UNIT-II

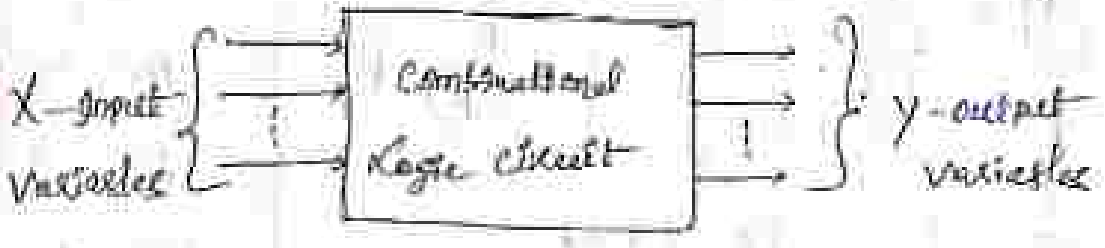
⇒ Combinational circuits :- Digital Logic circuits

→ Logic circuits for digital systems may be combinational (or) Sequential.

→ In combinational circuits, the output variables at any instant of time are dependent only on the present input variables.

→ A combinational circuit consists of input variables, logic gates and output variables.

→ Logic gates accept signals from the input and generate signals to the outputs.



→ Design procedure of combinational circuit :-

The design of combinational circuits starts from the specification of the problem that can be implemented in a logic circuit diagram or a set of Boolean functions.

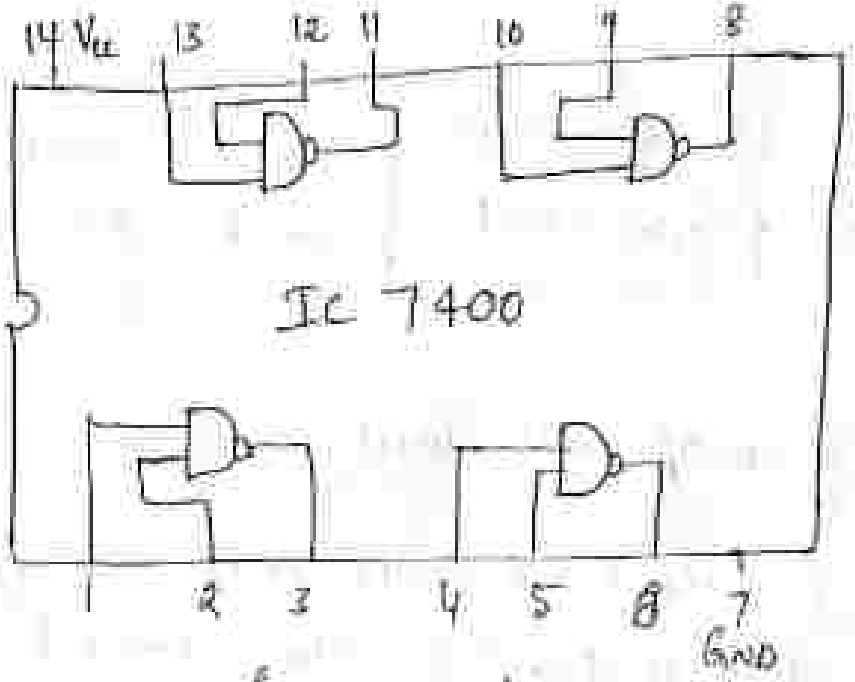
(1) From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.

(2) Derive the truth table that defines the required relationship between inputs and outputs.

(3) Obtain the boolean functions and draw the logic diagram.

⇒ Integrated NAND-NOR Gates :-

NAND gate is actually a series of AND gate with NOT gate. If we connect the output of an AND gate to the i/p of a NOT gate, this combination will work as NOT-AND (or) NAND gate. The output is 1 when any or all inputs are 0, otherwise o/p is 1.



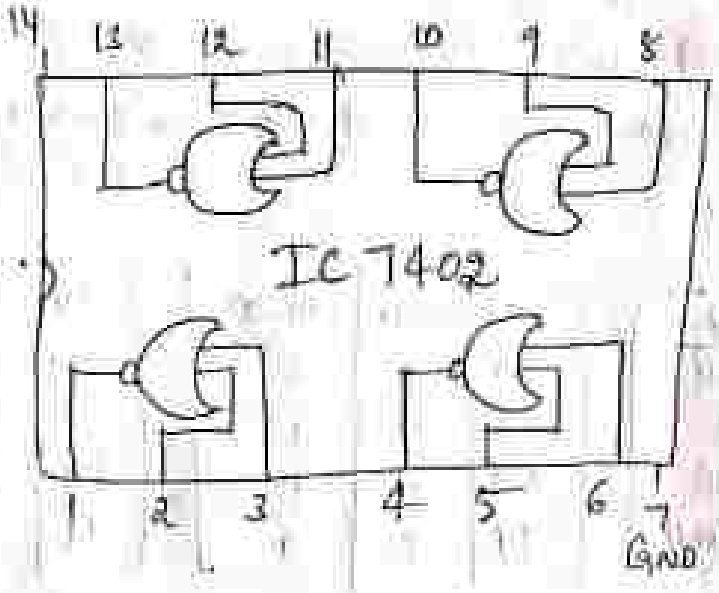
Truth Table

$Y = \overline{A \cdot B}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

fig IC 7400

NOR gate is actually a series of OR gate with NOT gate. If we connect the output of an OR gate to the i/p of a NOT gate, this combination will work as NOT-OR or NOR gate. The output is 0 when any (or) OR all inputs are 1, otherwise output is 1.

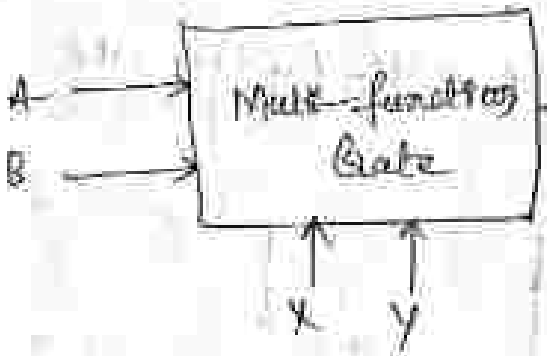


Truth Table

$Y = \overline{A+B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

⇒ **Multifunction gates** :— The gates which are doing multi-function, those type of gates are called multifunction gates. Many functions will perform by these gates.



⇒ **Design Specification plan** :— The idea is to design a multi-function gate that will have sets of inputs, and one output F. The function F will be instructed to perform four different logic operations. A and B are the data inputs. X and Y are control what the gate will do. X and Y are the operation selection lines.

⇒ **Design methodology** :— A and B tell the gate what operation to perform. If A and B both are 0, the gate will act as an AND gate.

If A=0, B=1, the gate will operate as OR. If A=1, B=0 the gate will operate as NOR. If A and B are both 1, the gate operates as NAND.

∴

A B	XY
0 0	AND
0 1	OR
1 0	NOR
1 1	NAND

XY	AND	NAND
00	0	1
01	0	1
10	0	1
11	1	0

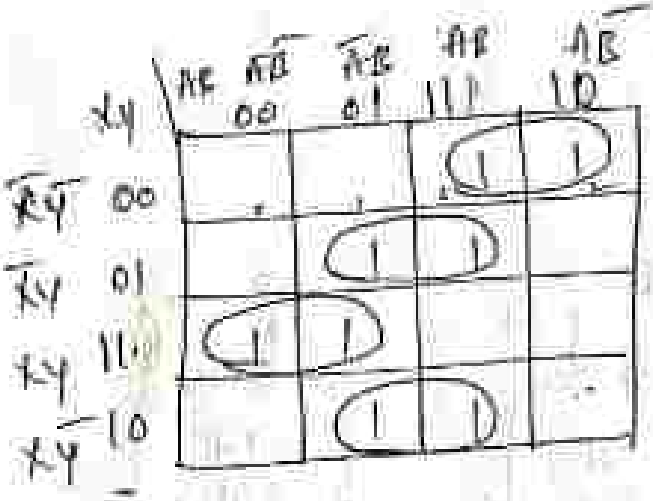
XY	OR	NOR
00	0	1
01	1	0
10	1	0
11	1	0

X and Y depend upon what A and B are doing. The truth tables for AND, NAND, OR and NOR can be seen for above figures. The three above figures can be condensed into a lengthy truth table as shown below figure.

Truth Table of multi-function gate

Gates	AND gate	OR gate	NOR gate	NAND gate
Z	0 0 0 1	0 1 1 1	1 0 0 0	1 1 1 0
XY	00 01 10 11	00 01 10 11	00 01 10 11	00 01 10 11
AB	00 00 00 00	01 01 01 01	10 10 10 10	11 11 11 11

It can easily be seen how the truth table can get rather complicated. Karnaugh (K-map) map would be a more convenient way to represent the multi-functioning gate.



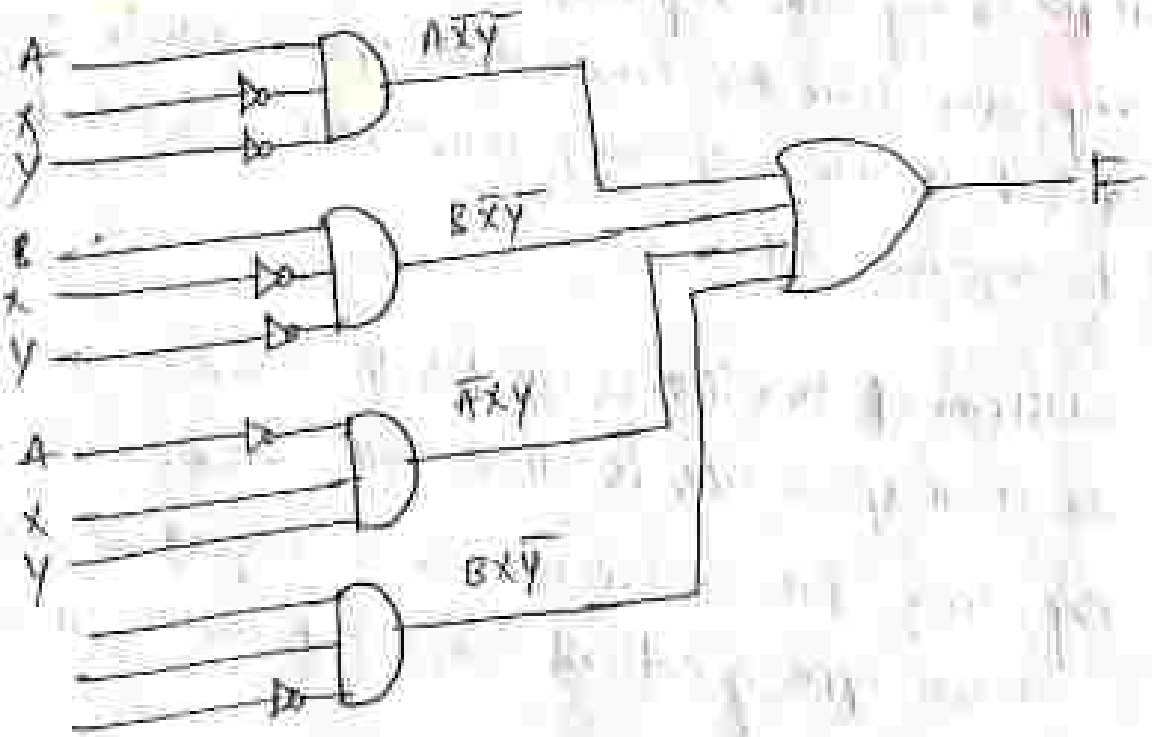
$\Sigma m(2,3,5,7,9,11,12,13)$

By using the ones on the table, the function can be written as a sum of products (SOP).

To do this you need ABXY to multiply out to equal 1. Therefore, the unsimplified equation for f is:

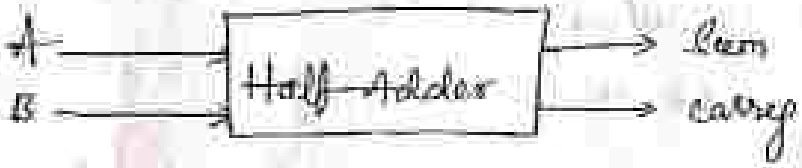
$$F = A\bar{X}\bar{Y} + B\bar{X}\bar{Y} + \bar{A}XY + BX\bar{Y}$$

⇒ Schematic diagram:



⇒ **Half Adder :-** A half adder is a combinational circuit with two binary inputs and two binary outputs (Sum and Carry). It adds the two inputs (A & B) and produces the Sum (S) and carry (C) as a output bits.

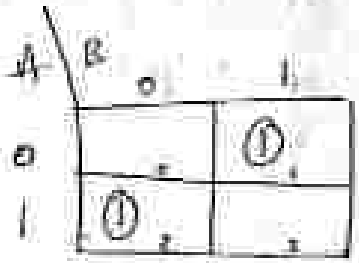
⇒ **Block diagram :-**



⇒ **Truth table :-**

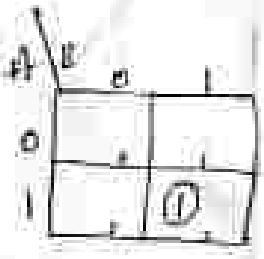
Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-map for (S)



$S = \bar{A}B + A\bar{B}$

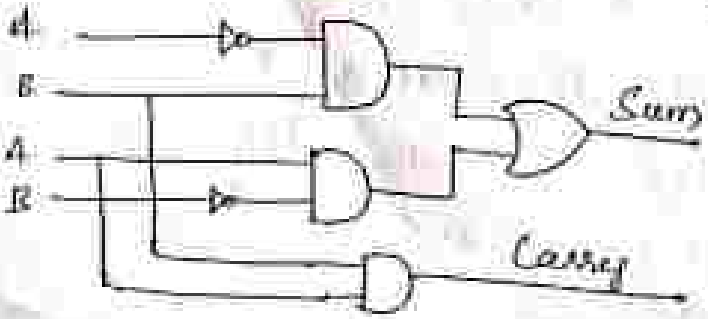
K-map for (C)



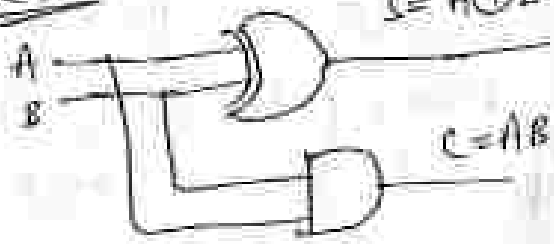
$C = AB$

$S = A \oplus B$

⇒ **Logic diagram :-**



(or)



→ **Full Adder :-** A full adder is a combinational circuit that adds two bit and a carry and outputs a sum bit and carry bit

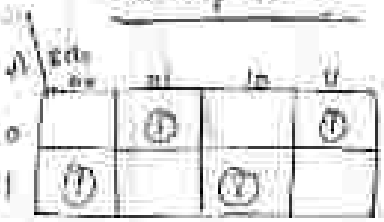
→ The full adder adds the bits A and B and the carry from the previous column called the "carry in" (C_{in}) and outputs the sum bit 'S' and carry bit called (C_{out}).

Block diagram

→ **Truth Table :-**

Inputs			Outputs	
A	B	C_{in}	S	C_{out}
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

K-map for S

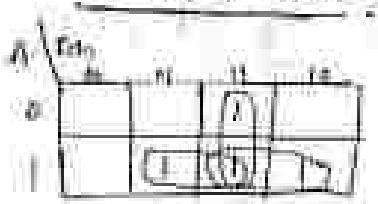


$S = \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + AB C_{in} + \bar{A}\bar{B}C_{in}$

$= \bar{A}(B \oplus C_{in}) + A(\overline{B \oplus C_{in}})$

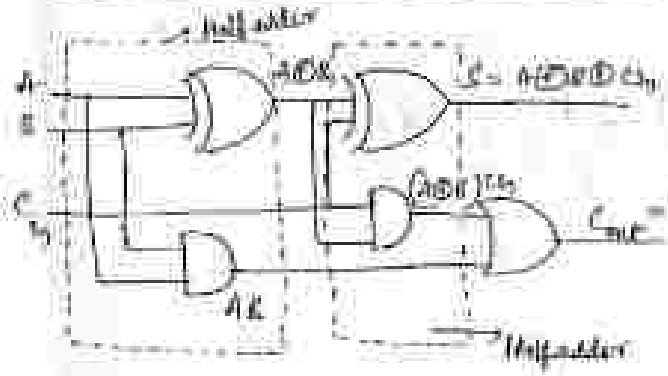
$= A \oplus B \oplus C_{in}$

K-map for C_{out} :-



$C_{out} = BC_{in} + AC_{in} + AB$

→ **Logic diagram :-** (Using two half adders)



carry

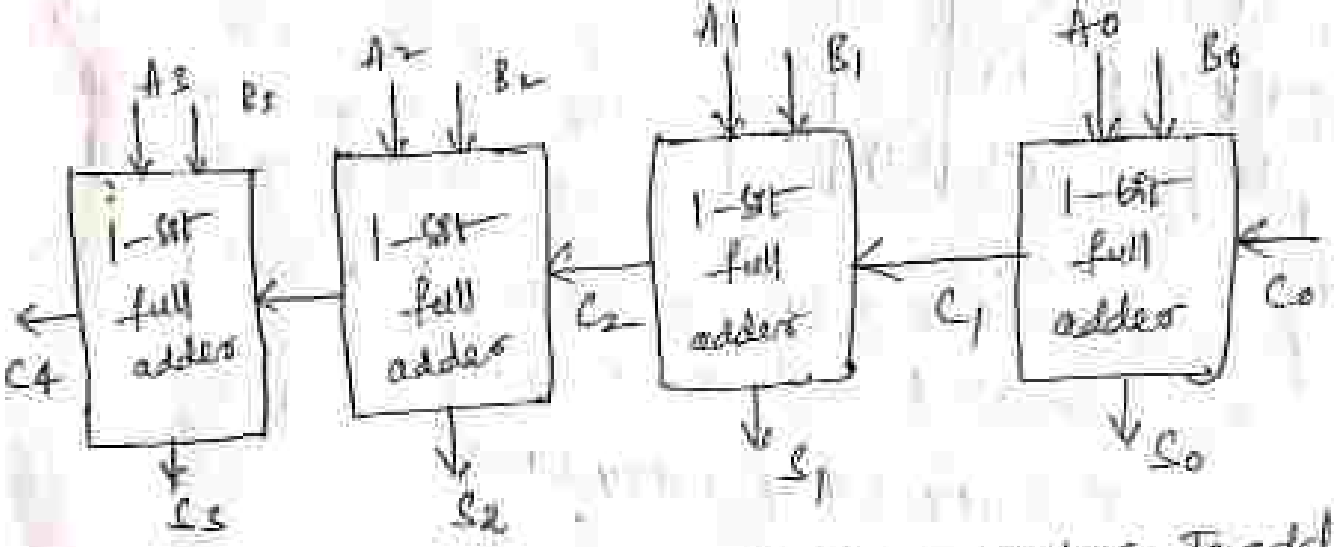
$C = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$= C(\bar{A}B + A\bar{B}) + AB(C + \bar{C})$

$= C(A \oplus B) + AB$

⇒ **Multi-bit adder** :— The most basic arithmetic operation is the addition of binary digits. A combinational circuit that performs the addition of more bits (multi) is called multi-bit adder.

⇒ **Block Diagram** :—



A circuit for adding two 4-bit binary numbers. To add multiple binary bits together, we must include a possible carry over from the lower order of magnitude, and send it as an input carry to the next higher order of magnitude bit.

* Addition of two bits is called half adders
* Addition of 3 bits is called full adder.
* By using full adders we are adding more bits. These type of adders are called multi-bit adders.

→ Multiplexers :- (Multiplexing means sharing)

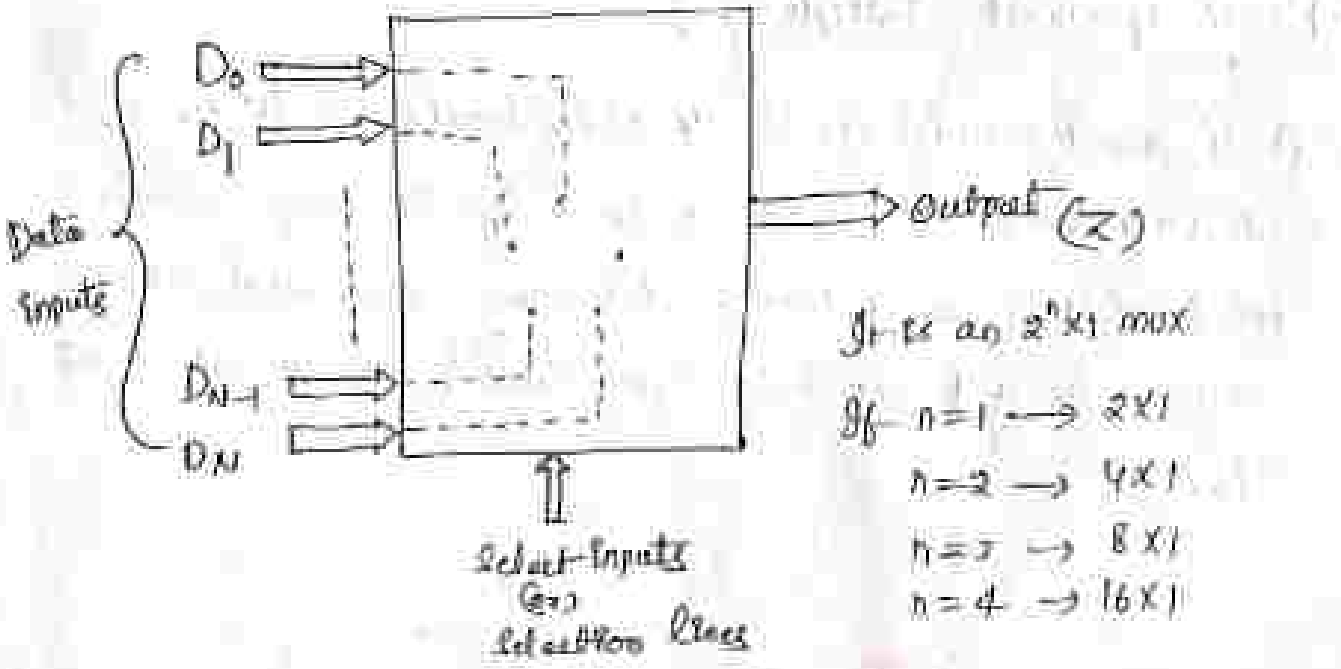
→ A multiplexer is a combinational circuit that selects binary information from one to many input lines and direct to a single output line.

→ The routing of the desired data inputs to the output is controlled by select inputs (or) selection lines.

→ For $2^n \times 1$ multiplexer 2^n input lines, '1' output line, 'n' selection lines.

* Multiplexer is a combinational Logic circuit.
* It is a parallel to serial converter.

→ As it is selecting one of the input data at a output. It is also called as **Data selector**



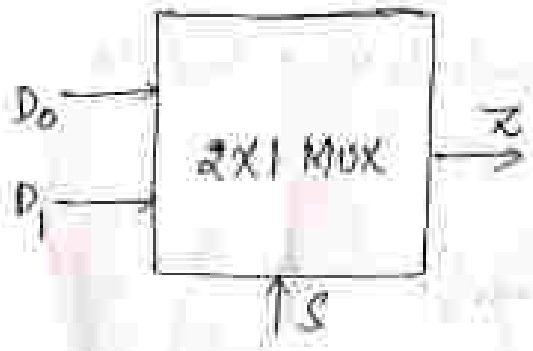
It is an $2^n \times 1$ mux

If $n=1 \rightarrow 2 \times 1$
$n=2 \rightarrow 4 \times 1$
$n=3 \rightarrow 8 \times 1$
$n=4 \rightarrow 16 \times 1$

⇒ Basic 2-Input multiplexer :-

A 2-Input multiplexer has two data inputs D_0 & D_1, one selection line S and one output Z

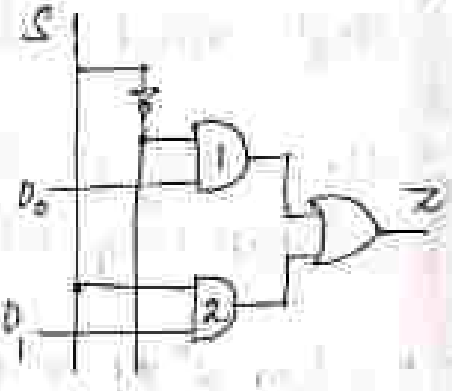
Block diagram



Truth Table

S	Z
0	D_0
1	D_1

Logic diagram



$Z = \bar{S}D_0 + SD_1$

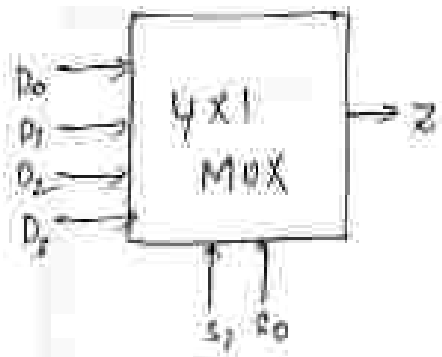
→ When $S=0$, AND gate '1' is enabled & AND gate '2' is disabled so $Z = D_0$.

→ When $S=1$, AND gate '2' is enabled & AND gate '1' is disabled so $Z = D_1$.

⇒ Basic 4-Input Multiplexer :-

A 4-Input multiplexer has 4 data inputs D_0, D_1, D_2, D_3 and two data select inputs S_0 & S_1. The Logic Levels applied to the S_0, S_1 inputs determine which AND gate is enabled, so that its data passes through the OR gate to the o/p.

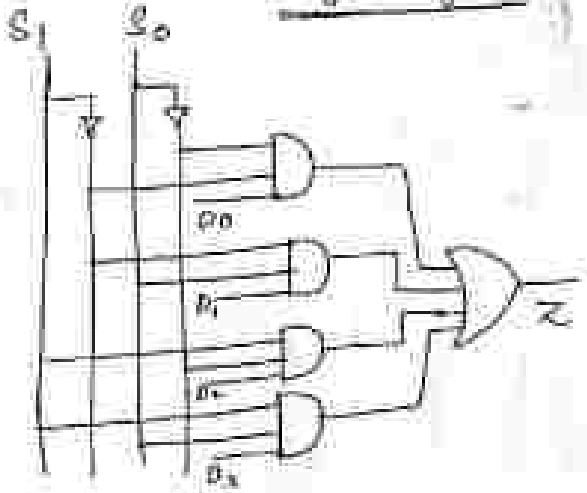
Block diagram



Truth table

S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Logic diagram



$\therefore Z = \bar{S_1}\bar{S_0}D_0 + \bar{S_1}S_0D_1 + S_1\bar{S_0}D_2 + S_1S_0D_3$

→ **16-Input multiplexer from two 8-Input multiplexer :-**

→ To design a 16-Input multiplexer from two 8-Input multiplexer one OR gate and one inverter is required. Here four select inputs S_3, S_2, S_1, S_0 will select one of 16 inputs to pass through to X.

→ The S_3 input determines which multiplexer is enabled. when $S_3 = 0$, the left multiplexer is enabled and S_2, S_1 and S_0 inputs determine which of its data input will appear at the output and pass through the OR gate to 'X'. when $S_3 = 1$ the right multiplexer is enabled and S_2, S_1 & S_0 inputs select one of the data inputs for passage to output X.

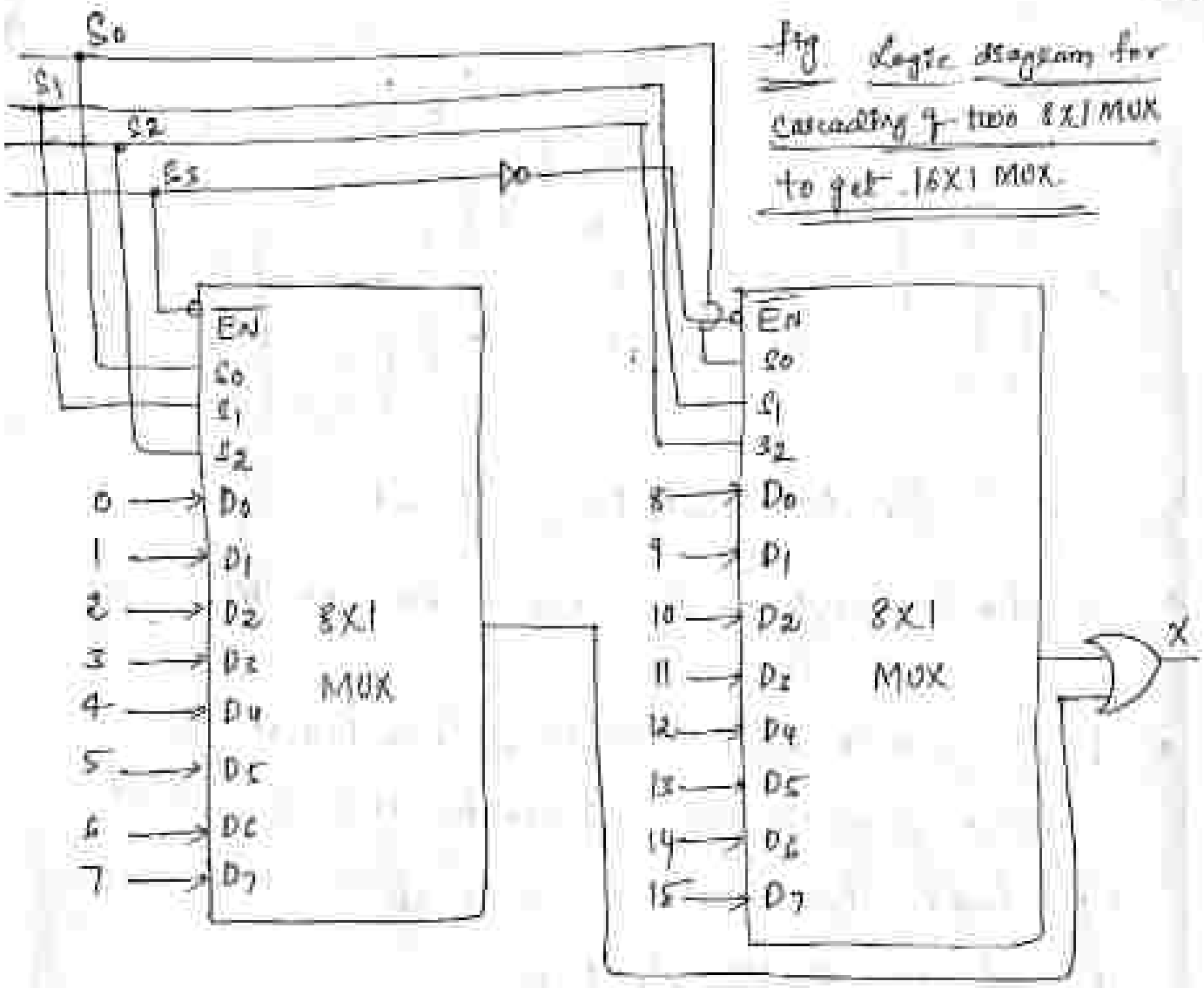


Fig Logic diagram for cascading of two 8X1 MUX to get 16X1 MUX.

⇒ Design of a 16:1 MUX using 4:1 MUX :-

To design a 16:1 MUX using 4:1 MUX, five 4:1 MUX is needed. four inputs are applied successively and 4 select input are required, select input C & D are applied to S_1 & S_0 terminals of the four multiplexers. the outputs of these mxers are connected as data inputs to the 5th 4X1 MUX & select lines A & B are applied to S_1 & S_0.

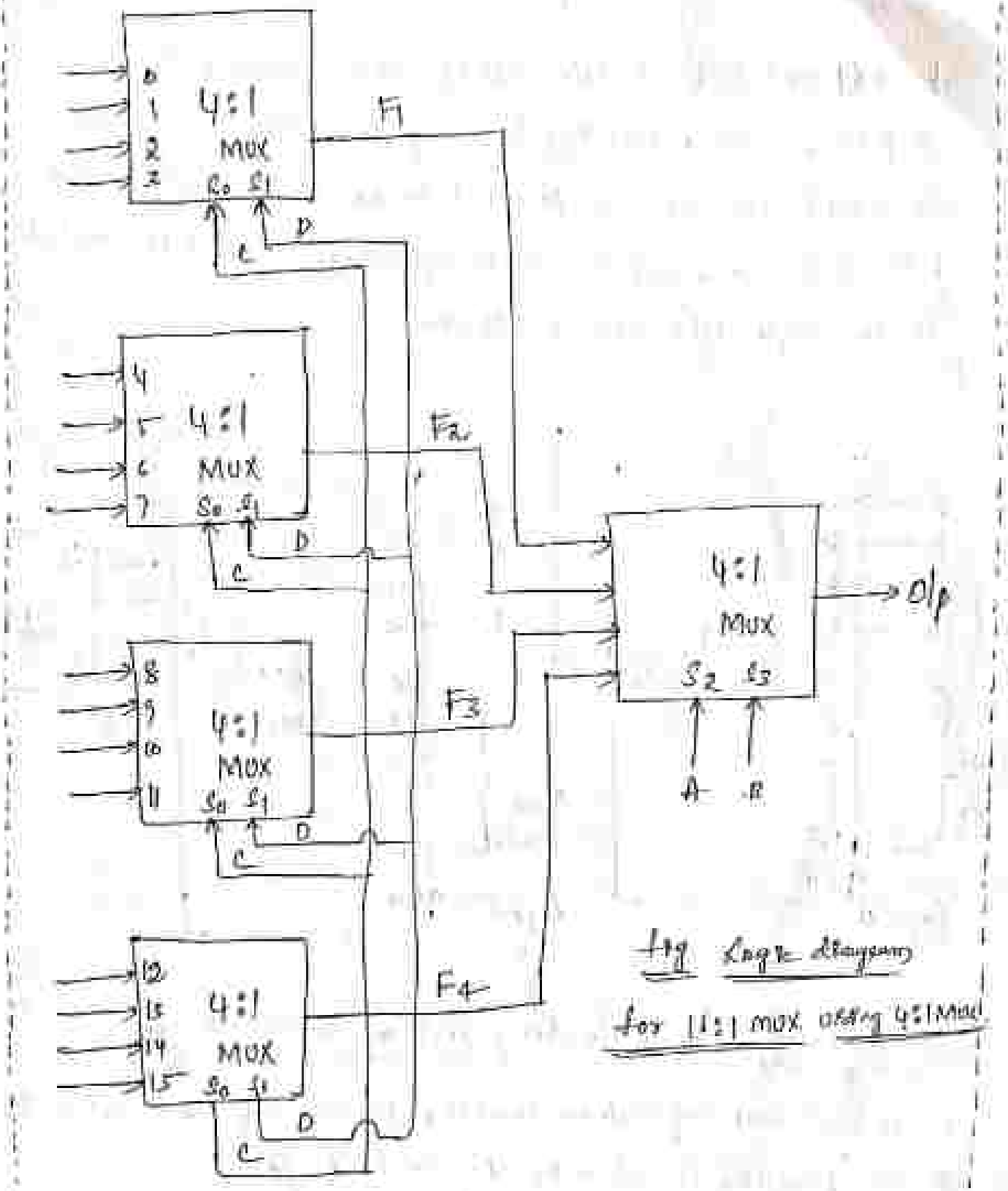
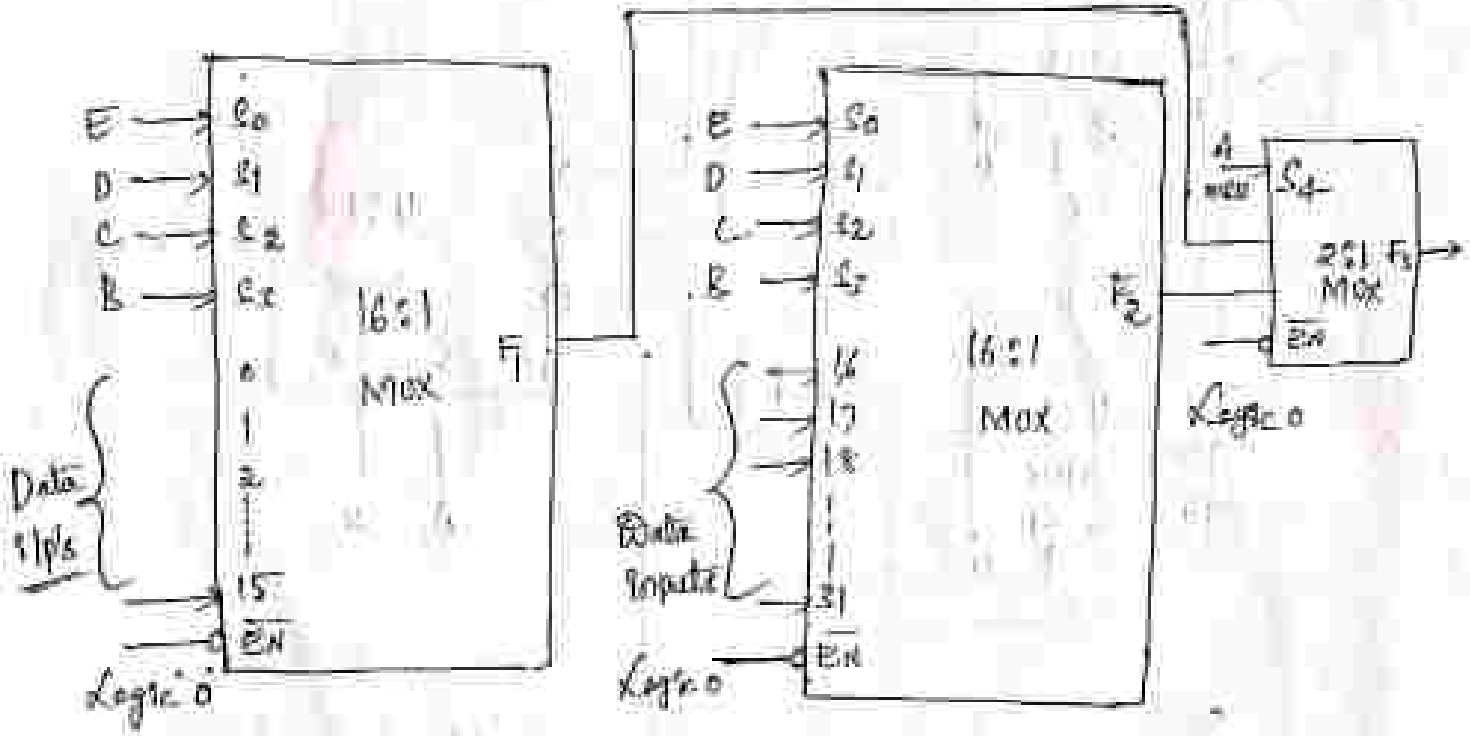


Fig Logic diagram

for 16:1 mux using 4:1 Mux

⟹ Design of 32X1 MUX using two 16X1 MUX & one 2X1 MUX :—

A 32X1 mux has 32 data inputs. So it requires five data select lines. Since a 16X1 MUX has only four data select lines, the inputs B, C, D & E are connected to the data select lines of both 16X1 muxes and the most significant input A is connected to the single data line of 2X1 mux.



⟹ Considering points while doing problems on multiplexers :—

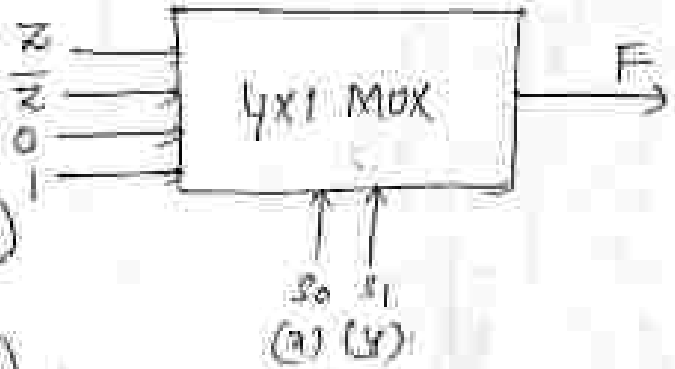
* To implement any boolean function should follow the procedure i.e
  1. the function should be in standard SOP form
  2. Based on number of variables 'n' select multiplexer having (n-1) no. of selection lines.
  3. Take any two variables as selection lines.

Q) $F = \Sigma m(1,2,6,7)$

Truth table

S_0 (x)	S_1 (y)	z	F	
0	0	0	0	F = z
0	0	1	1	
0	1	0	1	F = $\bar{z}$
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	

Block Diagram

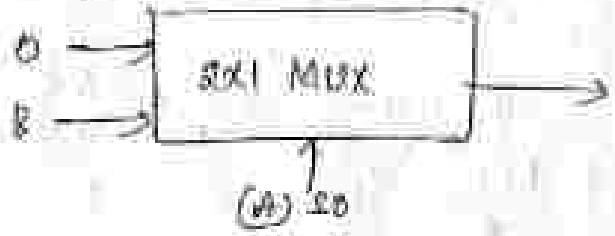


Implemented truth table

S_0	S_1	F
0	0	z
0	1	$\bar{z}$
1	0	0
1	1	1

Q) Implementation of And gate using MUX

Block Diagram



Truth table

S_0 (A)	B	Y = AB	
0	0	0	Y = 0
0	1	0	
1	0	0	Y = B
1	1	1	

Implementation truth table

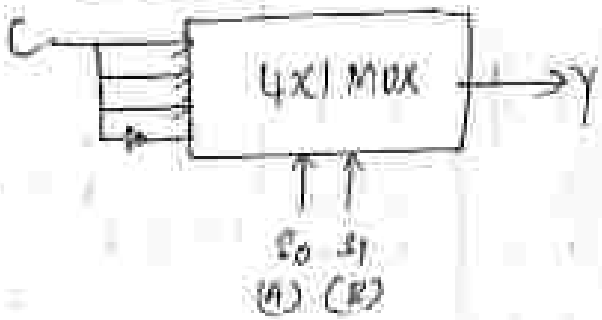
S_0	Y
0	0
1	B

(8a) Using 4x1 MUX, implement the Logic function $F(a,b,c) = \Sigma m(1,2,5,6)$

Sol

(S_0) A	(S_1) B	C	F	
0	0	0	0	F = C
0	0	1	1	
0	1	0	0	F = C
0	1	1	1	
1	0	0	0	F = C
1	0	1	1	
1	1	0	1	F = $\bar{C}$
1	1	1	0	

Block Diagram



Implementation table

S_0	S_1	Y
0	0	C
0	1	C
1	0	C
1	1	$\bar{C}$

(8b) Implement the function $F(a,b,c) = ab + \bar{b}c$ using 4x1 MUX. Convert to the canonical form.

Sol $ab(c+\bar{c}) + \bar{b}c(a+\bar{a})$

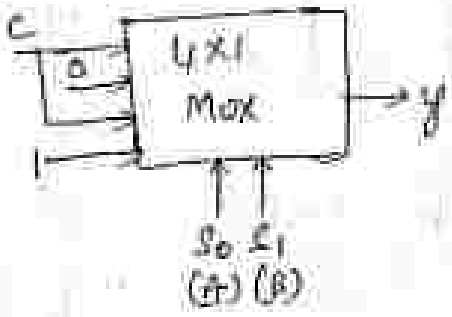
$abc + ab\bar{c} + a\bar{b}c + \bar{a}\bar{b}c$

111 110 101 001

7 6 5 1

$F(a,b,c) = \Sigma m(1,5,6,7)$

Block Diagram

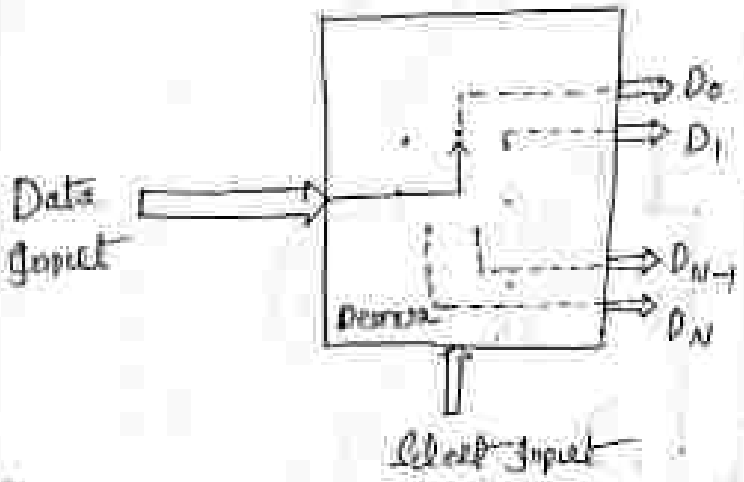


⇒ Implementation Table:-

S_0	S_1	y
0	0	c
0	1	0
1	0	c
1	1	1

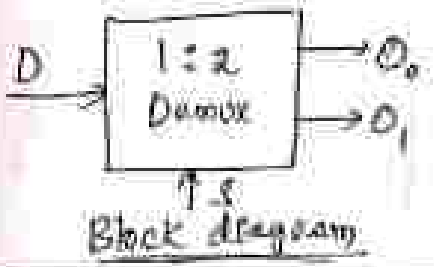
⇒ Demultiplexer :- (Data Distributor)

A Demultiplexer performs the reverse operation of multiplexer. It takes a single input and distributes it over several output. So, Demultiplexer can be called as "data distributor". Since it transmits same data to different destinations, a demultiplexer is 1 to N device.



⇒ (1 x 2) 1 to 2 Demultiplexer :-

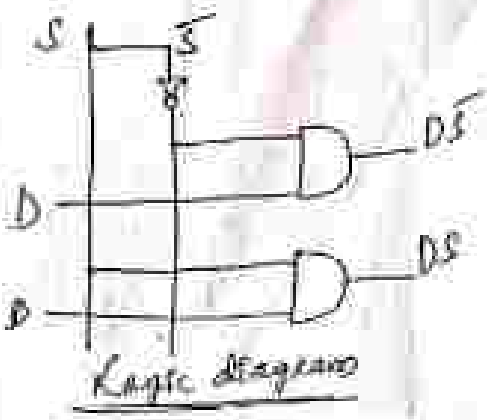
The input data line goes to all of the AND gates. The select line enable only one gate at a time, and the data appearing on the input will pass through the selected gate to the associated output line.



Block diagram

S	0
0	O_0
1	O_1

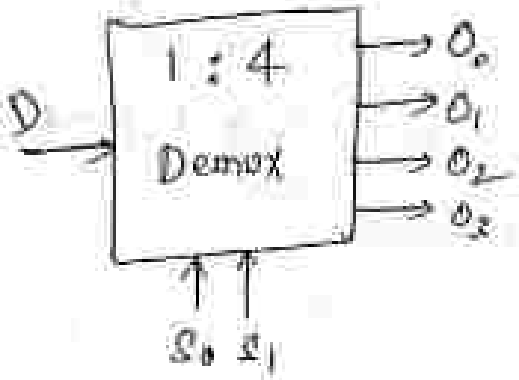
Truth Table



Logic diagram

⇒ 1 Line to 4 Line Demultiplexer :-

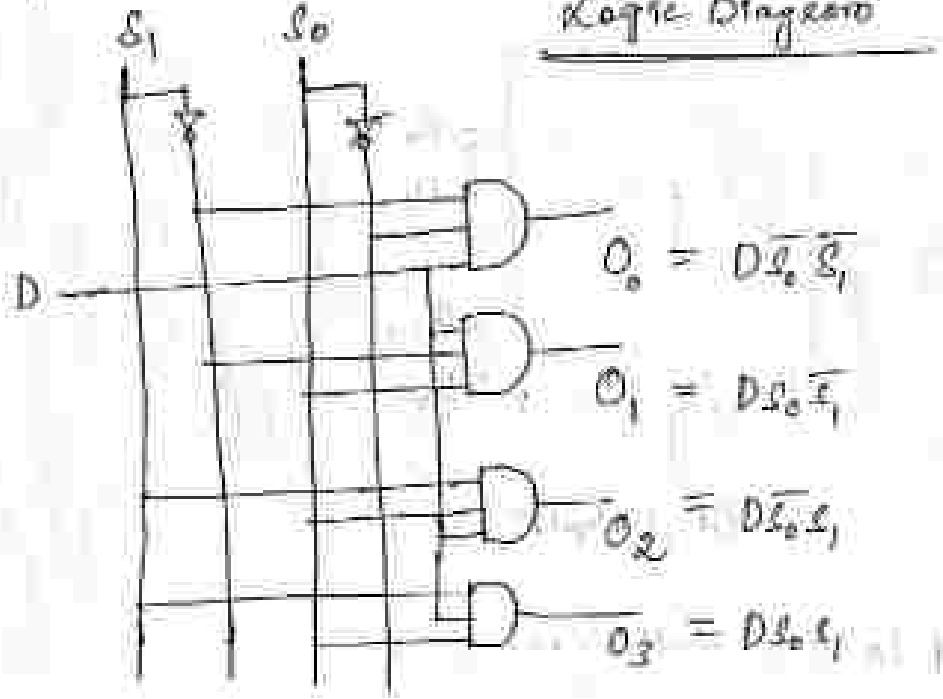
Block diagram



Truth table

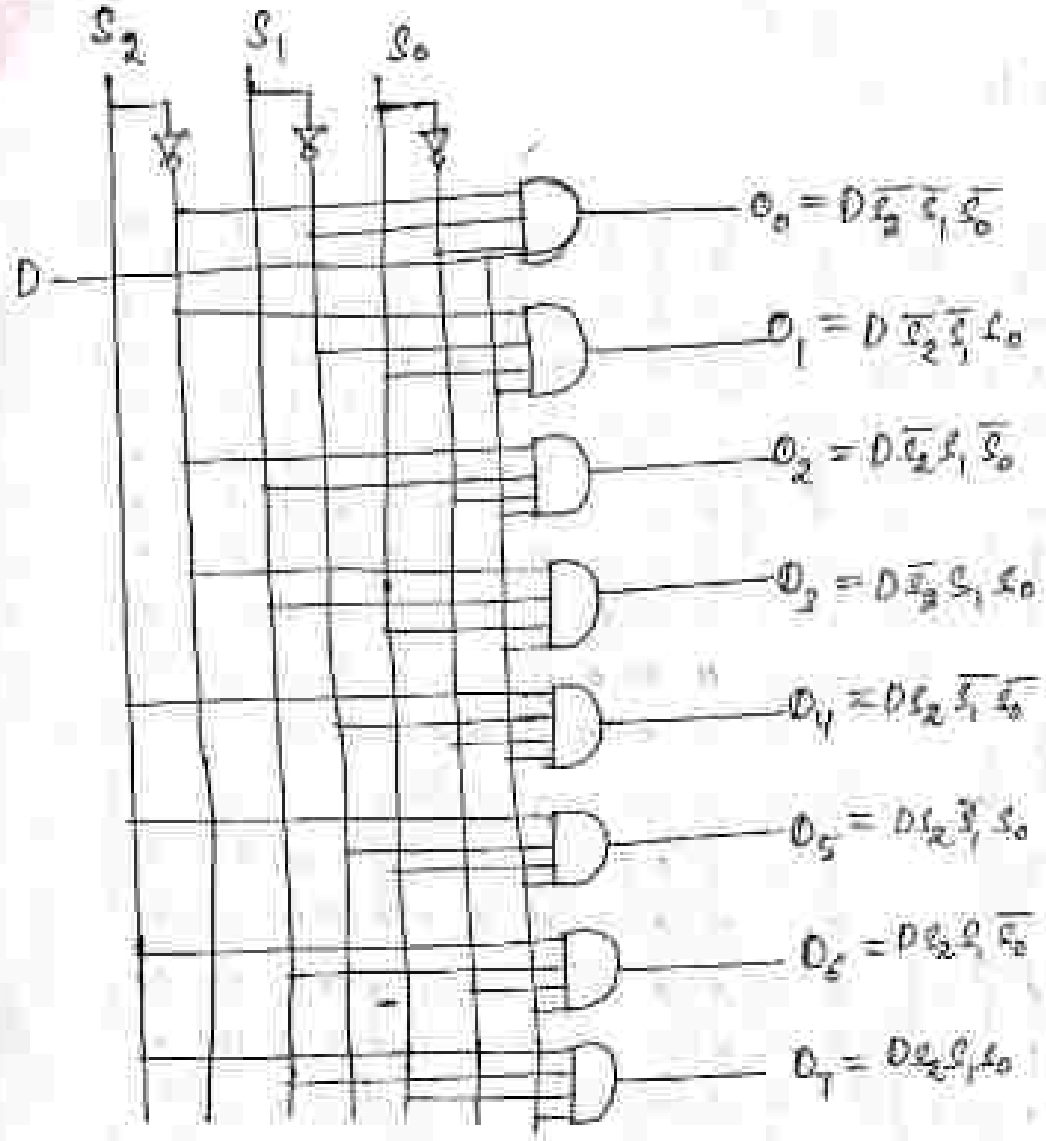
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

Logic Diagram



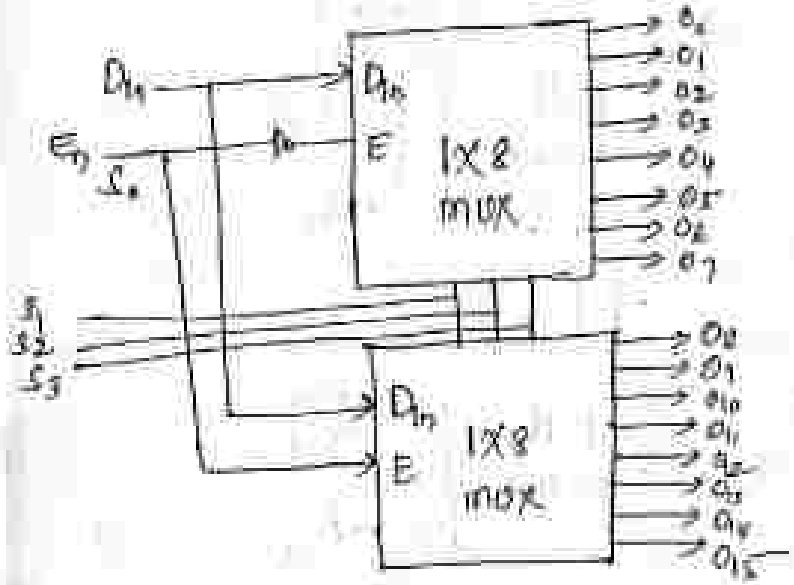
⇒ 1 Line to 8 Line Demultiplexer :-

S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	D
0	0	1	0	0	0	0	0	0	D	0
0	1	0	0	0	0	0	0	D	0	0
0	1	1	0	0	0	0	D	0	0	0
1	0	0	0	0	0	D	0	0	0	0
1	0	1	0	0	D	0	0	0	0	0
1	1	0	0	D	0	0	0	0	0	0
1	1	1	D	0	0	0	0	0	0	0



⇒ Design and Implementation of 1X16 Demultiplexer using

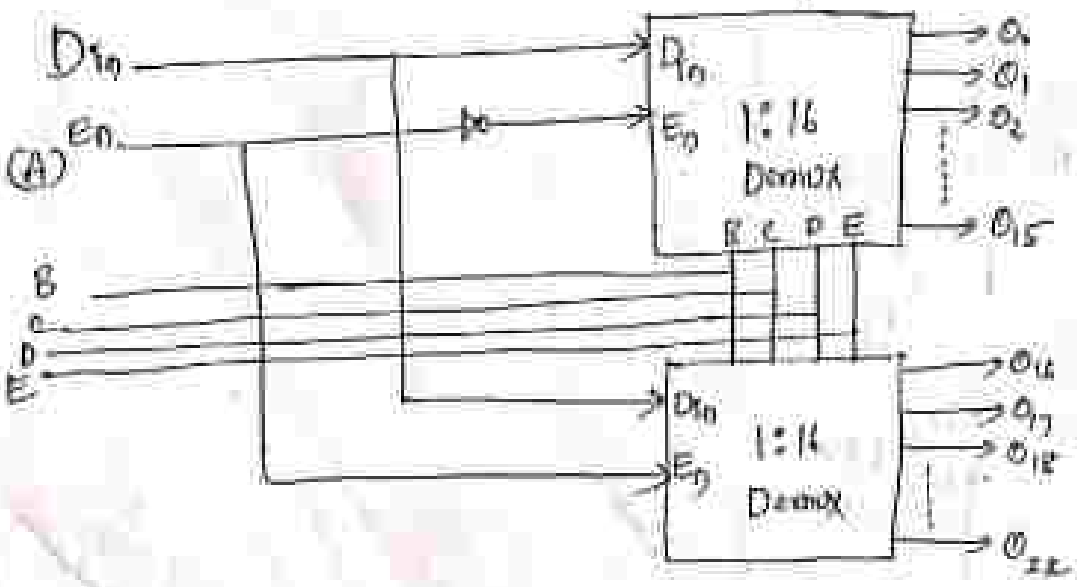
1X8 Demultiplexer :-



⇒ Truth Table :-

S_0	S_1	S_2	S_3	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}	O_{11}	O_{12}	O_{13}	O_{14}	O_{15}
0	0	0	0	D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	D	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	D	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	D	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	D	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	D	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	D	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	D	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	D	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	D	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	D	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	D	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	D	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	D	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	D	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	D

⇒ Design of 1:32 demux using two 1:16 Demux :-



⟹ Decoder :—

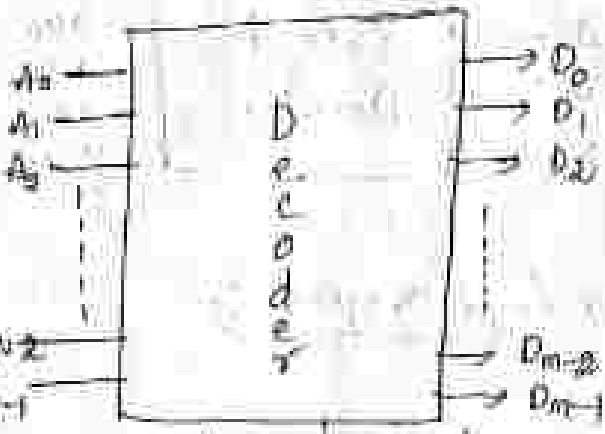
A Decoder is a logic device that converts an N-bit binary input code into 2^N output lines such that only one output line is activated for each one of the possible combinations of inputs.

⟶ Each of N inputs, there are 2^N possible input combinations.

(or) codes. For each of those input combinations only one of 2^N output lines will be active high, all other outputs will remain inactive.

⟶ Some decoders are designed to produce active low o/p, while all the other outputs remains high.

N X 2^N Decoder



2^N Input code figure

* $N = 3 \Longrightarrow 3 \times 2^3 \Longrightarrow$ 3X8 Decoder
* $n = 2 \Longrightarrow 2 \times 2^2 \Longrightarrow$ 2X4 Decoder

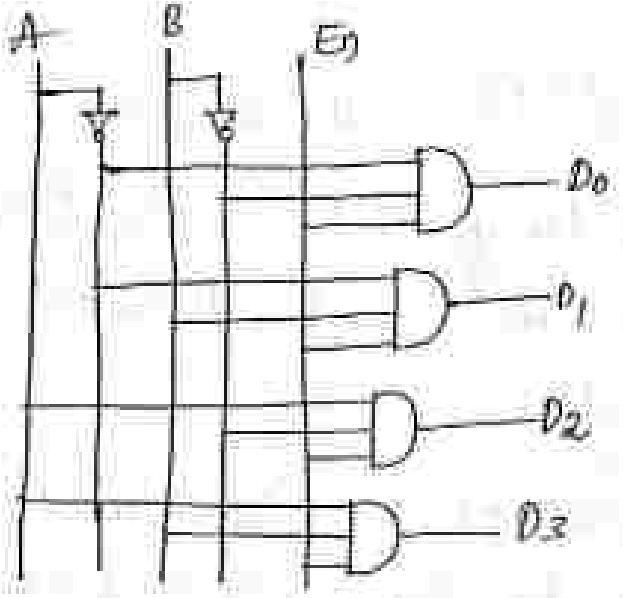
(Q) Design and implementation of 2 to 4 Decoder with active high output (or) using AND gates.

Sol

Block diagram →

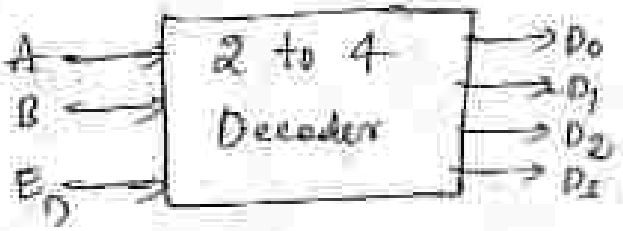
Truth Table

En	A	B	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



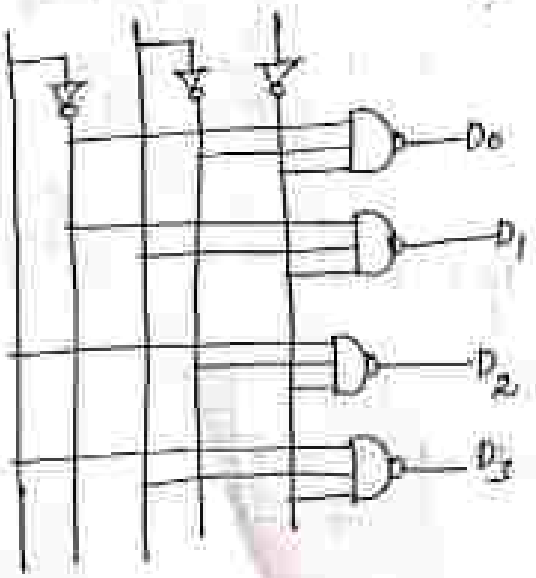
(8) Design & Implementation of 2 to 4 Decoder with active Low output (or) Using NAND gates.

Block Diagram



En	A	B	D_0	D_1	D_2	D_3
1	x	x	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

⇒ Logic Diagram :-



Note ①

→ This decoder consists of '2' input lines A & B and 4 outputs D_0 D_1 D_2 D_3. As it uses all AND gates the outputs are Active High.

Note ②

For active Low outputs NAND gates are used.

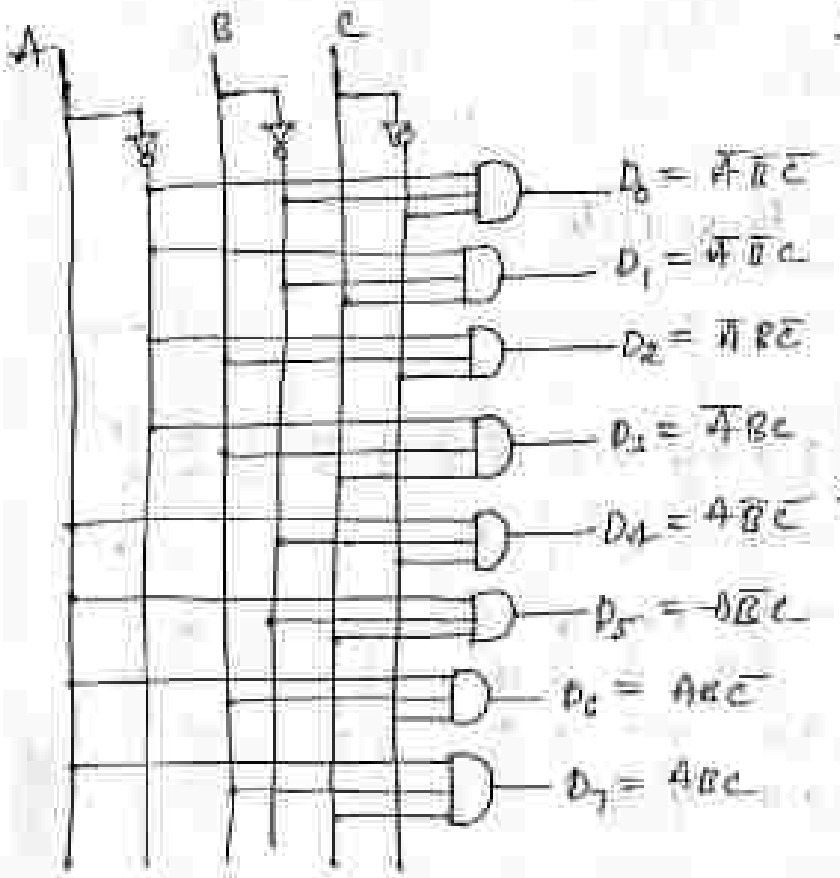
⇒ 3X8 Decoder :-

A 3 to 8 Decoder has 3 inputs and 8 outputs. It uses all AND gates, Therefore the outputs are active high. For active low outputs, NAND gates are used. It can be called a 3line to 8 line decoder because it has three input lines and eight output lines. It can also be called as a binary to Octal Decoder.

Block Diagram

Inputs			Outputs							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Truth Table

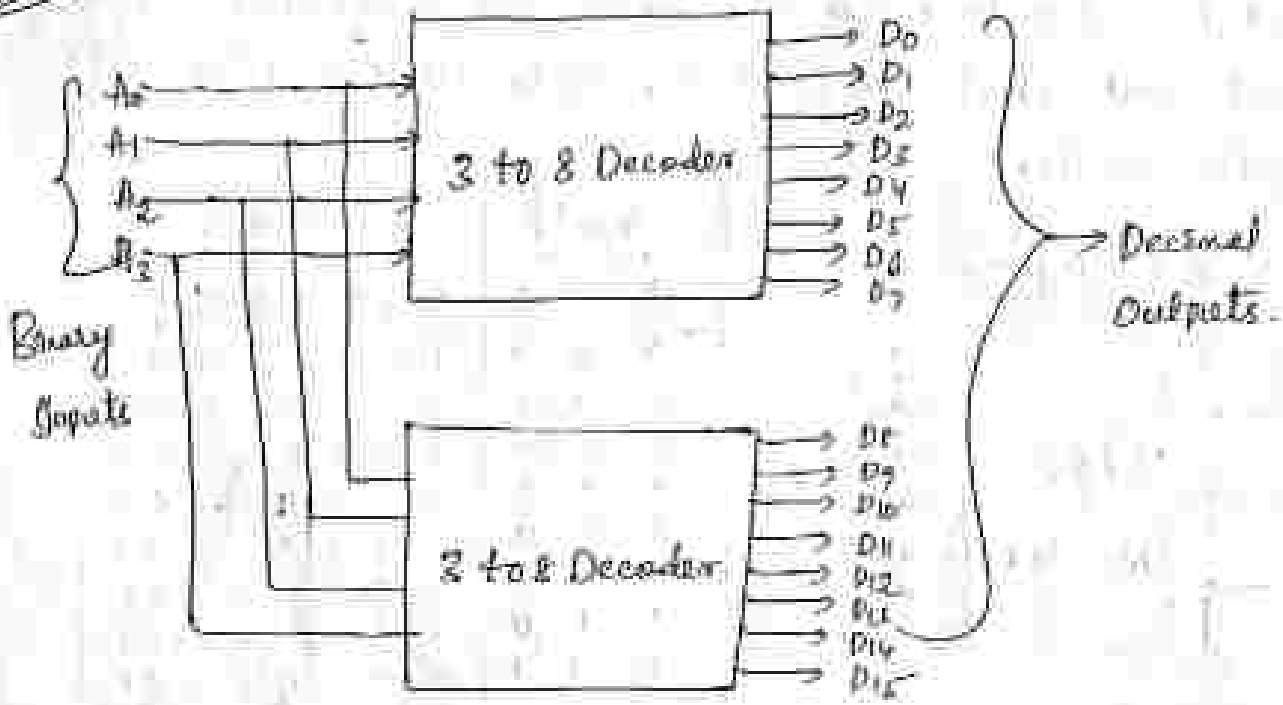


Logic diagram

(Q1) Design & implement 4 to 16 line Decoder from two 3 to 8 line Decoders

Sol

Block diagram



Truth Table

Binary Inputs				Decimal Outputs													
A3	A2	A1	A0	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Sequential Circuits - I

→ In Sequential logic circuits, the output is a function of the present inputs as well as the past inputs and outputs.

→ It consists of combinational circuits and memory elements the past values is provided by feedback from the output back to the input.

Examples of Sequential circuits are

* Counters
* Shift registers
* Serial adders
* Sequence generators

⇒ Block diagram :-

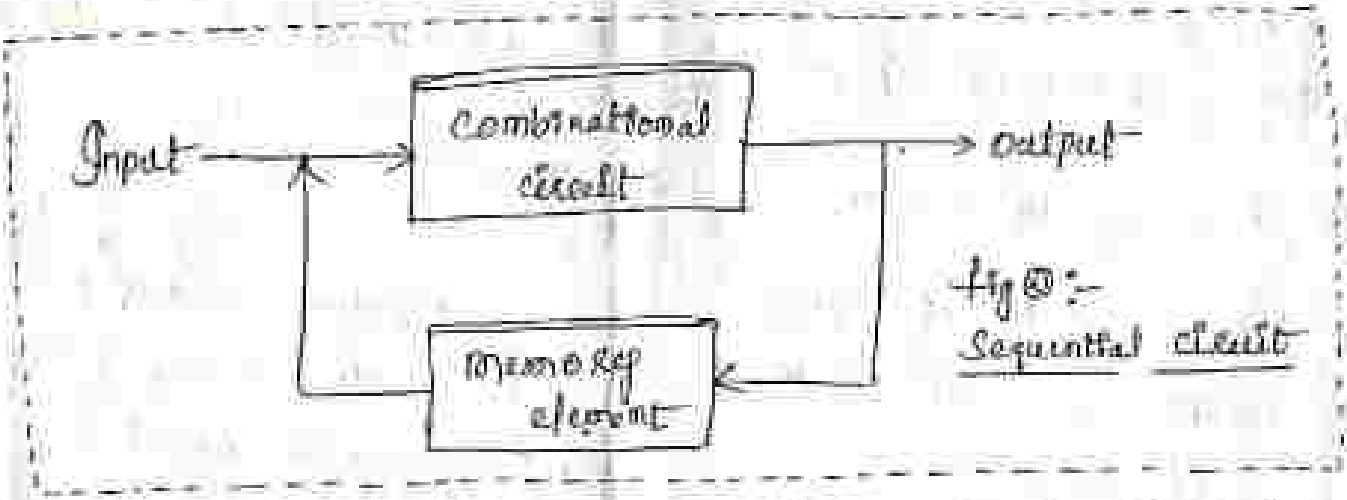


fig ① :- Sequential circuit

→ Sequential circuit includes memory element to store the past data the information stored in the memory element at any given time defines the present state of the sequential circuits.

Combinational circuit

① In combinational circuits, the output variables at any instant of time are dependent only on the present I/p variables.

② Memory unit is not required in combinational circuits.

③ combinational circuits are faster because the delay b/w the I/p and O/p is due to propagation delay of gates only.

④ It is easy to design.

⑤ The combinational logic circuits are independent of the clock.

⑥ The combinational digital circuit don't require any feedback.

⑦ It's behaviour is described by the set of output functions.

⑧ Block diagram :-

Sequential circuit

① In sequential circuits, the output variables at any instant of time are dependent not only on the present I/p variables, but also on the present state, i.e on past values of the circuit.

② Memory unit is required to store the past values of the I/p variables in sequential circuits.

③ Sequential circuits are slower than combinational circuits.

④ It is harder to design.

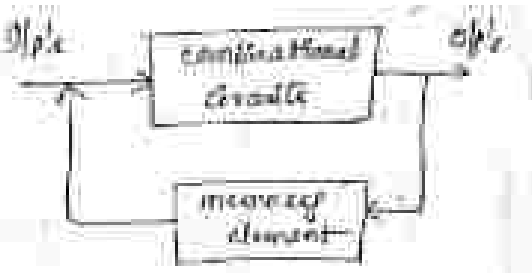
⑤ The maximum sequential logic circuits use a clock for triggering the flip-flop operation.

⑥ The sequential digital logic circuits utilize the feedback from outputs to inputs.

⑦

⑦ It's behaviour is described by the set of next state fn's and the set of o/p functions.

⑧ Block diagram :-



Q Comparision between Synchronous and Asynchronous Sequential Circuits :-

Synchronous Sequential circuits	Asynchronous Sequential circuits
(1) In synchronous circuits, memory elements are clocked Flip-Flops (FF's).	(1) In asynchronous circuits, memory elements are either unclocked Flip-Flops or time delay elements.
(2) In this, the change in i/p signals can affect memory elements upon activation of clock signal.	(2) In this, change in i/p signals can affect memory elements at any instant of time.
(3) The maximum operating speed of the clock depends on time delays involved.	(3) Because of the absence of the clock, asynchronous circuits can operate faster than synchronous circuits.
(4) They are easier to design.	(4) These are more difficult to design.

⟹ **Latch** :- The term 'Latch' is used for certain flip flops. It refers to non-clocked flip-flops.

→ Latch is a sequential device that checks all the inputs continuously and changes the outputs accordingly at anytime independent of clock signal.

→ Gated Latches (or) Clocked flip-flops are latches which respond to the inputs and Latch onto a '1' (or) '0' only when they are enabled, when the enable signal (or) gating signal is high.

→ In the absence of Enable (or) gating signal the latch doesn't respond to the changes in the inputs (here the gating signal should be a clock pulse).

→ Latch may be an 'Active high' input latch (or) an active low input latch.

→ An Active Latch (High) constructed with NOR gates.
An Active latch (low) constructed with NAND gates.

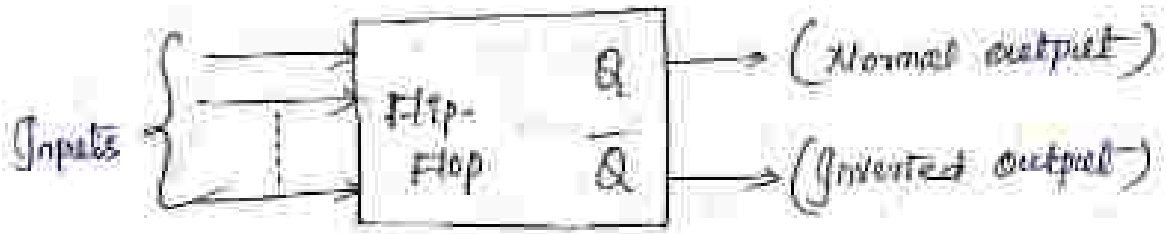
⟹ **Flip-Flops** :- The most important memory element is the flip-flop, which is made up of an assembly of logic gates.

→ There are several different gate arrangements that are used to construct flip-flops in a wide variety of ways.

→ Each type of flip-flop has special features (or) characteristics necessary for particular applications.

→ Flip-flops are the basic building blocks of sequential circuits. Actually flip-flop is an one bit memory device it can store either '1' (or) '0'.

⇒ General flip flop Symbol :-



→ The flip-flop can have one (or) more inputs, the flip flop inputs which command the flip-flop to change state are called excitations.

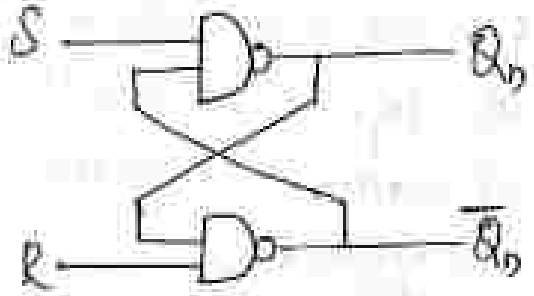
→ The applications of flip-flops are serves as a storage device if stores a '1' when its output 'Q' is a '1' and it stores a '0' its output Q is '0'. These are mainly used in Registers and Counters.

Q. Difference between Latches & Flip-Flops :-

Latch	Flip Flop
(1) A Latch is an electronic sequential logic circuit used to store information in an asynchronous arrangement.	(1) A Flip-flop is an electronic sequential logic circuit used to store information in an synchronous arrangement
(2) One Latch can store one bit information, but output state changes only in response to data input.	(2) One flip-flop can store one bit data, but output state changes with clock pulse only.
(3) Latch is an asynchronous device and it has no clock input.	(3) Flip-flop has clock input and its output is synchronised with clock pulse.
(4) Latch holds a bit value and remains constant until new inputs force it to change.	(4) Flip Flops holds a bit value and it remains constant until clock pulse is received.
(5) Latches are Level sensitive and the o/p level is high therefore as long as the level is logic Level 1 the o/p can change if the i/p	(5) Flip Flops are edge sensitive they can store the i/p only when there is either a rising (or) falling edge of clock.

⇒ SR Latch with NAND Gates / Active-Low latch :-

Circuit diagram :-

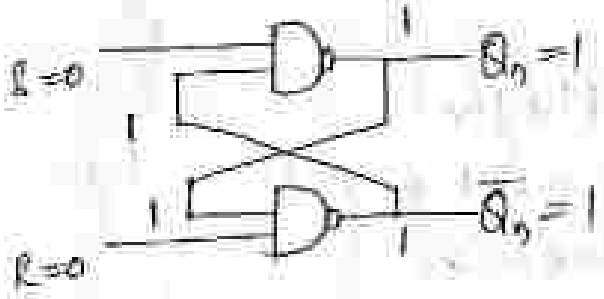


Truth Table

S	R	Q_n	$\overline{Q}_n$
0	0	Invalid state	
0	1	1	0 (set)
1	0	0	1 (Reset)
1	1	No change	

Case (i) :-

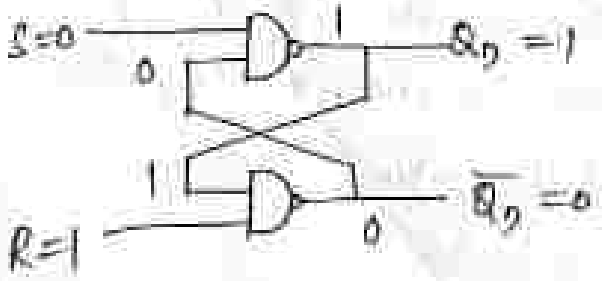
S=0, R=0



When S=0, R=0 the outputs are $Q_n=1$ & $\overline{Q}_n=1$. So, this is Invalid state / Indetermined.

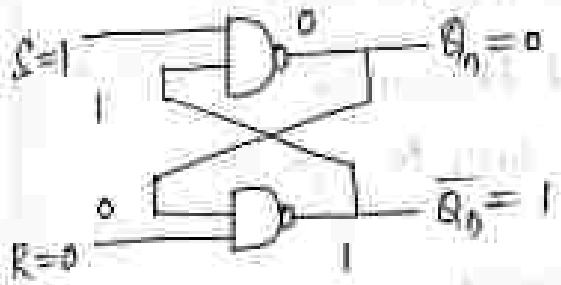
Case (ii) :-

When S=0, R=1



∴ When S=0, R=1 the outputs are $Q_n=1$ & $\overline{Q}_n=0$.

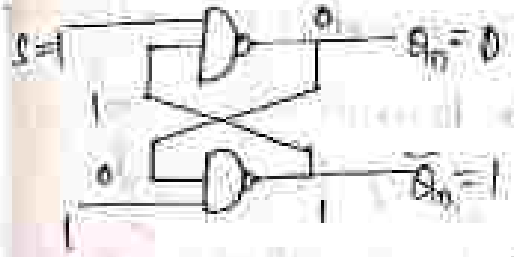
Case III :- $S=1, R=0$



∴ When $S=1, R=0$ the outputs are $Q_n=0$; $\bar{Q}_n=1$

Case IV :-

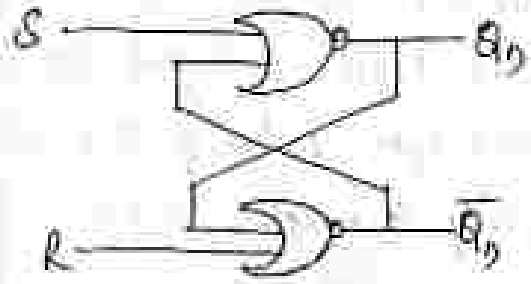
$S=1$; $R=1$



∴ When $S=1$ & $R=1$ the outputs are $Q_n=0$ & same as previous state. So, the o/p is no change state.

⟹ SR Latch with NOR Gates / Active high Latch :-

⟹ Circuit diagram :-



Truth Table

S	R	Q_n	$\bar{Q}_n$
0	0	No Change	
0	1	1	0
1	0	0	1
1	1	Invalid	

Note :- The above four cases case I, II, III, & IV, also present in the above procedure is same here also. Once we do same operation we get above truth table. NOR gates.

→ Flip-Flops :-

It is a memory element, made up of an assembly of logic gates. Flip Flop also known more formally as bistable multivibrator. Flip Flop is a one bit memory element.

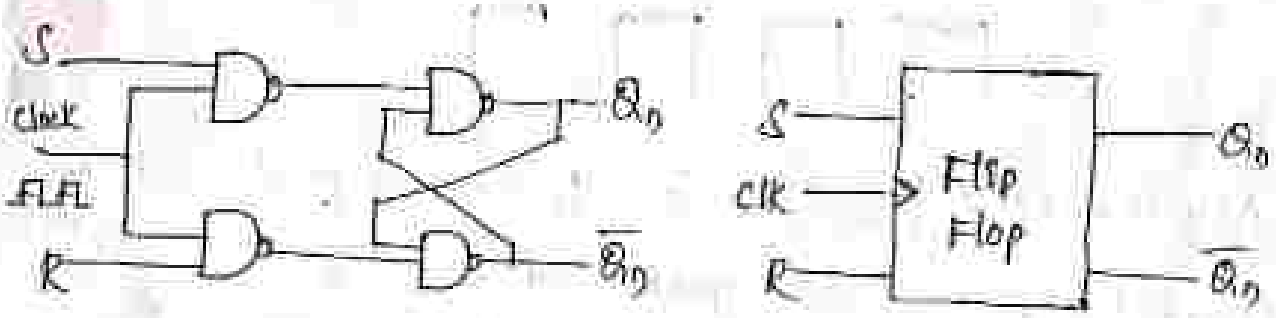
→ Flip-Flops are classified into 4 types

① SR Flip-Flop ③ JK Flip-Flop

② D Flip-Flop ④ T Flip-Flop

→ Clocked SR FlipFlop :-

→ Block diagram of SR Flip-Flop :-



(a) Circuit diagram (b) Block diagram

(c) Truth Table

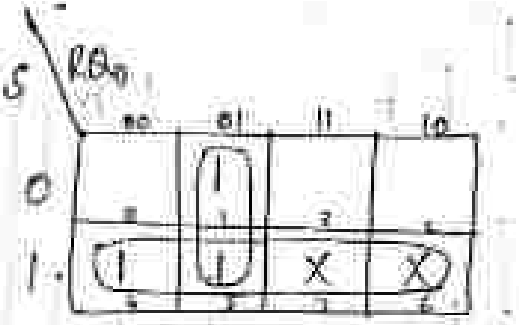
CLK	S	R	Q_n $\overline{Q_n}$
1	0	0	No change
1	0	1	0 1 (Reset)
1	1	0	1 0 (Set)
1	1	1	Invalid state

(d) **Characteristic Table :-**

From truth table find characteristic table.

CLK	S	R	Q_n	Q_{n+1}
↓	0	0	0	0
↓	0	0	1	1
↓	0	1	0	0
↓	0	1	1	0
↓	1	0	0	1
↓	1	0	1	1
↓	1	1	0	X
↓	1	1	1	X

(e) **Characteristic equation**

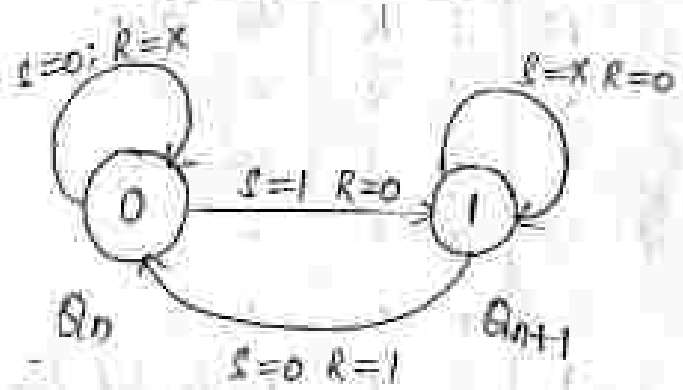


$$Q_{n+1} = S + \bar{R} Q_n$$

(f) **Excitation table :-**

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(g) **State diagram**



① $Q_n\ Q_{n+1}$: 0 0 → S R : 0 X ✓

② $Q_n\ Q_{n+1}$: 0 1 → 1 0 ✓

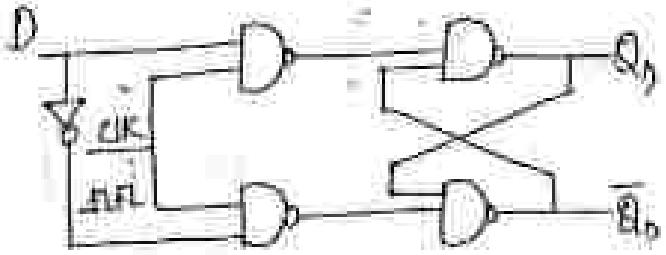
③ 1 0 → 0 1 ✓

④ 1 1 → 0 X ✓

* In SR Flip Flop S=1 & R=1 then the state is indeterminate state. This drawback is avoided in JK Flip Flop.

⇒ D-Flip-Flop :- (Delay flip flop)

(a) Circuit diagram :-



(b) Block diagram

(c) Truth Table :-

CLK	D	Q_n	$\bar{Q}_n$
1	0	0	1
1	1	1	0

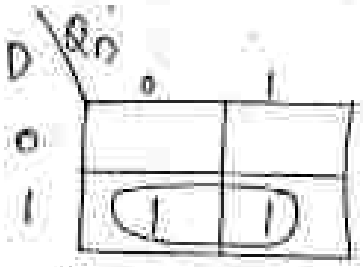
→ D Flip-Flop is also called as Delay (or) Data Flip Flop. It is used to find delay b/w like Set and Reset.

(d) Characteristic Table :-

It is used to find the next state.

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

(e) Characteristic equation

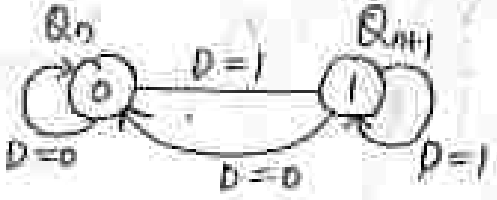


$$Q_{n+1} = D$$

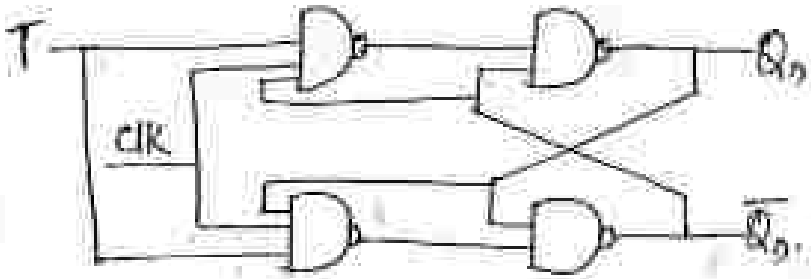
(f) Excitation Table

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

(g) State diagram :-



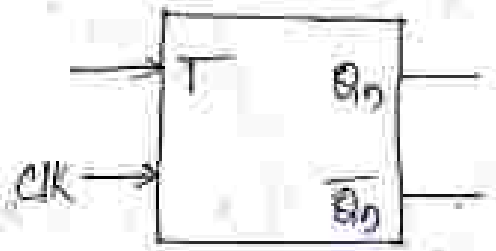
⇒ T- Flip Flop :-



(a) Logic diagram

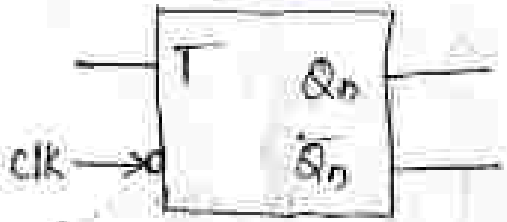
(b) positive edge T-F/F

Symbol



Negative edge T-F/F

Symbol



(c) Truth Table

CLK	T	Q_n $\overline{Q_n}$	
1	0	Q_n $\overline{Q_n}$	→ no change state
1	1	Toggle state	

(d) Characteristic Table

CLK	T	Q_n	Q_{n+1}
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(e) Excitation Table

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

(f) Characteristic equation

T \ Q_n	0	1
0		①
1	①	

$$Q_{n+1} = T\overline{Q_n} + \overline{T}Q_n$$

COUNTERS :-

→ A digital counter is a set of flip-flop (FFs) whose state change in response to pulses applied at the input to the counter.

→ A counter is used to count pulses.

→ A counter can also be used as a frequency divider to obtain wave-forms with frequencies that are specific fractions of the clock frequency.

→ They are also used to perform the timing function as in digital watches, to create delays, to Produce non-sequential binary counts, to generate pulse trains, and to act as frequency counters.

→ counters are classified into
  (i) Asynchronous counters
  (ii) Synchronous counters.

→ Asynchronous counters also called as ripple counters (or) series counters.

Comparison of Synchronous and Asynchronous counters

Asynchronous counters	Synchronous counters
1. In this type of counter FFs are connected in such a way that the output of first FF drives the clock for the second FF, the output of the second is the clock of the third and so on.	1. In this type of counter there is no connection between the output of first FF and clock input of next FF and so on.
2. All the FFs are not clocked simultaneously	2. All the FFs are clocked simultaneously.
3. Design and implementation is very simple even for more number of states	3. Design and implementation becomes tedious and complex as the number of states increases
4. Main drawback of these counters is their low speed as the clock is propagated through a number of FFs before it reaches the last FFs	4. Since clock is applied to all the FFs simultaneously the total propagation delay is equal to the propagation delay of only one FF. Hence they are faster.

Asynchronous Counters :-

→ To design an asynchronous counter, first write the counting sequence, then tabulate the values of reset signal R for various states of the counter and obtain the minimal expression for R or $\overline{R}$ using k-map or any other method.

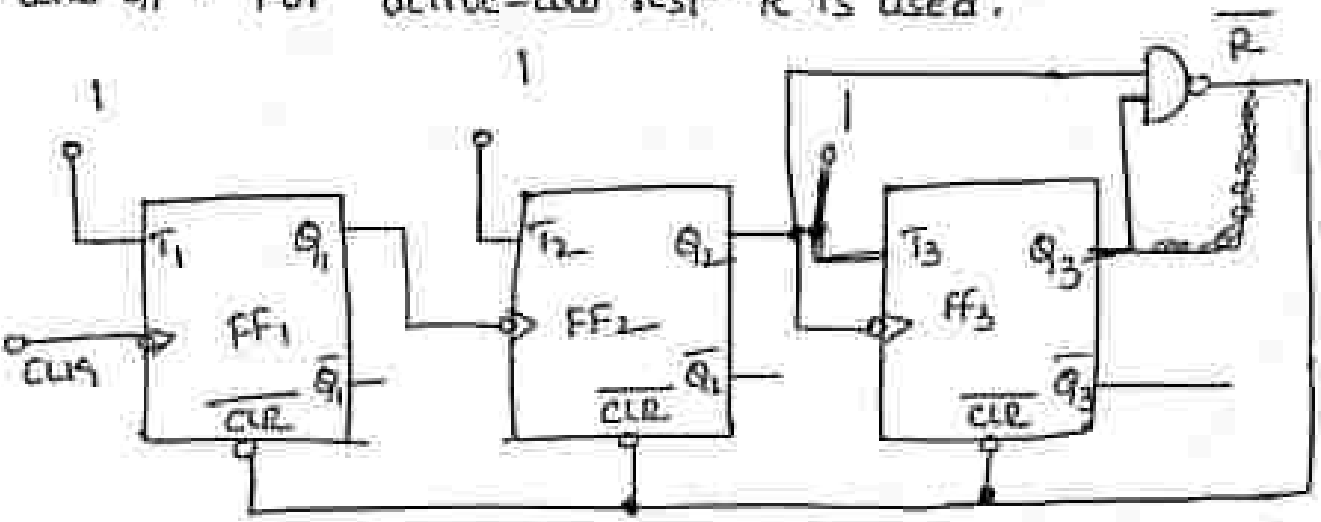
→ Provide a feedback such that R or $\overline{R}$ resets all the FFs after the desired count.

Design of a mod-6 asynchronous counter using T FFs

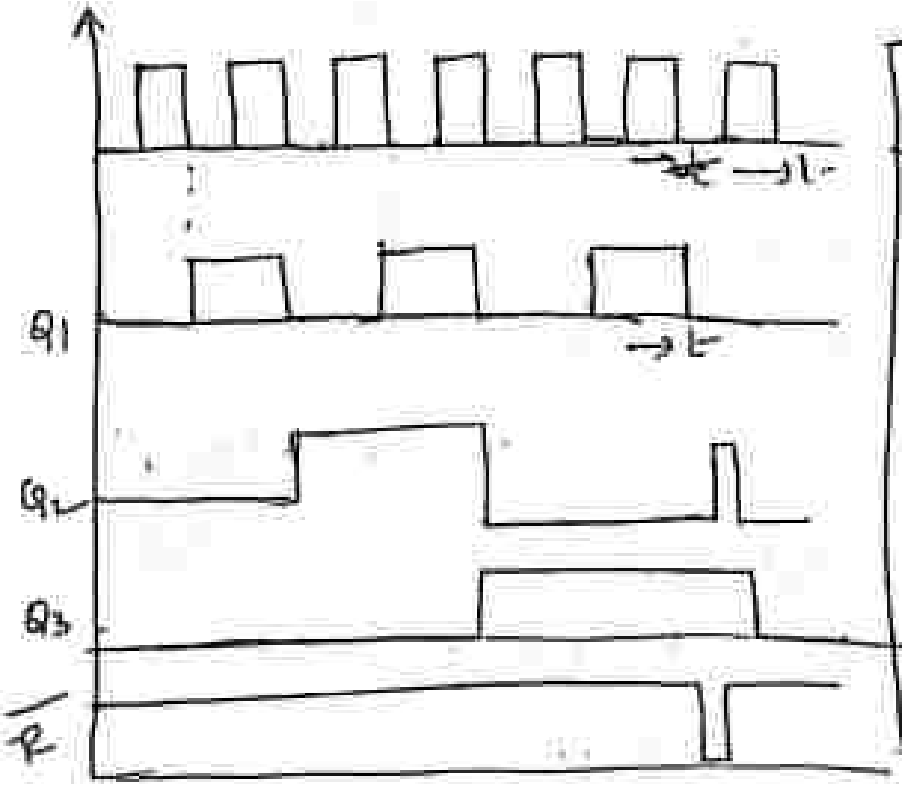
A mod-6 counter has six stable states 000, 001, 010, 011, 100, and 101. When the sixth clock pulse is applied, the counter temporarily goes to 110 state, but immediately resets to 000 because of the feed back provided.

→ Here we are using 3 FFs for designing, three FFs can have eight possible states, out of which only six utilized and the remaining two states 110 and 111 are invalid.

→ For the design, write a truth table with the present state outputs Q_3, Q_2 and Q_1 as the variables, and reset R as the output and obtain an Expression for R in terms of Q_3, Q_2 and Q_1. For active-low rest $\overline{R}$ is used.



Logic diagram.



After Pulses	State Q_3	Q_2	Q_1	$\overline{R}$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
	↓	↓	↓	
	0	0	0	0
	0	0	1	0

Truth table.

→ Design of mod-10 asynchronous counter using T FFs :-

→ A mod-10 counter is a decade counter. It is also called a BCD counter or a divide-by-10 counter. It requires 4 FFs.

→ This counter has ten stable states 0000 through 1001, i.e it counts from 0 to 9.

→ The initial state is 0000 and after nine clock pulses it goes to 1001. When the tenth clock pulse is applied, the counter goes to state 1010 temporarily, but because of the feed back provided, it resets to initial state 0000.

→ So, there will be a glitch in the waveform of Q_2. The state 1010 is a temporary state for which the reset signal R=1, R=0 for 0000 to 1001, and R=X (don't care) for 1011 to 1111.

Shift Register counters :-

→ one of the applications of shift registers is that they can be arranged to form several types of counters.

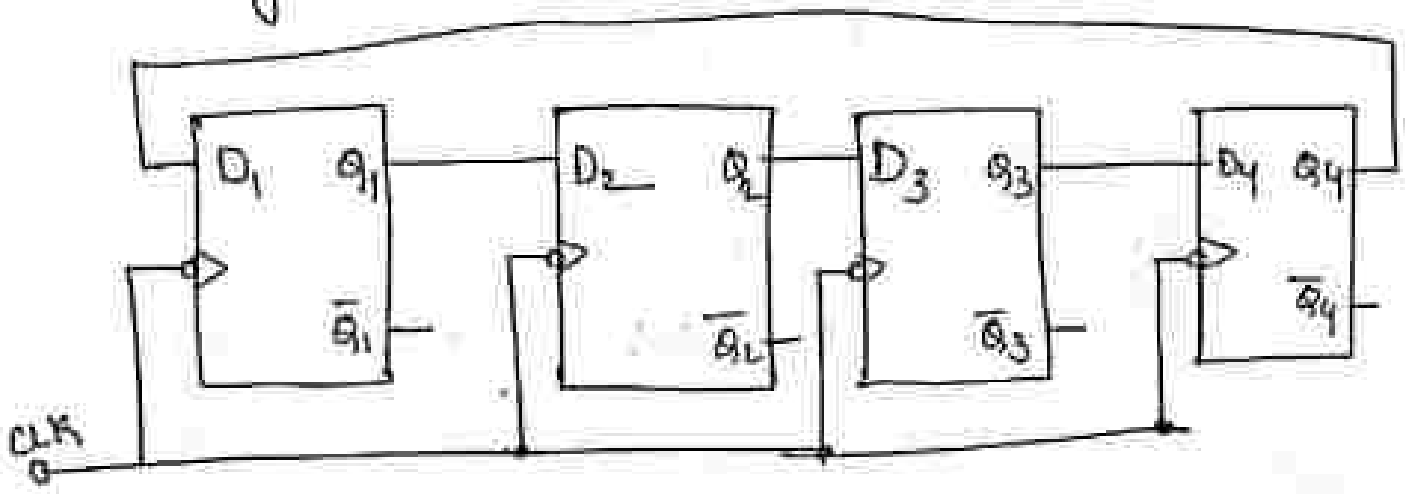
→ Shift register counters are obtained from serial-in, serial-out shift registers by providing feedback from the output of the last FF to the input of the first FF. These devices are called counters because they exhibit a specified sequence of states.

→ The mostly used shift register counter is the ring counter, as well as the twisted ring counter.

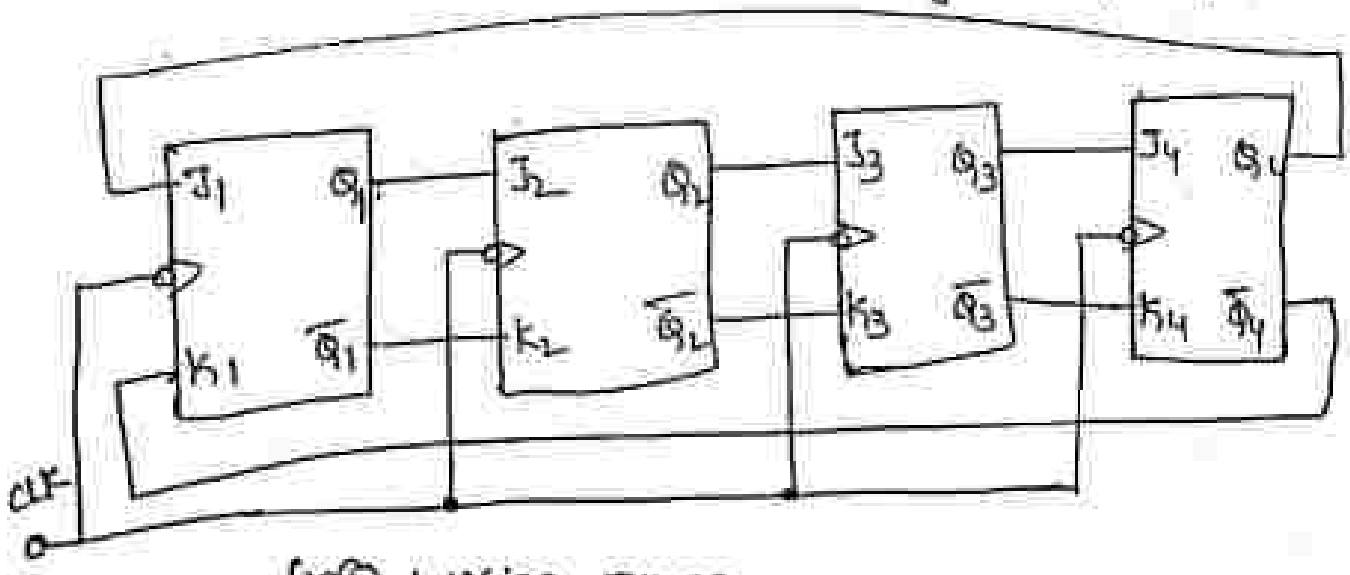
Ring counter :-

→ This is the simplest shift register counter. the basic ring counter using DFFs is shown below-fig (a). The realization of this counter using J-K FFs is shown in fig (b).

→ The FFs are arranged as in a normal shift register, i.e the Q output of each stage is connected to the D input of the next stage, but the Q output of the last FF is connected back to the D input of the first FF such that the array of FFs is arranged in a ring and therfore, the name ring counter.



fig(a):- logic diagram of 4-bit ring counter using D-flip-flop



fig(b): using JK FFs.

→ In most instances, only a single 1 is in the register and is made to circulate around the registers as long as clock pulses are applied.

→ Initially, the first FF is preset to a 1. So, the initial state is 1000, i.e $Q_1=1$, $Q_2=0$, $Q_3=0$ and $Q_4=0$. After each clock pulse, the contents of the register are shifted to the right by one bit and Q_4 is shifted back to Q_1.

→ The sequence repeats after four clock pulses. The number of distinct states in the ring counter, i.e the mod of the ring counter is equal to the number of ffs used in the counter.

→ An n-bit ring counter can count only n bits, where as n-bit ripple counter can count 2^n ~~states~~ bits.

→ It is entirely a synchronous operation and requires no gates external to FFs, it has the further advantage of being very fast.

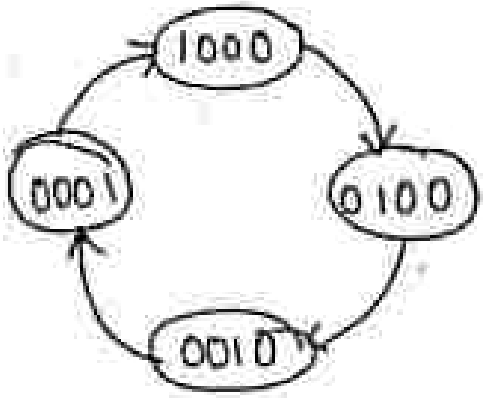
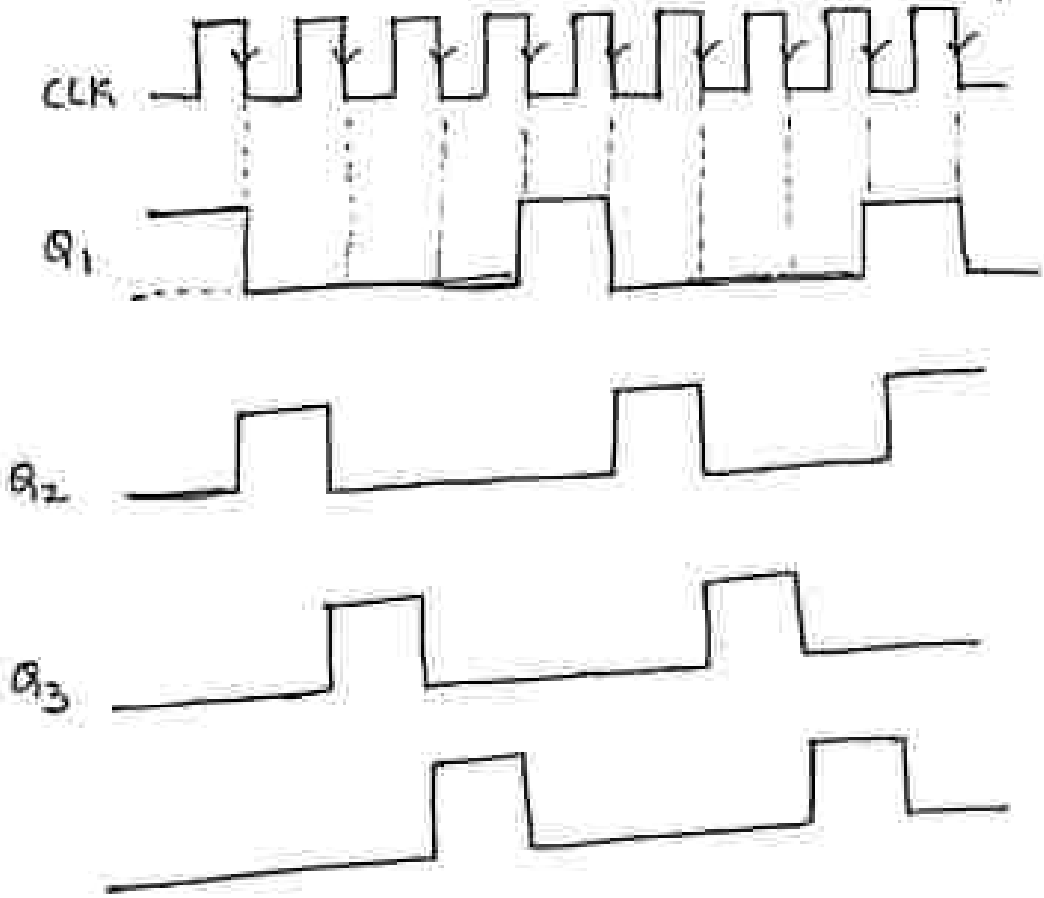


Fig :- State diagram.

Q_1	Q_2	Q_3	Q_4	After clock pulse
1	0	0	0	0
0	1	0	0	1
0	0	1	0	2
0	0	0	1	3
1	0	0	0	4
0	1	0	0	5
0	0	1	0	6
0	0	0	1	7

Sequence table



Timing diagram of a 4-bit ring counter