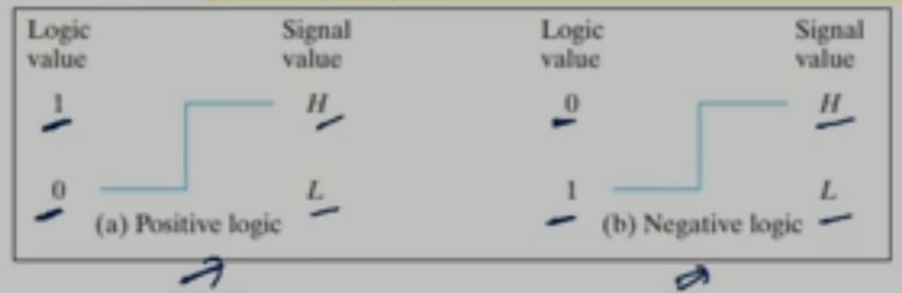
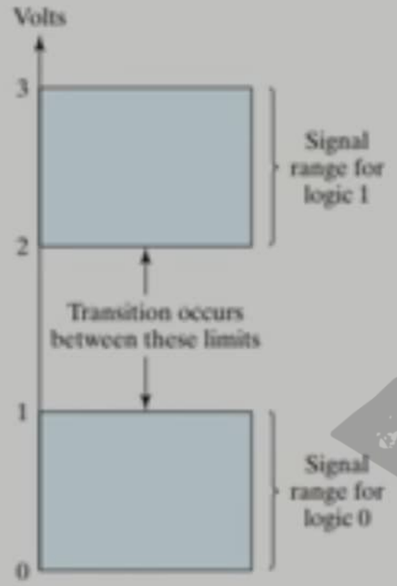
		Module - 1	M	L	C
Q.1	a	Demonstrate the positive and negative logic using AND gate	5	L1	CO1



- The binary signal at the inputs and outputs of any gate has one of two values, except during transition.
- One signal value represents logic 1 and the other logic 0.
- Choosing the high-level H to represent logic 1 defines a positive logic system.
- Choosing the low-level L to represent logic 1 defines a negative logic system.



x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

(c) Truth table for positive logic

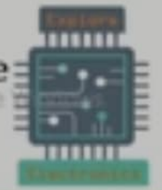
(d) Positive logic AND gate

x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

(e) Truth table for negative logic

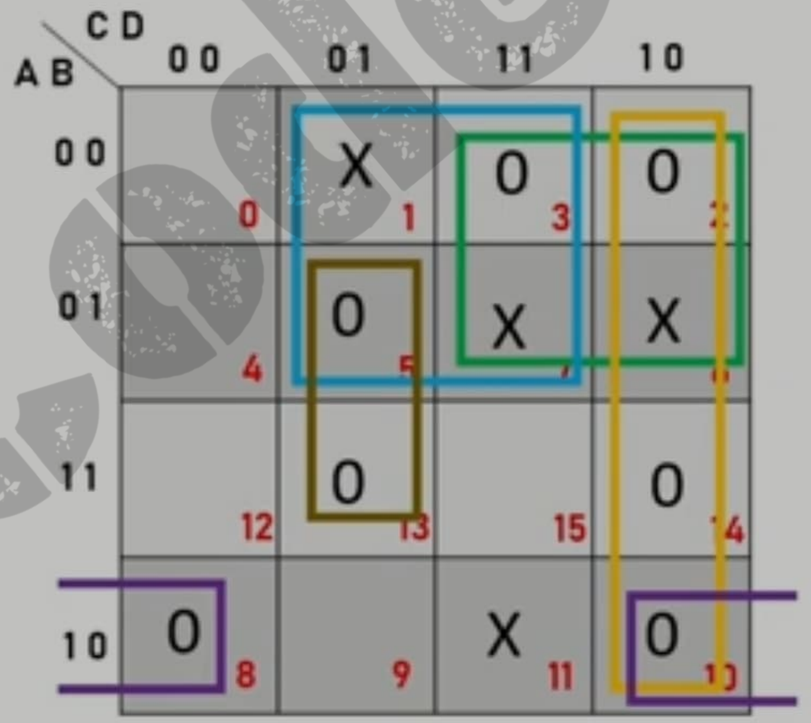
(f) Negative logic OR gate

The conversion from positive logic to negative logic and vice versa is essentially an operation that changes 1's to 0's and 0's to 1's in both the inputs and the output of a gate. Since this operation produces the dual of a function, the change of all terminals from one polarity to the other results in taking the dual of the function. The upshot is that all AND operations are converted to OR operations (or graphic symbols) and vice versa



b	Find the POS expression for F(a,b,c,d) = Π(2,3,5,8,10,13,14) + d(1,6,7,11) and realize it using NOR gates.	5	L3	CO1

$$F(a,b,c,d) = \Pi(2,3,5,8,10,13,14) + d(1,6,7,11)$$



Blue: (A + D')
Green: (A + C')
Orange : (C' + D)
Brown : (B' + C + D`)
Purple : (A` + B + D)

F= (A + D') (A + C') (C' + D) (B' + C + D') (A' + B + D)

$(A+\bar{D})(A+\bar{C})(C+\bar{D})(\bar{B}+C+\bar{D})(\bar{A}+B+D)$

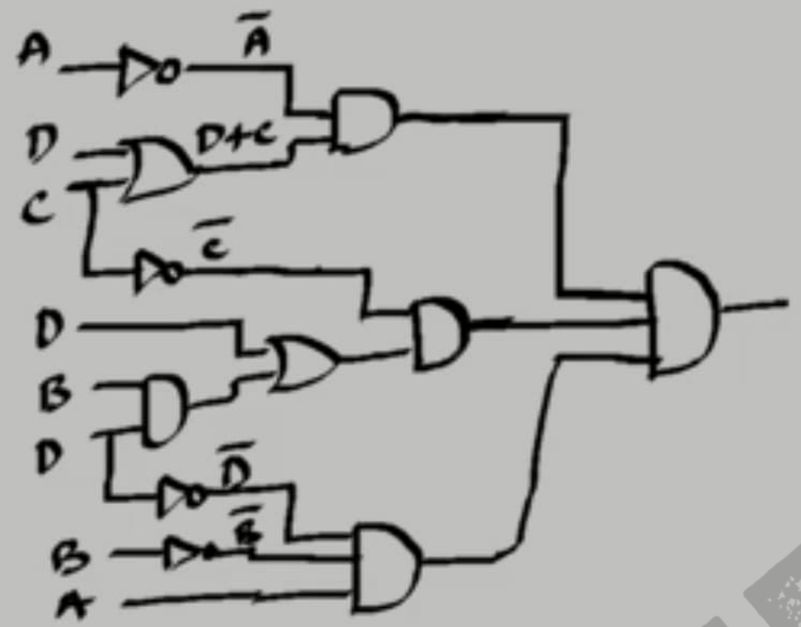
$$\Rightarrow \overline{\overline{(A+\bar{D})(A+\bar{C})(C+\bar{D})(\bar{B}+C+\bar{D})(\bar{A}+B+D)}}$$

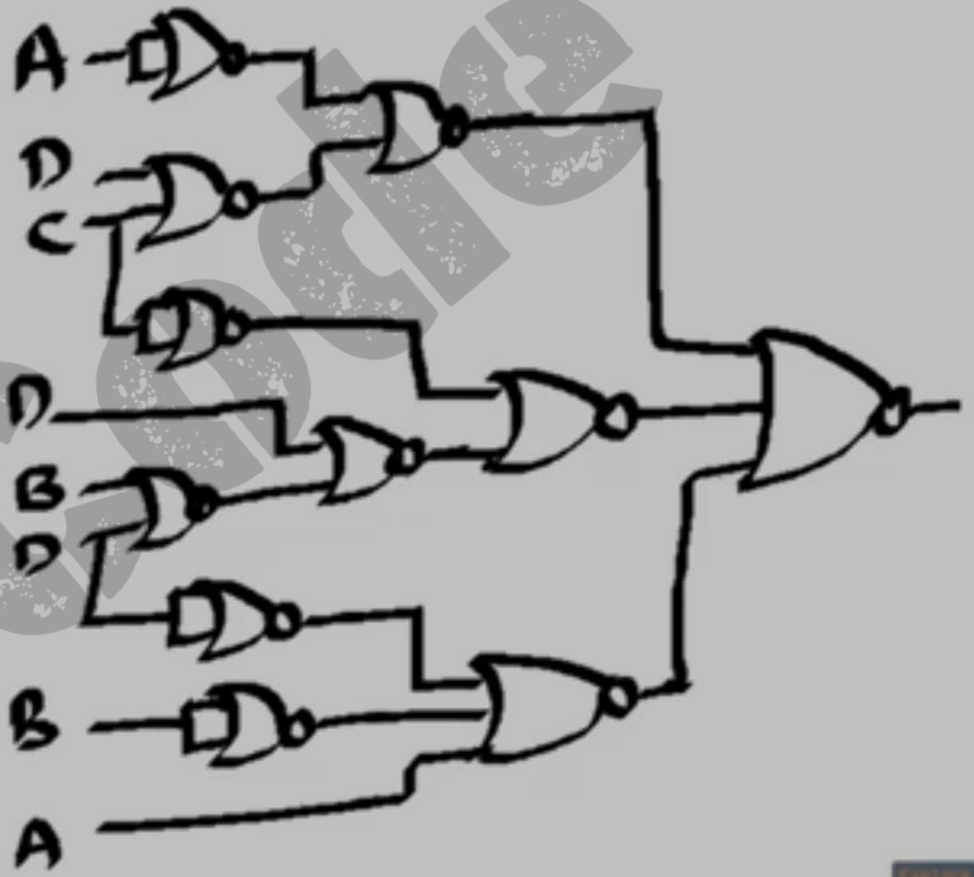
Demorgan

$$\Rightarrow \overline{(A+\bar{D})} + \overline{(A+\bar{C})} + \overline{(C+\bar{D})} + \overline{(\bar{B}+C+\bar{D})} + \overline{(\bar{A}+B+D)}$$

$$\Rightarrow \bar{A}\cdot D + \bar{A}\cdot C + \bar{C}\cdot D + B\cdot\bar{C}\cdot D + A\cdot\bar{B}\cdot\bar{D}$$

$$\Rightarrow \bar{A}(D+C) + \bar{C}(D+BD) + A\bar{B}\bar{D}$$

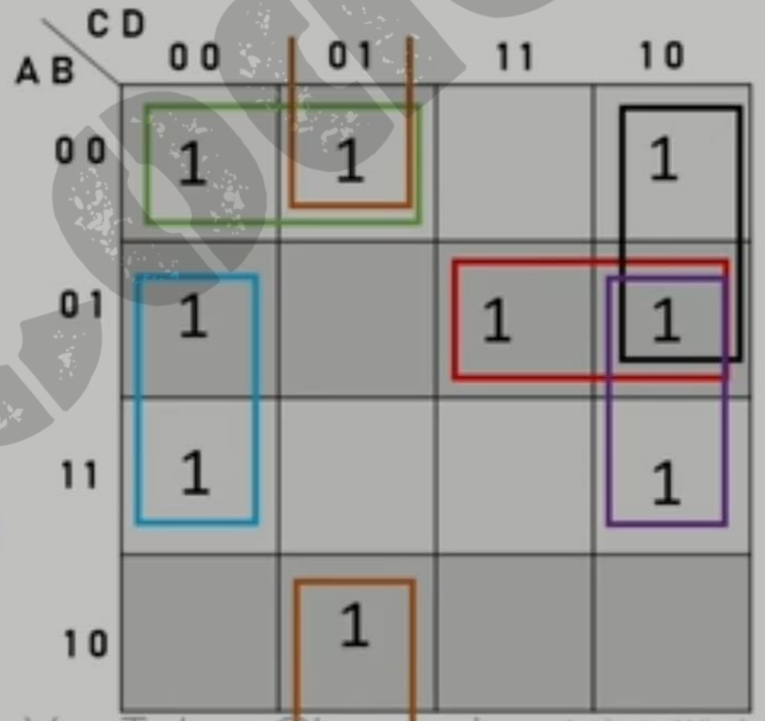




c	Simplify the Boolean function, F(w,x,y,z) = ∑(0,1,2,4,6,7,9,12,14) using k-map and Write the Verilog Program for realizing the minimized expression.	10	L3	CO1

$(x\bar{y}\bar{z}) + (\bar{w}\bar{x}\bar{y}) + (\bar{x}\bar{y}z) +$

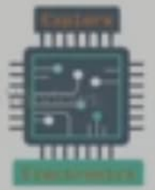
$(\bar{w}xy) + (\bar{w}x\bar{y}) + (xy\bar{z})$



$(x\bar{y}\bar{z}) + (\bar{w}\bar{x}\bar{y}) + (\bar{x}\bar{y}z) + (\bar{w}xy) + (\bar{w}x\bar{y}) + (xyz)$

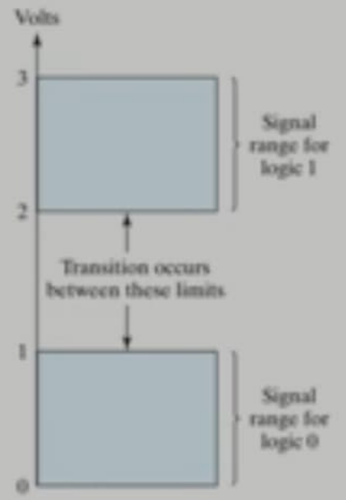
⇒

```
module code (f, w, x, y, z);
input w, x, y, z;
output f;
assign f = (x & ~y & ~z) | (~w & ~x & ~y) |
           (~x & ~y & z) | (~w & x & y) | (~w & x & ~y) |
           (x & y & ~z);
endmodule
```

Q.2	a	What is Binary logic? List out any 4 Laws of Logic.	5	L1	CO1

Binary logic is a system that uses only two digits, 1 and 0, to represent the operation of binary logic gates.



Commutative Law states that the interchanging of the order of operands in a Boolean equation does not change its result. For example:

1. OR operator → A + B = B + A
2. AND operator → A * B = B * A

Associative Law of multiplication states that the AND operation are done on two or more than two variables. For example:

A * (B * C) = (A * B) * C

Distributive Law states that the multiplication of two variables and adding the result with a variable will result in the same value as multiplication of addition of the variable with individual variables. For example:

A + BC = (A + B) (A + C).

Idempotent law:

A + A = A

A.A = A

Identity law:

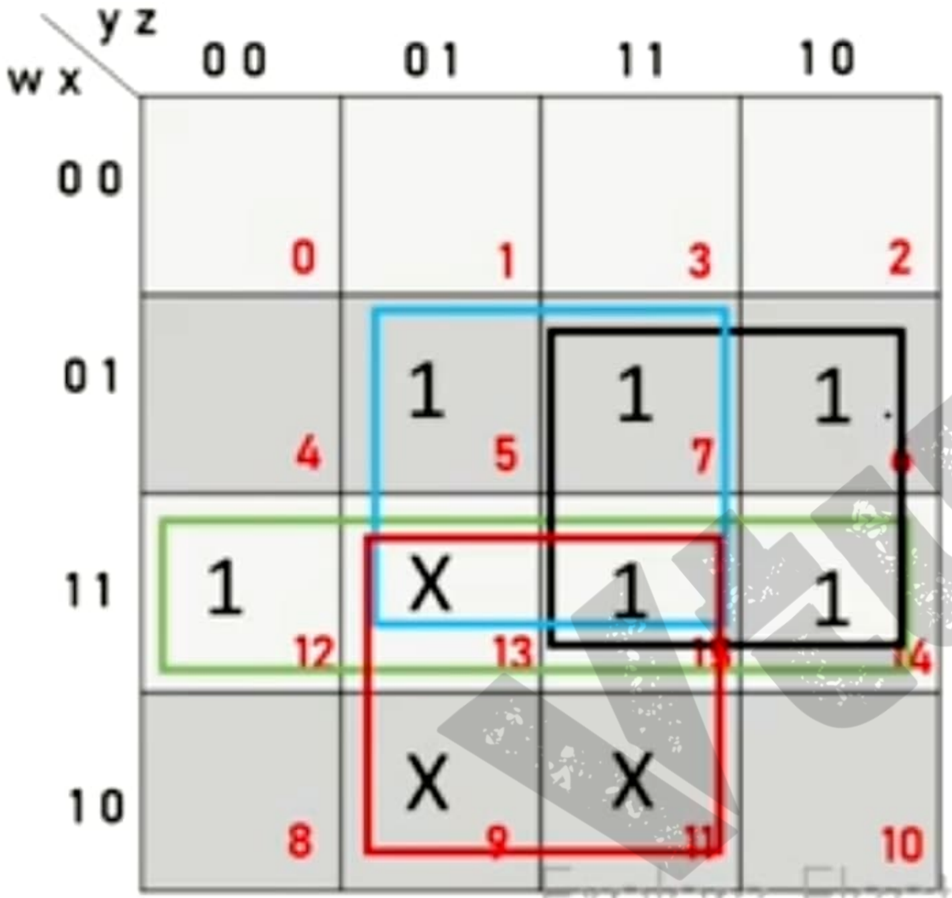
A.1 = A

A + 0 = A

b	Simplify the following Boolean function and find its SOP: i) $F(x,y,z) = \sum(0,1,4,5,6) + d(2,3,7)$ ii) $F(w,x,y,z) = \sum(5,6,7,12,14,15) + d(13,9,11)$	**10**	**L3**	**CO1**

F = 1

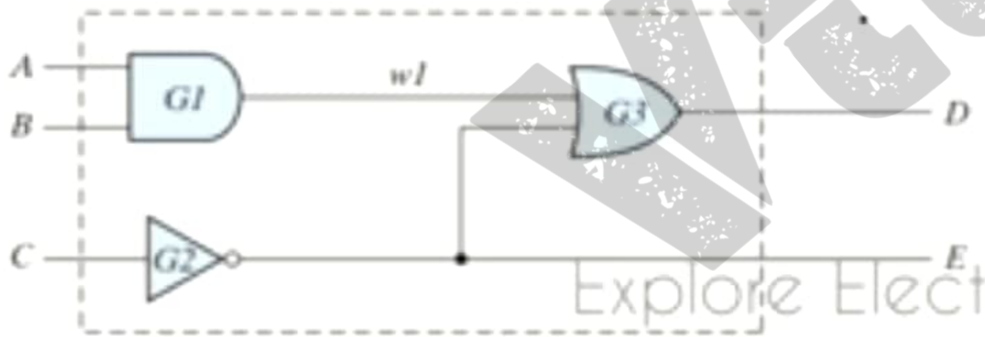
b	**Simplify the following Boolean function and find its SOP:** i) $F(x,y,z) = \sum(0,1,4,5,6) + d(2,3,7)$ ii) $F(w,x,y,z) = \sum(5,6,7,12,14,15) + d(13,9,11)$	**10**	**L3**	**CO1**



$F = (wx) + (xz) + (wz) + (xy)$

c	Write a short note on Hardware Description Language.	5	L1	CO1

- A hardware description language (HDL) is a computer-based language that describes the hardware of digital systems in a textual form.
- It resembles an ordinary computer programming language, such as C, but is specifically oriented to describing hardware structures and the behavior of logic circuits.
- It can be used to represent logic diagrams, truth tables, Boolean expressions, and complex abstractions of the behavior of a digital system.
- One way to view an HDL is to observe that it describes a relationship between signals that are the inputs to a circuit and the signals that are the outputs of the circuit.
- For example, an HDL description of an AND gate describes how the logic value of the gate's output is determined by the logic values of its inputs.
- HDLs are used in several major steps in the design flow of an integrated circuit: design entry, functional simulation or verification, logic synthesis, timing verification, and fault simulation.




```
module  Simple_Circuit (A, B, C, D, E);
 output         D, E;
 input          A, B, C;
 wire           w1;

 and            G1 (w1, A, B); // Optional gate instance name
 not            G2 (E, C);
 or             G3 (D, w1, E);
endmodule
```

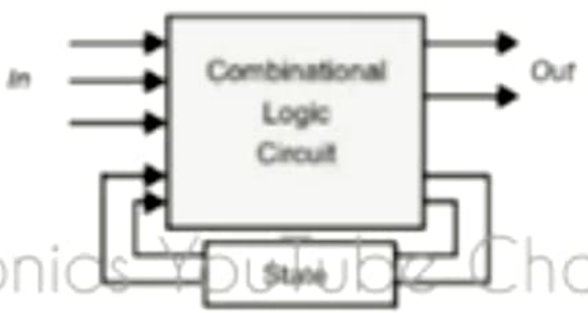
Q.3	a	Explain the differences between Combinational and Sequential Circuits with their block diagrams and examples.	5	L3	CO2

	Combinational Circuits	Sequential Circuits
1.	The outputs of the combinational circuit depend only on the present inputs.	The outputs of the sequential circuits depend on both present inputs and present state(previous output).
2.	The feedback path is not present in the combinational circuit.	The feedback path is present in the sequential circuits.
3.	In combinational circuits, memory elements are not required.	In the sequential circuit, memory elements play an important role and require.
4.	The clock signal is not required for combinational circuits.	The clock signal is required for sequential circuits.
5.	The combinational circuit is simple to design.	It is not simple to design a sequential circuit.

Combinational circuits:
Demultiplexer, Decoder, Full adder encoder, Half adder, Magnitude comparator

Sequential Circuits:
Registers, Flip Flops, Counters

(a) Combinational



(b) Sequential

b	What are decoders? Implement the following Boolean functions with a decoder: $F1(A,B,C) = \sum m(1, 3,4,7)$, $F2(A,B,C) = \sum m(0,2,3,6)$ and $F3(A,B,C) = \sum m(2,3,6,7)$	**5**	**L3**	**CO2**

A *decoder* is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n-bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.

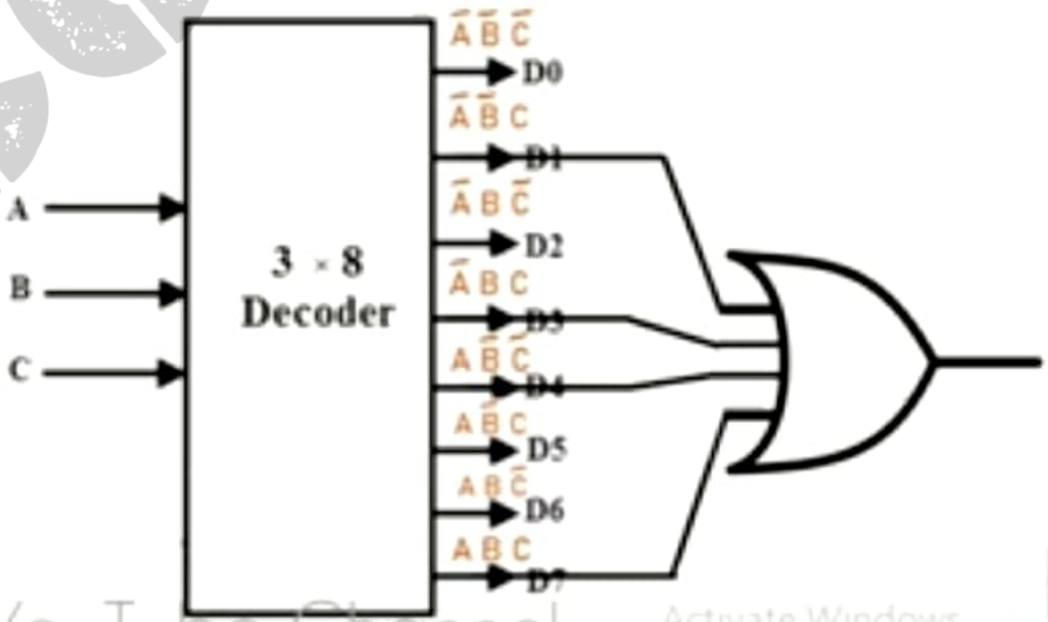
$F_1(A,B,C) = \sum m(1,3,4,7)$

$1 \rightarrow 001 \rightarrow \bar{A}\bar{B}C$

$3 \rightarrow 011 \rightarrow \bar{A}BC$

$4 \rightarrow 100 \rightarrow A\bar{B}\bar{C}$

$7 \rightarrow 111 \rightarrow ABC$



c	What are Multiplexers? Implement the Boolean function F(A,B,C,D) = $\sum m(1,3,4,11,12,13,\overline{14},15)$ with a 8:1 MUX	10	L3	CO2

	A	B	C	D	F
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

$I_0 = D$

$I_1 = D$

$I_2 = \overline{D}$

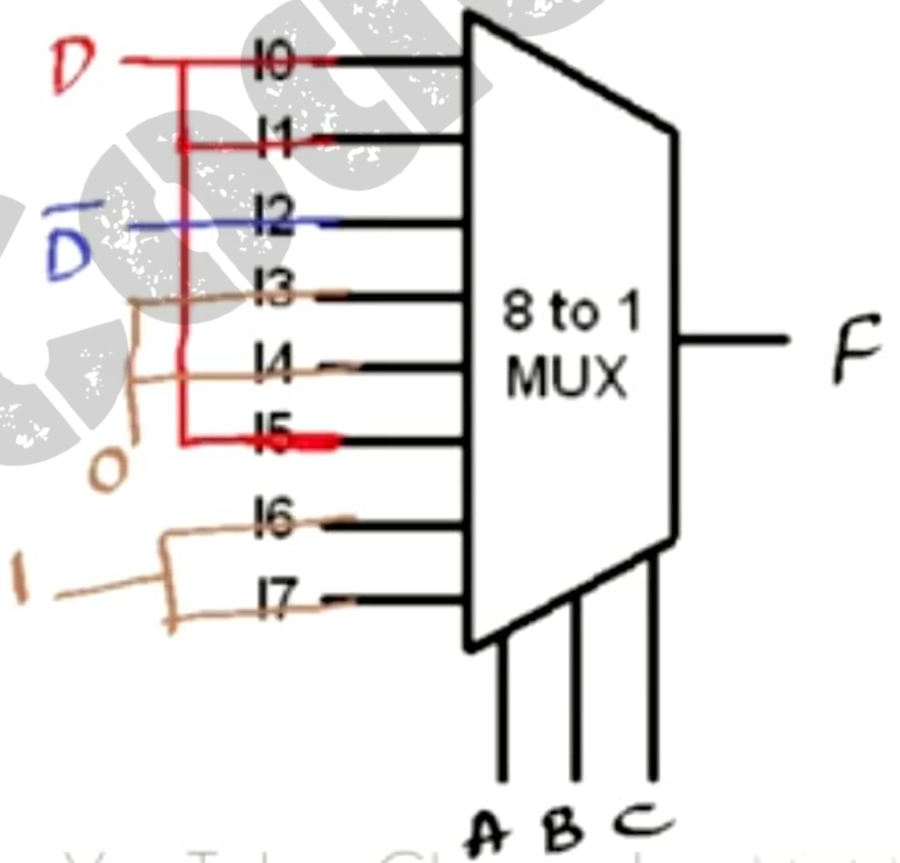
$I_3 = 0$

$I_4 = 0$

$I_5 = D$

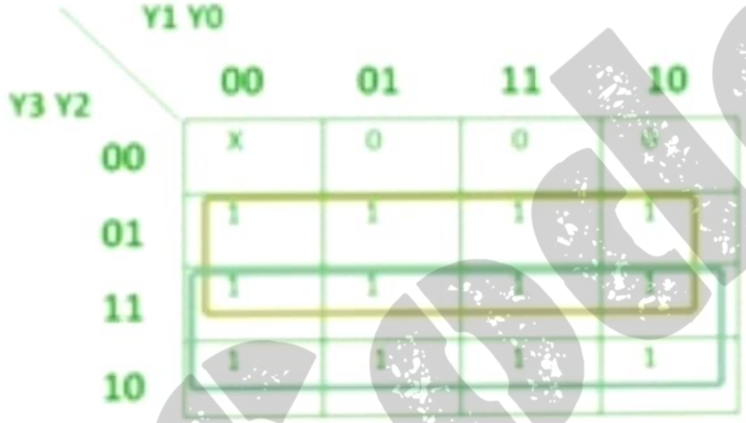
$I_6 = 1$

$I_7 = 1$

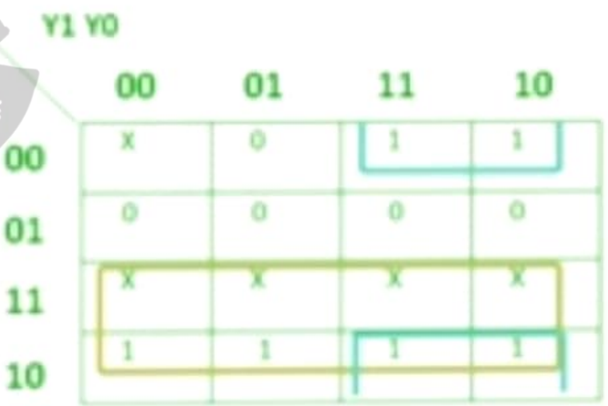


Q.4	a	Define Encoder. Design a Four-input Priority Encoder.	5	L3	CO2

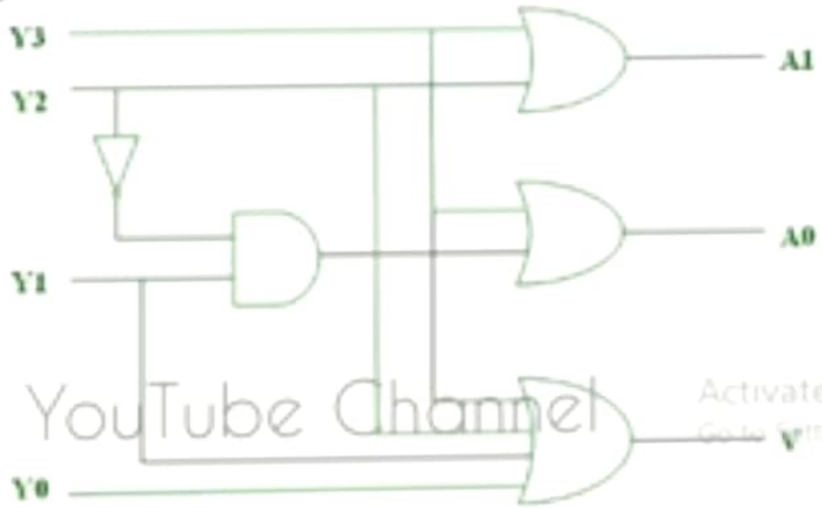
INPUTS				OUTPUTS		
Y3	Y2	Y1	Y0	A1	A0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1



A1=Y3 + Y2



A0=Y3 + Y2' Y1



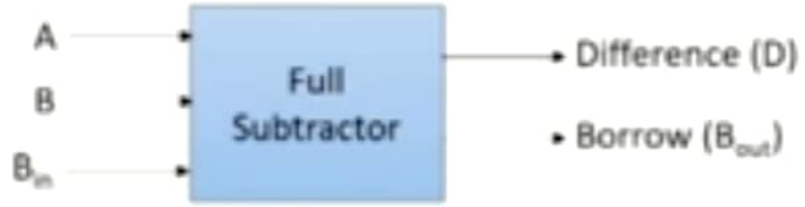
b	Write the Verilog program to Implement Full Adder and Subtractor Circuits.	5	L3	CO2

A	B	Cin	Sum (S)	Carry (Cout)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = A \wedge B \wedge Cin$

$Cout = A \cdot B + B \cdot Cin + A \cdot Cin$

```
module full_adder(input a, b, cin, output S, Cout);
assign S = a ^ b ^ cin;
assign Cout = (a & b) | (b & cin) | (a & cin);
endmodule
```



$D = A \wedge B \wedge Bin$

$Bout = A' \cdot B + (A \wedge B)' \cdot Bin$

A	B	Bin	Difference (D)	Borrow (Bout)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

```
module full_subtractor(input a, b, Bin, output D, Bout);
assign D = a ^ b ^ Bin;
assign Bout = (~a & b) | (~(a ^ b) & Bin);
endmodule
```

c	Write the Characteristic Table and Equations of SR, JK, D and T Flip Flops.	10	L1	CO2

Flip-Flop Characteristic Tables

(a) *JK* Flip-Flop

J	K	$Q(t+1)$	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}(t)$	Complement

(b) *SR* Flip-Flop

S	R	$Q(t+1)$	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined

(c) *D* Flip-Flop

D	$Q(t+1)$	Operation
0	0	Reset
1	1	Set

(d) *T* Flip-Flop

T	$Q(t+1)$	Operation
0	$Q(t)$	No change
1	$\overline{Q}(t)$	Complement

Here are the characteristic equations

- SR flip flop: Qn+1 = S + QnR'
- D flip flop: Qn+1 = D
- JK flip flop: Qn+1 = Q'nJ + QnK'
- T flip flop: Qn+1 = Q'nT+ QnT'

Q.5	a	What do you mean by an Addressing Mode? Explain any 5 Addressing Modes.	10	L1	CO3

- Addressing modes are an aspect of the instruction set architecture in most central processing unit designs.
- The various addressing modes that are defined in a given instruction set architecture define how the machine language instructions in that architecture identify the operand of each instruction

Immediate addressing mode

```
MOV AL, 35H (move the data 35H into AL register)
```

Register mode

```
MOV AX,CX (move the contents of CX register to AX register)
```

Register Indirect mode

```
MOV AX, [BX](move the contents of memory location s addressed by the register BX to the register AX)
```

Direct addressing/ Absolute addressing Mode

```
ADD AL,[0301]   //add the contents of offset address 0301 to AL
```

Indirect addressing Mode

```
MOV A, @R1 (effective address is in the register)
```


Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

b	Describe the functionality of the following: MAR, PC, IR, MDR and ALU	**5**	**L1**	**CO3**

Memory Address Register (MAR) is a register in the CPU that stores the memory address from which data is fetched to the CPU registers. It can also store the address to which data is sent and stored via system bus.

Program Counter (PC) is a register that manages the memory address of the instruction to be executed next.

Instruction Register (IR) is a part of a CPU's control unit that holds the instruction that is being executed or decoded.

Memory Data Register (MDR) is a register in a computer's CPU that stores data being transferred to and from immediate access storage. It's also known as the Memory Buffer Register (MBR)

Arithmetic Logic Unit (ALU) performs basic arithmetic and logical operations.

c	Explain Basic Performance Equation and SPEC rating.	5	L2	CO3

$$T = \frac{N \times S}{R}$$

This is often referred to as the *basic performance equation.*

The performance parameter T for an application program is much more important to the user than the individual values of the parameters N, S, or R. To achieve high performance, the computer designer must seek ways to reduce the value of T, which means reducing N and S, and increasing R. The value of N is reduced if the source program is compiled into fewer machine instructions. The value of S is reduced if instructions have a smaller number of basic steps to perform or if the execution of instructions is overlapped. Using a higher-frequency clock increases the value or R, which means that the time required to complete a basic execution step is reduced.

We must emphasize that N, S, and R are not independent parameters; changing one may affect another. Introducing a new feature in the design of a processor will lead to improved performance only if the overall result is to reduce the value of T. A processor advertised as having a 900-MHz clock does not necessarily provide better performance than a 700-MHz processor because it may have a different value of S.

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

- The performance of a program depends on the algorithm, the language, the compiler, the architecture, and the actual hardware.
- Clock cycle Also called tick, clock tick, clock period, clock, or cycle.
- The time for one clock period, usually of the processor clock, which runs at a constant rate.
- Clock period - The length of each clock cycle.

SPEC- System Performance Evaluation Corporation, Its a non profit organization selects and publishes bench marks.

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i\right)^{\frac{1}{n}}$$

Q.6	a	Demonstrate the Branching operations using a loop to add n numbers with block diagram.	8	L3	CO3

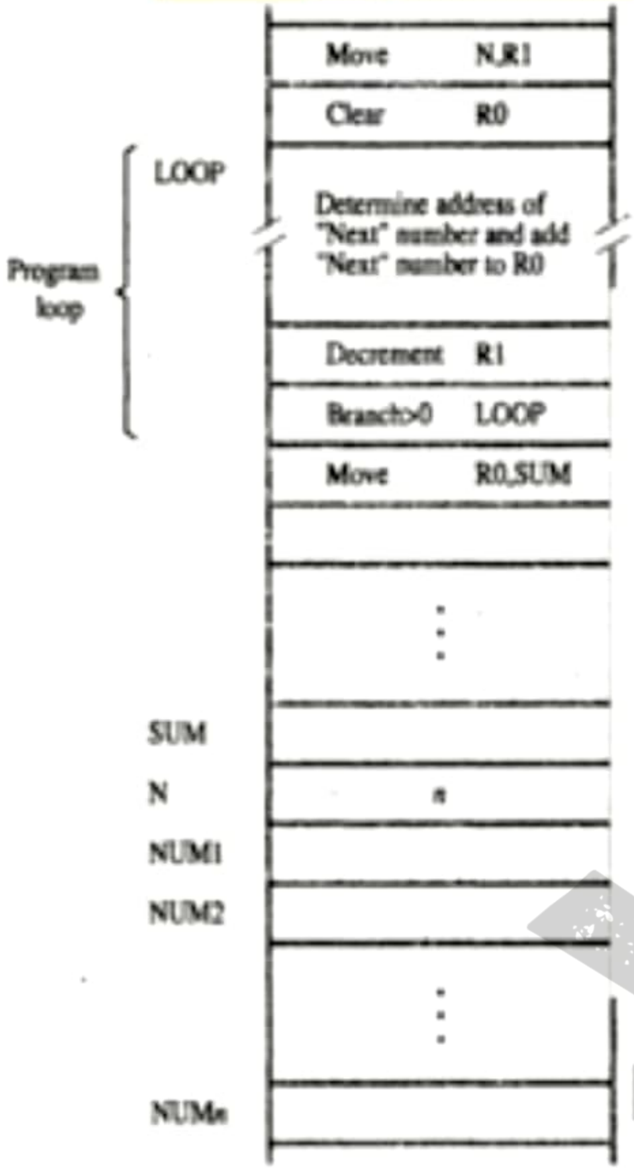


Figure 2.10 Using a loop to add *n* numbers.

In the program in Figure 2.10, the instruction

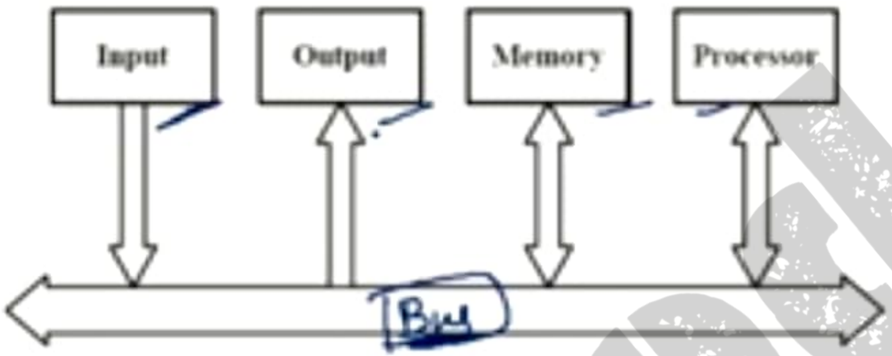
Branch>0 LOOP

(branch if greater than 0) is a conditional branch instruction that causes a branch to location LOOP if the result of the immediately preceding instruction, which is the decremented value in register R1, is greater than zero. This means that the loop is repeated as long as there are entries in the list that are yet to be added to R0. At the end of the *n*th pass through the loop, the Decrement instruction produces a value of zero, and, hence, branching does not occur. Instead, the Move instruction is fetched and executed. It moves the final result from R0 into memory location SUM.

b	The Registers R1 and R2 has decimal values 1200 and 4600. Calculate the effective address of the memory operand in each of the following instructions when they are executed in sequence. i) Load 20(R1), R5 ii) Move #3000, R5 iii) Store R5, 30(R1,R2) iv) Add –(R2), R5 v) Subtract (R1)+, R5	**7**	**L3**	**CO3**

- ***Registers R1 and R2 of a computer contain the decimal values 1200 and 4600, we have to find effective address of associated memory operand in each instruction:***
- **Load 20(R1),R5 :** This means load 20+R1 into R5 .
  R1= 1200 , R1 + 20 = 1220 , so R5 have 1220 , **Effective address of R5 is 1220.**
- **Move #3000,R5 :** This means move value 3000 into R5
  so effective address is part of the instruction whose value is 3000. **Now R5 = 3000**
- **Store R5,30(R1,R2) :** This means 30+R1+R2 and store the result into R5 .
  so R5 = 30+1200+4600 = 5830 , so now R5 value is 5830 , the **effective address is 5830.**
- **Add -(R2),R5 :** This means -1 from R2 value and store the result into R5 . So R5= 4600 - 1 = 4599 , effective address of R5 is 4599 . It is pre decrement addressing.
- **Subtract (R1)+,R5 :** This means effective address is contents of R1 so EA = 1200 .
  It is post increment addressing.

c	Explain Single Bus Structure.	5	L2	CO3



- A single bus structure is a computer architecture that uses a common bus to communicate between the processor, memory, and I/O devices.
- one common bus is used to communicate between peripherals and microprocessors
- Only one transfer at a time: The bus can only be used for one transfer at a time, so only two units can actively use the bus at any given time.
- A single bus structure is primarily found in mini and microcomputers.

- One significant advantage of using a one bus data path is its simplicity.
- Single bus structure has disadvantages of limited speed since usually only two units can participate in a data transfer at any one time. This means that an arbitration system is required and that units will be forced to wait.

Q.7	a	Explain Memory mapped I/O and I/O interface for an input device with a diagram.	10	L2	CO4

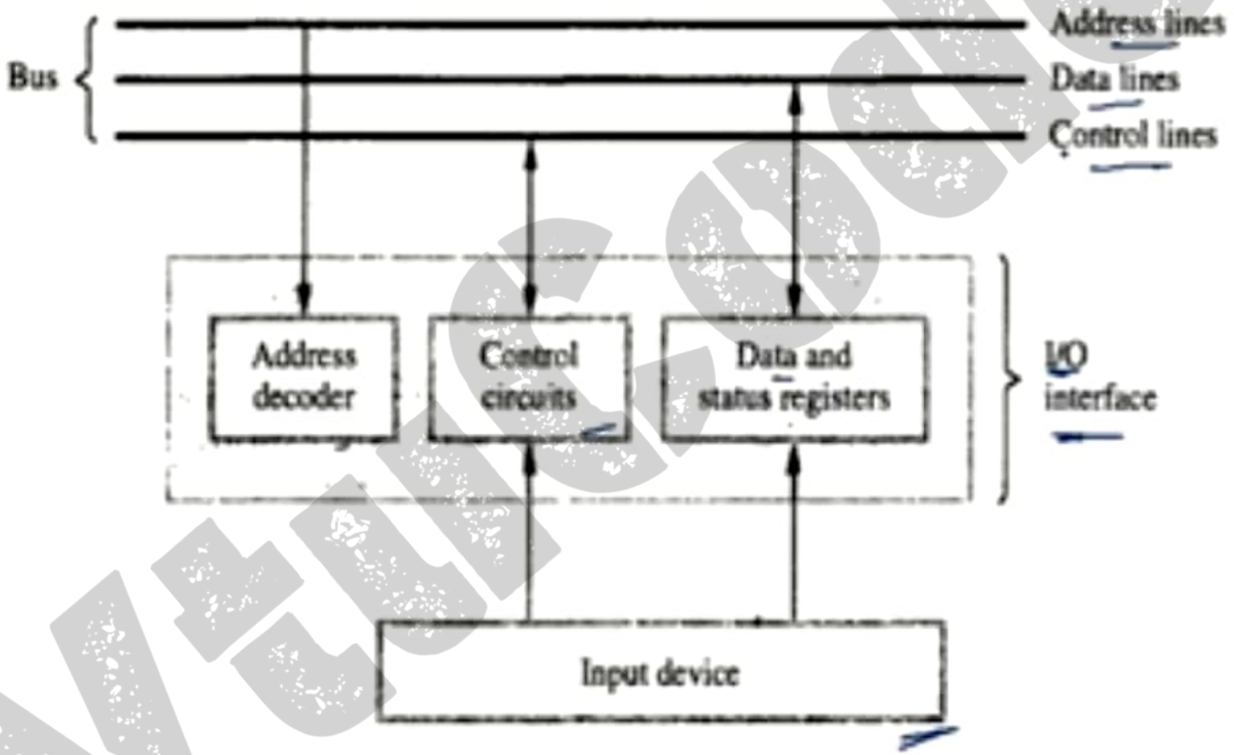


Figure 4.2 I/O interface for an input device.

b	Explain I/O operations involving a keyboard and display device with a program that reads one line from keyboard, stores it in buffer and echoes it back to display.	**10**	**L4**	**CO4**

```
        Move        #LINE,R0          Initialize memory pointer.
WAITK   TestBit     #0,STATUS         Test SIN.
        Branch=0    WAITK             Wait for character to be entered.
        Move        DATAIN,R1         Read character.
WAITD   TestBit     #1,STATUS         Test SOUT.
        Branch=0    WAITD             Wait for display to become ready.
        Move        R1,DATAOUT        Send character to display.
        Move        R1,(R0)+          Store charater and advance pointer.
        Compare     #$0D,R1           Check if Carriage Return.
        Branch≠0    WAITK             If not, get another character.
        Move        #$0A,DATAOUT      Otherwise, send Line Feed.
        Call        PROCESS           Call a subroutine to process the
                                        the input line.
```

Figure 4.4 A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display.

To review the basic concepts, let us consider a simple example of I/O operations involving a keyboard and a display device in a computer system. The four registers shown in Figure 4.3 are used in the data transfer operations. Register STATUS contains two control flags, SIN and SOUT, which provide status information for the keyboard and the display unit, respectively. The two flags KIRQ and DIRQ in this register are used in conjunction with interrupts. They, and the KEN and DEN bits in register CONTROL, will be discussed in Section 4.2. Data from the keyboard are made available in the DATAIN register, and data sent to the display are stored in the DATAOUT register.

The program in Figure 4.4 is similar to that in Figure 2.20. This program reads a line of characters from the keyboard and stores it in a memory buffer starting at location LINE. Then, it calls a subroutine PROCESS to process the input line. As each character is read, it is *echoed back* to the display. Register R0 is used as a pointer to the memory buffer area. The contents of R0 are updated using the Autoincrement addressing mode so that successive characters are stored in successive memory locations.

Each character is checked to see if it is the Carriage Return (CR) character, which has the ASCII code 0D (hex). If it is, a Line Feed character (ASCII code 0A) is sent to move the cursor one line down on the display and subroutine PROCESS is called. Otherwise, the program loops back to wait for another character from the keyboard.

Q.8	a	Explain how to handle interrupt from multiple devices using daisy chain and priority scheme.	10	L3	CO4

Daisy chaining is a method of interrupt handling that involves connecting devices that can request an interrupt in a serial manner. The devices are connected in a serial form, with the device with the highest priority placed first, followed by lower-priority devices.

(a) Daisy chain

shown in Figure 4.8*a*. The interrupt-request line $\overline{\text{INTR}}$ is common to all devices. The interrupt-acknowledge line, INTA, is connected in a daisy-chain fashion, such that the INTA signal propagates serially through the devices. When several devices raise an interrupt request and the $\overline{\text{INTR}}$ line is activated, the processor responds by setting the INTA line to 1. This signal is received by device 1. Device 1 passes the signal on to device 2 only if it does not require any service. If device 1 has a pending request for interrupt, it blocks the INTA signal and proceeds to put its identifying code on the data lines. Therefore, in the daisy-chain arrangement, the device that is electrically closest to the processor has the highest priority. The second device along the chain has second highest priority, and so on.

b	Explain Centralized and Distributed Bus Arbitration approaches.	10	L2	CO4

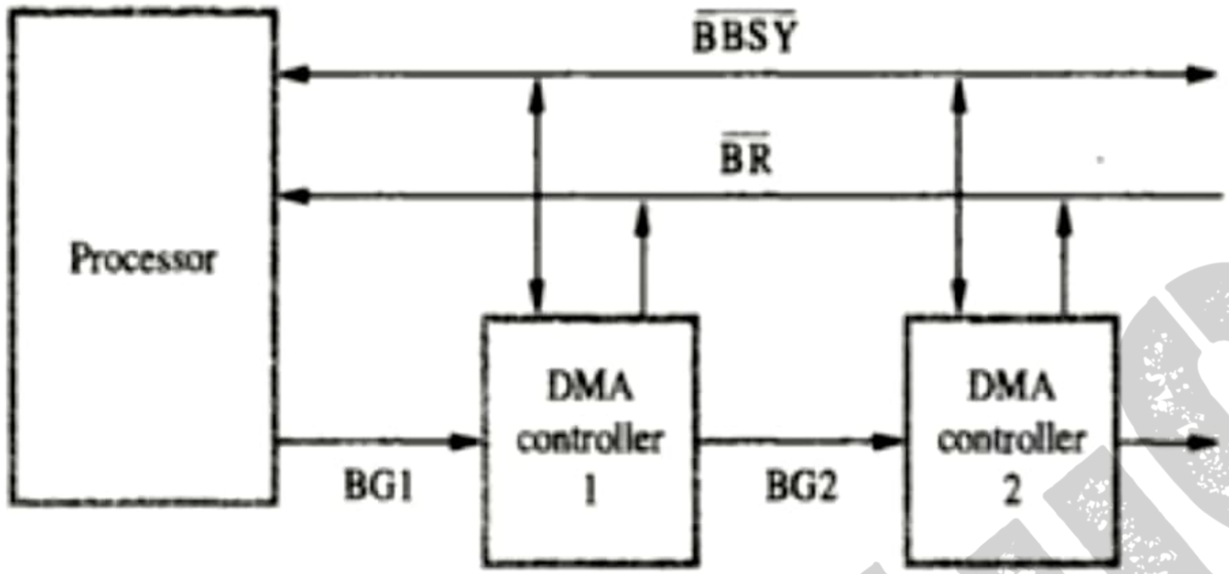


Figure 4.20 A simple arrangement for bus arbitration using a daisy chain.

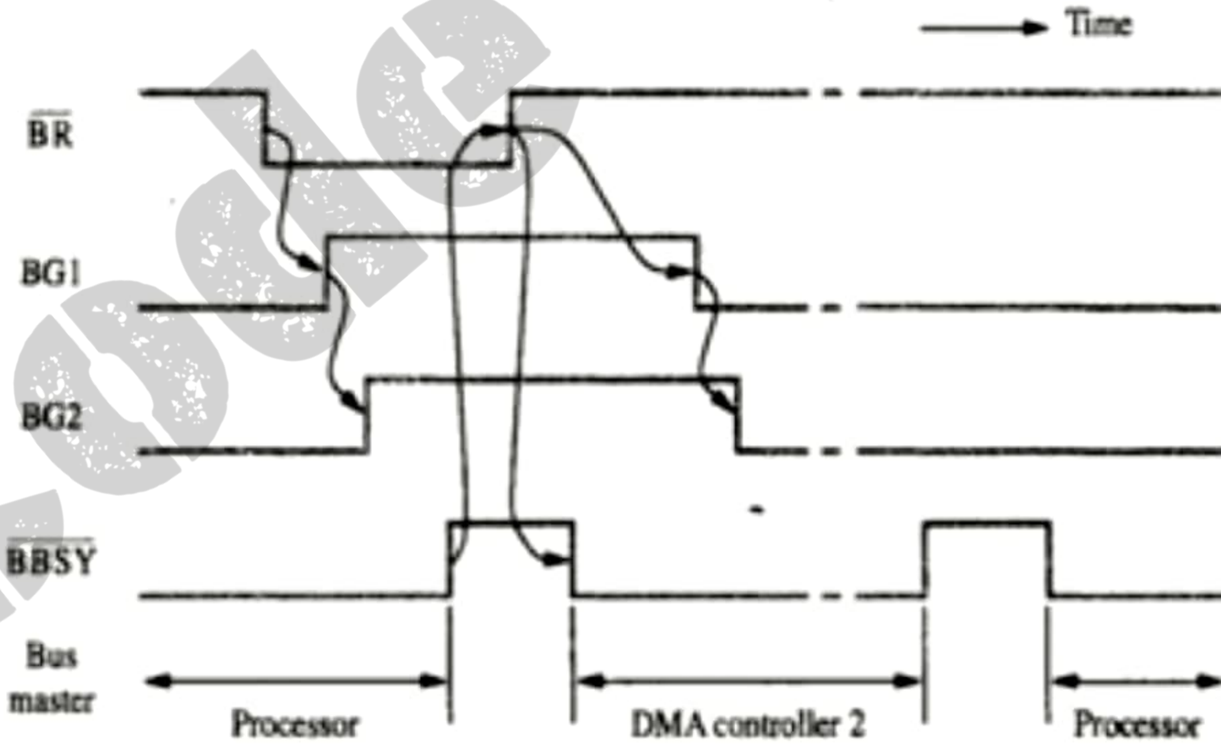


Figure 4.21 Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

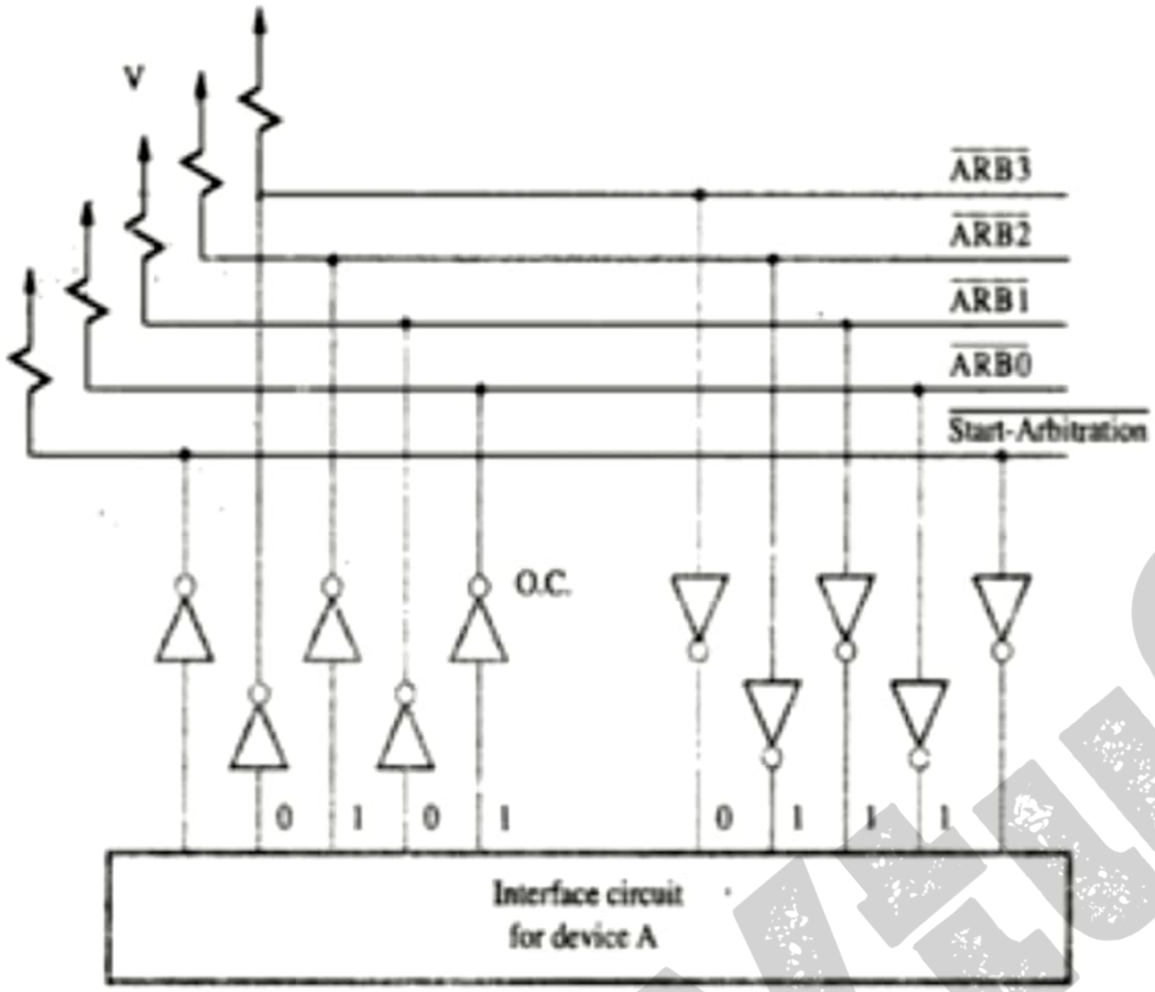
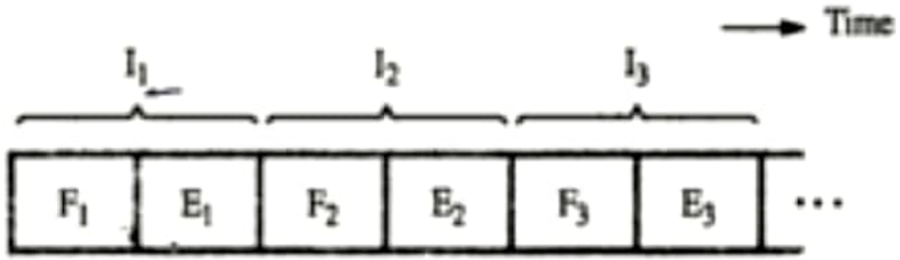


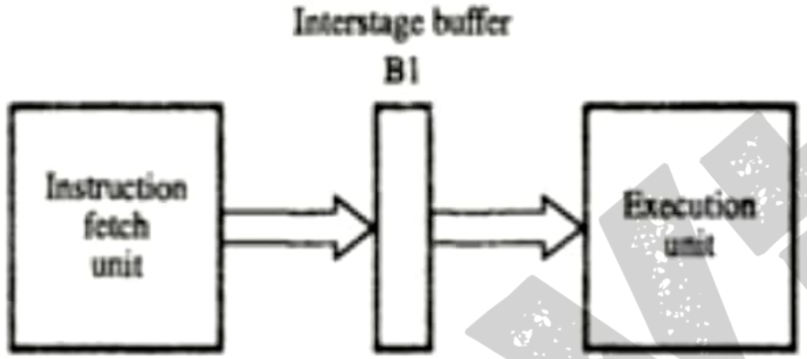
Figure 4.22 A distributed arbitration scheme.

Distributed arbitration means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter. A simple method for distributed arbitration is illustrated in Figure 4.22. Each device on the bus is assigned a 4-bit identification number. When one or more devices request the bus, they assert the Start-Arbitration signal and place their 4-bit ID numbers on four open-collector lines, ARB0 through ARB3. A winner is selected as a result of the interaction among the signals transmitted over these lines by all contenders. The net outcome is that the code on the four lines represents the request that has the highest ID number.

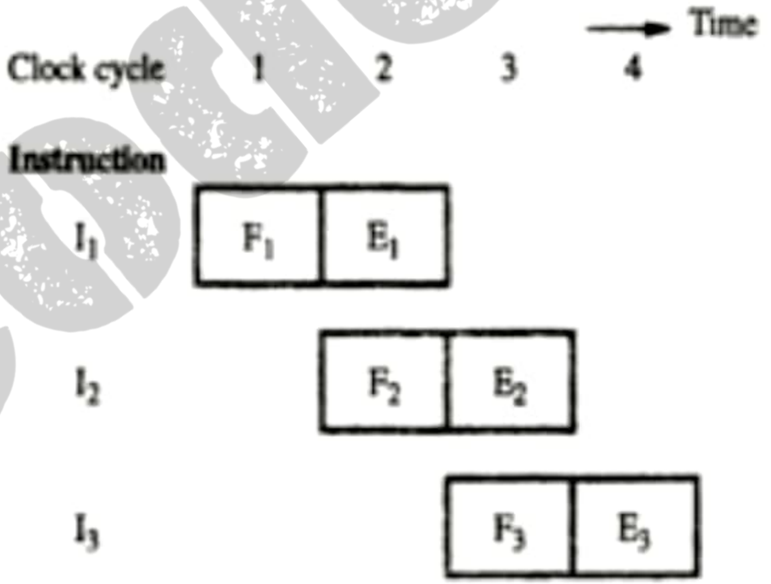
b	Describe the Basic idea of Instruction Pipeline.	**10**	**L2**	**CO5**



(a) Sequential execution



(b) Hardware organization



(c) Pipelined execution

Figure 8.1 Basic idea of instruction pipelining.

Q.10	a	Explain the process of Fetching Word from Memory in processor.	10	L4	CO5

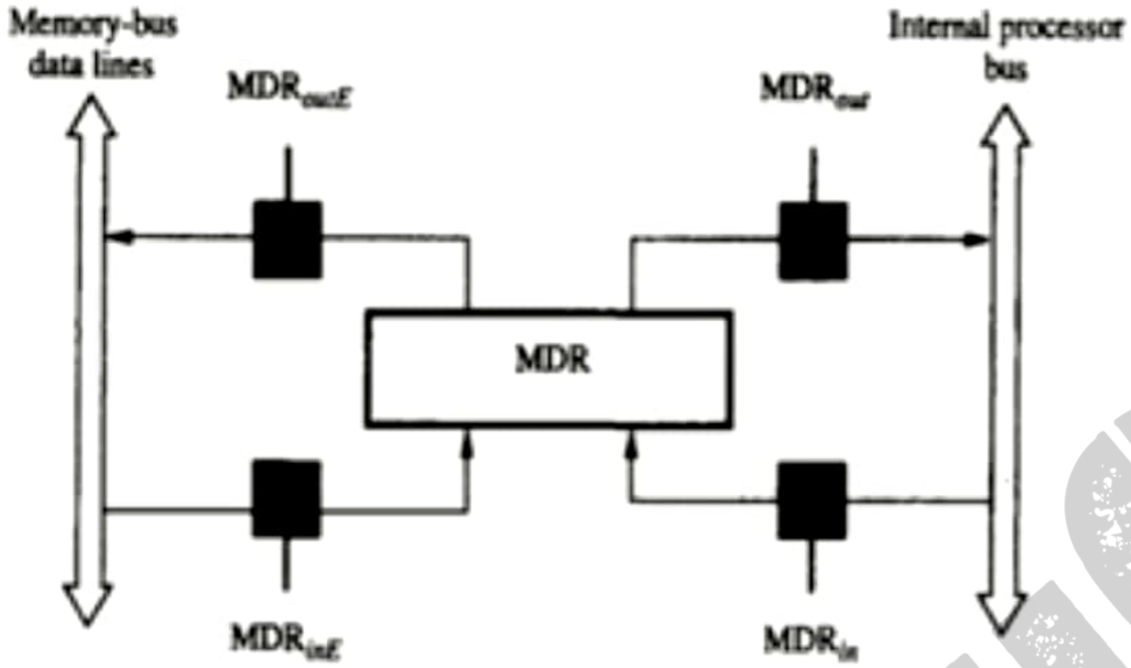


Figure 7.4 Connection and control signals for register MDR.

As an example of a read operation, consider the instruction Move (R1),R2. The actions needed to execute this instruction are:

1. MAR ← [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory bus
5. R2 ← [MDR]

To fetch a word of information from memory, the processor has to specify the address of the memory location where this information is stored and request a Read operation. This applies whether the information to be fetched represents an instruction in a program or an operand specified by an instruction. The processor transfers the required address to the MAR, whose output is connected to the address lines of the memory bus. At the same time, the processor uses the control lines of the memory bus to indicate that a Read operation is needed. When the requested data are received from the memory they are stored in register MDR, from where they can be transferred to other registers in the processor.

The connections for register MDR are illustrated in Figure 7.4. It has four control signals: MDR_{in} and MDR_{out} control the connection to the internal bus, and MDR_{inE} and MDR_{outE} control the connection to the external bus. The circuit in Figure 7.3 is easily modified to provide the additional connections. A three-input multiplexer can be used, with the memory bus data line connected to the third input. This input is selected when $MDR_{inE} = 1$. A second tri-state gate, controlled by MDR_{outE} can be used to connect the output of the flip-flop to the memory bus.

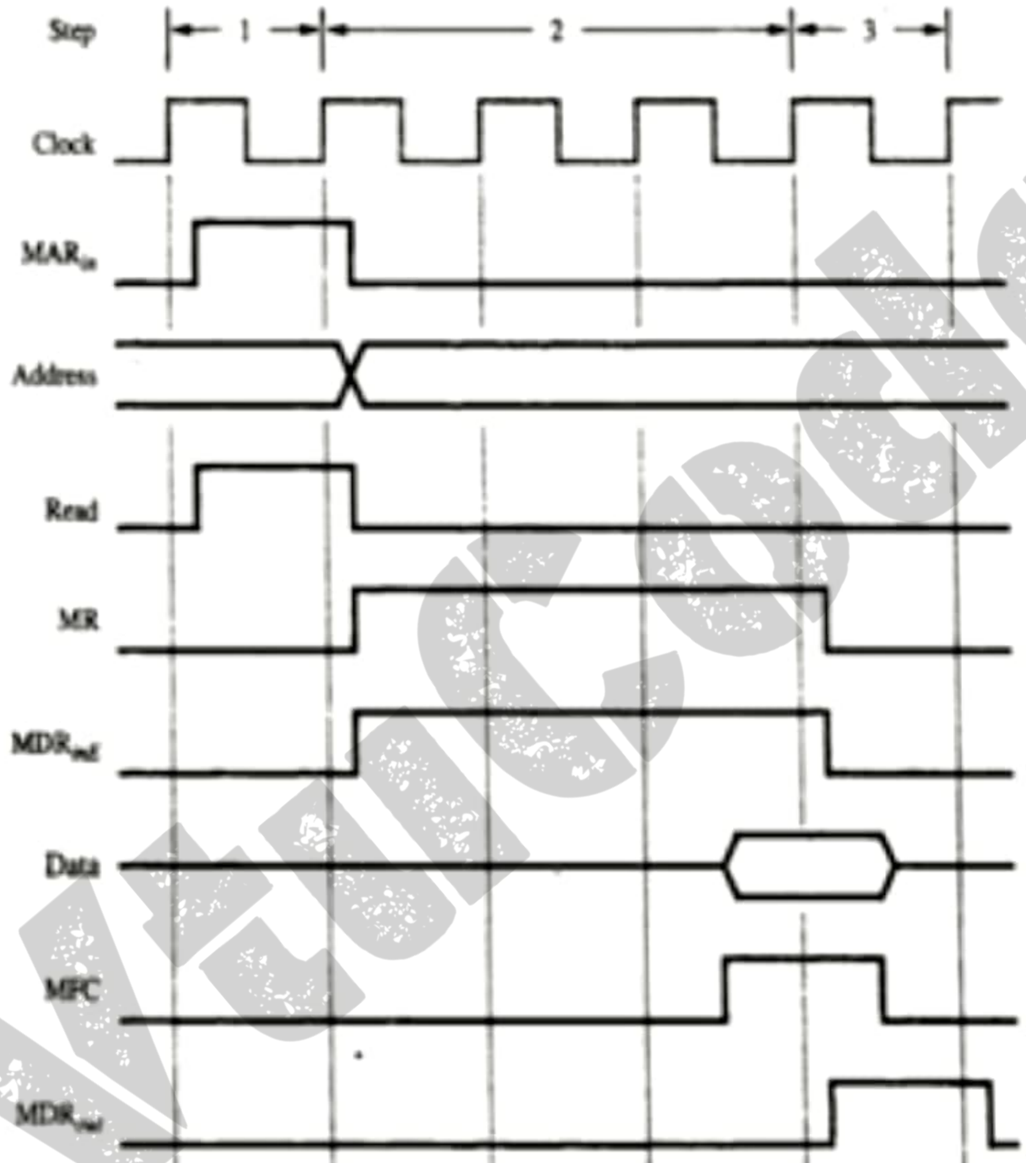
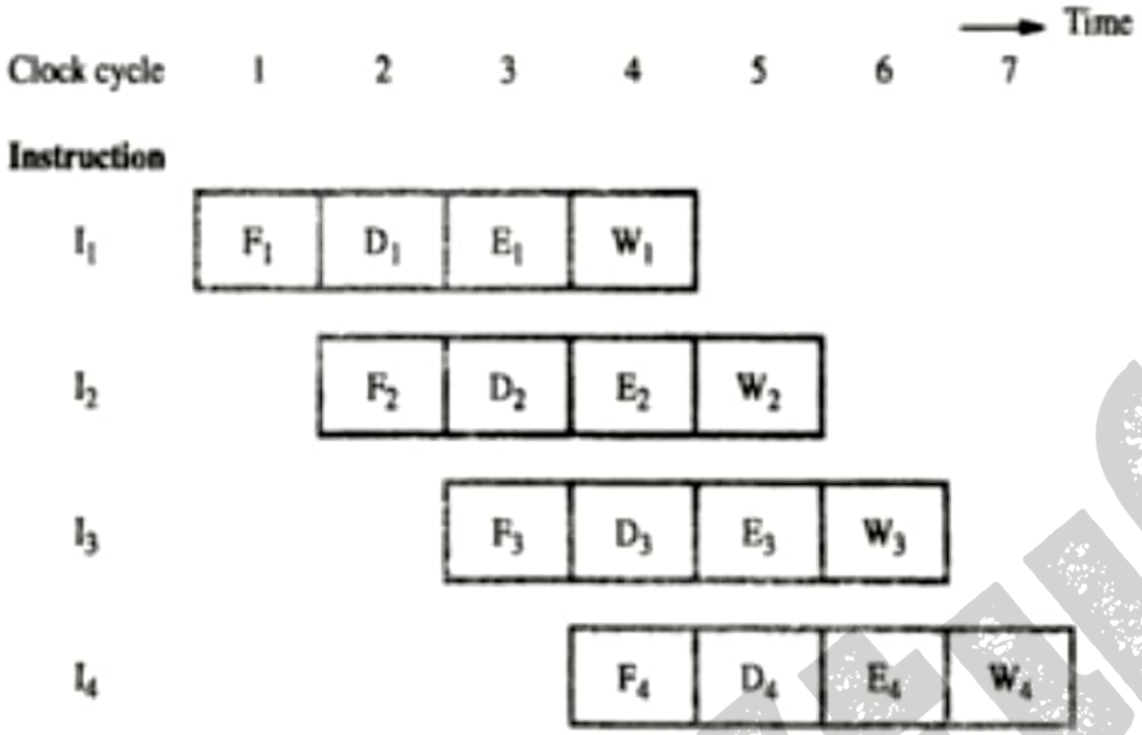
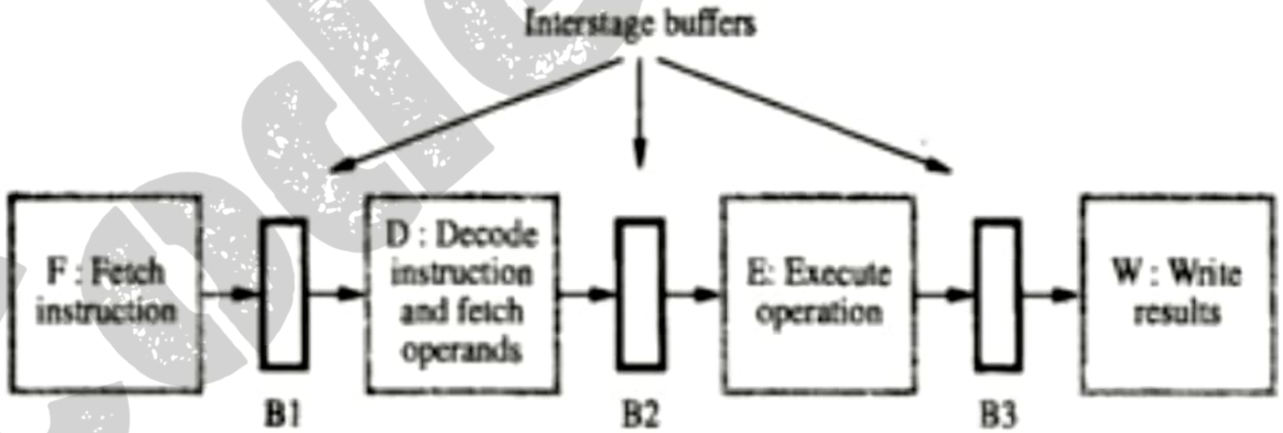


Figure 7.5 Timing of a memory Read operation.

b	Explain the Pipeline Performance of a Processor and pipeline stalls.	10	L2	CO5



(a) Instruction execution divided into four steps



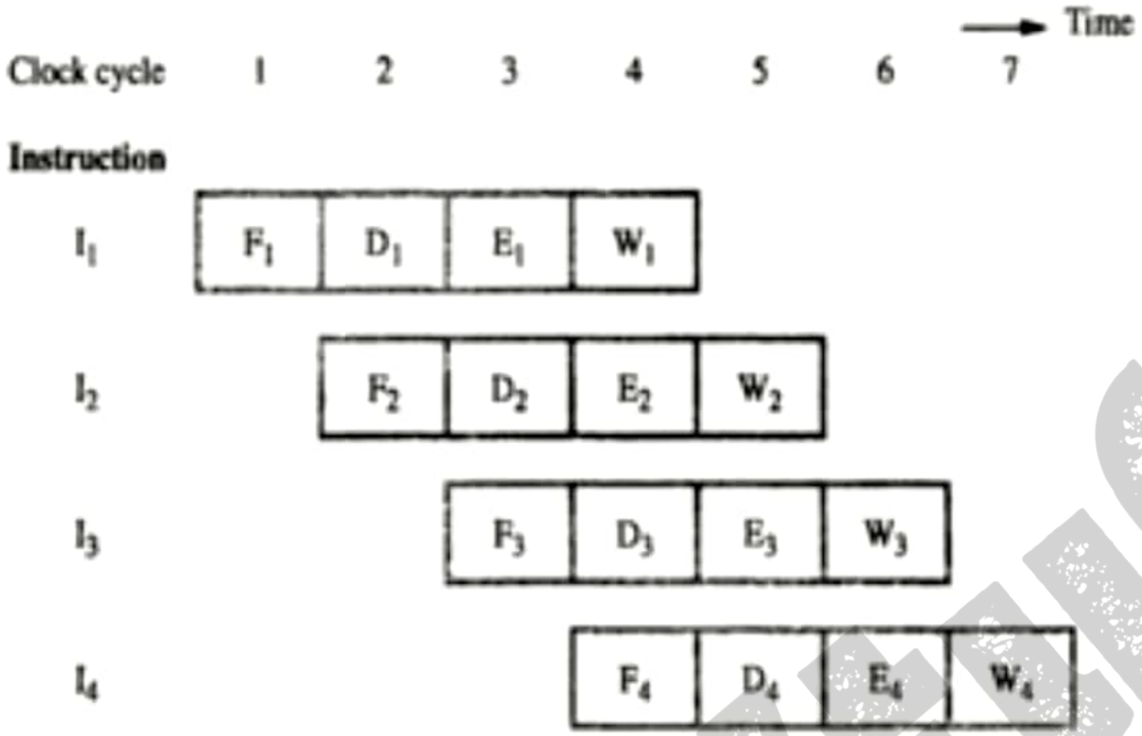
(b) Hardware organization

Figure 8.2 A 4-stage pipeline.

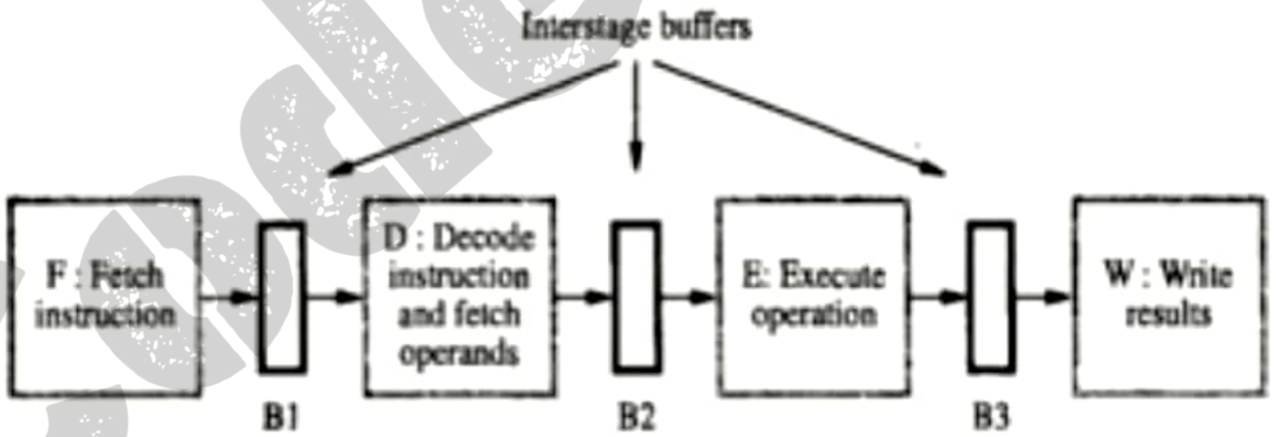
The pipelined processor in Figure 8.2 completes the processing of one instruction in each clock cycle, which means that the rate of instruction processing is four times that of sequential operation. The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages. However, this increase would be achieved only if pipelined operation as depicted in Figure 8.2*a* could be sustained without interruption throughout program execution. Unfortunately, this is not the case.

For a variety of reasons, one of the pipeline stages may not be able to complete its processing task for a given instruction in the time allotted. For example, stage E in the four-stage pipeline of Figure 8.2*b* is responsible for arithmetic and logic operations, and one clock cycle is assigned for this task. Although this may be sufficient for most operations, some operations, such as divide, may require more time to complete. Figure 8.3 shows an example in which the operation specified in instruction I_2 requires three cycles to complete, from cycle 4 through cycle 6. Thus, in cycles 5 and 6, the Write stage must be told to do nothing, because it has no data to work with. Meanwhile, the information in buffer B2 must remain intact until the Execute stage has completed its operation. This means that stage 2 and, in turn, stage 1 are blocked from accepting new instructions because the information in B1 cannot be overwritten. Thus, steps D_4 and F_5 must be postponed as shown.

b	Explain the Pipeline Performance of a Processor and pipeline stalls.	**10**	**L2**	**CO5**



(a) Instruction execution divided into four steps



(b) Hardware organization

Figure 8.2 A 4-stage pipeline.