Model Answers for Class Test 1
CSEN 3104
held on
12/09/2019

Dr. Debranjan Sarkar

Question No. 1

- Latencies can be {1, 2, 3, 4, 5}
- To find out Forbidden Latency
  - S1: 6 – 1 = 5 (diff. between entries in S
  - S2: 5 – 2 = 3
  - S3: Nil
  - S4: Nil
  - S5: 6 – 2 = 4

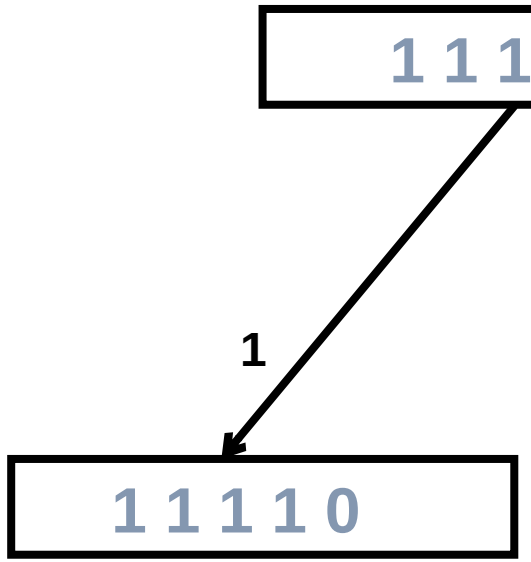
	1	2	3	4	5	6
S1	X					X
S2		X			X	
S3			X			
S4				X		
S5		X				X

- So Forbidden Latencies = {3, 4, 5}
- Permission Latencies = {1,2} [other than the forbidden latencies]
- Initial Collision Vector: $C_5C_4C_3C_2C_1$

  1 1 1 0 0 [Bit positions corresponding to permissible latencies are 0 and those corresponding to forbidden latencies are 1]

(b) State Diagram (step 1)

State Diagram (step 2)



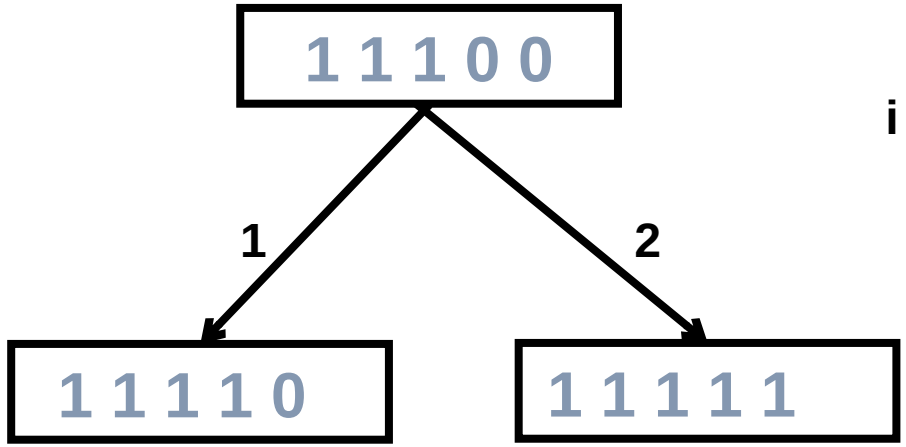
i = 1

ICV – 11100 OR

CV_i – 01110

CV* 11110

State Diagram (step 3)



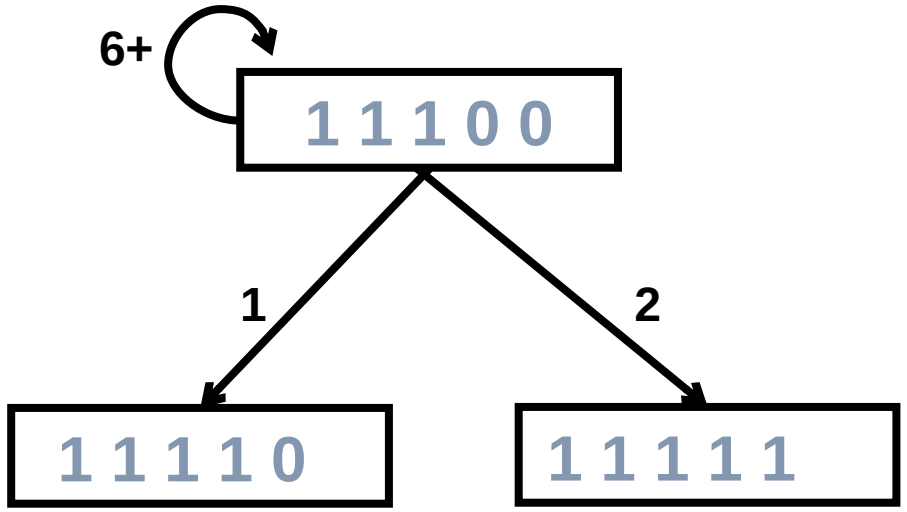
i = 2

ICV – 11100 OR

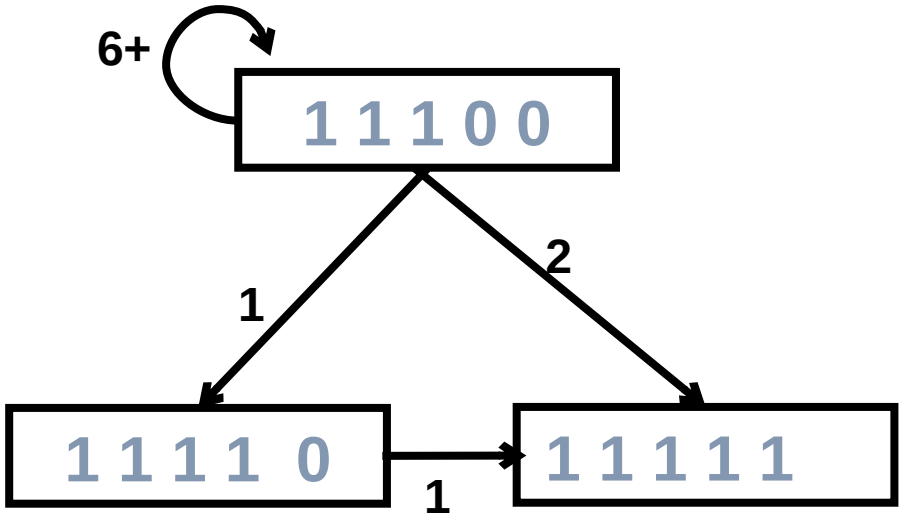
CV_i – 00111

CV* 11111

State Diagram (step 4)



State Diagram (step 5)



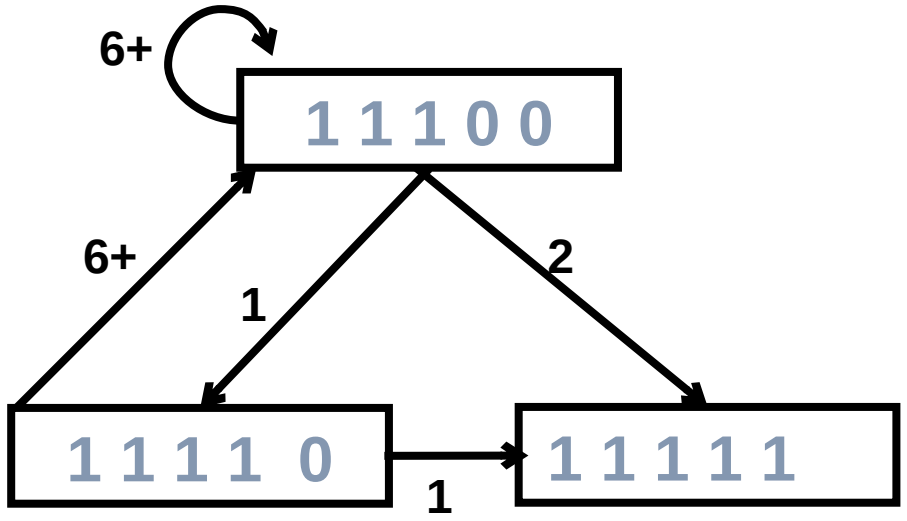
i =1

ICV – 11110 OR

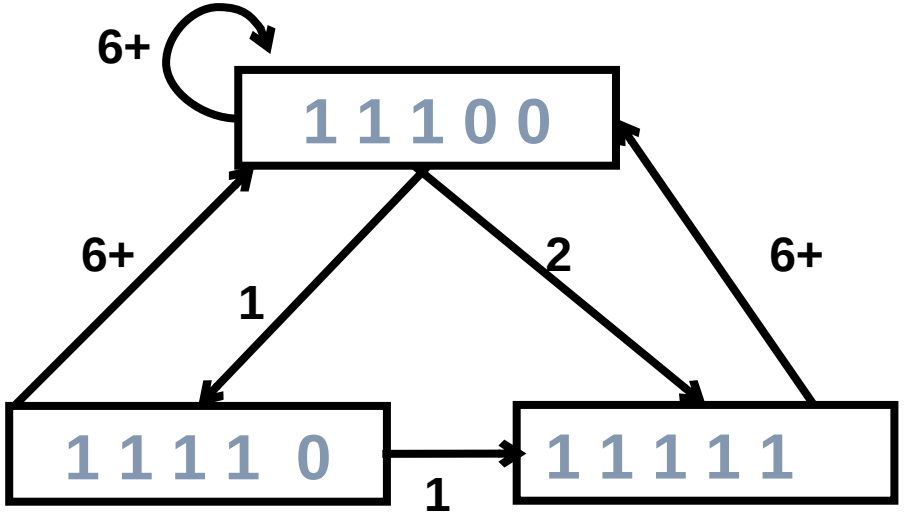
CV_i – <u>01111</u>

CV* 11111

State Diagram (step 6)



State Diagram (step 7)



This is the required State Diagram

Question 1

- (c) Simple Cycles: latency cycles in which each state is encountered only once. Here it is (6), (1,6), (2,6), (1,1,6)
- To point out the Greedy cycles, we find out
  - The minimum latency outward from the state 11100 is 1
  - The minimum latency outward from the state 11110 is 1
  - The minimum latency outward from the state 11111 is 6
- So, there is only one greedy cycle, given by (1,1,6)
- (d) Minimum average latency (MAL) = minimum of the average latencies of all the simple cycles.
- Here, the average latencies of the simple cycles are respectively:
- 6/1 = 6; (1+6)/2 = 3.5, (2+6)/2 = 4; (1+1+6)/3 = 2.67
- So, the minimum average latency (MAL) is 2.67
- The GC corresponding to this MAL is (1,1,6)

Question 1

- (e) Lower bound on the MAL is the maximum number of checkmarks in any row of the Reservation Table
- Here number of checkmarks in the 1st row: 2

  2nd row: 2

  3rd row: 1

  4th row: 1

  5th row: 2
- So, maximum number of checkmarks is 2
- Lower bound on the MAL = 2
- Upper bound of MAL = the number of 1's in the initial collision vector + 1

  = 3 + 1 = 4
- Upper bound on the MAL = 4

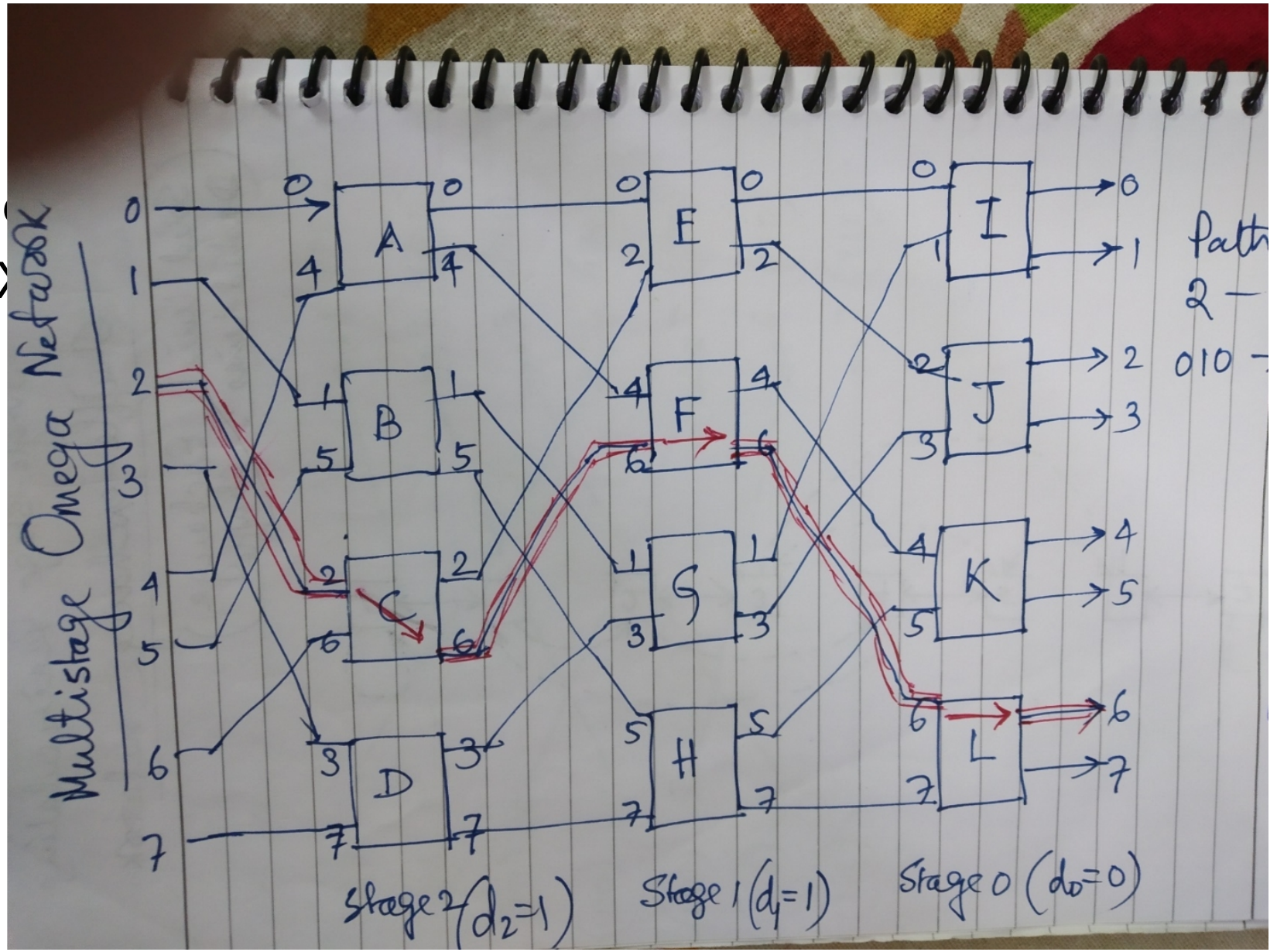
Question 2(a)

- Omega network is a type of Shuffle Exchange Network
- Let A be the address of a Processing Element, given by

  $A = a_{n-1}a_{n-2}.....a_2a_1a_0$
- Shuffle Exchange Network is based on 2 routing functions
  - Shuffle, given by $S(A) = a_{n-2}.....a_2a_1a_0a_{n-1}$
  - Exchange, given by $E(A) = a_{n-1}a_{n-2}.....a_2a_1a'_0$
- For N = 8, $n = \log_2 N = 3$, the shuffle-exchange function may be implemented with a multi-stage Omega Network shown in the following
- There is a perfect shuffle interconnection between 2 adjacent stages
- Each stage has N/2 = 4 switch boxes under independent box control
- Each box has 4 functions viz. straight, exchange, upper broadcast and lower broadcast

Question 2(

Diagram of 8 X 8 Om
Network built with 2 X
switching elements

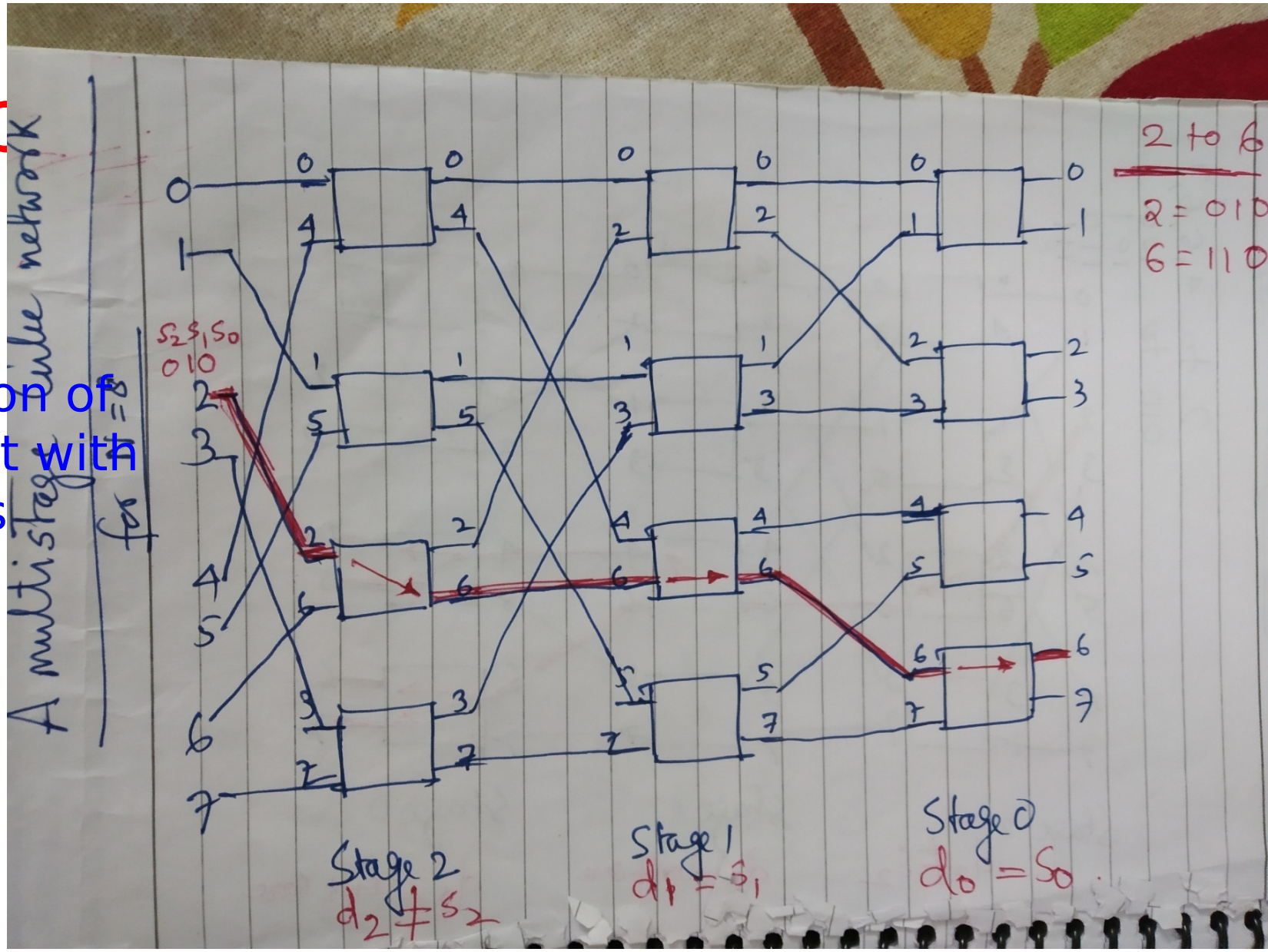


Question 2(b)

- Let us consider a cube network with N = 8 numbers of processing elements (PE)
- Let A be the address of a Processing Element, given by A = $a_2a_1a_0$
- There are n = 3 routing functions, as given below:
  - $C_0(A) = a_2a_1a'_0$
  - $C_1(A) = a_2a'_1a_0$
  - $C_2(A) = a'_2a_1a_0$
- A cube network may be implemented as a multistage network as shown below
- There will be n = $\log_2 N$ = 3 numbers of stages
- In each stage, there are 4 numbers of two-function (straight and exchange) switch boxes
- Stages are numbered as 0 at the input end and 2 at the output end
- Stage 0 implements the routing function C_0, Stage 1 implements the routing function C_1 and Stage 2 implements the routing function C_2
- That means, switch boxes at stage 0 connect an input line to the output line that differs from it only at the 0^{th} bit position and so on.
- Because of this interconnection requirement, individual box control is assumed in a multi-stage cube network

Question 2(b

Multistage implementation of
8 X 8 Cube Network built with
2 X 2 switching elements



3(a) Branch Prediction

- Refer to Lecture 8 dated 23/07/2019
- We know Branch Prediction is one of the effective methods to mitigate Control Hazards in the Pipeline
- Control Hazard occurs in a pipeline because there are conditional branch instructions in the program
- Until the execution of the conditional branch instruction is complete, one cannot surely tell whether the branch would be taken or not (i.e. whether the condition would be satisfied or not)
- Branch Prediction technique uses some additional logic (heuristic) to predict the outcome of a conditional branch instruction before it is executed

Techniques to mitigate Control Hazards: Branch Prediction

- Normally two strategies are followed
  - Static Branch Strategy: Probability of branch
  - Dynamic Branch Strategy: Branch history is taken into consideration
- Instructions are speculatively fetched and executed down the predicted path
- But results are not written back to the register file until the branch is executed and the prediction is verified
- When a branch is predicted, the processor enters a *speculative mode* in which results are written to another register file that mirrors the architected register file
- Another pipeline stage called the *commit* stage is introduced to handle writing verified speculatively obtained results back into the "real" register file
- Branch predictors cannot be 100% accurate
- So there is still a penalty for branches if the prediction is found to be incorrect

3(b) OR Strip Mining

- Refer to Lecture 12 dated 01/08/2019
- In Vector Processing, vector length (L_V) may not be equal to the length of the vector registers (L_{VR}).
- If $L_V < L_{VR}$, a vector length register (VLR) may be used to control the length of any vector operation
- If $L_V > L_{VR}$, we can split the long vector into multiple vectors so that each vector operation is done for a size $\leq$ the maximum vector length (MVL) i.e. the length of the vector registers
- The process generating code by splitting the long vector into multiple vectors as above, is called strip-mining
- Length of any vector operation cannot be > the length of vector registers (L_{VR})

```
for (i=0;i<n;i++)
    y[i] = a * x[i] + y[i];
```

- As the value of n is known only during runtime, strip mining technique tackles

Example of Strip Mining

- Let n = 135, MVL = 64
- 1st loop iterates for (n mod MVL) = 7 times
- 2nd and 3rd loop iterates for MVL = 64 times each
- There is a loop overhead

```
    low = 0;
vl = n mod mvl;   /*find the odd size piece*/
for(j = 0; j <= n/mvl; j++)  /*outer loop*/
{
for(i = low; i < low+vl; i++)    /*runs for length VL*/
{
y[i] = a*x[i] + y[i];   /*main operation*/
}
low = low + vl;/*start of next vector*/
vl = mvl;  /*reset the length to max*/
}
```

3(b) OR Vector Chaining

- Refer to Lecture 12 dated 01/08/2019
- Data Forwarding is a technique to mitigate the Data Hazard in pipeline
- The data obtained at the Execution phase of the first instruction may be forwarded to the operand fetch unit of the 2nd instruction
- Instruction I: ADD R1, R2, R3
- Instruction J: SUB R4, R1, R6

	1	2	3	4	5	6	7
IF	I	J					
ID		ADD	SUB				
OF			R_2, R_3	Delay	$R_{2+}R_3$, R_6		
EX				R_2+R_3		$R_{2+}R_3$- R_6	
WB					Res to		

Delay Reduced to 1 cycle
By Data Forwarding

Vector Chaining

- Vector Chaining allows the results of one vector operation to be directly used as input to another vector operation
- It is the equivalent to Data forwarding in pipelined processors
- Vector Chaining is used in case of data dependency among vector instructions.
- Let us take the example of a simple vector sequence as follows:

```
MULTV V1,V2,V3
ADDV   V4,V1,V5
```

- As the ADDV instruction is dependent on the MULTV instruction, these two instructions are to be put into two separate convoys

Vector Chaining

- A convoy is a set of vector instructions that can potentially execute together.
- If the vector register, V1 in this case, is treated not as a single entity but as a group of individual registers, then the ideas of forwarding can be conceptually extended to work on individual elements of a vector.
- This insight, which will allow the ADDV instruction to start earlier in this example, is called chaining
- Chaining allows a vector operation to start as soon as the individual elements of its vector source operand become available
- The results from the first functional unit in the chain are “forwarded” to the second functional unit

Thank you