both processor, what is the speedup of processor Y compared with that of processor X.
- Calculate the MIPS rate of each processor during the execution of this particular program.

Answer :

a. $CPI_X = 4$, $f_X = 25 \text{ MHz}$ and $I_C = 100$.

Find the CPU (T) time for each processor as followed :

$$\begin{aligned} T_X &= CPI_X \cdot I_c \cdot \tau_x \\ &= \frac{CPI_X \cdot I_c}{f_x} \\ &= \frac{(4)(100)}{25 \cdot 10^6} \\ &= 16 \mu s \end{aligned}$$

and

$$\begin{aligned} T_Y &= [k + (n - 1)] \tau_Y \\ &= \frac{k + (n - 1)}{f_Y} \\ &= \frac{5 + (100 - 1)}{20 \cdot 10^6} \\ &= 5.2 \mu s \end{aligned}$$

so that the **speedup** is

$$\begin{aligned} S_k &= \frac{T_X}{T_Y} \\ &= \frac{16}{5.2} \\ &= 3.078 \approx 3.08 \end{aligned}$$

b. The MIPS rate for each processor is

$$\begin{aligned} MIPS_X &= \frac{f}{CPI \cdot 10^6} \\ &= \frac{25 \cdot 10^6}{4 \cdot 10^6} \\ &= 6.25 \text{ MIPS} \end{aligned}$$


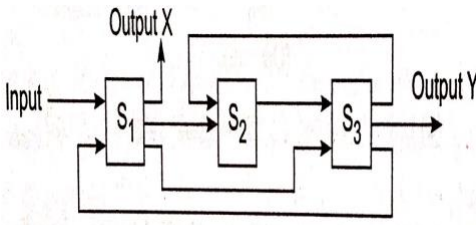
Find the **CPI** for Processor Y as followed :

$$\begin{aligned} T_Y &= CPI_Y \cdot I_c \cdot \tau_Y \quad \leftrightarrow \quad CPI_Y = \frac{T_Y}{I_c \cdot \tau_Y} \\ CPI_Y &= \frac{T_Y \cdot f_Y}{I_c} \\ &= \frac{(5.2 \cdot 10^{-6})(20 \cdot 10^6)}{100} \\ &= 1.04 \end{aligned}$$

Then, the **MIPS** rate is

$$\begin{aligned}
 MIPS_Y &= \frac{f_Y}{CPI \cdot 10^6} \\
 &= \frac{20 \cdot 10^6}{(1.04) \cdot 10^6} \\
 &= \mathbf{19.23 \text{ MIPS}}
 \end{aligned}$$

4.3 NON LINEAR PIPELINE DESIGN

STATIC / LINEAR Pipeline	DYNAMIC or NON-LINEAR Pipeline
<p>1. Linear pipelines are static pipelines because they perform fixed function</p> <p>2. Allows only streamline connection</p> <p>3. </p> <p>4. Reservation Table for Linear Pipeline.</p>	<p>1. Dynamic pipeline can be re-configured to perform variable functions at different times</p> <p>2. Allow feed-forward and feedback connections in addition to stream line connection</p> <p>3.  A three-stage pipeline</p> <p>The 3 stage pipeline has Streamline connection from S1 to S2, Feedforward from S1 to S3 Feedback from S3 to S2 and S3 to S1</p> <p>4. Reservation Tables for Non-Linear Pipeline</p>

		Time (clock cycles)			
		1	2	3	4
Stages	S ₁	X			
	S ₂		X		
	S ₃			X	
	S ₄				X

- Only streamline connection – thus only one function evaluation
- Specified by a single reservation table
- Output always obtained from last stage
- All initiations to static pipeline use same reservation table.
- Function portioning is easy-since only streamline connection

		Time							
		1	2	3	4	5	6	7	8
Stages	S ₁	X					X		X
	S ₂		X		X				
	S ₃			X		X		X	

Reservation table for function X

		Time					
		1	2	3	4	5	6
Stages	S ₁	Y				Y	
	S ₂			Y			
	S ₃		Y		Y		Y

Reservation table for function Y

- Multiple reservation table can be generated for evaluation of different functions
- Allows different initiations to follow a mix of reservation table.
- Output not necessarily from last stage.
- Following different dataflow patterns we can use same pipeline to evaluate different functions
- Function portioning is difficult because pipeline stage interconnected with loops in addition to streamline connection

4.3.1 RESERVATION AND LATENCY ANALYSIS

RESERVATION TABLES (Dynamic Pipeline)

- Can have multiple reservation table corresponding to diff function evaluation.
- No of columns in a reservation table is called **evaluation time of a function**.

		Time							
		1	2	3	4	5	6	7	8
Stages	S ₁	X					X		X
	S ₂		X		X				
	S ₃			X		X		X	

Reservation table for function X

		Time					
		1	2	3	4	5	6
Stages	S ₁	Y				Y	
	S ₂			Y			
	S ₃		Y		Y		Y

Reservation table for function Y

Figure:4.3.1

- Abv fig func X require 8 clock cycles, Y requires 6 clock cycles

- Checkmarks in each row correspond to **time instant (cycle) that a particular stage** will be used.
- Multiple checkmarks in a row indicate repeated usage of same stage in diff cycles
- Multiple checkmarks in a column indicate multiple stages are used in parallel during a particular clock cycle.

LATENCY ANALYSIS

Definitions

latency – The no: of time units(clock cycles) between two initiations of a pipeline. A latency of K means that 2 initiations are separated by k clock cycles.

Collision – Attempt by 2 or more initiations to use same pipeline stage at the same time. A collision implies resource conflicts between 2 initiations in the pipeline and must be avoided

Forbidden Latencies – latencies that cause collision. Ex: in fig above on evaluating function X, latencies 2 and 5 are forbidden.

→ Time

	1	2	3	4	5	6	7	8	9	10	11	
S_1	X_1		X_2		X_3	X_1	X_4	X_1, X_2		X_2, X_3		
Stages S_2		X_1		X_1, X_2		X_2, X_3		X_3, X_4		X_4		...
S_3			X_1		X_1, X_2		X_1, X_2, X_3		X_2, X_3, X_4			

(a) Collision with scheduling latency 2

→ Time

	1	2	3	4	5	6	7	8	9	10	11	
S_1	X_1					X_1, X_2		X_1				
Stages S_2		X_1		X_1			X_2		X_2			...
S_3			X_1		X_1		X_1	X_2		X_2		

(b) Collision with scheduling latency 5

.6.4 Collisions with forbidden latencies 2 and 5 in using the pipeline in Fig. 6.3 to evaluate the function X

The i th initiation is denoted as X_i in Fig. above. **With latency 2**, initiations X_1 and X_2 collide in **stage 2** at **time 4**. At **time 7**, these initiations collide in **stage 3**. Similarly, other collisions are shown at times 5, 6, 8, ..., etc.

How to detect Forbidden Latencies?

Detected by checking the distance between any 2 checkmarks in the same row of the reservation table.

		Time →							
		1	2	3	4	5	6	7	8
Stages	S ₁	X					X		X
	S ₂		X		X				
	S ₃			X		X		X	

Reservation table for function X

Ex: distance btw 1st and 2nd checkmark in row s1 is 5 (ie 6-1) – implies 5 is a forbidden latency

Other forbidden latencies are – 2 (in row S2 and S3),

- latency 4 (row S3(btw 3 and 7)) and
- Latency 7 (row S1(btw 1 and 8))
- Thus forbidden latencies are -2,4,5,and 7
- all others are **permissible latencies** – 1,3,6

Latency Sequence – a sequence of permissible non-forbidden latencies between successive task initiations. Ex - (1,3,6) or (3,6)

Latency Cycle – a latency sequence which repeats the same subsequence(cycle) indefinitely. Ex: latency cycle (1,8) – 1,8,1,8,1,8 – it implies successive initiations of new tasks separated by 1 cycle and 8 cycles alternately.

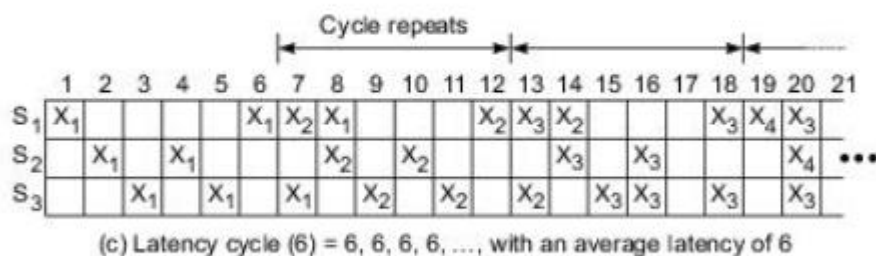
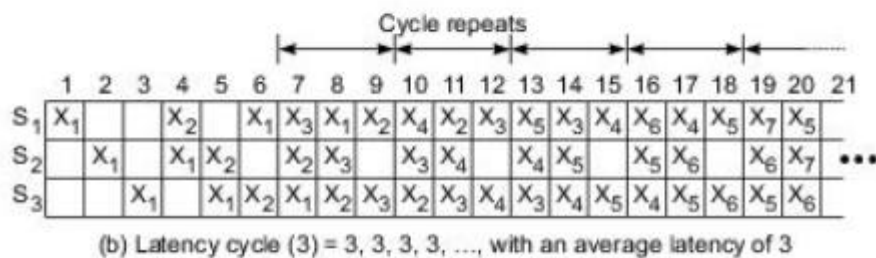
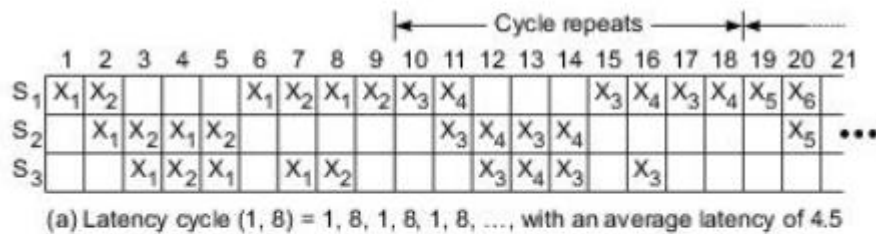


Fig. 6.5 Three valid latency cycles for the evaluation of function X

Average Latency – Obtained by dividing the sum of all latencies along the cycle. Latency cycle(1,8) thus has an average latency $(1+8)/2 = 4.5$

Constant Cycle – a latency cycle which contains only one latency value. ex: cycle(3) and (6).

4.3.2 COLLISION FREE SCHEDULING

The main objective of scheduling events in a pipeline is to obtain shortest average latency between initiations without collision.

Collision Vectors – by examining the reservation table we can distinguish the set of permissible latencies from the set of forbidden latencies. Collision vectors represent permissible and forbidden latency.

For a reservation table with n columns ,**maximum forbidden latency (m)** $\leq n-1$

Permissible latency p should be as small as possible $1 \leq p \leq m-1$. $P=1$ is the ideal case.

Definition : **COLLISION VECTOR**

- The combined set of permissible and forbidden latencies can be easily displayed by a Collision Vector, which is a m-bit binary vector (C)

$$C = (C_m C_{m-1} \dots C_2 C_1)$$

- $C_i = 1$, if latency i causes a collision
- $C_i = 0$, if latency i is permissible

→ Time

	1	2	3	4	5	6	7	8
Stages								
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

Reservation table for function X

For the above Reservation table **Collision Vector $C_x = (1\ 0\ 1\ 1\ 0\ 1\ 0)$**

Latencies – **1,3,6 are permissible** (shown with 0's in collision vector, starts from right)

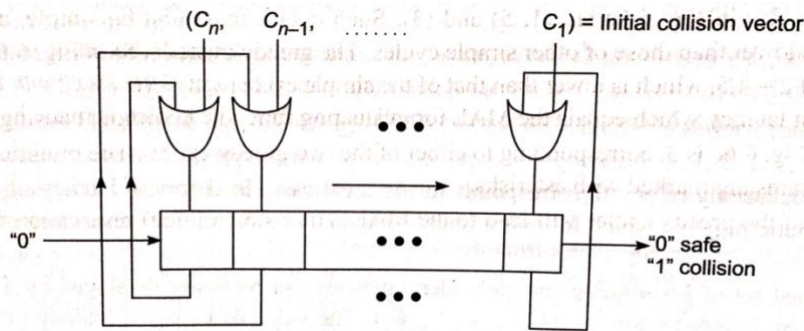
Latencies 2,4,5,7 are not permissible (shown with 1's in collision vector)(here 7 is the maximum forbidden latency ,so 7 bits in collision vector)

STATE DIAGRAMS

We can construct a state diagram specifying the permissible state transition among successive initiations.

The Collision vector C_x (above) corresponds to initial state of pipeline at time 1 and thus it is called **initial collision vector**.

The next state of pipeline at time $t+p$ is obtained with assistance of an **m bit right shift register** (shown below)



State transition using an n -bit right shift register, where n is the maximum forbidden latency

Working

1. The initial collision vector C is initially loaded into the register
2. The register is then shifted to the right. Each 1-bit shift corresponds to an increase in latency by 1.
3. When 0 bit emerges from right end after 'p' shifts, it means p is permissible latency.
4. If 1 bit is being shifted out, it means a collision and thus – forbidden latency.
5. Logical 0 enters from left end of shift register in each shift
6. The next state after 'p' shifts is obtained by Bitwise OR-ing the initial collision vector with shifted register contents.

PROBLEM ASKED from Collision Free Scheduling (can also be drawn/ written if Collision Free Scheduling with ex is asked)

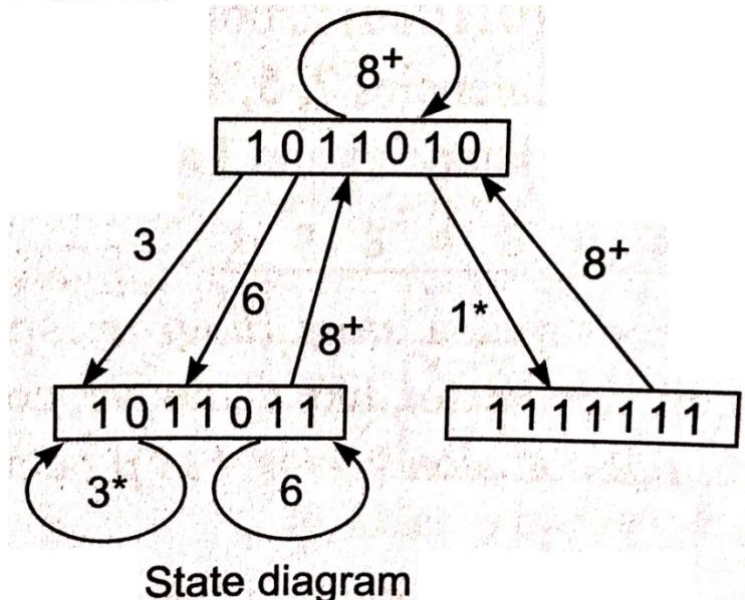
Q) Consider the reservation table for the pipelined processor and answer the questions that follow.

		Time							
		1	2	3	4	5	6	7	8
Stages	S_1	X					X		X
	S_2		X		X				
	S_3			X		X		X	

Reservation table for function X

1. List the set of forbidden latencies and write the collision vector
2. Draw the state transition diagram
3. List all simple and greedy cycles
4. Find minimum average latency MAL
5. Find throughput of pipeline
 - **Forbidden latencies** – 2,4,5,7 (calculated by checking distance btw checkmarks)
 - **Permissible latencies** – 1,3, 6 (all other latencies other than forbidden)
 - **Collision vector C_x** = 1011010 (0-permissible, 1- forbidden, starting from right end)

Constructing State Transition Diagram for a pipeline unit



- Initial collision vector $C_x = 1011010$
- On right shifting 1 bit and inserting 0 bit from left we get 0101101
- 7. Ie:- a 0 bit pop's out from right - thus no collision – it's a permissible state – and has a state in the state transition diagram. [When 0 bit emerges from right end after 'p' shifts, it means p is permissible latency.]
- To find the state transition of permissible latency , we OR the shifted content with initial vector

$$\begin{array}{r}
 1011010 \\
 \text{OR } 0101101 \\
 \hline
 1111111
 \end{array}$$

- Ie: for latency 1, the transition state is 1111111, thus an arrow from 1011010 to 1111111 for latency = 1
- Now on 2nd right shift of initial collision vector we get 0010110, popping out a '1' bit – thus cause collision – forbidden latency – no state transition – thus no need to OR with initial vector.
- 3rd right shift of 0010110 pop's out a 0 – and we get 0001011, since 0 pops out it's a permissible latency and has a transition state.
- To find the transition state we **OR** 0001011 with initial collision vector 1011010

$$\begin{array}{r}
 1011010 \\
 \text{OR } 0001011 \\
 \hline
 1011011
 \end{array}$$

- Ie:- for latency 3, the transition state is 1011001, thus an arrow from 1011010 to 1011011 for latency = 3
- 4th shift and 5th shift pops out 1 thus - not permissible latency – has no state transition.

- 6th shift leads to 0000001 popping out 0 bit.
- Transition state for latency 6 is

```

    1011010
OR  0000001
-----
    1011011

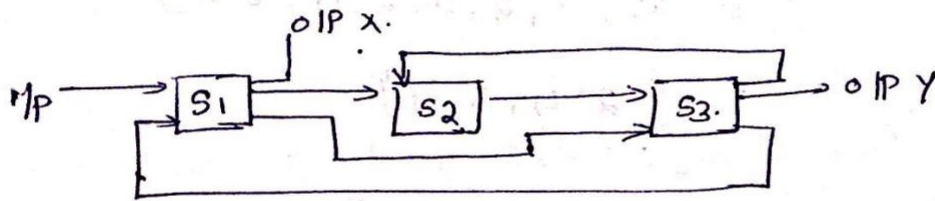
```

- Thus a line from initial collision vector to 1011011 for latency = 6
- 7th shift – pops out 1 – not permissible.
- For 8 or more shifts all transitions are redirected back to initial state since ORing initial collision vector with 0000000 returns initial vector itself.
- Similarly from state 1011011, we reach same state itself after 3 shifts or 6 shifts since 0 pops out on 3rd and 6th shift.
- When the number of shifts is m+1 or greater, all transitions are redirected back to the initial state, regardless of which state the transition starts from.
- **Simple Cycles** – is a latency cycle in which each state appears only once. In the above state diagram (3), (6), (8), (3,8) and (6,8) are simple cycles. The cycle (1,8,6,8) is not a simple cycle because it repeats state 1011010 twice.
- **Greedy Cycles** – is one whose edges are all made with minimum latencies from their respective starting state. Such cycles must be simple and their average latencies must be lower than other simple cycles.. cycle (1,8) and (3) are greedy cycle has an average latency $(1+8)/2 = 4.5$ which is lower than other simple cycles. The greedy cycle (3) has a constant latency and is the **minimum average latency (MAL)**.
- Thus **MAL** = 3
- **Pipeline throughput** = $1/\text{MAL} = 1/3$

Quick reference

Non-linear Pipeline processors

Reservation and Latency analysis.



A three stage pipeline processor.

we have given stage diagram

↓
From state diagram Draw Reservation tables

↓
Find collision vector. (Using Forbidden latency {permissible latency})

↓
Draw state diagrams

↳ From state diagram Find
{ Simple cycles
greedy cycles &
Minimal Average latency.

* Draw Reservation table. for function X.

	1	2	3	4	5	6	7	8
stages S1	X					X		X
S2		X		X				
S3			X		X		X	

* Find forbidden latency.

To find forbidden latency, one need to simply need to check the distance between any two checkmarks in the same row of RT.

$$\begin{aligned}
 \text{forbidden latency} &= \{(6-1), (8-1), (8-6) \\
 &\quad (4-2), (5-3), (7-5), (7-3)\} \\
 &= \{5, 7, 2, 2, 2, 4, 2\} \\
 &= \underline{\underline{\{2, 4, 5, 7\}}}
 \end{aligned}$$

$$\text{Permissible latency} = \{1, 3, 6\}$$

* Collision vector

Collision vector will contain no. of bits = the highest forbidden latency. Here $= 7$.

$$\text{Collision vector} = \{c_7, c_6, c_5, c_4, c_3, c_2, c_1\}$$

$$C_x = \{1, 0, 1, 1, 0, 1, 0\}$$

↳ mark 1 for forbidden latency.

* Draw state diagram

From collision vector, one can construct a state diagram specifying the permissible transitions among successive iterations.

The above collision vector corresponds to the initial state of the pipeline at time 1 and thus is called an initial collision vector.

Let 'P' be a permissible latency, $1 \leq P \leq m-1$
Cm \rightarrow maximum forbidden latency.

\rightarrow Next state of pipeline at time $t+P$ is obtained with the assistance of an m -bit ~~shift~~ right shift register.

\rightarrow Initial collision vector is loaded into the registers.

\rightarrow The register is then right shifted. ~~to the~~.

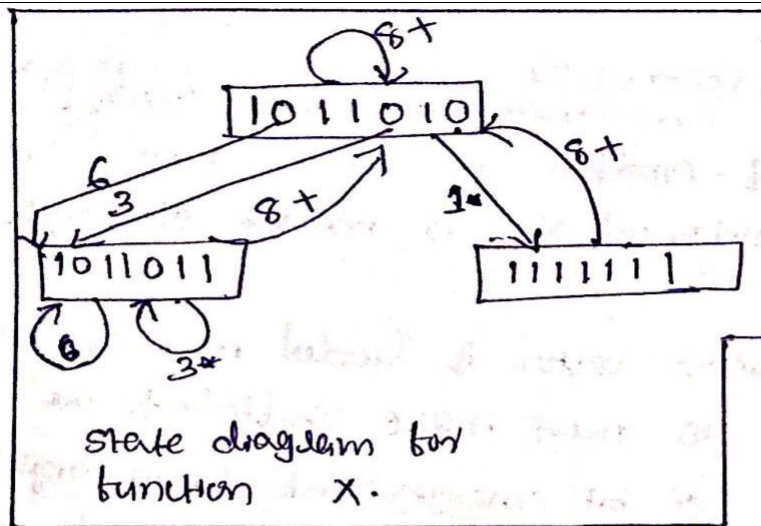
\rightarrow ~~Each~~ when '0' bit emerges out from right, after P shifts, it means 'P' is a permissible latency. So a new state will arise.

\rightarrow when a '1' bit being shifted out means, a collision, and thus the corresponding latency should be forbidden.

\rightarrow The next state after P shifts is obtained by bitwise-ORing the initial collision vector with shifted register contents.

NOTE = when the no. of shifts is $(m+1)$ or greater [i.e. $(m+1)$ or greater], all transitions are redirected back to the initial state. denoted as $g+$.

* minimum latency edges are marked with asterisk (*).



new states will be obtained after 1st, 3rd & 6th shift from initial collision vector

1st shift

$$\begin{array}{r}
 0101101 \oplus \\
 1011010 \\
 \hline
 1111111
 \end{array}$$

3rd shift

$$\begin{array}{r}
 0001011 \oplus \\
 1011010 \\
 \hline
 1011011
 \end{array}$$

6th shift

$$\begin{array}{r}
 0000001 \\
 1011010 \\
 \hline
 1011011
 \end{array}$$

From this state, new states will be obtained after 2nd & 5th shifts.

2nd shift

$$\begin{array}{r}
 0000011 \\
 1011010 \\
 \hline
 1011011
 \end{array}$$

5th shift

$$\begin{array}{r}
 0000001 \\
 1011010 \\
 \hline
 1011011
 \end{array}$$

Simple cycles & greedy cycles

Simple cycle :- A latency cycle, in which each state appears only once.

eg: (3), (6), (8), (1,8), (3,8), (6,8).

Greedy cycle :- one whose edges are all made with minimum latency from their respective starting states.

average latency of greedy cycles must be lower than those of other simple cycles.

List of all simple cycles

cycles	Avg latency.
(3)	3
(6)	6
(8)	8
(1,8)	4.5
(3,8)	5.5
(6,8)	7.

OR
greedy cycles are those in which the avg. latency \leq no. of '1' bits in the initial collision vector.
In one eg:
no. of 1's = 4.

Greedy cycles are those in which the average latency \leq no. of 1's in the initial collision vector + 1

In our example $4+1$

Greedy cycles are :- 3, & (1,8).

MAL :- Either of the two greedy cycles.
or, smallest entry in avg. latency column

ie: 3

i. Pipeline Schedule Optimization

An optimization technique based on the MAL is given below. The idea is to insert noncompute delay stages into the original pipeline. This will modify the reservation table, resulting in a new collision vector and an improved state diagram. The purpose is to yield an optimal latency cycle, which is absolutely the shortest.

Bounds on the MAL: In 1972, Shar determined the following bounds on the minimal average latency (MAL) achievable by any control strategy on a statically reconfigured pipeline executing a given reservation table:

1. The MAL is *lower-Bounded* by the *maximum number of checkmarks* in any row of the reservation table.
2. The MAL is lower than or equal to the average latency of any greedy cycle in the state diagram.
- 3 The average latency of any greedy cycle is *upper-bounded* by the *number of 1's in the initial collision vector plus 1*. This is also an upper bound on the MAL.

- These results suggest that the **optimal latency cycle** must be selected from **one of the lowest greedy cycles**. However, **a greedy cycle is not sufficient to guarantee the optimality of the MAL**.
- **The lower bound guarantees the optimality.** (ie, optimal latency cycle should be equal to maximum number of checkmarks in any row of the reservation table.)
- For example, the MAL = 3 for both function X and function Y and has met the lower bound of 3 from their respective reservation tables.
- From the above figure b, the upper bound on the MAL for function X is equal to $4+1=5$, a rather loose bound. On the other hand above fig C shows a rather tight upper bound of $2+1=3$ on the MAL. therefore all greedy cycles for function Y leads to optimal latency value of 3, which cannot be lowered further.

To *optimize the MAL*, one needs to find the lower bound by modifying the reservation table. The approach is to reduce the maximum number of checkmarks in any row. The modified reservation table must preserve the original function being evaluated. Patel and Davidson (1976) have suggested the insertion of **non compute delay stages to increase pipeline performance with shorter MAL**.

Delay Insertion

The purpose of delay insertion is to modify the reservation table, yielding a new collision vector. This leads to a modified state diagram, which may produce **greedy cycles meeting lower bound on the MAL**.

The below reservation table corresponds to a collision vector $C=(1011)$, corresponding to forbidden latencies 1, 2 and 4. The corresponding state diagram fig C contains only one self-reflecting state with a greedy cycle of latency 3 equal to the MAL. Based on the given reservation table, the maximum number of *checkmarks in any row is 2*. Therefore, the **MAL = 3** so obtained in Fig. C is **not optimal**.

Inserting noncompute delay to reduce MAL

To insert a non compute stage D1 after stage S3 will delay both X1 and X2 operations one cycle beyond time 4. To insert another non compute stage D2 after the second usage of S1 will delay the operation X2 by another cycle.

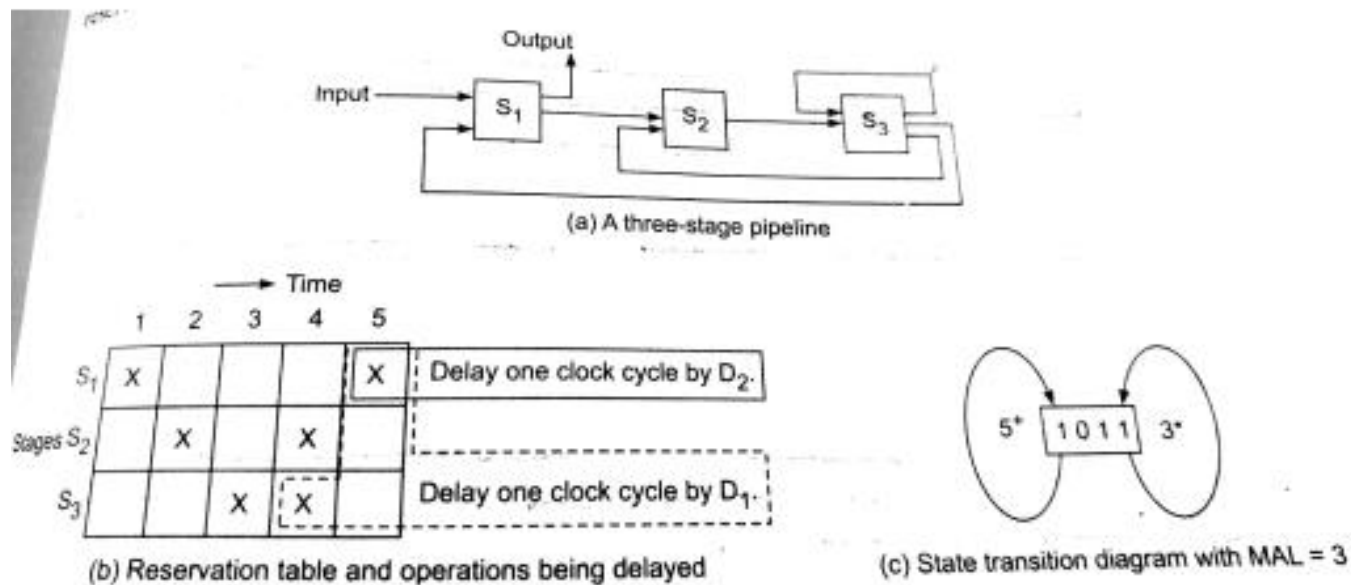


Fig. 6.7 A pipeline with a minimum average latency of 3

These delay operations as grouped in the fig6.7b result in a new pipeline configuration in the following figure 6.8a. Both delay elements D_1 and D_2 are inserted as extra stages as shown in fig 6.8b with an enlarged reservation table having $3+2 = 5$ rows and $5+2=7$ columns.

In total, the operation **X1** has been delayed one cycle from time 4 to time 5 and the operation **X2** has been delayed two cycles from time 5 to time 7. All remaining operations (marked as X in Fig 6.8b) are unchanged. This new table leads to a new collision vector (100010) and a modified state diagram in fig 6.8c.

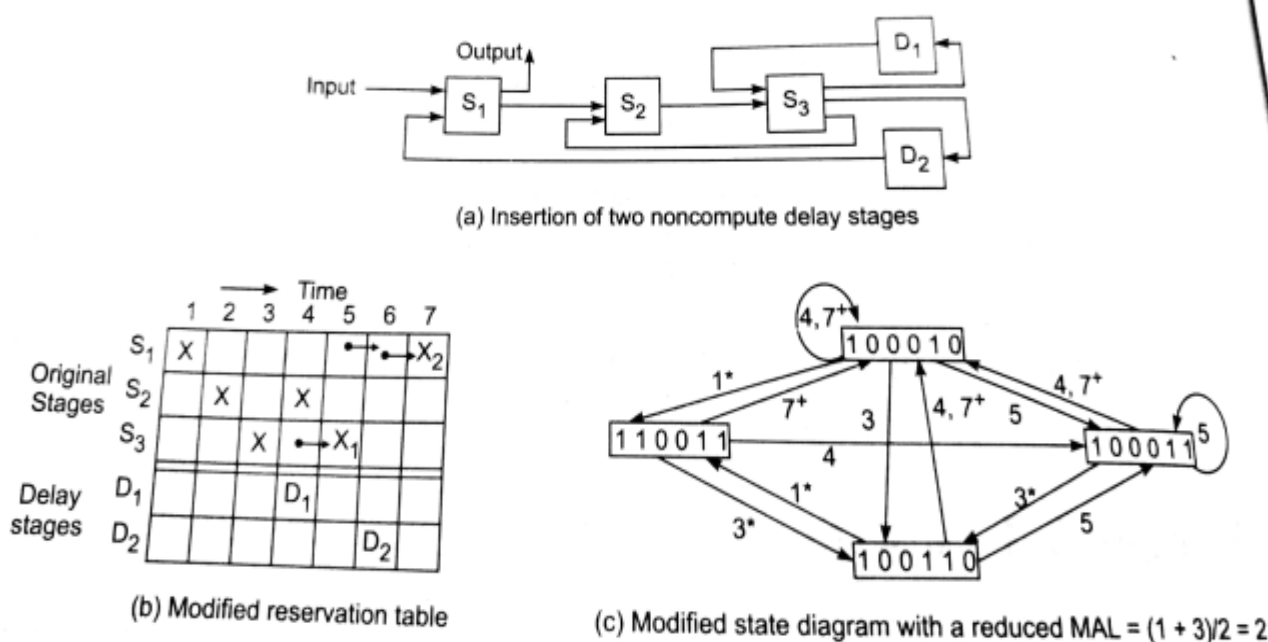


Fig. 6.8 Insertion of two delay stages to obtain an optimal MAL for the pipeline in Fig. 6.7

This diagram displays a **greedy cycle (1,3) resulting in a reduced $MAL=(1+3)/2=2$** . The delay insertion thus improves the pipeline performance, yielding a lower bound for the MAL.

Pipeline Throughput

This is the **initiation rate or the average number of task initiations per clock cycle**. If N tasks are initiated within n pipeline cycles, then the initiation rate or pipeline throughput is measured as N/n . This rate is determined primarily by the *inverse of the MAL adapted*. Therefore, the scheduling strategy does affect the pipeline performance.

In general, the *shorter the adapted MAL, the higher the throughput that can be expected*. The highest achievable throughput is one task initiated per cycle, when the MAL equals 1 since $1 \leq MAL \leq$ the shortest latency of any greedy cycle. Unless the MAL is reduced to 1, the pipeline throughput becomes a fraction.

Pipeline Efficiency

It is the percentage of time that each pipeline stage is used over a sufficiently long series of task initiations is the stage utilization. The accumulated rate of all stage utilizations determines the pipeline efficiency.

Consider the latency cycle (3) in fig 6.5 b. Within each latency cycle of three clock cycles, there are two pipeline stages S1 and S3, which are completely and continuously utilized after time 6. The pipeline stage S2 is used for two cycles and is idle for one cycle.

Therefore entire pipeline can be considered $8/9=88.8\%$ efficient for latency cycle(3). On the other hand, the pipeline is only $14/27 = 51.8\%$ efficient for a latency cycle (1,8) and $8/16 = 50\%$ efficient for latency cycle(6) as illustrated in fig 6.5 a and 6.5c respectively.

The pipeline throughput and pipeline efficiency are related to each other. Higher throughput results from a shorter latency cycle. Higher efficiency implies less idle time for pipeline stages.

At least one stage of the pipeline should be fully (100%) utilized at the steady state in any acceptable initiation cycle. Otherwise pipeline capability has not been fully explored. In such cases, the initiation cycle may not be optimal and another initiation cycle should be examined for improvement.

Additional questions and answers

1.

Problem 6.5 – Prove the lower bound and upper bound on the minimal average latency (MAL)

Answer :

[Shar72] states the MAL as followed :

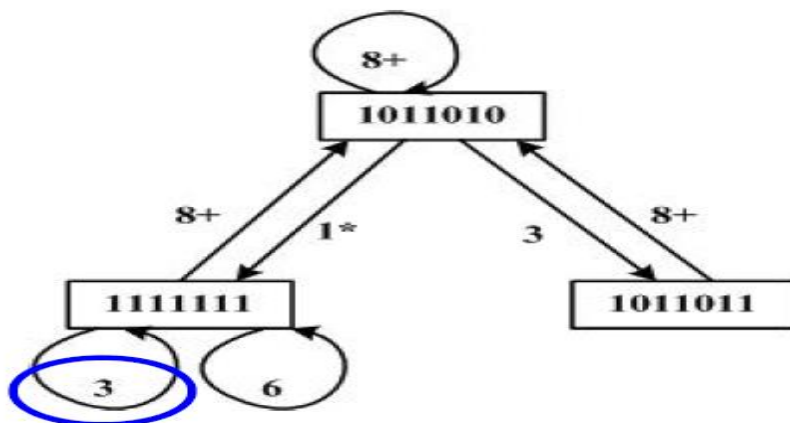
1. The **MAL is lower-bounded by the maximum number of checkmarks in any row of the reservation table.**
2. The average latency of any greedy cycle is upper-bounded by **the number of 1's in the initial collision vector plus 1**. This is also an **upper bound on the MAL**.

Proof :

Take the case on the function X's reservation table on page 271 as followed :

	1	2	3	4	5	6	7	8
S1	X					X		X
S2		X		X				
S3			X		X		X	

From the table we see that S1 has 3 checkmarks, S2 has 2 checkmarks and S3 has 3 checkmarks which means that the minimum MAL for function X is 3. The forbidden latencies are 2, 4, 5 and 7 while the permissible latencies are 1, 3 and 6. We, then, can obtain the collision vector $C_X = 1011010$. Let's shift this vector bit-by-bit to the right to find the state transition diagram.



- The **simple cycles** are (3), (6), (1,8) and (3,8), such that the **greedy cycles** are (3) and (1,8). The lowest greedy cycle is the **MAL** and in this case is (3), so the **MAL is 3**..... (*1st statement proved*).
- The **collision vector** $C_X = 1011010$ has 4 1-bits and the maximum MAL is the number of this 1-bit plus 1 or equal to 5. The largest average latency of any greedy cycle on function X's state diagram is **3.5** which is taken from greedy cycle (1,8) (*2nd statement proved*).

2.

Problem 6.6 – Consider the following reservation table for a **four-stage pipeline** with a **clock cycle** $\tau = 20$ ns.

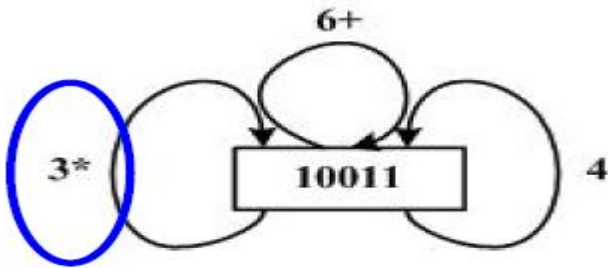
	1	2	3	4	5	6
S1	X					X
S2		X		X		
S3			X			
S4				X	X	

- What are the forbidden latency and the initial collision vector ?
- Draw the state transition diagram for the scheduling the pipeline.
- Determine the MAL associated with the shortest greedy cycle.
- Determine the pipeline throughput corresponding to the MAL and given τ .
- Determine the lower bound on the MAL for this pipeline.

Answer :

a. The **forbidden latencies** are 1, 2, and 5 ($S_1 : 5; S_2 : 2; S_3 : 0; S_4 : 1$) and the **collision vector** is, $C_x = C_5 C_4 C_3 C_2 C_1 = 10011$. The **permissible latencies** are 3 and 4.

b. State diagram can be obtained by tracing each C_x shift as followed :



c. The **greedy cycle** is 3 and so is the **MAL**.

d. The pipeline **throughput** according to :

1) The MAL is $\frac{1}{3} = 0.33$

2) The τ is $\frac{1}{2} = 0.5$ (at clock cycle 4 there are two task initiated)

NOTE: If N tasks are initiated within n pipeline cycles, then the initiation rate or pipeline throughput is measured as N/n . This rate is determined primarily by the inverse of the MAL adapted.

e. The MAL's lower bound of this function according to [Shar72] is **2** (the maximum number of checkmarks in any row). The optimal latency have not been found – needs a modification on the reservation table.

3.

Problem 6.7 – You are allowed to insert one noncompute delay stage into the pipeline in Problem 6.6 to make latency of 1 permissible in the shortest greedy cycle. The purpose is to yield a new reservation table leading to an optimal latency equal to the lower bound.

- Show the modified reservation table with five rows and seven columns.
- Draw the new state transition diagram for obtaining the optimal cycle.
- List all the simple cycles and greedy cycles from the state diagram.
- Prove that the new MAL equals to the lower bound.
- What is the optimal throughput of this pipeline ? Indicate the percentage of throughput improvement compared with that obtained in part (d) of Problem 6.6.

	1	2	3	4	5	6
S1	X					X
S2		X		X		
S3			X			
S4				X	X	

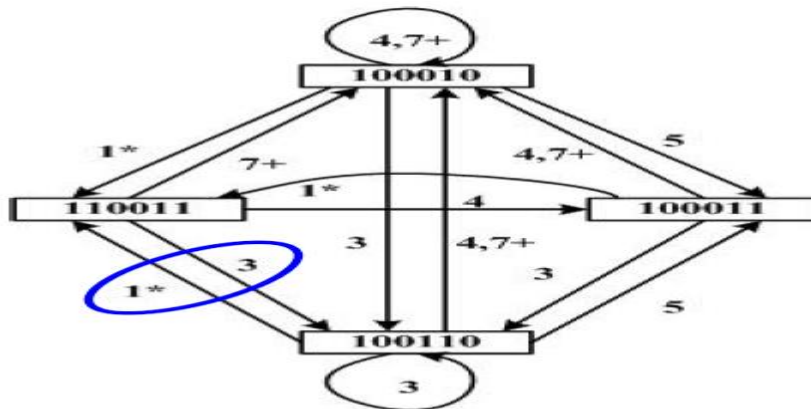
Reservation table-prob 6.6

a. The modified reservation table is :

	1	2	3	4	5	6	7
S1	X						X
S2		X		X			
S3			X				
S4				X		X	
D					D ₁		

The **forbidden latencies** are 2 and 6, then the **collision vector** is $C_x = 100010$. The **permissible latencies** are 1, 3, 4, and 5.

b. State diagram can be obtained by tracing each C_x shift as followed :



The state diagram from the modified reservation table.

- The **simple cycles** are (3), (4), (5), (1,3), (1,4), (1,7), (3,4), (3,5), (3,7), (4,5), (5,7), (1,3,4), (1,3,7), (1,4,4), (1,4,7), (3,5,4), (3,5,7), and (5,1,7). The **greedy cycles** are (3) and (1,3).
- The greedy cycle (1,3) has the lowest average latency which is equal to 2. This greedy cycle leads to the MAL of this pipeline machine. It can be seen on the reservation table that **this MAL is equal to the maximum number of checkmarks in any row in the reservation table (proved).**

e. The **throughput** is $\frac{1}{MAL} = \frac{1}{2} = 0.5$ and it has improvement percentage of $\frac{0.5}{0.33} = 1,52 \times 100\% = 152\%$.

4.

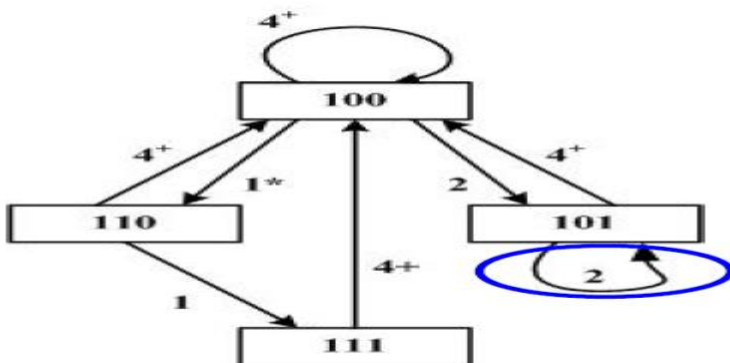
Problem 6.9 – Consider the following **pipeline** reservation table :

	1	2	3	4
S1	X			X
S2		X		
S3			X	

- What are the forbidden latency ?
- Draw the state transition diagram.
- List all the simple cycles and greedy cycles.
- Determine the optimal constant latency cycle and the minimal average latency (MAL).
- Let the pipeline clock period be $\tau = 20$ ns. Determine the throughput of this pipeline.

Answer :

- The **forbidden latency** is 3. From the table we can obtain the **collision vector**, $C_x = C_3 C_2 C_1 = 100$. The **permissible latencies** are 1 and 2.
- State diagram can be obtained by tracing each C_x shift as followed :



- The **simple cycles** are (2), (4), (1,4), (2,4), and (1,1,4). The **greedy cycles** are (2) and (1,4)
- The **optimal constant latency** is (2), which is equal to MAL.

e. The maximum **throughput** is for this is liner pipeline is :

$$H_k = \frac{n}{[k + (n - 1)] \tau}$$

$$H_3 = \frac{2}{[3 + (2 - 1)] 20 \cdot 10^{-9}}$$

$$= \frac{1}{40 \cdot 10^{-9}}$$

$$= 25 \text{ MIPS}$$

5;

Problem 6.10 – Consider the **five-staged pipelined** processor specified by the following reservation table:

	1	2	3	4	5	6
S1	X					X
S2		X			X	
S3			X			
S4				X		
S5		X				X

- List the set of forbidden latencies and the collision vector.
- Draw the state transition diagram showing all possible initial sequence (cycles) without causing a collision in the pipeline.
- List all the simple cycles from the state diagram.
- Identify the greedy cycles among the simple cycles.
- What is the MAL of this pipeline ?
- What is the minimum allowed constant cycle in using this pipeline ?
- What will be the maximum throughput of this pipeline ?
- What will be the throughput if the minimum constant cycle is used ?

Answer :

- The **forbidden latencies** are 3, 4, and 5 (S1 : 5; S2 : 3; S3 : 0; S4 : 0 and S5 : 4), so that the **collision vector** is $C_X = 11100$ where the **permissible latencies** are 1 and 2.
- State diagram can be obtained by tracing each C_X shift as followed :
- The **simple cycles** are (2), (6), (1,6), and (2,6).
- The **greedy cycles** are (2) and (1,6).
- According to the lowest greedy cycle's average latency, the **MAL** is 2.
- The **greedy cycle** is also the constant cycle which is equal to the MAL.
- The maximum throughput is $\frac{1}{MAL} = \frac{1}{2} = 0.5$ or only **50%**.
- The **minimum constant cycle** is 2, so that the he maximum throughput does not change, only **50%**