SYLLABUS

	TOPIC
1.	**Number System & Boolean Algebra Number System :** Binary, Octal, Decimal, Hexadecimal; Conversion of Number System; Binary Arithmetic & Complement, Binary Codes: Weighted & Non Weighted, Gray Code, Excess-3 Code. Boolean Function; Boolean Postulates; De-Morgan's Theorem; Boolean Expressions: Sum of Product, Product of Sum, Minimization of Boolean Expressions using K-Map; Logic Gates: AND, OR, NOT, NAND, NOR, XOR, XNOR; Implementations of Logic Functions using Gates; NAND- NOR Implementations; Multilevel gate Implementations.
2.	**Combinational Circuits Adders & Subtractors :** Half Adder, Full Adder, Binary Adder, Half Subtractor, Full Subtractor, Adder Subtractor; Magnitude Comparator: Two Bit Magnitude Comparator, Three Bit Magnitude Comparator; Multiplexer & De-Multiplexer: 4*1 Multiplexer, 8*1 Multiplexer; Decoder & Encoder; Parity Checker & Generator; Code Converter.
3.	**Sequential Circuit: Introduction to Flip Flops :** SR, JK, T, D, Master Slave Flip Flops; Conversion of Flip Flops; Characteristic Table & Equation; Edge Triggering & Level Triggering; Excitation Table; State Diagram; State Table; State Reduction; Design of Sequential Circuits.
4.	**Registers Introduction of Registers :** Classification of Registers; Register with Parallel Load; Shift Registers; Bidirectional Shift Register with Parallel Load. Counters Introduction of Counter; Asynchronous/Ripple Counters; Synchronous Counters; BCD Counter; 4-bit Binary Counter with Parallel Load; Design of Synchronous Counters; Ring Counter; Johnson Counter.
5.	**Memory Organization :** Basic cell of static and dynamic RAM; Building large memories using chips; Associative memory; Cache memory organization and Virtual memory organization.

r transmitted
r mechanical
ther.
pital,
@gmail.com
book neither
d omissions.

NUMBER SYSTEM & BOOLEAN ALGEBRA

1. ***Convert the given decimal number to equivalent octal number :***
$(125)_{10} \rightarrow (?)_8$ **(2023-24)**

Solution :

8	125	5
8	15	7
	1	1

$(125)_{10} \rightarrow (175)_8$

2. ***Simplify the following Boolean expression using Boolean algebra-***
$B\,(A + B')\,(B + C)$ **(2023-24)**

Solution : To simplify the boolean expression $B(A+B')(B+C)$, we can use Boolean algebra laws such as distributive law, complement law, and identity law.

(1) Apply the Distributive Law : $XY + XZ = X(Y + Z)$
$B(A + B')(B + C) \Rightarrow B(A\cdot(B + C) + B'\cdot(B + C))$

(2) Apply the Distributive Law Again : $AB + AC = A(B + C)$
$B(A\cdot(B + C) + B'\cdot(B + C)) \Rightarrow B((AB + AC) + (B'\cdot(B + C)))$

(3) Apply the Identity Law : $AB + A'B = A + B$
$B((AB + AC) + (B'\cdot(B + C))) \Rightarrow B((A + B) + (B'\cdot(B + C)))$

(4) Apply the Distributive Law Again : $XY + XZ = X(Y + Z)$
$B((A + B) + (B'\cdot(B + C))) \Rightarrow B((A + B) + (B'B + B'C))$

(5) Apply the Complement Law : $XX' = 0$
$B((A + B) + (B'B + B'C)) \Rightarrow B((A + B) + (0 + B'C))$

(6) Apply the Identity Law : $A + 0 = A$
$B((A + B) + (0 + B'C)) \Rightarrow B((A + B) + B'C)$

[A-2]

(7) **Apply the Complement Law :** $XX' = 0$
$B((A + B) + B'C) \Rightarrow B(A + B + B'C)$

(8) **Apply the Absorption Law :** $X + XY = X$
$B(A + B + B'C) \Rightarrow B(A + B)$

(9) **Apply the Absorption Law Again :** $X + XY = X$
$B(A + B) = B$

So, the simplified expression is B.

3. ◆ ***Explain all Logic gates with diagram & truth table.*** **(2023-24)**

◆ ***Explain all logic gates with truth table and logic circuit design.*** **(2016)**

There are seven basic logic gates : AND, OR, XOR, NOT, NAND, NOR, and XNOR.

(1) AND Gate : The AND gate is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. The output is "true" when both inputs are "true." Otherwise, the output is "false."

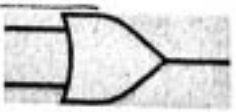
A·B
AB
A AND B

(Figure : AND Gate)

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

(2) OR Gate : The OR gate gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false".

A+B
A OR B



(Figure : OR gate)

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

(3) XOR
the sa
"true"
The o
both i
circui
are di

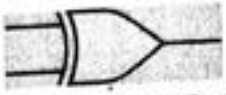
(4) NOT
some
othe
inpu

(5) NA
gat
the
ou
th

(6) N
fo
in

SCAN AnyScanner

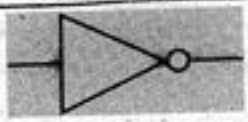
(3) XOR Gate : The XOR (exclusive - OR) gate acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true." The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observe that the output is 1 if the inputs are different, but 0 if the inputs are the same.



(Figure : XOR Gate)

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

(4) NOT Gate or Inverter : A logical inverter, sometimes called a NOT gate to differentiate it forms other types of electronic inverter devices, has only one input. It reverses the logic state.



(Figure : Inverter or NOT Gate)

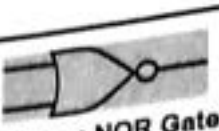
Input	Output
1	0
0	1

(5) NAND Gate : The NAND gate operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true." Otherwise, the output is "true."

(Figure : NAND Gate)

Input 1	Input 2	Output
0	0	1
0	1	1
1	0	1
1	1	0

(6) NOR Gate : The NOR gate is a combination OR gate followed by an inverter. Its output is "true" if both inputs are "false." Otherwise, the output is "false."



(Figure : NOR Gate)

Input 1	Input 2	Output
0	0	1
0	1	0
1	0	0
1	1	0

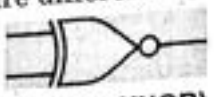
(7) XNOR Gate : The XNOR (exclusive-NOR) gate is a combination XOR gate followed by an inverter. Its output is "true" if the inputs are the same, and "false" if the inputs are different.

compliment of XOR

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

$\bar{A}\bar{B}$

$(A \odot B)$



(Figure : XNOR)

Input 1	Input 2	Output
0	0	1
0	1	0
1	0	0
1	1	1

even no. of 1 => true

4. *Simplify the POS using K-map :*
F(A, B, C, D) = $\Pi M(1,3,5,7,8,9,10,11,12,13,14,15)$
(2023-24)

Solution :

To simplify the given Boolean expression *F(A, B, C, D)* using a Karnaugh map (K-map), we first need to plot the given minterms and then group them into the largest possible groups of 1s. Each group should have a size that is a power of 2 (1, 2, 4, 8, etc.). Let's start by creating the K-map :

cd \ ab	00	01	11	10
00		1	1	
01		1	1	
11	1	1	1	1
10	1	1	1	1

Now, let's group the minterms :

Group 1 : $(a'c' +$
Group 2 : $(a'c +$
Group 3 : $(ac' +$
Group 4 : $(ac +$
So, the simplifie
$F(a,b,c,d) = (a'c$

5. *How many*
provide a mem

Data :
RAM chip
Memory
Formula :
No of chi
Calculation :
No of chi

6. *Define min*

(1) Minter
comple
minter
Examp
A and
A
A
~

(2) Maxte
compl
maxte
Exam
The
be:
$A + B$
$A + \sim$
$\sim A +$

Group 1 : $(a'c' + bd')$
Group 2 : $(a'c + b'd)$
Group 3 : $(ac' + bd)$
Group 4 : $(ac + bd')$
So, the simplified expression is :
$F(a,b,c,d) = (a'c' + bd')(a'c + b'd)(ac' + bd)(ac + bd')$

5. *How many 32 K × 1 RAM chips are needed to provide a memory capacity of 256 K bytes?* (2023-24)

Data :

RAM chip's capacity = $32\,K \times 1 = 32 \times 1024 \times 1$ bits

Memory Capacity = $256\,K$ bytes $= 256 \times 1024 \times 8$ bits

Formula :

$$\text{No of chips needed} = \frac{\text{Memory Capacity}}{\text{RAM Chip's Capacity}}$$

Calculation :

$$\text{No of chips needed} = \frac{256 \times 1024 \times 8}{32 \times 1024 \times 1} = 64 \text{ chips}$$

6. *Define min terms and max terms.* (2022-23)

(1) **Minterm :** The product of all literals, either with complement or without complement, is known as minterm.

Example : The minterm for the Boolean variables A and B is :

A.B
A.~B
~A.B

(2) **Maxterm :** The sum of all literals, either with complement or without complement, is known as maxterm.

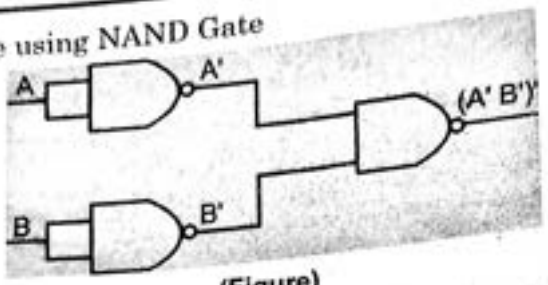
Example :

The maxterm for the Boolean variables A and B will be:

$A + B$
$A + \sim B$
$\sim A + B$

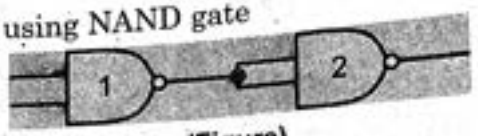
7. *How can AND OR circuit converted to NAND logic?* (2022-23)

OR gate using NAND Gate



(Figure)

AND gate using NAND gate



(Figure)

8. *Express the following function :*

$F_1 = A\bar{C} + AB + BC$ ***in canonical S.O.P.***

$F_2 = (A+B)(B+C)(A+C)$ ***in canonical P.O.S.***

(2022-23)

$F_1 = (AC' + AB + BC)$:

Here is the truth table for the function F_1 :

A	B	C	$F_1 = (AC' + AB + BC)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

From the truth table, we can see that the minterms corresponding to the true outputs (1) are :

Minterm 1 : A'BC

Minterm 2 : ABC'

Minterm 3 : AB'C'

Minterm 4 : ABC

Now, we can express the function in canonical SOP form by combining these minterms using logical OR :

$F_1 = A'BC + ABC' + AB'C' + ABC$

For

$F_2 = (A+$

$(A+B) = (A+$

$(B+C) = (B+$

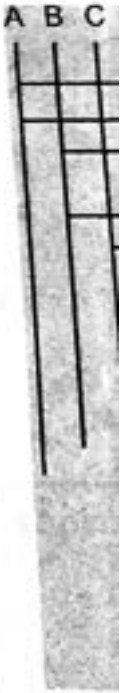
$(A+C) = (A+$

Combining all

$F_2 = (A+$

9. ◆ *Design*
code to
◆ *Design*

The log
equivalent g
converter an
an $n-1$ bit
the MSB of
axis.



Truth T

	B
b_3	b_2
0	0
0	0
0	0

For

$$F_2 = (A+B)(B+C)(A+C)$$

$$(A+B) = (A+B+C'C) = (A+B+C')(A+B+C)$$

$$(B+C) = (B+C+A'A) = (A+B+C)(A'+B+C)$$

$$(A+C) = (A+C+B'B) = (A+B+C)(A+B'+C)$$

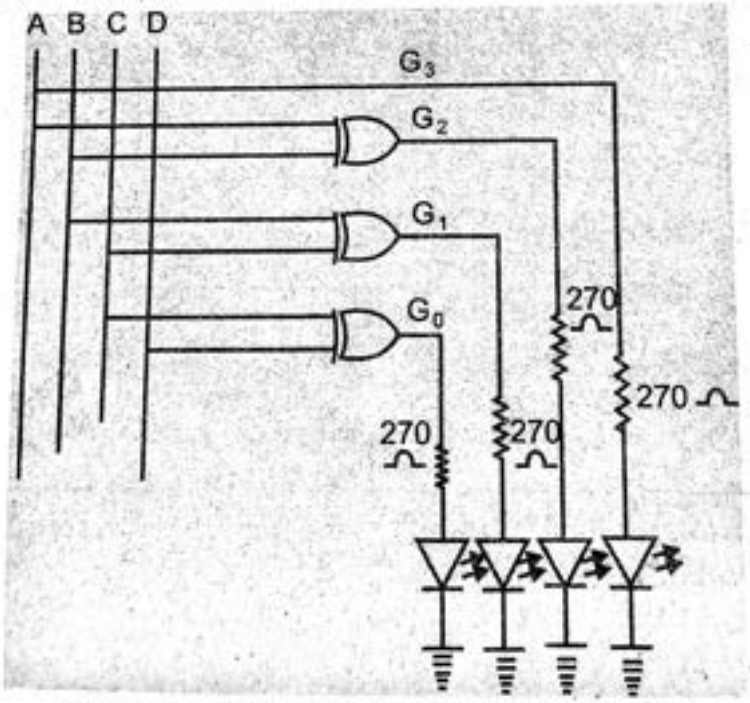
Combining all terms, we get :

$$F_2 = (A+B+C')(A+B+C)(A'+B+C)(A+B'+C)$$

9. ◆ ***Design a code converter that converts Binary code to Gray code.*** **(2022-23)**

◆ ***Design a binary to gray code converter.*** **(2019)**

The logical circuit which converts the binary code to equivalent gray code is known as binary to gray code converter an n - bit gray code can be obtained by reflecting an $n-1$ bit code about an axis after 2^{n-1} rows and putting the MSB of 0 above the axis and the MSB of 1 below the axis.



(Figure : Binary to Gray Code Converter)

Truth Table :

Binary				Gray Code			
b_3	b_2	b_1	b_0	g_3	g_2	g_1	g_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1

0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	1	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	0
1	0	1	1	1	1	1	1

10. ***Simplify the Boolean expression using*** $F(A, B, C, D) = \Pi M(4,5,6,7,8,12)$. $d(0, 1, 2, 3, 4, 11, 14)$ **(2022-23)**

The K-map is as follows :

AB \ CD	C+D	C+D'	C+D	C'+D
A+B	d	d	d	d
A+B'	0	0	0	0
A'+B	0			d
A'+B'	0		d	

(Figure)

Octet = A

Quad = $C + D$

Therefore, $F(A,B,C,D) = A(C + D)$

11. ***Convert the following as per instructions given below :*** **(2017)**

(1) $(1101)_2 \rightarrow (\ )_{10}$

(2) $(169)_{10} \rightarrow (\ )_2$

(3) $(786)_{10} \rightarrow (\ )_8$

(4) $(01101110010110)_2 \rightarrow (\ )_{16}$

(5) $(10110.1110)_2 = (\ )_{10}$

(6) ***Find 2's complement of*** $(1001)_2$

(1) $(1101)_2 = (\ )_{10}$

Step 1 : Binary number :

110

Step 2 :

1×2

Step 3 :

1×8

Step 4 :

8 +

This is

1101.

(2) $(169)_{10}$

Step :

quotien

169/2 =

84/2 =

42/2 =

21/2 =

10/2 =

5/2 =

2/2 =

1/2 =

=

This is th

(3) $(786)_1$

L

binar

rema

(4) $(011$

Conv

as :

num

1101

Step 2 :

$1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

Step 3 :

$1\times8 + 1\times4 + 0\times2 + 1\times1 = 8 + 4 + 0 + 1$

Step 4 :

$8 + 4 + 0 + 1 = 13.$

This is the decimal equivalent of the binary number 1101. **(Ans.)**

(2) $(169)_{10} \rightarrow (\ )_2$

Step : Divide $(169)_{10}$ successively by 2 until the quotient is 0 :

169/2 = 84, remainder is 1
84/2 = 42, remainder is 0
42/2 = 21, remainder is 0
21/2 = 10, remainder is 1
10/2 = 5, remainder is 0
5/2 = 2, remainder is 1
2/2 = 1, remainder is 0
1/2 = 0, remainder is 1
=10101001.

This is the binary equivalent of decimal number 169. **(Ans.)**

(3) $(786)_{10} \rightarrow (\ )_8$

Let us convert decimal 786 or $(786)_{10}$ into its binary equivalent

8	786	R
8	98	2
8	12	2
8	1	4
8	0	1

Now write the result starting from the last remainder obtained. Therefore, $(786)_{10} = (1422)_8$ **(Ans.)**

(4) $(01101110010110)_2 \rightarrow (\ )_{16}$

Convert the 01101110010110 into hexadecimal form as :

0001	1011	1001	0110
1	12	9	6
1	B	9	6

Therefore the hexadecimal value of the binary number is : 1B96 **(Ans.)**

(5) $(10110.1110)_2 = (\ )_{10}$

$(10110.1110)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4}$

$= 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 + 1 \times ½ + 1 \times ¼ + 1 \times 1/8 + 0 \times 1/16$

$= 16 + 0 + 4 + 2 + 0 + 0.5 + 0.25 + 0.125 + 0$

$= 22.875$ **(Ans.)**

(6) **Find 2's Complement of $(1001)_2$**

Step 1 : Subtrahend = 1001

One's complement of subtrahend = 0110

Two's complement of subtrahend = 0110 + 1 = 0111

Step 2 : Add minuend and two's complement of subtrahend

```
  1011
+ 0111
 10010
```

Now discard the carry from the last bit, the result = 0010 **(Ans.)**

(2016)

12. ***Convert the following :***

(1) $(111111110010)_2 \rightarrow (?)_6$

(2) $(57.4)_{16} \rightarrow (?)_{10}$

(3) $(BAD)_{16} \rightarrow (?)_{10}$

(4) $(25.625)_{10} \rightarrow (?)_2$

(1) $(111111110010)_2 \rightarrow (?)_6$

```
 1111   1111   0010
   ↓      ↓      ↓
   F      F      2
```

$= (F\ F\ 2)_{16}$ **(Ans.)**

(2) $(57.4)_{16} \rightarrow (?)_{10}$

$5 \times 16^1 + 7 \times 16^0 + .4 \times 16^{-1}$

$80 + 7 + 0.25 = (87.25)_{10}$ **(Ans.)**

(3) $(BAD)_{16} \rightarrow (?)_{10}$

$B \times 16^2 + A \times 16^1 + D \times 16^0$

$11 \times 256 + 10 \times 16 + 13 \times 1$

$2816 + 160 + 13 = (2989)_{10}$ **(Ans.)**

(4) $(25.625)_{10} \rightarrow (?)_2$

A Power of 2	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
B Remainder of division	25.625	25.625	9.625	1.625	1.625	1.625	.625	.125	.125
C lake Value (A result)	32	16	8	4	2	1	0.5	0.25	0.125
Binary digit (B ÷ C)	0	1	1	0	0	1	1	0	1

$= (11001.101)_2$ **(Ans.)**

13. ◆ *What code*

◆ *Writ*

(1)
(2)
(3)
(4)

(1) **BCD** code from **Exa** BCD code bina oper **Not** INV

dec by coc vir di ca di ra p

(2) **E** I (

(

SCAN AnyScanner

13. ◆ ***What is the most widely used alphanumeric code for input and output? Explain.*** **(2016)**

◆ ***Write in short about the following :*** **(Dec. 2013, 2014)**

(1) *BCD*
(2) *EBCDIC*
(3) *ASCII*
(4) *UNICODE*

(1) BCD (Binary-Coded Decimal) Code : Four-bit code that represents one of the ten decimal digits from 0 to 9.

Example : $(37)_{10}$ is represented as 0011 0111 using BCD code, rather than $(100101)_2$ in straight binary code. Thus BCD code requires more bits than straight binary code. Still it is suitable for input and output operations in digital systems.

Note : 1010, 1011, 1100, 1101, 1110, and 1111 are INVALID CODE in BCD code.

Binary-coded decimal (BCD) is an encoding for decimal numbers in which each digit is represented by its own binary sequence. The binary coded decimal codes are one of the early memory codes. Its main virtue is that it allows easy conversion to decimal digits for printing or display and faster decimal calculations. It is based on the idea of converting each digit of a decimal number into its binary equivalent rather than converting the entire decimal value into a pure binary form.

(2) EBCDIC (Extended Binary Coded Decimal Interchange Code) Code :

(a) 8-bit alphanumeric code developed by IBM, supports 256 symbols.

(b) It was mainly used in IBM mainframe computers.

EBCDIC (Extended Binary Coded Decimal Interchange Code) is a character encoding set used by IBM mainframes. Unlike virtually every computer system in the world which uses a variant of ASCII, IBM mainframes and midrange systems such as the AS/400 tend to use a wholly incompatible character set primarily designed for ease of use on punched

cards. The character encoding is based on Binary Coded Decimal (BCD), so the contiguous characters in the alphanumeric range are formed up in blocks of up to 10 from 0000 binary to 1001 binary. Non alphanumeric characters are almost all outside the BCD range. EBCDIC uses the full 8 bits available to it, so parity checking cannot be used on an 8 bit system. Also, EBCDIC has a wider range of control characters than ASCII.

(3) ASCII (American Standard Code Information Interchange) Code :

(a) It is 7-bit or 8-bit alphanumeric code.

(b) 7-bit code is standard ASCII supports 127 characters.

(c) Standard ASCII series starts from 00h to 7Fh, where 00h-1Fh are used as control characters and 20h-7Fh as graphics symbols.

(d) 8-bit code is extended ASCII supports 256 symbols where special graphics and math's symbols are added.

(e) Extended ASCII series starts from 80h to FFh.

(4) UNICODE : Now a day's most of the computers use a new code called Unicode and it defines an International Character Set. The Unicode can represent all of the characters found in all human languages. This becomes possible because Unicode uses 16 bits to represent a character. Many languages such as Latin, Greek, Arabic, Hebrew etc. have adopted this coding scheme.

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, Just Systems, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others. Unicode is required by modern standards such as XML, Java, ECMA Script (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646. It is supported in many operating systems, all modern browsers, and many other products. The emergence of the Unicode Standard, and the availability of tools supporting it,

is among the most significant recent global software technology trends.

Incorporating Unicode into client-server or multi-tiered applications and websites offers significant cost savings over the use of legacy character sets. Unicode enables a single software product or a single website to be targeted across multiple platforms, languages and countries without re-engineering. It allows data to be transported through many different systems without corruption.

14. ***Solve the following :*** **(2015)**

(1) ***Divide $(1101)_2$ by $(101)_2$.***

(2) ***Multiply binary number 1100 & 1010***

(3) ***Subtract $(25)_{10}$ from $(50)_{10}$ using complementary method.***

(1) Divide $(1101)_2$ by $(101)_2$:

$$\begin{array}{r|l} & 10 \\ \hline 101 & 1101 \\ & -\ 101 \\ \hline & 11 \end{array}$$

Quotient = 10 and remainder = 11

(2) Multiplication of 1100 & 1010 :

$$\begin{array}{cccccccc} & & & & 1 & 1 & 0 & 0 \\ & & & * & 1 & 0 & 1 & 0 \\ \hline & & & & 0 & 0 & 0 & 0 \\ + & & & 1 & 1 & 0 & 0 & * \\ + & & 0 & 0 & 0 & 0 & * & * \\ + & 1 & 1 & 0 & 0 & * & * & * \\ \hline 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{array}$$

(3) To Convert Decimal Number to 9's and 10's Complement : 9's complement of decimal number can be obtained by $((10^n - 1)$-number) where n represents the number of digits in given number. 10's complement can be obtained by $(10^n$ – number) where n represents the number of digits in given number.

Consider – $(1234)_{10}$ 9's complement = $10^4 - 1 - 1234 = 9999\text{-}1234 = 8765$

10's complement = $10^4 - 1234 = 8766$.

Always the number to be subtracted or negative number is converted to 9's or 10's complement.

Subtraction Using 10's Complement :

$A - B$

$A = (50)_{10}$

$B = (25)_{10}$

10's complement of $B = 25$, Adding 10's complement of B to A.

10's complement of $B = 25$, $10^2 - 25 = 100 - 25 = 75$

$$\begin{array}{r} 50 \\ +\ 75 \\ \hline 125 \\ \hline = \quad 25 \end{array}$$ (Ans.)

Important Notes :

(1) If there is any end carry, just ignore it and sum obtained is the answer.

(2) If there is no carry, answer is - (10's complement of the sum obtained).

(2016)

15. *Solve the following :*

(1) $(1111)_2 \times (1111)_2$ {*Multiplication*}

(2) $(110110)_2 + (101101)_2$ {*Addition*}

(3) $(110011)_2 - (101100)_2$ {*subtraction*}

(1) $(1111)_2 \times (1111)_2$ {Multiplication}

$$\begin{array}{r} 1111 \times 1111 \\ \hline 1111 \\ 1111\times \\ 1111\times\times \\ 1111\times\times\times \\ \hline 11100001 \end{array}$$ (Ans.)

(2) $(110110)_2 + (101101)_2$ {Addition}

$$\begin{array}{r} 110110 \\ +\ 101101 \\ \hline 1100011 \end{array}$$ (Ans.)

(3) $(110011)_2 - (101100)_2$ {Subtraction}

$$\begin{array}{r} 110011 \\ +\ 101100 \\ \hline 000111 \end{array}$$ (Ans.)

16. *Convert the following :* (2015)

(1) $(FAB)_{16} \rightarrow (?)_2$

(2) $(1001.011)_2 \rightarrow (?)_{10}$

(3) $(125)_6 \rightarrow (?)_4$

(4) $(247.65)_8 \rightarrow (?)_{10}$

(1) $(FAB)_{16} = (?$

$(F)_{16} = (15)_{10}$

$(A)_{16} = (10)_{10}$

$(B)_{16} = (11)_{1}$

So, $(FAB)_{1}$

(2) (1001.011):

1 0

2^3 2

$= (1 * 2^3) +$

$+ (1 *$

$= (1 * 8) +$

$+ (0 *$

$= 8 + 0 + 0$

$= 9 + 0.25$

(3) $(125)_6 = ($

It is a two

Step 1 : (

Step 2 : (

Step –

Convers

$= (1$

$= (1$

$= 3$

Step –

Conve

$(53)_{10} =$

$(53)_{10}$

(4) (247.6

$= (2 *$

$= (2 *$

$= (12$

$= 128$

$= 167$

$= (16$

(1) $(FAB)_{16} = (?)_2$

$(F)_{16} = (15)_{10} = (1111)_2$

$(A)_{16} = (10)_{10} = (1010)_2$

$(B)_{16} = (11)_{10} = (1011)_2$

So, $(FAB)_{16} = (1111\ 1010\ 1011)_2$

(2) $(1001.011)_2 = (?)_{10}$

1	0	0	1	0	1	1
2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}

$= (1 * 2^3) + (0 * 2^2) + (0 * 2^1) + (1 * 2^0) + (0 * 2^{-1}) + (1 * 2^{-2}) + (1 * 2^{-3})$

$= (1 * 8) + (0 * 4) + (0 * 2) + (1 * 1) + (0 * ½) + (1 * ¼) + (1 * 1/8)$

$= 8 + 0 + 0 + 1 + 0 + ¼ + 1/8$

$= 9 + 0.25 + 0.125 = 9.375 = (9.375)_{10}$

(3) $(125)_6 = (?)_4$

It is a two steps process :

Step 1 : $(125)_6 = (?)_{10}$ Senary to Decimal conversion

Step 2 : $(?)_{10} = (?)_4$ Decimal to Quaternary conversion

Step – 1 : Senary System to Decimal System Conversion :

1	2	5
6^2	6^1	6^0

$= (1 * 6^2) + (2 * 6^1) + (5 * 6^0)$

$= (1 * 36) + (2 * 6) + (5 * 1) = (36) + (12) + (5)$

$= 36 + 12 + 5 = 53 = (53)_{10}$

Step – 2 : Decimal System to Quaternary System Conversion :

$(53)_{10} = (?)_4$

4 | 53		
4 | 13		1
4 | 3		1

$(53)_{10} = (311)_4$

(4) $(247.65)_8 = (?)_{10}$

2	4	7	6	5
8^2	8^1	8^0	8^{-1}	8^{-2}

$= (2 * 8^2) + (4 * 8^1) + (7 * 8^0) + (6 * 8^{-1}) + (5 * 8^{-2})$

$= (2 * 64 + (4 * 8) + (7 * 1) + (6 * 1/8) + (5 * 1/64)$

$= (128) + (32) + (7) + (6 * 0.125) + (5 * 0.015625)$

$= 128 + 32 + 7 + (0.75) + (0.078125)$

$= 167 + 0.75 + 0.078125 = 167.828125$

$= (167.828125)_{10}$

17. *Multiply* $(1010)_2$ *and* $(1001)_2$. **(2014)**

Multiply the binary numbers 1010 and 1001

$$\begin{array}{r} 1\ 0\ 1\ 0 \\ \times\ 1\ 0\ 0\ 1 \\ \hline 1\ 0\ 1\ 0 \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 0 \\ \hline 1\ 0\ 1\ 1\ 0\ 1\ 0 \end{array}$$

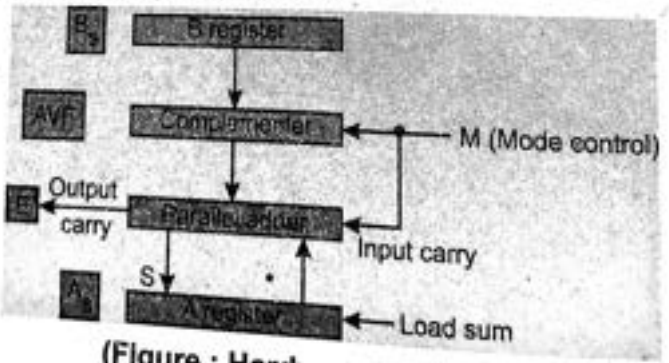
18. *Show the logic behind addition and subtractio with signed magnitude data show the hardwa components needed to perform these operations.* **(201**

Signed magnitude addition and subtraction

(1) For an add operation, identical signs dictate that magnitudes be added, different signs require that magnitudes be subtracted.

(2) For subtraction operation, different signs dictate magnitudes be added, identical signs require magnitudes be subtracted AVF.

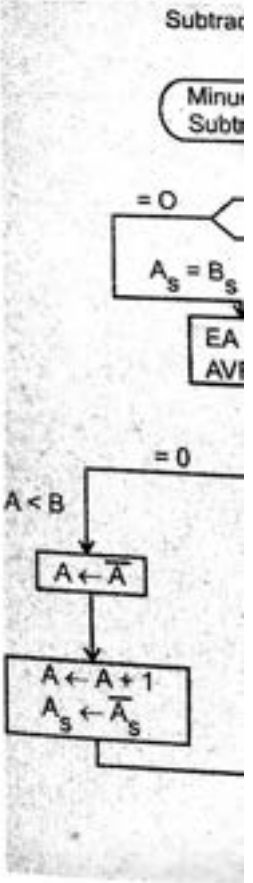
(3) Add-overflow flip-flop holds the overflow bit whe and B are added Addition of A and B is done thro parallel adder. Flowchart for add and subt operation is shown as :



(Figure : Hardware Implementation)

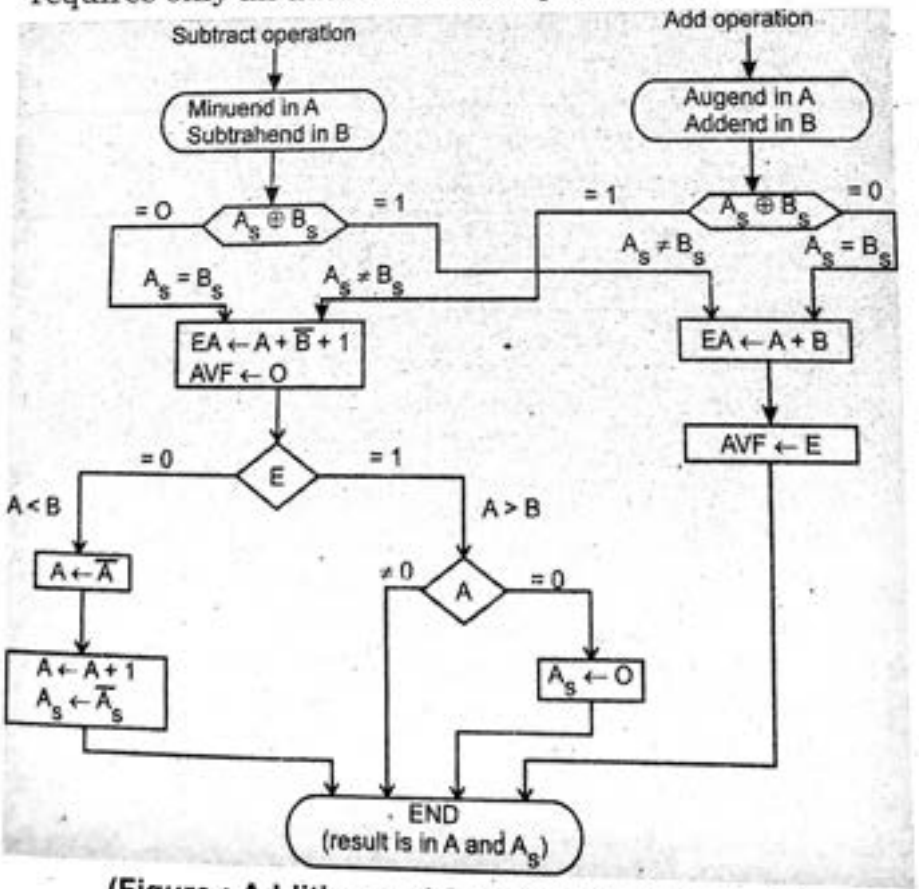
To implement the two arithmetic operations hardware, it is first necessary that the two number stored in registers. Let A and B be two registers that the magnitudes of the numbers, and A_s and B_s, be two flops that hold the corresponding signs. The result o

operation may a saving is achi A_s. Thus A and Consider now algorithms abo perform the m circuit is neede Third, two pa perform the m relationship ca with A, and magnitude co However, a dif less equipmen accomplished the result of a carry after th alternatives r subtraction ar requires only



(Figure

operation may be transferred to a third register: however, a saving is achieved if the result is transferred into A and A_s. Thus A and A_s together form an accumulator register. Consider now the hardware implementation of the algorithms above. First, a parallel-adder is needed to perform the microoperation $A + B$. Second, a comparator circuit is needed to establish if $A > B$, $A = B$, or $A < B$. Third, two parallel-subtractor circuits are needed to perform the microoperations $A - B$ and $B - A$. The sign relationship can be determined from an exclusive OR gate with A_s and B_s as inputs. This procedure requires a magnitude comparator, an adder, and two subtractors. However, a different procedure can be found that requires less equipment. First, we know that subtraction can be accomplished by means of complement and add. Second, the result of a comparison can be determined from the end carry after the subtraction. Careful investigation of the alternatives reveals that the use of 2's complement for subtraction and comparison is an efficient procedure that requires only an adder and a complementer.



(Figure : Addition and Subtraction Flowchart)

The Figure shows a block diagram of the hardware for implementing the addition and subtraction operations. It consists of registers A and B and sign flip-flops A, and B,. Subtraction is done by adding A to the 2' s complement of B. The output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitudes of the two numbers. The add-overflow flip-flop AVF holds the overflow bit when A and B are added. The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm. The addition of A plus B is done through the parallel adder. The S (sum) output of the adder is applied to the input of the A register. The complementer provides an output of B or the complement of B depending on the state of the mode control M. The complementer consists of exclusive-OR gates and the parallel adder consists of full-adder circuits. The M signal is also applied to the input carry of the adder. When $M = 0$, the output of B is transferred to the adder, the input carry is 0, and the output of the adder is equal to the sum $A + B$. When M a I, the l's complement of B is applied to the adder, the input carry is I, and output $S = A + H + 1$. This is equal to A plus the 2's complement of B, which is equivalent to the subtraction $A - B$.

19. *Divide $(100001)_2$ by $(110)_2$ and write down the quotient and remainder.* **(2014)**

Divide 100001 by 110

```
          0101    Quotient
      ┌──────
110   │ 100001    Dividend
      │ 110       ──────── 1
      │ ----
      │ 1000      ──────── 2
      │  110      ──────── 3
      │ ----
      │  100      ──────── 4
      │  110      ──────── 5
      │ ----
      │ 1001      ──────── 6
      │  110      ──────── 7
      │ ----
      │   11      Remainder
```

(1) Divisor greater than 100, so put 0 in quotient.
(2) Add digit from dividend to group used above.
(3) Subtraction possible so put 1 in the quotient.
(4) Remainder from subtraction plus digit from dividend
(5) Divisor greater, so put 0 in quotient.
(6) Add digit from dividend to group.
(7) Subtraction possible, so put 1 in quotient.

20. *What do you mean by BCD Number? Draw circuit to add two BCD numbers.* **(2019)**

The digital system handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD Adder Circuit that adds two BCD digits and produces a sum digit also in BCD. BCD numbers use 10 digits, 0 to 9 which are represented in the binary form 0 0 0 0 to 1 0 0 1, i.e. each BCD digit is represented as a 4-bit binary number. When we write BCD number say 526, it can be represented as

5	2	6
↓	↓	↓
0101	0010	0110

Here, we should note that BCD cannot be greater than 9.

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

(1) At first the given number are to be added using the rule of binary. For example,

Case 1 :

$$\begin{array}{r} 1010 \\ +0101 \\ \hline 1111 \end{array}$$

Case 2 :

$$\begin{array}{r} 0001 \\ +0101 \\ \hline 0110 \end{array}$$

(2) In second step we have to judge the result of addition. Here two cases are shown to describe the rules of BCD Addition. In case 1 the result of addition of two binary number is greater than 9, which is not valid for BCD number. But the result of addition in case 2 is less than 9, which is valid for BCD numbers.

(3) If the four bit result of addition is greater than 9 and if a carry bit is present in the result then it is invalid and we have to add 6 whose binary equivalent is (0110)2 to the result of addition. Then the resultant that we would get will be a valid binary coded number. In case 1 the result was $(1111)_2$, which is greater than 9 so we have to add 6 or $(0110)_2$ to it.

$$(1111)_2 + (0110)_2 = 0001\ 0110 = 15$$

As you can see the result is valid in BCD. But in case 2 the result was already valid BCD, so there is no need to add 6. This is how BCD Addition could be. Now a question may arrive that why 6 is being added to the addition result in case BCD Addition instead of any other numbers. It is done to skip the six invalid states of binary coded decimal i.e. from 10 to 15 and again return to the BCD codes.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given

BCD Adder Truth Table :

Input				Output
S_3	S_2	S_1	S_0	y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Solving for Y by k-map we get :

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

(Figure)

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given BCD Adder Truth Table.

$Y = S$

Steps :

(1) Add
addi

(2) If fou
is ne

(3) If t
carr
inva

(4) To
sum
the
Th
(a)
(b
(c

W

Adder

Output
Carry

firs
bir

$$Y = S_3 S_2 + S_3 S_1$$

Steps :

(1) Add two BCD numbers using ordinary binary addition.

(2) If four-bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.

(3) If the four-bit sum is greater than 9 or if a carry is generated from the four-bit sum, the sum is invalid.

(4) To correct the invalid sum, add 01102 to the four-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.

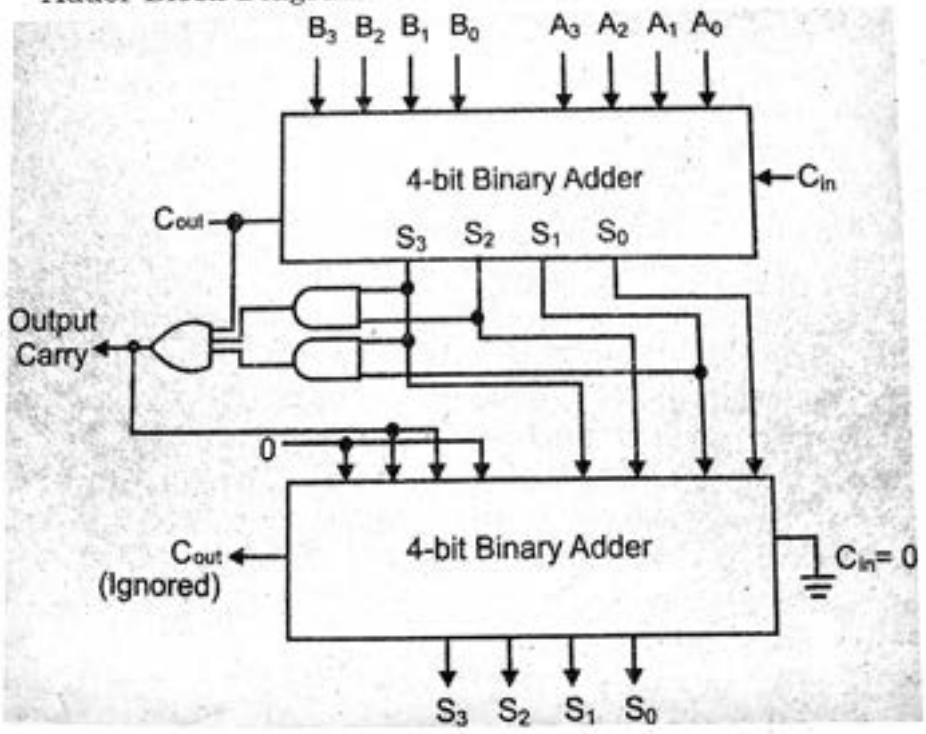
Thus to implement BCD Adder Circuit we require :

(a) 4-bit binary adder for initial addition

(b) Logic circuit to detect sum greater than 9 and

(c) One more 4-bit adder to add 01102 in the sum if sum is greater than 9 or carry is 1.

With this design information we can draw the BCD Adder Block Diagram



(Figure)

The two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum.

(1) When the output carry is equal to zero (i.e. when sum ≤ 9 and $C_{out} = 0$) nothing (zero) is added to the binary sum.

(2) When it is equal to one (i.e. when sum > 9 or $C_{out} = 1$), binary 0110 is added to the binary sum through the bottom 4-bit binary adder.

(3) The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.

21. ◆ ***Define De-Morgan's law of Boolean Algebra and apply it to simplify ((A' + B)'.C)'.*** **(2019)**

◆ ***State and prove the De – Morgan's theorem.*** **(2016, 2018)**

◆ ***Define various Boolean Laws.***

Boolean algebra, proposed by George Bole, is an attempt of representing the true-false logic of human in mathematical form. It is used to design and analyze logical circuits used in computers. It is helpful to describe complicated computer circuitry. Generally in boolean algebra, we use letters and symbols to represent statements and their logical connections. Variables in boolean algebra can hold and their logical connections. Variables in boolean algebra can hold binaries (i.e. 1 or 0) only thus also known as binary algebra. Some of the fundamental laws of boolean algebra is given below :

(1) **OR Laws :** This law is subjected to OR logical operation. The symbol for this operation is '+', (which is certainly not the plus sign of ordinary mathematics), which when operated with the given operands produced corresponding result example,

$A + B = C$ means that if A is true (i.e. 1) OR if B is true (i.e. 1) then C will be true. Here, it doesn't mean that A plus B equal to C. Some of the OR laws may expressed as :

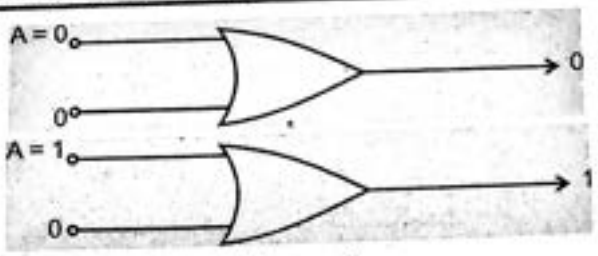
(a) $A + 0 = A$

(b) $A + 1 = 1$

(c) $A + A = A$

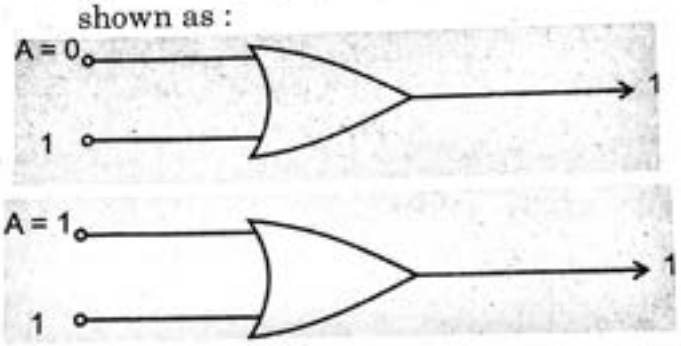
(d) $A + \overline{A} = 1$

(a) Physical meaning of this law is that ORing of input A with 0, always outputs A (i.e., it depends upon the value of A). This may be shown as :



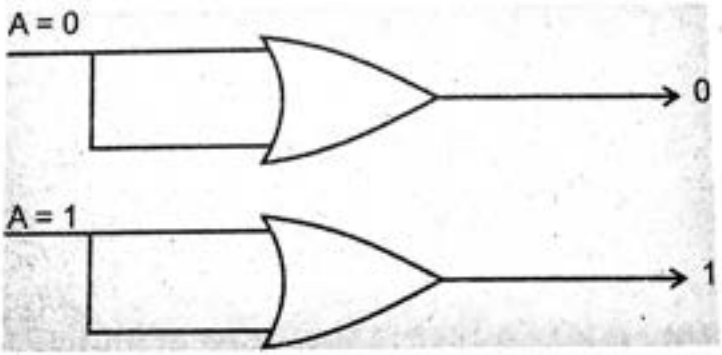
(Figure)

(b) This law states that ORing of an input A with 1 (i.e. a zero input) always produces 1. This may be shown as :



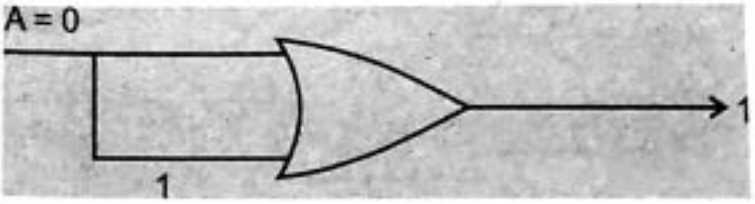
(Figure)

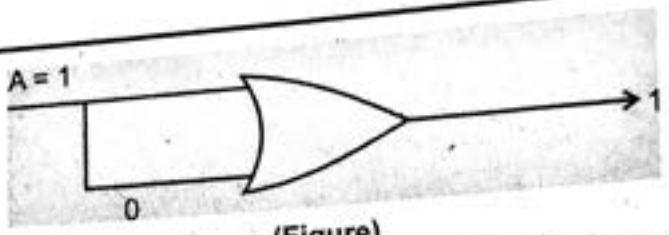
(c) It states that output of ORing of A with A will be A as shown in figure :



(Figure)

(d) This law states that Oring of A with its complement (if $A = 0$ then $\overline{A} = 1$ and if $A = 1$ then $\overline{A} = 0$) alrways outputs 1 this may be shown as :





(Figure)

(2) AND Laws : This law is subjected to AND logical operation, the symbol of the operation is "." (dot) which doesn't mean the multiplication as an ordinary mathematics.

Example : $AB = C$ means that if A is true AND if B is true (i.e. 1) then C will be true. Some of the AND laws are given below :

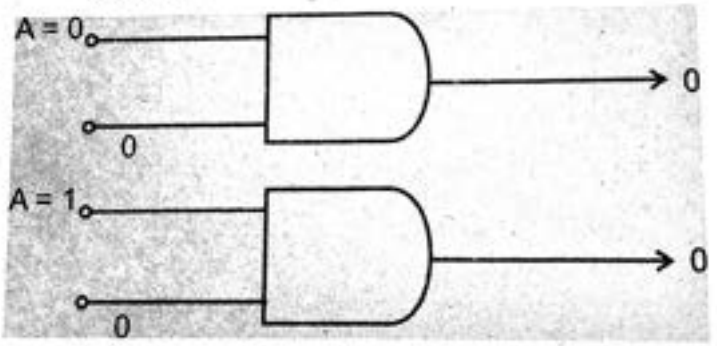
(a) $A.0 = 0$

(b) $A.1 = A$

(c) $A.A = A$

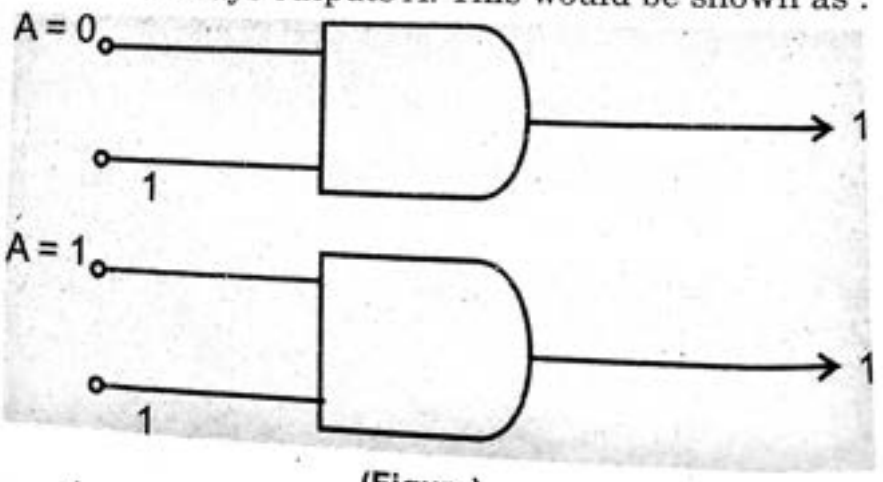
(d) $A.\overline{A} = 0$

(a) This law physically states the ANDing of an input A with 0 always produces 0 as shown below :



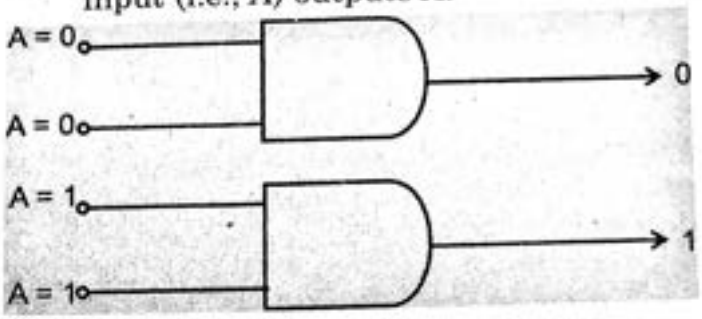
(Figure)

(b) It states that the ANDing of an input A with 1 always outputs A. This would be shown as :



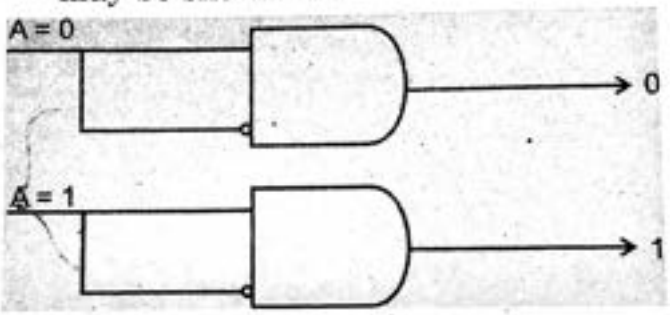
(Figure)

(c) This law means that ANDing of A with the same input (i.e., A) outputs A.



(Figure)

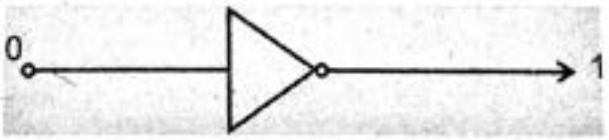
(d) This law states that ANDing of an input A with its complement (i.e. $\overline{A}$) always outputs 0. This may be shown as :



(Figure)

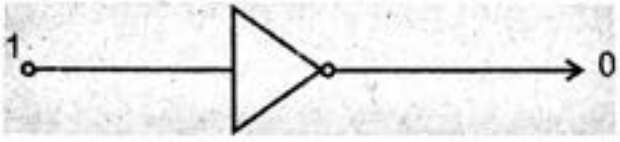
(3) **NOT Laws :** It is subjected to NOT logical operation which negates its input as given :

(a) $\overline{0} = 1$



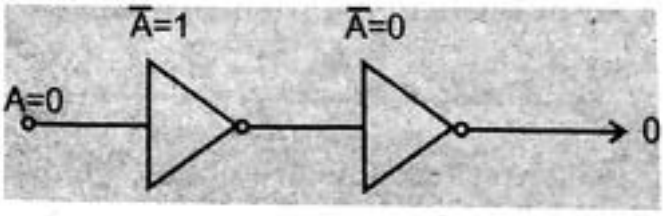
(Figure)

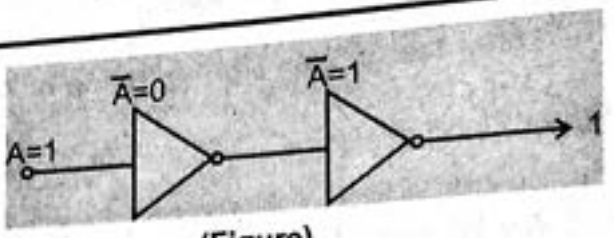
(b) $\overline{1} = 0$



(Figure)

(c) $\overline{A} = A$





(Figure)

(4) **Commutative Laws :** These laws state that the order in which a combination of inputs is given, doesn't affect the final result i.e.

$$A + B = B + A$$
$$A.B = B.A$$

(5) **Associative Laws :**

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

(6) **Distributive Laws :**

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

(7) **Absorptive Laws :**

$$A + AB = A$$
$$A.(A + B) = A$$
$$A.(\overline{A} + B) = B$$

(8) **De'morgan's Theorems :**

(a) The complement of a sum equal to product of complements i.e.,

$$\overline{A + B + C} = \overline{A}.\overline{B}.\overline{C}$$

(b) The complement of a product equal to sum of complement i.e.,

$$\overline{A.B.C} = \overline{A} + \overline{B} + \overline{C}$$

Proof of De-Morgan's Theorem

(1) **De Morgan's Theorem 1 :** The complement of the sum of two or more variables is equal to the product of the complement of the variables.

(2) **De Morgan's Theorem 2 :** The complement of the product of two or more variables is equal to the sum of the complements of the variables.

For two variables A and B these theorems are written in Boolean notation as follows :

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

The two theorems are proved below.

To prove

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

Since each variable can have a value either 0 or 1, the following four cases arise :

(a) When $A=0,\ B=0,$

$$\overline{A+B}=\overline{0+0}=\overline{0}=1$$

And $\overline{A}\cdot\overline{B}=\overline{0}\cdot\overline{0}=1\cdot1=1$

Hence $\overline{A+B}=\overline{A}\cdot\overline{B}$

(b) When $A=0,\ B=1,$

$$\overline{A+B}=\overline{0+1}=\overline{1}=0$$

And $\overline{A}\cdot\overline{B}=\overline{0}\cdot\overline{1}=1\cdot0=0$

Hence $\overline{A+B}=\overline{A}\cdot\overline{B}$

(c) When $A=1,\ B=0,$

$$\overline{A+B}=\overline{1+0}=\overline{1}=0$$

And $\overline{A}\cdot\overline{B}=\overline{1}\cdot\overline{0}=0\cdot1=0$

Hence $\overline{A+B}=\overline{A}\cdot\overline{B}$

(d) When $A=1,\ B=1,$

$$\overline{A+B}=\overline{1+1}=\overline{1}=0$$

and $\overline{A}\cdot\overline{B}=\overline{1}\cdot\overline{1}=0\cdot0=0$

Hence $\overline{A+B}=\overline{A}\cdot\overline{B}$

Since, in every case the left hand side equals the right hand side, the theorem is proved.

To prove

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

As all possible combinations of A and B are exhausted, the theorem is proved.

Numerical Solution : Now

$$((A'+B').C)'$$

$=(A'+B')''+C'$ Apply De – Morgan's law

$=A'+B'+C'$ **(Ans.)**

22. ***Find the equivalent expression in canonical S–O–P for :*** **(2019)**

$$AB+A'B'C+BC'.$$

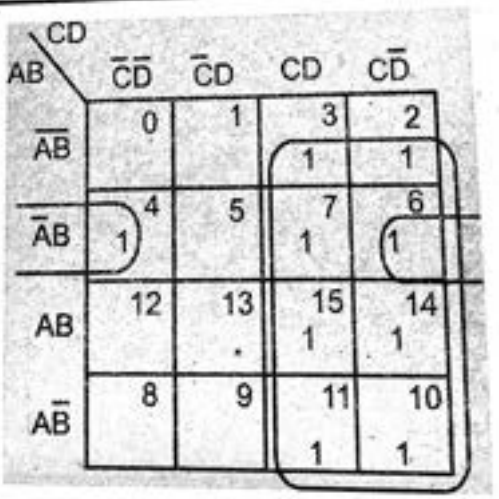
$$F(A,B,C)=AB+A'B'C+BC'$$

$$= AB(C + C')A'B'C + BC'(A + A')$$
$$= ABC + ABC' + A'B'C + ABC' + A'BC'$$
$$= ABC + ABC' + A'B'C + A'BC' \quad \textbf{(Ans.)}$$

23. *Express Boolean function $F = xy + x'z$ as product of max terms from truth table.* **(2019)**

$$F = xy + \bar{x}z$$
$$F = xy + \bar{x}z$$
$$= xy(z + \bar{z}) + \bar{x}z(y + \bar{y})$$
$$= xyz + xy\bar{z} + \bar{x}yz + \bar{x}\bar{y}z$$
$$= \sum m(7,6,3,1)$$
$$\prod M(0,2,4,5)$$
$$F = (x + y + z)(x + \bar{y} + z)(\bar{x} + y + z)(\bar{x} + y + \bar{z}) \quad \textbf{(Ans.)}$$

24. *Simplify the following Boolean function using K – map :*
$F(A, B, C, D) = \sum 2,4,6,7,11,14,15$
with don't care $\sum_d 3,10,13$ ***in S – O – P form.*** **(2019)**



(Figure)

$$Y = C + \bar{A}B\bar{D}$$

25. *State the absorption law of Boolean Algebra.* **(2018)**

Absorptio

Absor

(1) $A + A$
L.H.S
$= A +$
$= A(1$
$= A.$
$= A$
$= R.$

(2) $A\ (A$
L.H
$= A$
$= A$
$= A$
$= A$
$=$

26. Expre
produ

$F = x +$
$= x($
$= (x$
$= x$
$= x$
In Ma
$F = ($
$= ($
$= ($
$F = \pi$

27. Si
[((

[((A

[((

Absorption Law of Boolean Algebra

Absorption law of Boolean algebra are :

(1) $A + AB = A$

L.H.S

$= A + AB$

$= A(1 + B)$ (by distributive law)

$= A.\ 1$

$= A$

= R.H.S.

(2) $A\ (A + B) = A$

L.H.S.

$= A\ (A + B)$

$= A.A + A.B$ (by distributive law)

$= A + AB$ (by idempotency law)

$= A$ (Absorption law)

= R.H.S.

26. *Express the Boolean function $F = x + y'z$ as a product of max term.* **(2018)**

$F = x + y'z$

$= x(y + y')(z + z') + (x + x')y'z$

$= (xy + xy')(z + z') + xy'z + x'y'z$

$= xyz + xyz' + xy'z + xy'z' + xy'z + x'y'z$

$= xyz + xyz' + xy'z + xy'z' + x'y'z'$

In Max Term,

$F = (x' + y' + z')(x' + y' + z)(x' + y + z')(x' + y + z)(x + y + z')$

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

$= (M_7, M_6, M_5, M_4, M_1)$

$F = \pi\ (1, 4, 5, 6, 7)$ **(Ans.)**

27. *Simplify the expression :* **(2018)**
$[((AB)'C)'D]'$

$[((AB)'C)'D]' = [((A' + B').C)'D]'$

$= [((A' + B')' + C').D]'$

$= [((A''B'') + C').D]'$

$= [(A.B + C').D]'$

$= (A.B + C')'.D'$

$= (AB)' + C + D'$

$[((AB)'C)'D]' = A' + B' + C + D'$ **(Ans.)**

28. ***Find the minterms of the logical expression :*** **(2018)**

$$Y = A'B'C' + A'B'C + A'BC + ABC'$$

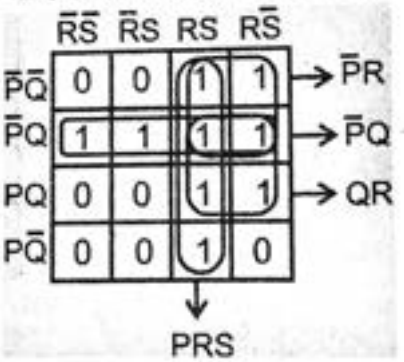
$$\begin{array}{lcccc} Y = & A'B'C' & + & A'B'C & + & A'BC & + & ABC' \\ = & 000 & & 001 & & 011 & & 110 \\ = & m_0 & & m_1 & & m_3 & & m_6 \end{array}$$

$Y = \Sigma m\ (0, 1, 3, 6)$ **(Ans.)**

29. ***Simplify the following Boolean function : $F(P, Q, R, S) = \Sigma(2, 3, 4, 5, 6, 7, 11, 14, 15)$ and implement of means of NAND Gate.*** **(2018)**

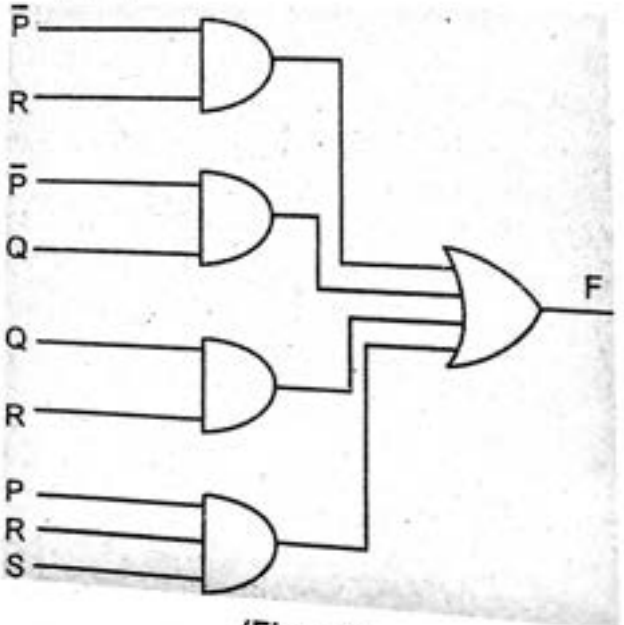
Boolean Function

$F(P, Q, R, S) = \Sigma(2, 3, 4, 5, 6, 7, 11, 14, 15)$



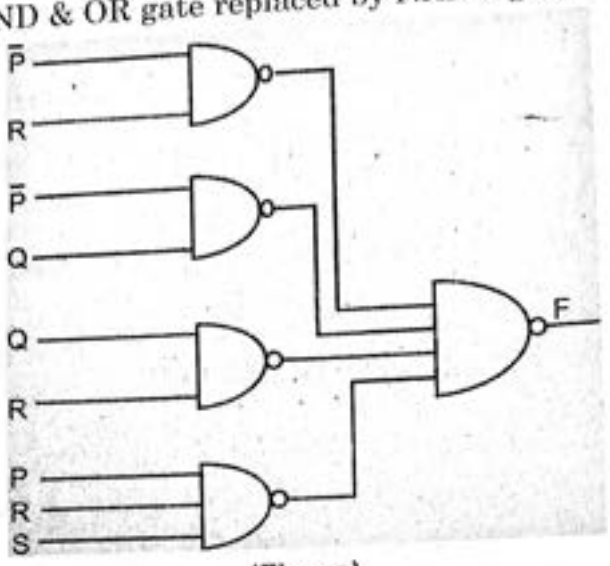
(Figure)

$F = \bar{P}R + \bar{P}Q + QR + PRS$ **(Ans.)**



(Figure)

Now AND & OR gate replaced by NAND gate.



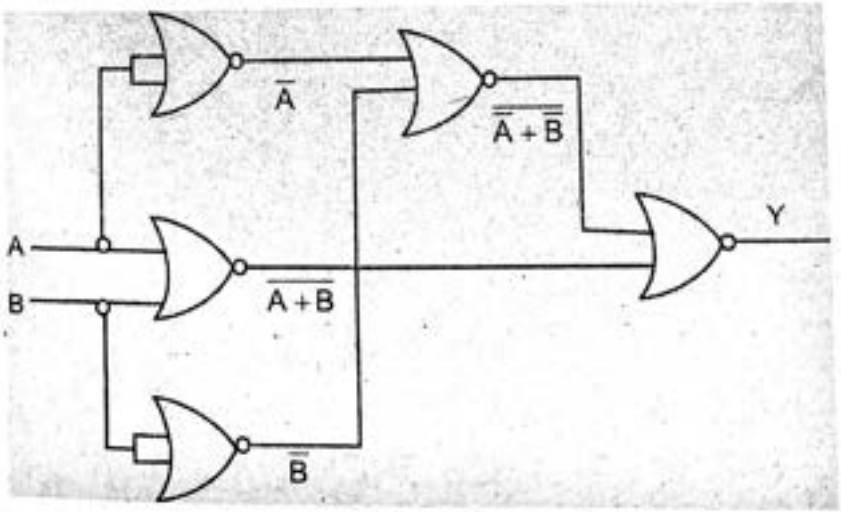
(Figure)

30. *Implement Ex - OR gate with NOR gate only.* (2018)

Implementation of Ex - OR Gate with NOR Gate

Ex - OR gate output expression is

$$Y = A\bar{B} + \bar{A}B$$
$$= A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B}))$$
$$= (\bar{A} + \bar{B})(A + B)$$
$$= \overline{\overline{(\bar{A} + \bar{B})(A + B)}}$$
$$Y = \overline{\overline{\bar{A} + \bar{B}} + \overline{A + B}}$$



(Figure)

31. ***Represent decimal number 8620 in BCD and Excess-3 code.*** **(2018)**

Decimal number = $(8620)_{10}$

$(8620)_{10}$ =(1001 0110 0010 0000)$_{BCD}$

$(8620)_{10}$ =(1011 1001 0101 0011)$_{Excess\text{-}3\ code}$

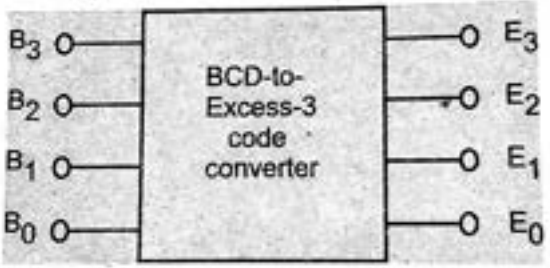
Note : Excess-3 code can be derived from the natural BCD code by adding 3 to each coded number. **(Ans.)**

32. ◆ ***Design a BCD to excess - 3 code converter.*** **(2018)**

◆ ***Draw the logic diagram for BCD to Excess-3 code converter. Discuss each step through truth table and K-map.*** **(May 2013)**

BCD-to-Excess-3 Code Converter

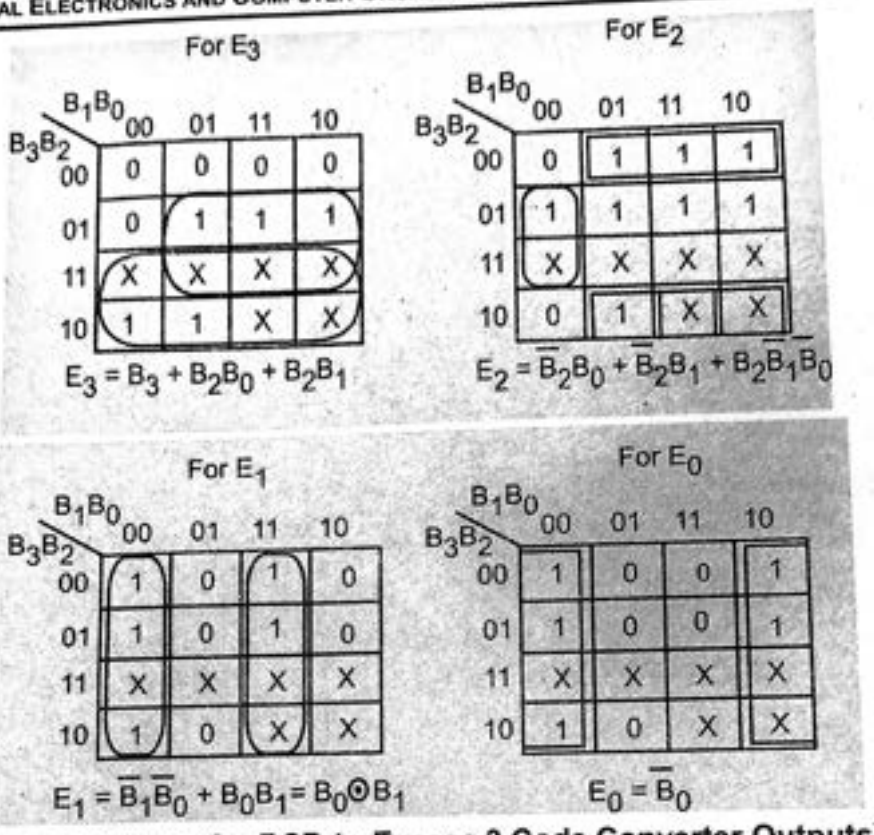
Excess-3 code can be obtained from the BCD code by adding 3 to each coded number. The code converter contains four inputs and four outputs as shown in figure.



(Figure : Block Diagram of BCD-to-EX-3 Code Converter)

Table : Truth Table for BCD-to-Excess 3 Code Converter :

Decimal number	BCD Number Code				Excess-3 Code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	1



(Figure : K-maps for BCD-to-Excess 3 Code Converter Outputs)

The logic diagram for the outputs

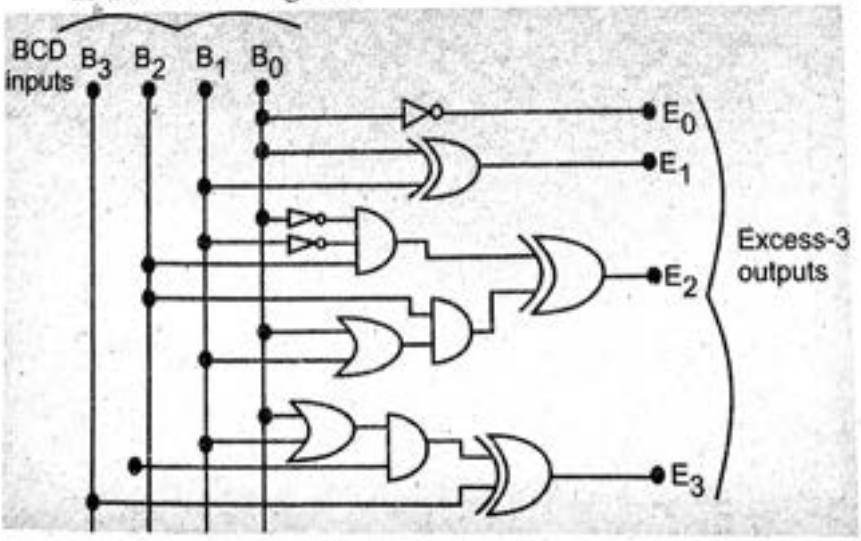
$$E_3 = B_3 + B_2(B_0 + B_1)$$

$$E_2 = B_2\overline{B}_1B_0 + \overline{B}_2(B_0 + B_1)$$

$$E_1 = (B_0 \Theta B_1)$$

$$E_0 = \overline{B}_0$$

is shown in figure.



(Figure : Logic Diagram for BCD-to-Excess 3 Code Converter)

33. ◆ *Simplify the Boolean function :* (2018)
$F(w, x, y, z) = \sum(0, 1, 2, 3, 7, 8, 10)$
$d(w, x, y, z) = \sum(5, 6, 11, 15)$

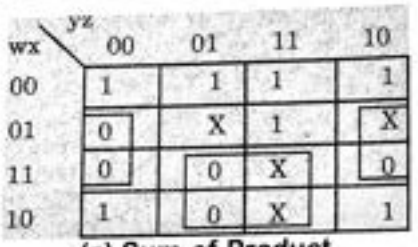
◆ *Simplify the Boolean function F together with the don't care condition D in :* (June 10)

(1) *Sum-of-products form*

(2) *Product-of-sums form*

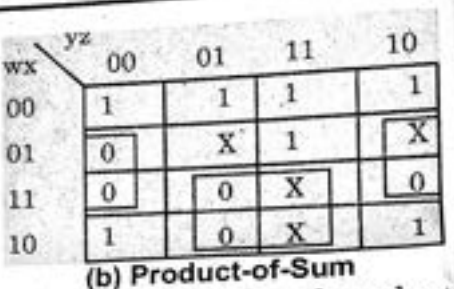
$F(w, x, y, z) = \Sigma(0, 1, 2, 3, 7, 8, 10)$

$D(w, x, y, z) = \Sigma(5, 6, 11, 15)$



wx \ yz	00	01	11	10
00	1	1	1	1
01	0	X	1	X
11	0	0	X	0
10	1	0	X	1

(a) Sum-of-Product



wx \ yz	00	01	11	10
00	1	1	1	1
01	0	X	1	X
11	0	0	X	0
10	1	0	X	1

(b) Product-of-Sum

This results in a simplified sum-of-products functions,

$$F = \overline{x}z + \overline{w}z$$

In (b), the 0's are combined with any X's convenient to simplify the complement of the function. The complement function is simplified to : $\overline{F} = wz + x\overline{z}$

Complementing again, we get a simplified product of sums function, $F = (\overline{w} + \overline{z})(\overline{x} + z)$

34. ◆ *Simplify the Boolean function :* (2017)
$F = X'Y' + XY + X'Y$

◆ *Simplify the Boolean function :* (May 2013)

(1) $x'y' + xy + x'y$

(2) $\overline{\overline{AB} + ABC + A\left(B + A\overline{B}\right)}$

(3) *Find the dual of :*
$F = A'BC' + A'B'C$

(1) $\overline{x}\overline{y} + xy + \overline{x}y$

$= \left(\overline{x}\overline{y} + \overline{x}y\right) + xy = \overline{x}\left(\overline{y} + y\right) + xy = \overline{x} + xy = \overline{x} + y$ **(Ans.)**

(2) $\overline{\left(\overline{AB} + ABC + A\left(B + A\overline{B}\right)\right)} = \overline{\overline{AB} + C + A\left(B + A\overline{B}\right)}$

$= \overline{\overline{AB} + C + A(B + A)} = \overline{\overline{AB} + C + A(1 + B)}$

$= \left(\overline{\overline{AB} \cdot \overline{C}}\right) + A = AB \cdot \overline{C} + A = A\left(1 + B\overline{C}\right) = A$ **(Ans.)**

(3) $F = \overline{A}B$

Dual of

35. *Implemen*
Gate. Sho

XOR Gate

To ma
that uses
operation.
NAND ga

$f = A \oplus B$

The
complemer
get each o

Fi
sum wit

36. *Expre*
of mi

P
F
F
Q' F

(3) $F = \overline{A}B\overline{C} + \overline{A}\overline{B}C$

Dual of $F = (\overline{A} + B + \overline{C}) = (\overline{A} + \overline{B} + C)$ **(Ans.)**

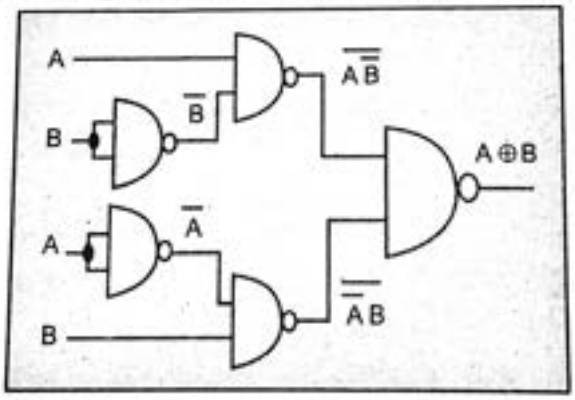
35. *Implement EX-OR Gate with the help of NAND Gate. Show the output at each step.*
(May 2013, 2014, 2017)

XOR Gate

To make a XOR gate we first derive at an expression that uses only NAND operations to realize an XOR operation. From that expression we can know how the NAND gates are to be connected. We know that;

$$f = A \oplus B = A\overline{B} + \overline{A}B = \overline{\overline{A\overline{B}} + \overline{\overline{A}B}} = \overline{\overline{A\overline{B}}.\overline{\overline{A}B}} = \overline{(A.\overline{B}.(\overline{A}.B)}\,.$$

The expression shows that first we have to complement each of the 2 variables to get $\overline{A}$ and $\overline{B}$. Then get each of the product terms by using NAND gates.



(Figure)

Finally the product terms are joined to get the final sum with another NAND gate.

36. *Express the Boolean function $F = P + Q'R$ in a sum of minterms.* **(2015, 2017)**

$$P = P(Q + Q') = PQ + PQ'$$
$$P = PQ(R + R') + PQ'(R + R')$$
$$P = PQR + PQR' + PQ'R + PQ'R'$$
$$Q'R = Q'R(P + P') = PQ'R + P'Q'R'$$
$$F = PQR + PQR' + PQ'R + PQ'R' + PQ'R + P'Q'R$$
$$F = P'Q'R + PQ'R' + PQ'R + PQR' + PQR$$

$= m_1 + m_4 + m_5 + m_6 + m_7$ (Ans.)

$F(P,Q,R) = \sum(1,4,5,6,7)$

37. *Let F (X, Y) = X' + Y. Simplify the expression for function :*
F (F (A + B, B), C) (2017)

$f(f(A + B, B), C) = f((A + B)' + B), C)$ $[\because (a + b)' = a'.b']$
$= ((A + B)' + B)' + C$
$= (A + B).B' + C = AB' + C$ (Ans.)

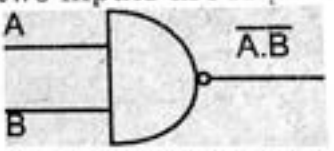
38. *Why NAND and NOR Gate is called Universal gate? Justify with example.* (2017)

NAND Gate as Universal Gate

The below diagram is of a two input NAND gate. The first part is an AND gate and second part is a dot after it represents a NOT gate. So it is clear that during the operation of NAND gate, the inputs are first going through AND gate and after that the output is reversed and we get the final output. Now we will look at the truth table of NAND gate.

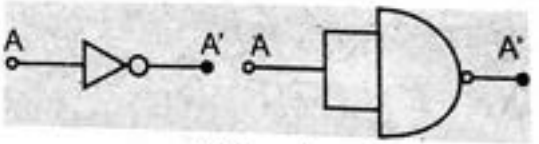
We will consider the truth table of the above NAND gate i.e. a two-input gate. The two inputs are *A* and *B*.

Inputs		Output
A	B	$X = \overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0



(Figure : NAND Gate)

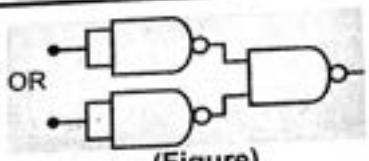
Now we will see how this gate can be used to make other gates.



(Figure)

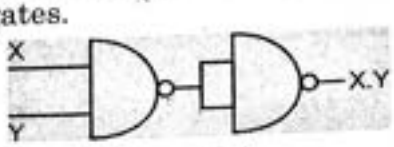
This is the circuit diagram of a NAND gate used to make work like a NOT gate, the original logic gate diagram of NOT gate is given beside.

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



(Figure)

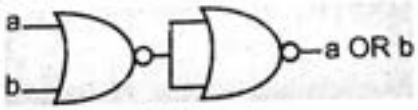
The above diagram is of an OR gate made from combinations of NAND gates, arranged in a proper manner. The truth table of an OR gate is also given beside the diagram. Now we will see the design of an AND gate from NAND gates.



(Figure)

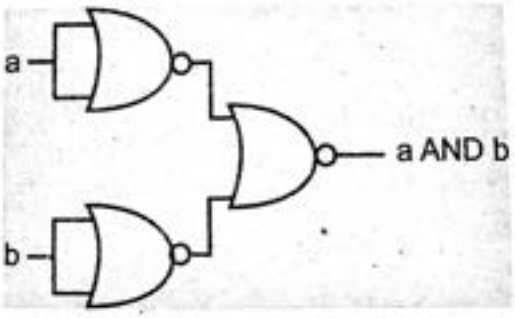
The above diagram is of an AND gate made from NAND gate. So we can see that all the three basic gates can be made by only using NAND gates, that's why this gate is called Universal Gate and it is appropriate.

NOR Gate as Universal Gate :



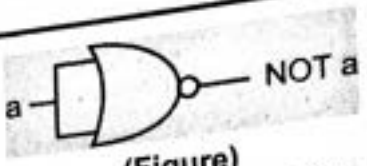
(Figure)

The above diagram is of an OR gate made by only using NOR gates. The output of this gate is exactly similar to that of a single OR gate. As we can see the circuit arrangement of OR gate using NOR gates is similar to that of AND gate using NAND gates.



(Figure)

The above diagram as the name suggests is of AND gate using only NOR gate, again we can see that the circuit diagram of AND gate using only NOR gate is exactly similar to that of OR gate using only NAND gates. Now we will finally see how a NOT gate can be made by using only NOR gates.



(Figure)

The above diagram is of a NOT gate made by using a NOR gate. The circuit diagram is similar to that of NOT gate made by using only NAND gate. So, from the above discussion it is clear that all the three basic gates (AND, OR, NOT) can be made by only using NOR gate. And thus, it can be aptly termed as Universal Gate.

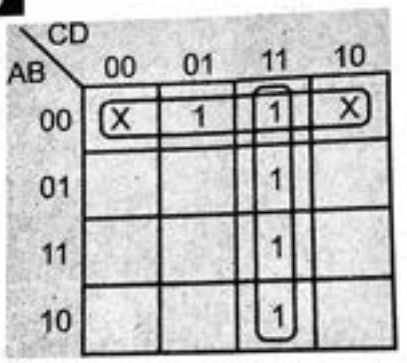
(2017)

39. ***Simplify using K – map :***

$F(A, B, C, D) = \Sigma(1, 3, 7, 11, 15)$

$d(A, B, C, D) = \Sigma(0, 2, 5)$

Using k-Map



(Figure)

The minterm m_0 and m_2 that is $A'B'C'D'$ and $A'B'CD'$ are the don't care terms which have been assumed as 1's while making a quad. The simplified SOP expression

$F = A'B' + CD.$

❑❑

COMBINATIONAL CIRCUITS ADDERS & SUBTRACTORS

1. ◆ *Explain half adder with block diagram and truth table.* (2023-24)
   ◆ *Explain half adder with truth table and logic circuit diagram.* (2016)

Half-Adder Circuit

A logic circuit for the addition of two one-bit numbers is referred to as an half-adder. The addition process is reproduced in truth table. Here, A and B are the two inputs and S (SUM) and C (CARRY) are the two outputs.

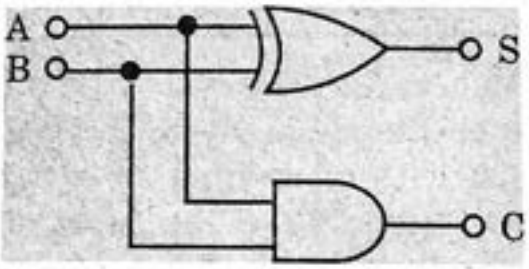
Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth table, we obtain the logical expression for S and C outputs as :

$$S = \overline{A}B + A\overline{B} = A \oplus B$$

$$C = AB$$

The realization of any half-adder using gates is shown in the figure.

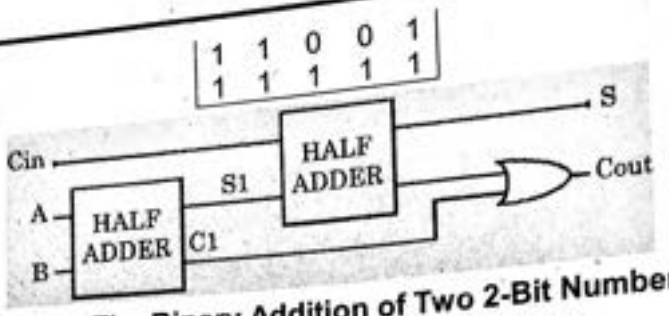


(Figure : Realization of an Half-Adder)

The Truth Table for Three Input Variables is Shown in Figure :

C_1	X_1	Y_1	Z_1	C_2
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

[B.2]



(Figure : The Binary Addition of Two 2-Bit Numbers)

From the truth table

$$C_2 = \overline{C}_1.X_1.Y_1 + C_1.\overline{X}_1.Y_1 + C_1.X_1.\overline{Y}_1 + C_1.X_1.Y_1$$

$$= X_1.Y_1 + C_1.X_1 + C_1.Y_1$$

2. ◆ ***What is Multiplexer? Construct a 16×1 line multiplexer with two 8×1 and one 2×1 line multiplexer.*** **(2023-24)**

◆ ***What is multiplexer? Draw the logic diagram of 8 x 1 multiplexer.*** **(2022-23)**

◆ ***What is a 4×1 MUX? Show its design detail. Design the following Boolean function using 4×1 MUX:***

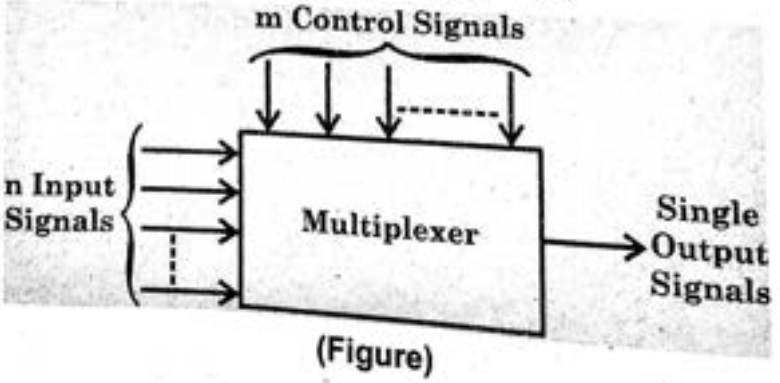
$$F(A, B, C) = \Sigma\, 1, 3, 5, 7$$ **(2019)**

◆ ***What is a multiplexer and demultiplexer? Explain how an 8 × 1 multiplexer can designed using two 4 × 1 multiplexers.*** **(2014, 2017)**

◆ ***What is Multiplexer?*** **(May 2012, 2014, 2016)**

Multiplexer

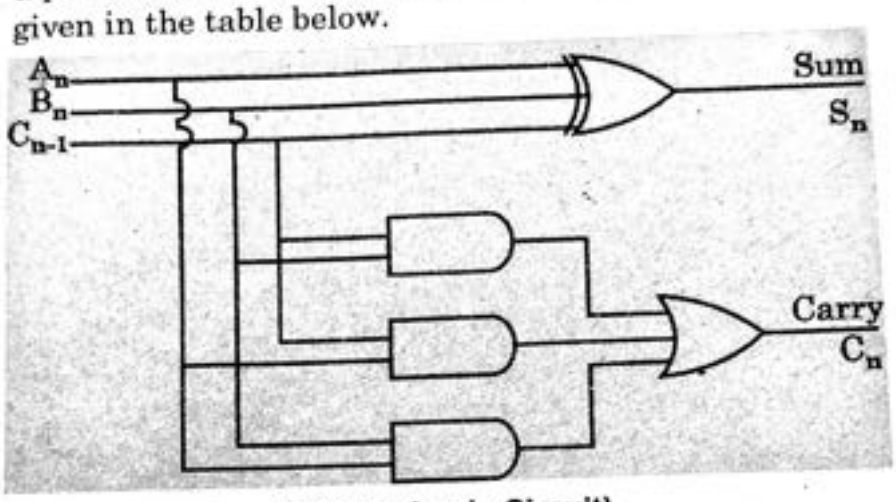
Multiplexer means many to 1. A multiplexer is an electronic circuit with many inputs but only one output. By applying control signals we can steer any input to the output. Figure shows the block diagram of a $n \times 1$ multiplexer. For a multiplexer with n inputs there must be at least m control signals where $2^m = n$.



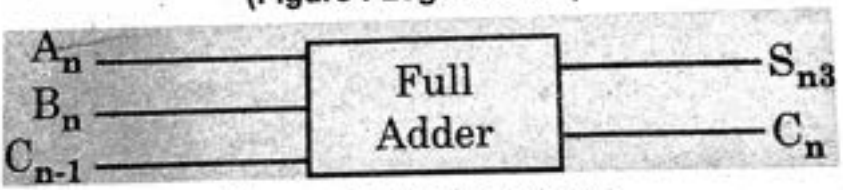
(Figure)

4 : 1 *MUX*

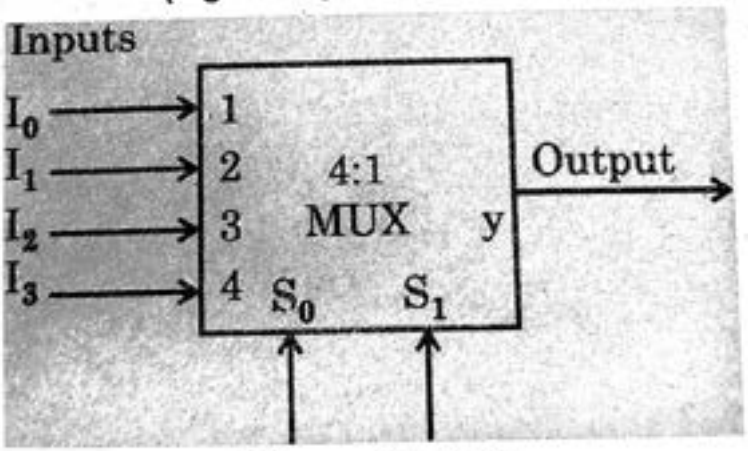
A 4 to 1 line multiplexer is shown in figure below, each of 4 input lines 10 to 13 is applied to one input of an AND gate. Selection lines S_0 and S_1 are decoded to select a particular AND gate. The truth table for the 4 : 1 mux is given in the table below.



(Figure : Logic Circuit)



(Figure : Symbolic Circuit)

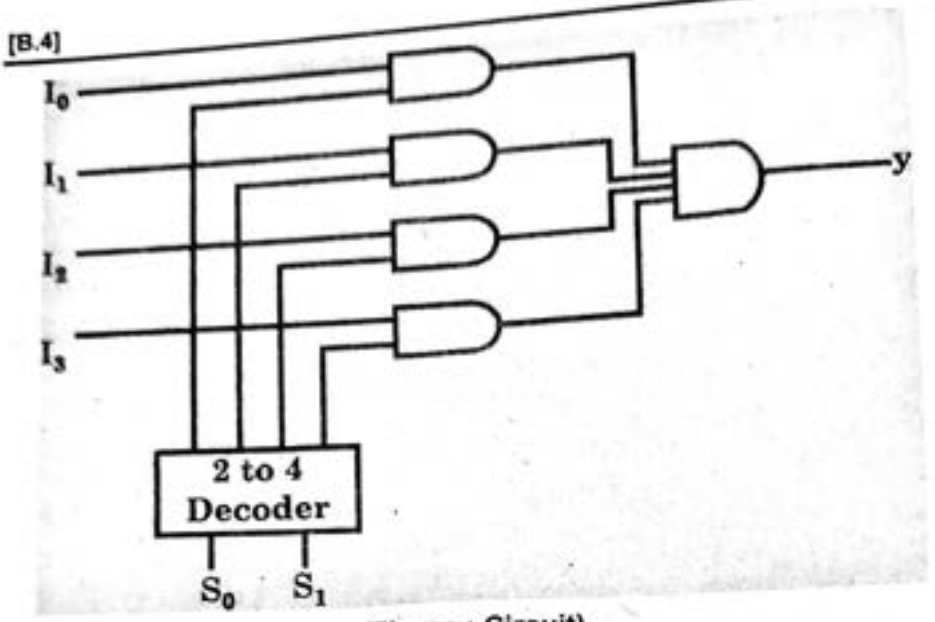


(Figure : Symbol)

Truth Table :

S_0	S_1	Y
0	0	10
0	1	11
1	0	12
1	1	13

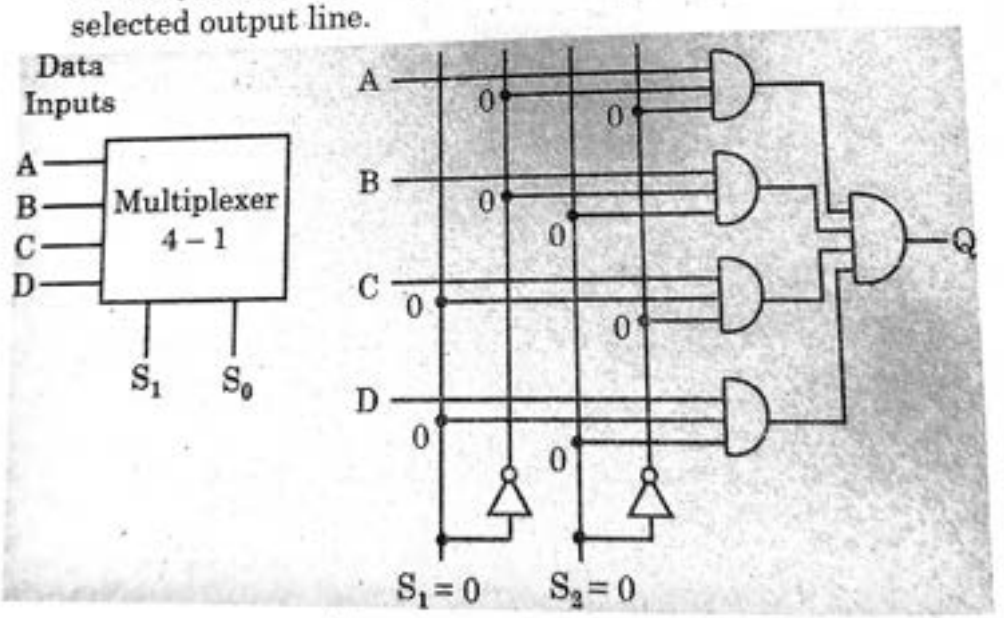
[B.4]



(Figure : Circuit)

Demultiplexer : A multiplexer or mux is a device that selects one of many data lines.

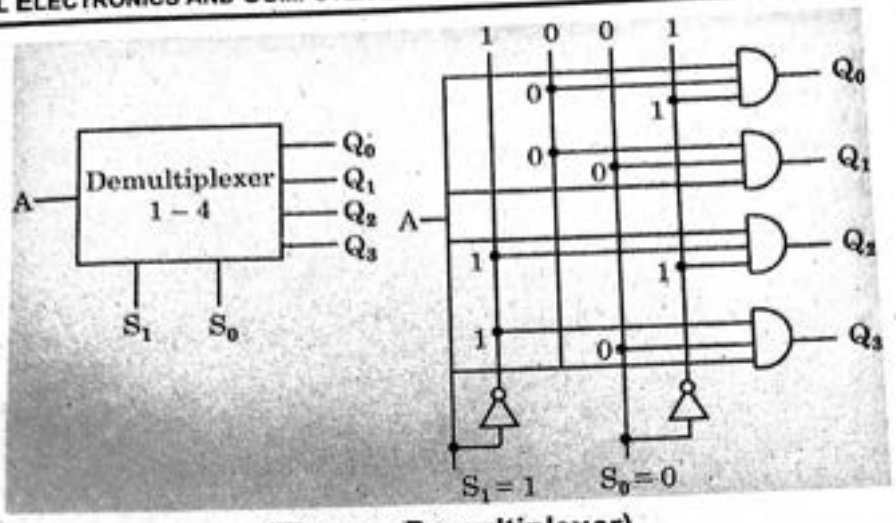
A demultiplexer or demux is a device that selects one of many output lines and connects the single input to the selected output line.



(Figure : Multiplexer)

Mulitplexer as a Universal Combinational Circuit : From the point of view of output(s) the multiplexer can be considered as a one level combinational circuit.

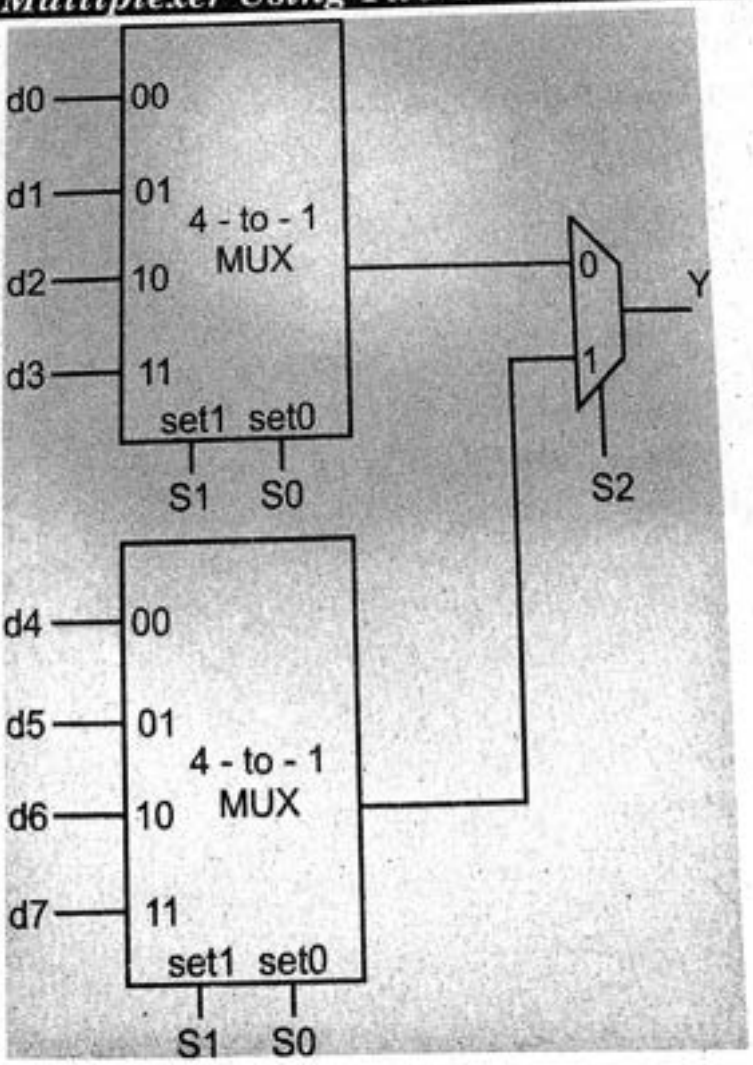
Its characteristics are the fast response time. For the selected input the time delay corresponds to the unit gate delay.



(Figure : Demultiplexer)

Demultiplexers are sometimes convenient for designing general purpose logic, because if the demultiplexer's input is always true(1), the demultiplexer acts as a decoder. This means that any function of the selection bits can be implemented by logically OR-ing the correct set of outputs.

8 × 1 *Multiplexer Using Two* 4 × 1 *Multiplexers*



(Figure)

Design of a circuit to implement an 8 – 1 mux using two 4 – 1 muxes and an OR Gate is given above

This is the truth table :

A	B	C	F
0	0	0	A0
0	0	1	A1
0	1	0	A2
0	1	1	A3
1	0	0	A4
1	0	1	A5
1	1	0	A6
1	1	1	A7

Truth Table for 8 × 1 Multiplexer

Numerical Solution :

Given function $F(A,B,C) = \Sigma m(1,3,5,7)$

	D_0	D_1	D_2	D_3	
$\overline{A}$	0	①	2	③	Row 1
A	4	⑤	6	⑦	Row 2
	0	1	0	1	

(Figure : Implementation Table)

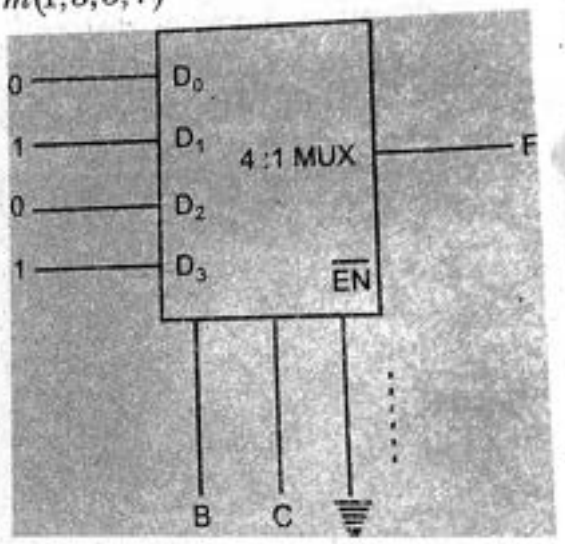


Figure :Multiplexer Implementation

Construction of 16-to-1 Line Multiplexer with Two 8-to-1 Line Multiplexers and one 2-to-1 Line Multiplexer :

Multiplexer is one of the basic building units of a computer system which in principle allows sharing of a common line by more than one input lines. It connects multiple input lines to a single output line. At a specific time one of the input lines is selected and the selected input is passed on to the output line.

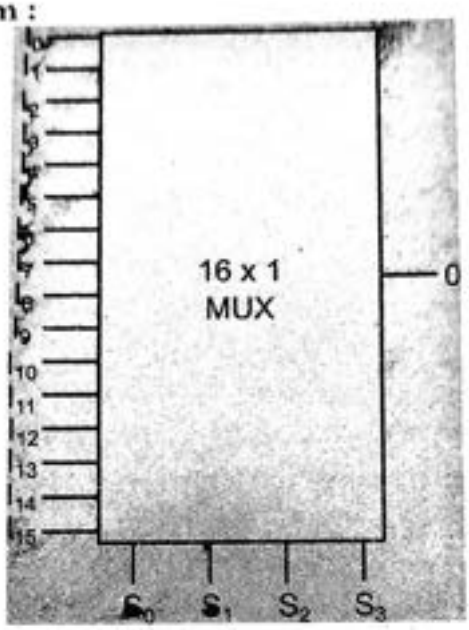
Relation between Multiple Input Lines and Selection Lines :

Input lines $16 = 2^4$ i.e. 4 Selection lines

Input lines will be $I_0 - I_{15}$

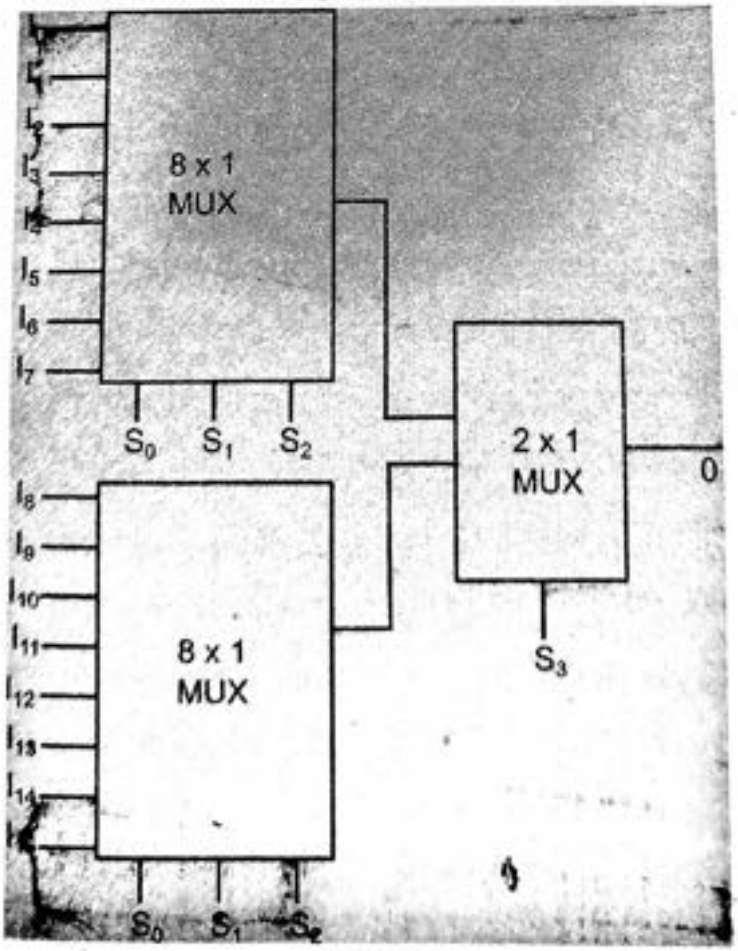
Selection lines will be $S_0 - S_3$

Block Diagram :



(Figure)

Constructed Diagram :



(Figure)

The diagram will be same as of the block diagram of 16-to-1 line multiplexer in which 8-to-1 line multiplexer Selection lines will be $S_0 - S_2$ and S_3 will be connected to 2-to-1 line multiplexer Selection and First 8-to-1 line multiplexer Input lines will be $I_0 - I_7$ and Second 8-to-1 line multiplexer Input lines will be $I_8 - I_{15}$.

3. ◆ ***Define full adder with logic circuit diagram & truth table.*** **(2023-24)**
   ◆ ***Write short note on Full Adder circuit.*** **(2018)**
   ◆ ***Explain full adder with truth table and logic circuit diagram.*** **(2017)**
   ◆ ***Explain full-adder with truth table. Construct logic diagram of full-adder using half-adder.*** **(2016)**
   ◆ ***Write a short note on Full adder.*** **(2015)**

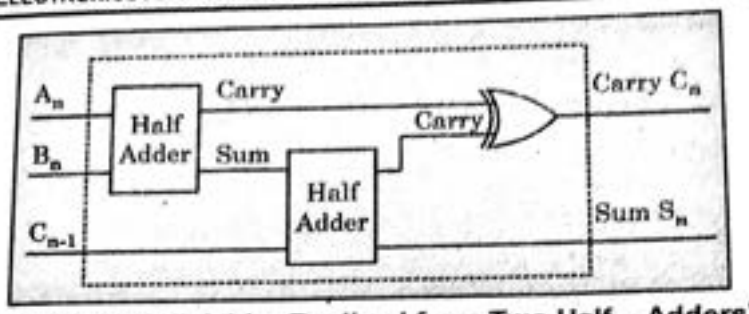
Full Adder

When two n-bit numbers are added there may be a carry from one stage to the next stage. The carry coming out from one stage needs to be added to the next stage. A half adder cannot add 3 bits as it has only 2 input terminals. Therefore, a logic circuit which can add 3 bits is required. The logic circuit which can add 3 bits is known as a full-adder. Table is the truth table for a full adder.

INPUTS			OUTPUTS	
A_n	B_n	C_{n-1}	SUM S_n	CARRY C_n
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

Truth Table for a Full-Adder

A full-adder can be built using two half adders and an XOR gate as shown in figure :



(Figure : A Full-Adder Realized from Two Half – Adders)

A logic circuit to behave according to the truth-table given in table given can be realized as seen in figure. The sum of product expression for sum and CARRY are written below.

It will be easy to write logic expression if we assume $A_n = A,\ B_n = B,\ C_{n-1} = C$

$$\text{SUM}(S_n) = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + ABC$$

$$= \overline{A}\,(\overline{B}\,C + B\,\overline{C}) + A\,(\overline{B}\,\overline{C} + B\,C)$$

$$= \overline{A}\,(B \oplus C) + A(\overline{B \oplus C}) = A \oplus B \oplus C$$

$$= A_n \oplus B_n \oplus C_{n-1} \quad \text{(In terms of } A_n, B_n \text{ and } C_{n-1})$$

Where C_{n-1} is the carry-in from the previous stage.

Hence, $\text{CARRY}(C_n) = \overline{A}\,BC + A\,\overline{B}C + AB\,\overline{C} + ABC$

This expression can easily be deducted by Karnaugh map method. It can also be deduced by algebraic method as given below :

If we add two mare ABC terms we have,

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

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

$$= BC(A + \overline{A}) + AC(B + \overline{B}) + AB(C + \overline{C})$$

$$= \text{AB} + \text{BC} + \text{AC} = A_nB_n + B_nC_{n-1} + A_nC_{n-1}$$

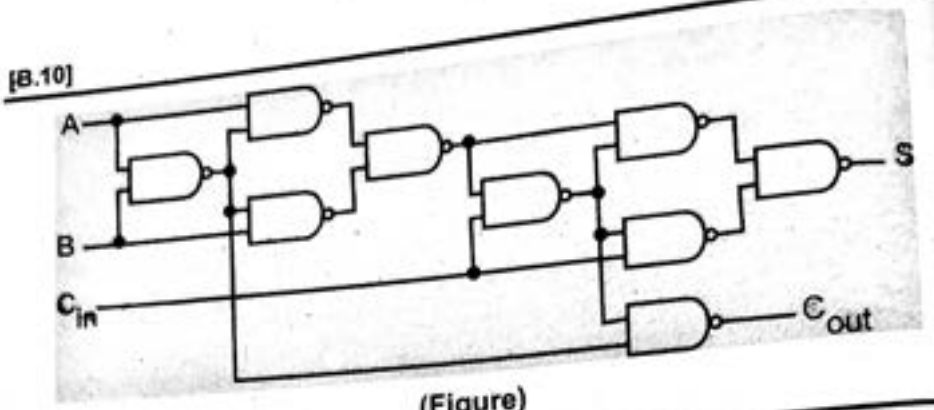
(Where C_{n-1} is the carry from the previous stage)

An electronic circuit for full adder can be realized as shown in figure.

Full Adder Circuit Using NAND Gates Only

We know that NAND gates are universal gates so that we can design any type digital circuit only with NAND gates. Here is the circuit to design Full adder with NAND gates.

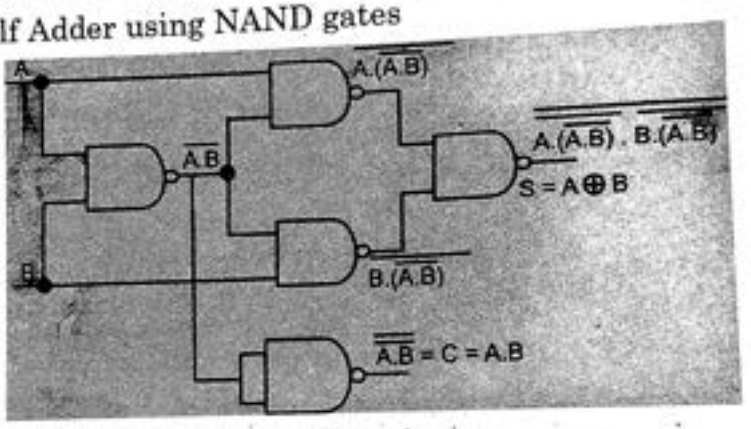
[B.10]



(Figure)

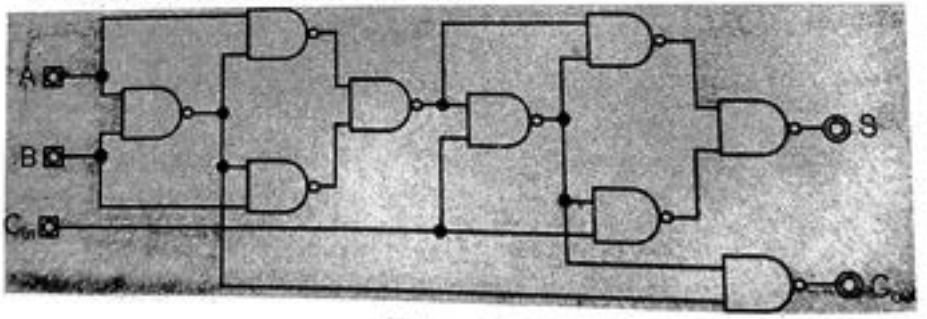
4. *Design half adder and full adder using only NAND gates.* (2022-23)

Half Adder using NAND gates



(Figure)

Full Adder using NAND gates



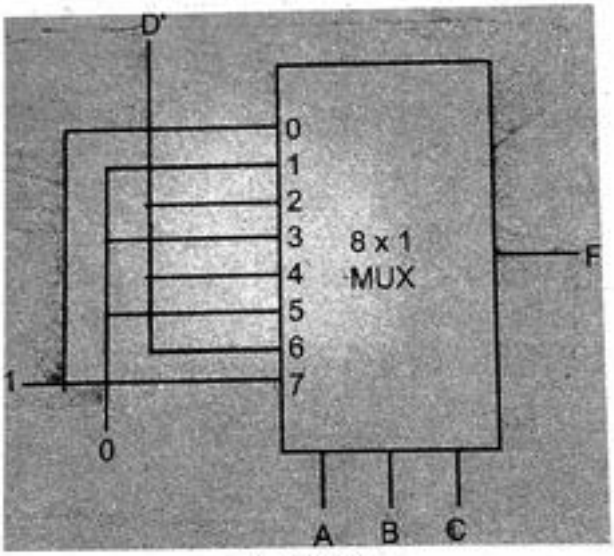
(Figure)

5. *Implement the following function using 8 : 1 MUX :*
$F(A, B, C, D) = \Sigma m(0,1, 4, 8,12, 14, 15)$ **(2022-23)**

Following table describes the truth values of A, B, C, D and function. For 8x1 multiplexer A, B and C are input to the select lines and D is considered for input.

Select Lines	Minterm	A	B	C	D	Function	Input
0	0	0	0	0	0	1	1
	1	0	0	0	1	1	
1	2	0	0	1	0	0	0
	3	0	0	1	1	0	
2	4	0	1	0	0	1	D'
	5	0	1	0	1	0	
3	6	0	1	1	0	0	0
	7	0	1	1	1	0	
4	8	1	0	0	0	1	D'
	9	1	0	0	1	0	
5	10	1	0	1	0	0	0
	11	1	0	1	1	0	
6	12	1	1	0	0	1	D'
	13	1	1	0	1	0	
7	14	1	1	1	0	1	1
	15	1	1	1	1	1	

As per the above table for the 8x1 multiplexer is as follows :



(Figure)

6. ◆ *What is Encoder? Explain in brief about octal to Binary Encoder.* (2022-23)
   ◆ *Show the logic diagram of 3×8 decoder with enable input (active when high i.e. enable input = 1).* (2019)
   ◆ *What is Encoder? Explain.*
   ◆ *Construct a logic diagram of 4 ×16 line decoder using 3 × 8 line decoder.* (2018)
   ◆ *What is decoder? Show the logic circuit of 3 × 8 decoder.* (2018)

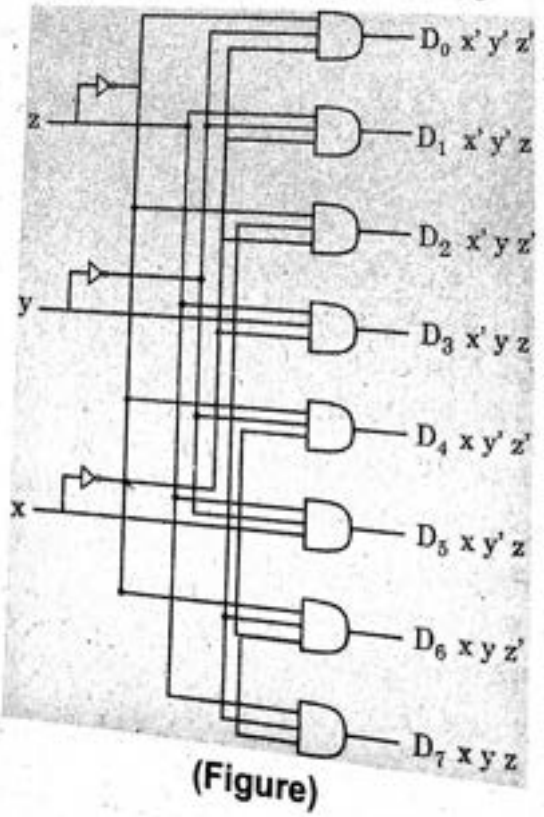
- *What is decoder? Design a BCD to decimal decoder.* (2017)
- *What is Decoder? Explain the Decoder expansion. Construct a logic diagram of 4 to 16 line decoder using 3 to 8 line decoder.* (2016)
- *What is decoder?* (2015)
- *Design 4 × 16 decoder with two 3 × 8 decoder.* (2015)

Decoders

A decoder is combination circuit that convert binary information from n input lines to a maximum of 2^n unique output lines. If n bit decoder information has unused or don't care combination. The decoder output will have less than 2^n output.

Truth Table :

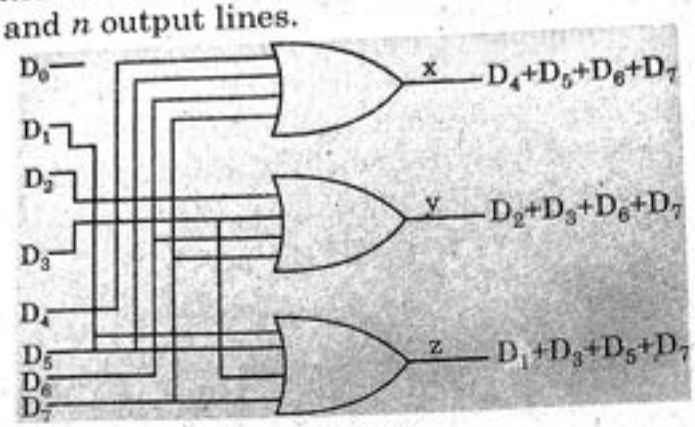
Input			D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
x	y	z								
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



(Figure)

Encoder

Encoder is a digital function that produces a reverse operation from that of decoder. An encoder has 2^n input lines and n output lines.

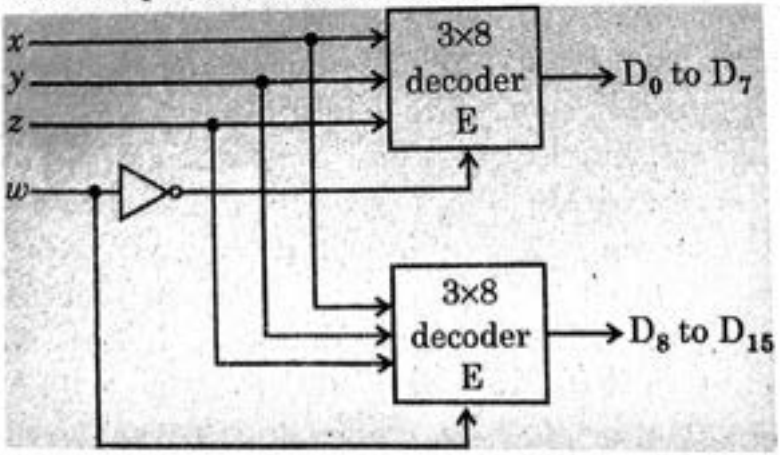


(Figure)

The output lines generate the binary codes for the 2^n input variables. An encoder in figure assumes that only one input line can be equal to 1 at any time.

Decoder Expansion and Construction of a Logic Diagram

Decoder/demultiplexer circuits can be connected together to form a larger decoder circuit. Figure shows two 3 × 8 decoders with enable inputs connected to form a 4 × 16 decoder. When $w = 0$, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's, and the top eight outputs generate minterms 0000 to 0111. When $w = 1$, the enable conditions are reversed; the bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's.



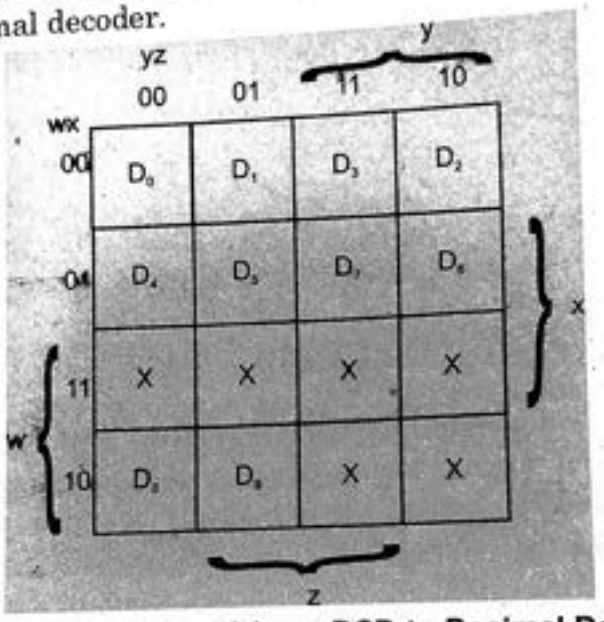
(Figure : A 4 × 16 Decoder Constructed with Two 3 × 8 Decoders)

This example demonstrates the usefulness of enable inputs in ICs. In general, enable lines are a convenient

feature for connecting two or more IC packages for the purpose of expending the digital function into a similar function with more inputs and outputs.

BCD to Decimal Decoder

The elements of information in this case are the ten decimal digits represented by the BCD code. The code itself has four bits. Therefore, the decoder should have four inputs to accept the coded digit and ten outputs, one for each decimal digit. This will give a 4-line to 10-line BCD-to-decimal decoder.



(Figure : Map for Simplifying a BCD-to-Decimal Decoder)

There is really no need to design such a decoder because it can be found in IC form as an MSI function. We will design it anyway for two reasons. First, it gives insight on what to expect in such an MSI function. Second, this is a good example for demonstrating the practical consequences of don't-care conditions.

Since the circuit has ten outputs, it would be necessary to draw ten maps to simplify each one of the output functions. There are six don't-care conditions here, and they must be taken into consideration when we simplify each of the output functions. Instead of drawing ten maps, we will draw only one map and write each of the output variables, D_0 to D_9 inside its corresponding minterm square as shown in Fig. Six input combinations will never occur, so we mark their corresponding minterm squares with X's.

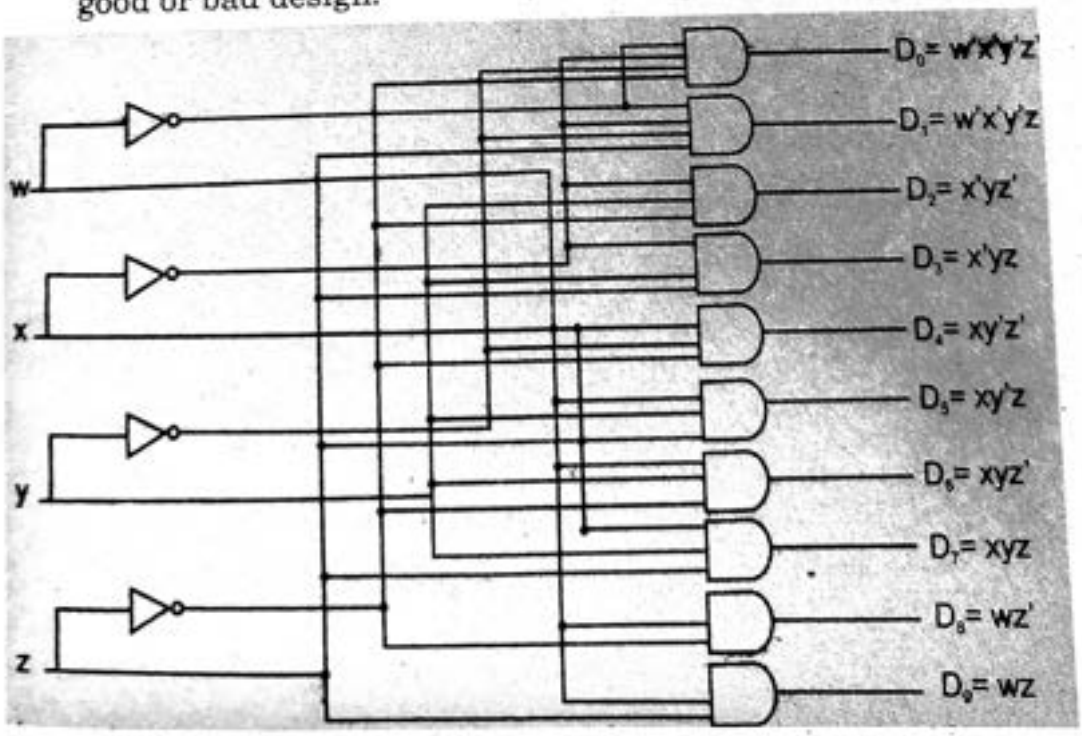
It is the designer's responsibility to decide on how to treat the don't-care conditions. Assume that it is decided to use them in such a way as to simplify the functions to the minimum number of literals. D_0 and D_1 cannot be combined with any don't-care minterms. D_2 can be combined with the don't-care minterm m_{10} to give :

The square $D_2 = x'yz'$ with D_9 can be combined with three other don't-care squares to give :

$$D_9 = wz$$

Using the don't-care terms for the other outputs, we obtain the circuit in Fig. Thus the don't-care terms cause a reduction in the number of inputs in most of the AND gates.

A careful designer should investigate the effect of the above minimization. Although it is true that under normal operating conditions the invalid six combinations will never occur, what if there is a malfunction and they do occur? An analysis of the circuit of Fig. shows that the six invalid input combinations will produce outputs as listed in table. The reader can look at the table and decide whether this is a good or bad design.



(Figure : BCD-to-Decimal Decoder)

Inputs				Output									
w	x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
1	0	1	0	0	0	1	0	0	0	0	0	1	0
1	0	1	1	0	0	0	1	0	0	0	0	0	1
1	1	0	0	0	0	0	0	1	0	0	0	1	0
1	1	0	1	0	0	0	0	0	1	0	0	0	1
1	1	1	0	0	0	0	0	0	0	1	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1	0	1

Truth Table of Octal to Binary Encoder is given below

Inputs								Outputs		
D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

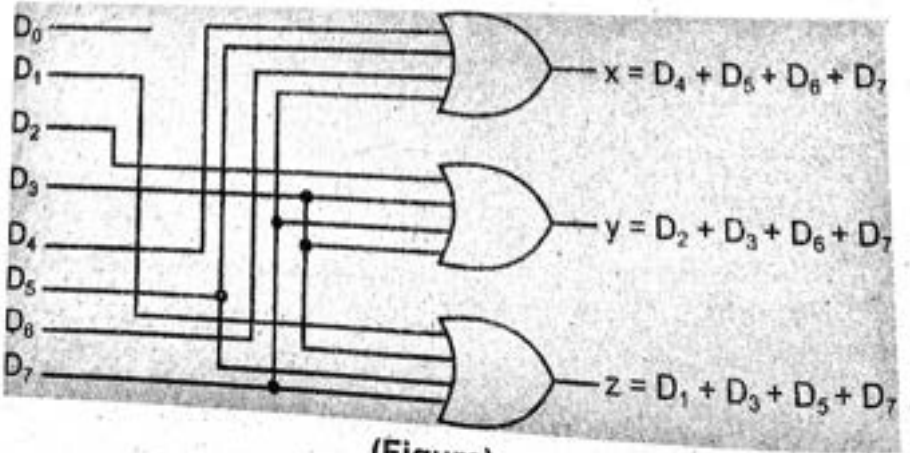
The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output $A_0 = 1$ if the input octal digit is 1 or 3 or 5 or 7. Similar conditions apply for other two outputs:

These conditions can be expressed by the following Boolean functions :

$$A_0 = D_1 + D_3 + D_5 + D_7$$

$$A_1 = D_2 + D_4 + D_6 + D_7$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$



(Figure)

7. *Design*
3 x 8 de

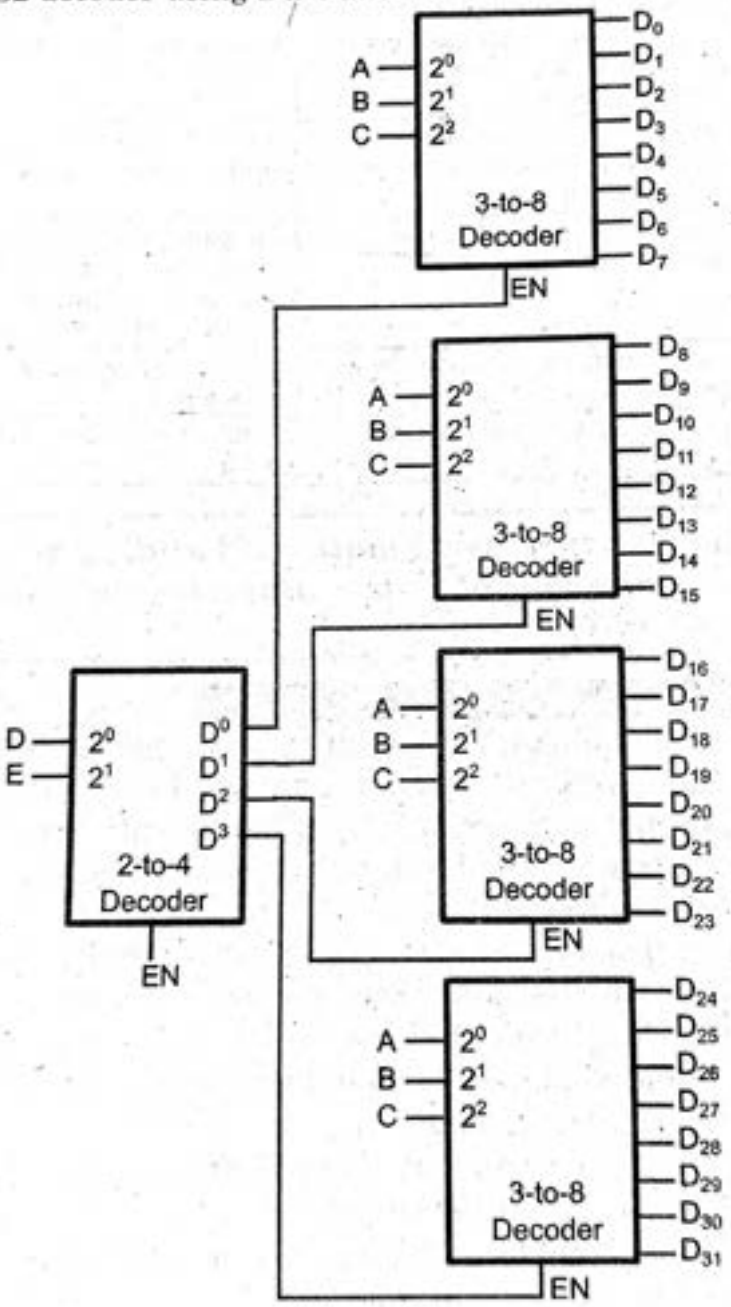
5 x 32 d

8. ◆

◆

7. *Design a 5 x 32 decoder using one 2 x 4 and four 3 x 8 decoder IC's.* **(2022-23)**

5 x 32 decoder using 2 x 4 and 3 x 8 decoders :



(Figure)

8. ◆ ***Differentiate between combinational and sequential circuit.*** **(2018, 2022-23)**

◆ ***Write short note on Combination and Sequential circuit.*** **(2014)**

Difference between Sequential and Combinational Circuit

Sequential Circuits	Combinational Circuits
In sequential circuits, the output variables dependent not only on the present input variables but they also depend upon the past history of these input variables.	In combinational circuits, the output variables are at all times dependent on the combination of input variables.
Memory unit is required to store the past history of input variables in the sequential circuit.	Memory unit is not required in combinational circuits.
Sequential circuits are slower than the combinational circuits.	Combinational circuits are faster in speed because the delay between input and output is due to propagation delay of gates.
Sequential circuits are comparatively harder to design	Combinational circuits are easy to design.
Serial adder is a sequential circuit.	Parallel adder is a combinational circuit.

9. *Name any three combinational building blocks and the purpose of its usage(s) in computer organization.* (2019)

Combinational Building Blocks

A combinational circuit is a system containing basic Boolean operation, some inputs and set of outputs. Since each output corresponds to an individual logic function, a combinational circuit after implements several different Boolean function.

In combinational circuits the output depends on previous inputs i.e. there is no memory element in combinational logic circuits.

Some combinational circuits are :

(1) Arithmetic circuits
(2) Multiplexer and Demultiplexer
(3) Decoders and Encoders

Arithmetic circuits are used to perform addition and subtraction.

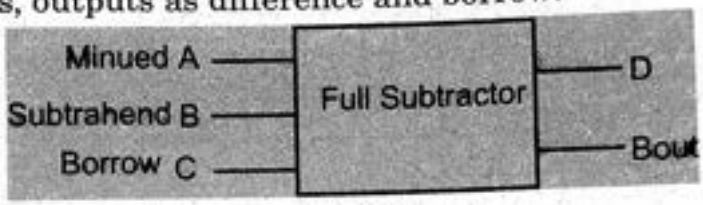
Multiplexers are used in data selection and data routing and demultiplexer use to performs reverse operation of a multiplexer.

Decoder used to implement combinational circuits and also used in memories to select particular register. Encoder performs the reverse operation of decoder.

10. *Write short note on the Full subtractor circuit.* **(2019)**

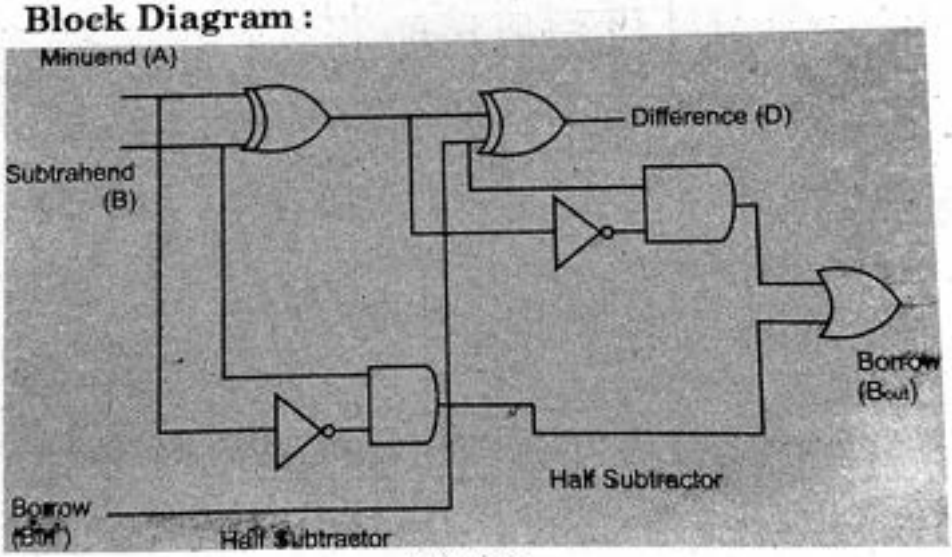
Full Subtractor Circuit

Full subtractor is an electronic device or logic circuit which performs subtraction of two binary digits. It is a combinational logical circuit used in digital electronics. It is famed by two half subtractors which involves three inputs such as minued, subtrahend and borrow, borrow bit among the inputs is obtained from subtraction of two binary digits and is subtracted from next higher order Pair of bits, outputs as difference and borrow.



(Figure)

Block Diagram :



(Figure)

Truth – Table :

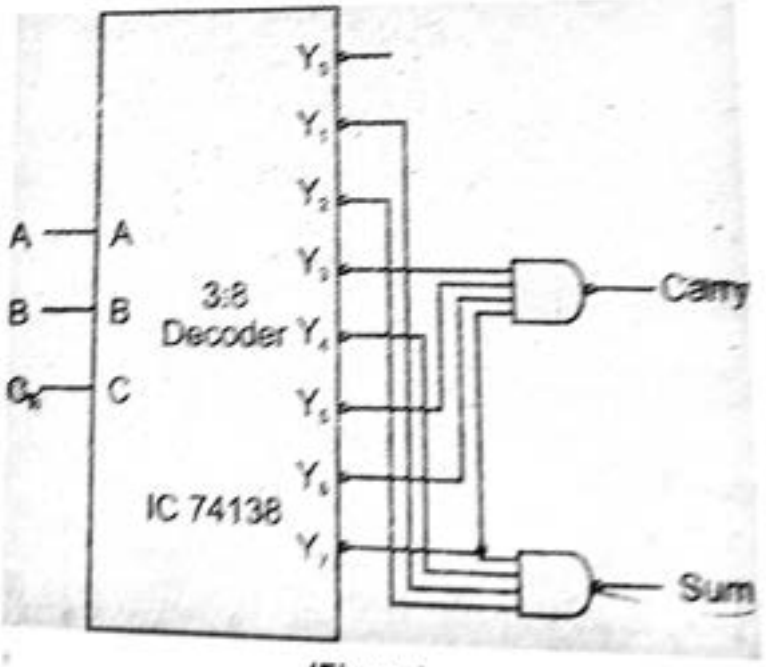
Inputs			Outputs	
Minued (A)	Subtrahend (B)	Borrow (Bin)	Difference (D)	Borrow (Bout)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

11. ◆ *Using a decoder and OR gates design the circuit of full adder.* (2013)
◆ *Implement Full Adder circuit with a decoder and two OR gates.* (2017)
◆ *Implement the full adder circuit with a decoder and two OR gate. Draw the truth table.* (May 2012)

Truth table for Full-adder is shown in the table.

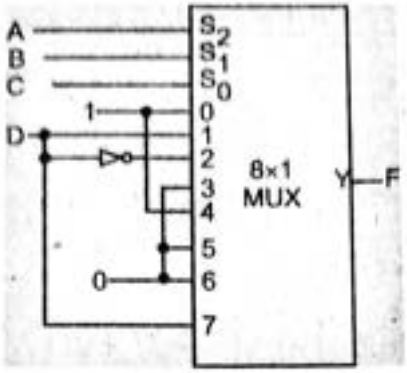
Inputs			Output	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table Truth Table for Full-adder



(Figure)

12. *Implement the following function with a multiplexer :*
$F(A, B, C, D) = \Sigma\ (0, 1, 3, 4, 8, 9, 15)$ (2017)



Truth Table :

A	B	C	D	F	
0	0	0	0	1	F = 1
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	F = D'
0	1	0	1	0	
0	1	1	0	0	F = D
0	1	1	1	0	
1	0	0	0	1	F = 1
1	0	0	1	1	
1	0	1	0	0	F = 0
1	0	1	1	0	
1	1	0	0	0	F = 0
1	1	0	1	0	
1	1	1	0	0	F = D
1	1	1	1	1	

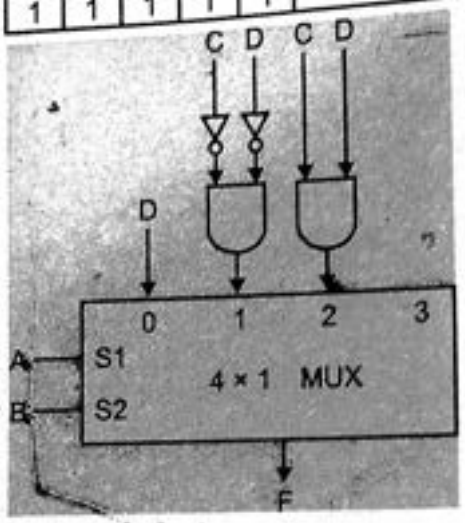
13. ***Realize the following Boolean expression using 4 × 1 multiplexer :*** **(2016)**

$$Z + \overline{ABCD} + \overline{A}BC\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D} + A\overline{BCD} + ABCD$$

Boolean Expression Using 4 × 1 Multiplexer

A	B	C	D	F	
0	0	0	0	0	
0	0	0	1	1	F = D
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	1	
0	1	0	1	0	F = C'D'
0	1	1	0	0	
0	1	1	1	0	

1	0	0	0	0	
1	0	0	1	0	F = CD
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	1	
1	1	0	1	1	F = 1
1	1	1	0	1	
1	1	1	1	1	



(Figure)

❑❑

SEQUENTIAL CIRCUIT

1. ***Differentiate between combinational & sequential circuit.*** **(2023-24)**

Difference between Combinational & Sequential Circuit

Feature	Combinational Circuit	Sequential Circuit
Output	Only depends on current input	Depends on current input and previous state
Feedback	No feedback path	Contains feedback path (memory elements)
Timing	No concept of clock	Operates with the concept of clock
Timing Requirements	No timing constraints	Timing constraints are critical
Memory	No memory elements (no storage of previous inputs)	Contains memory elements (storage of previous states)
Logic Function	Implemented using boolean logic gates and Boolean algebra	Can involve both combinational and sequential logic components
Examples	Adders, subtractors, multiplexers, decoders, encoders, etc.	Flip-flops, counters, registers, state machines, etc.

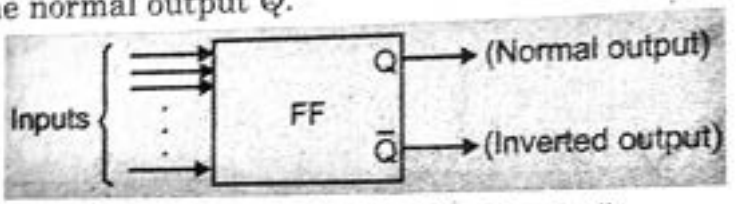
2. ◆ ***What is flip flop?*** **(2023-24,2013)**
   - ◆ ***What do you mean by Flip Flop? Show characteristic table of RS, JK, D and T flip flops. What is the excitation table by JK flip flop?*** **(2019)**
   - ◆ ***Explain the working of RS and D flip-flops.*** **(2018)**
   - ◆ ***What is flip – flop? Explain the working of RS flip- flop using logic diagram.*** **(2018)**
   - ◆ ***Write short note on D flip-flop.*** **(2016)**
   - ◆ ***Describe and discuss the operation of a D-type flip-flop.***
   - ◆ ***What is flip-flop? Explain the working of RS flip-flop using logic diagram and excitation table.*** **(2016)**

SCAN AnyScanner

Flip-Flop

A flip-flop (FF), known more formally as a bistable multivibrator, has two stable states. It can remain in either of the states indefinitely. Its state can be changed by applying the proper triggering signal. It is also called a binary or one-bit memory.

Figure shows the general type of symbol used for a flip-flop. The flip-flop has two outputs, labeled Q and $\overline{Q}$. Actually any letter can be used to represent the output, but Q is the one most often used. The Q output is the normal output of the flip-flop and $\overline{Q}$ is the inverted output. The outputs as well as its complement are available for each flip-flop. The state of the flip-flop always refers to the state of the normal output Q.



(Figure : General Flip-Flop Symbol)

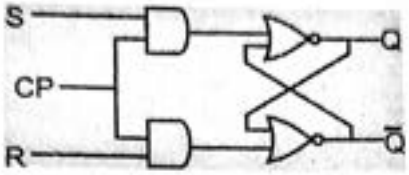
The inverted output $\overline{Q}$ is in the opposite state. A flip-flop is said to be in HIGH state or logic 1 state or SET state when $Q = 1$, and in LOW state or logic 0 state or RESET state or CLEAR state when $Q = 0$.

As the symbol in figure implies, a flip-flop can have one or more inputs. The input signals which command the flip-flop to change state are called excitations. These inputs are used to cause the flip-flop to switch back and forth (i.e. 'flip-flop') between its possible output states. A flip-flop input has to be pulsed momentarily to cause a change in the flip-flop output, and the output will remain in that new state even after the input pulse has been removed. This is the flip-flop's memory characteristic.

Working of RS Flip-Flop Using Logic Diagram and Excitation Table

The clocked RS flip flop shown in figure consists of a basic NOR flip-flop and two AND gates. The outputs of the two AND gates remain at 0 as long as the clock pulse is 0, regardless of the S and R input values. When the clock pulse goes to 1, information from the S and R input is allowed to reach the basic flip-flop. The set state is reached with $S = 1$, $R = 0$ and $CP = 1$. To change to the clear state,

the inputs must be $S = 0$, $R = 1$ and $CP = 1$. With both $S = R = 1$ the occurrence of a clock pulse causes both outputs to go to 0.



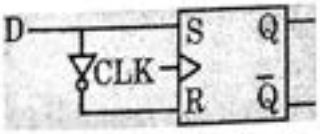
(Figure)

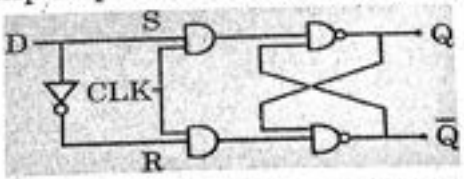
CP	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X
X = Indetermined			

Working of D Flip – Flop

The output of RS flip-flop is quite unpredictable when both the inputs are same 00 or 11. These input conditions can be avoided by making them complementary to each other. This is done by connecting an input to S and its complement to R at same time.

This modified SR flip-flop is termed as D flip-flop.





The D input goes directly to S and its complement is applied to R.

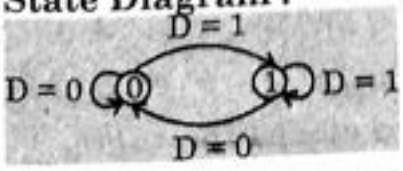
When, $D = 0$

$S = 0$ and $R = 1$ i.e., $Q_{n+1} = 0$

When, $D = 1$

$S = 1$ and $R = 0$ i.e., $Q_{n+1} = 1$

State Diagram :

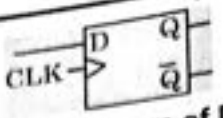


Truth Table :

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

As, in this flip-flop the output is same as that of input, that's why it is called Delay flip-flop.

[C.4]

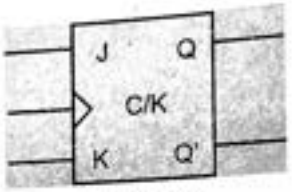


(Figure : Block Diagram of D Flip-flop)

Excitation Table :

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

JK – Flip Flop :



(Figure)

Characteristic Table :

J	K	$Q_{(next)}$
0	0	Q
0	1	0
1	0	1
1	1	Q'

Characteristic Equation :

$$Q_{(next)} = JQ' + K'Q$$

Excitation Table :

Q	$Q_{(next)}$	J	K
0	0	1	×
0	1	1	×
1	0	×	1
1	1	×	0

3. ◆ *Explain working of SR & JK flip flop with logic diagram and characteristics table.* **(2023-24)**

◆ *Explain the working of JK Flip – Flop using logic diagram and excitation table.* **(2017)**

◆ *Define J – K flip-flop with suitable diagram and truth table. How it is different from S – R flip-flop.* **(2016)**

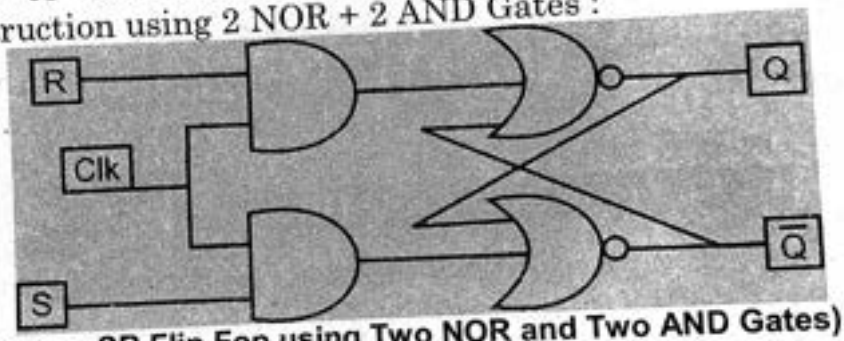
- *Explain the JK flip – flop with its working.* (May 2013, 2015)
- *Write short note on J-K flip-flop.* (2014)

Working of SR

It is a Flip Flop with two inputs, one is S and other is R. S here stands for Set and R here stands for Reset. Set basically indicates set the flip flop which means output 1 and reset indicates resetting the flip flop which means output 0. Here clock pulse is supplied to operate this flip flop, hence it is clocked flip flop.

Construction of SR Flip Flop : We can construct SR flip with two ways, one is with 2 NOR Gates + 2 AND Gates and other is with 4 NAND Gates.

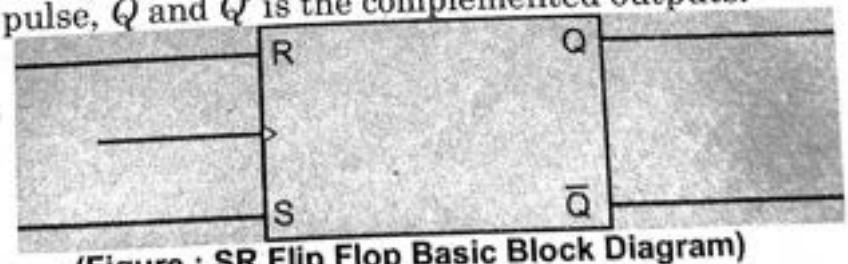
Ways to Construct SR Flip Flop : SR Flip Flop Construction using 2 NOR + 2 AND Gates :



(Figure : SR Flip Fop using Two NOR and Two AND Gates)

SR Flip Flop Construction using 4 NAND Gates :

Basic Block Diagram of SR Flip Flop : The basic block diagram contains S and R inputs, and between them is clock pulse, Q and Q' is the complemented outputs.



(Figure : SR Flip Flop Basic Block Diagram)

Working of SR Flip Flop :

(1) **Case 1 :** Let's say, $S = 0$ and $R = 0$, then output of both AND gates will be 0 and the value of Q and Q' will be same as their previous value, i.e, Hold state.

(2) **Case 2 :** Let's say, $S = 0$ and $R = 1$, then output of both AND gates will be 1 and 0, correspondingly the value of Q will be 0 as one of input is 1 and it is a NOR gate so it will ultimately gives 0, hence Q gets 0 value, similarly Q' will be 1.

(3) **Case 3 :** Let's say, $S = 1$ and $R = 0$, then output of both AND gates will be 0 and 1, correspondingly the value of Q' will be 0 as one of input to NOR gate is 1, so output will be 0 ultimately and this 0 value will go as input to upper NOR gate, and hence Q will become 1.

(4) **Case 4 :** Let's say, $S = 1$ and $R = 1$, then output of both AND gates will be 1 and 1 which is invalid, as the outputs should be complement of each other.

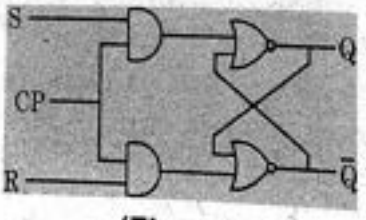
Truth Table of SR Flip Flop :

S	R	Qn+1	State
0	0	Qn	Hold
0	1	0	Reset
1	0	1	Set
1	1	X	Invalid

Here, S is the Set input, R is the reset input, $Qn+1$ is the next state and State tells in which state it enters.

JK Flip Flop with Logic Diagram :

A closed JK flip-flops is shown in figure. The working is same as S – R flip-flop. The only difference is the stage where $J = K = 1$.



(Figure)

A J – K flip flop is a refinement of the RS flip flop in that the indeterminate state of the RS type is defined in the JK type. A feedback connection in JK flip flop used to overcome this issue.

CP	J	K	Q(t + 1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

4. *What are the advantages of JK flip – flap over an SR flip – flop?* **(2022-23)**

JK flip-flops and SR flip-flops are both types of sequential logic circuits, but they have some differences in terms of functionality and advantages. Here are some advantages of JK flip-flops over SR flip-flops :

(1) No Invalid State : SR flip-flops can enter an invalid or indeterminate state when both the Set (S) and Reset (R) inputs are asserted simultaneously (i.e., $S = R = 1$). This condition is called the "forbidden" or "race" condition and can lead to unpredictable behavior. JK flip-flops, on the other hand, do not have this issue. When both J and K inputs are high ($J = K = 1$), the JK flip-flop toggles its state, eliminating the possibility of the forbidden state.

(2) Toggle Functionality : The JK flip-flop has a toggle or "T" functionality that allows it to change its output state when both J and K inputs are high ($J = K = 1$). This toggle behavior makes the JK flip-flop useful for applications where you need to alternate between two states or generate a periodic signal.

(3) Simplified Design : JK flip-flops offer a more streamlined design compared to SR flip-flops. With SR flip-flops, you need additional logic gates (NAND or NOR gates) to prevent the forbidden state. JK flip-flops inherently provide the toggle functionality and eliminate the need for extra gates, making them more efficient in terms of circuit complexity.

(4) Synchronous Operation : Both JK and SR flip-flops can be used in synchronous circuits, where all flip-flops are driven by a common clock signal. However, JK flip-flops are more commonly used in synchronous designs due to their simpler behavior and lack of invalid states. The J and K inputs can be changed during a specific clock edge to control the desired state transition, providing greater control and synchronization in sequential circuits.

5. *Explain the procedure to convert one flip – flop to another flip – flop.* (2022-23)

Follow these steps for converting one flip-flop to the other.

(1) Examine the characteristic table of the target flip-flop.

(2) Determine the excitation values required for each combination of present state and next state, filling in the excitation table for all flip-flops.

(3) Derive simplified expressions for each excitation input, using methods like Karnaugh maps if needed.

(4) Create the circuit diagram of the desired flip-flop using the given flip-flop and any necessary logic gates, based on the simplified expressions.

SR Flip-flop to D flip-flop Conversion

In this scenario, we have an initial flip-flop that is an SR flip-flop, and the desired flip-flop is a D flip-flop. Thus, let's examine the characteristic table of the D flip-flop to proceed.

D flip-flop input	Present State	Next State
D	Q_t	Q_{t+1}+1
0	0	0
0	1	0
1	0	1
1	1	1

Given that an SR flip-flop has inputs S and R, we can determine the excitation values of the SR flip-flop for different combinations of present state and next state values. The characteristic table of the D flip-flop, along with the corresponding excitation inputs for the SR flip-flop, is presented in the following table.

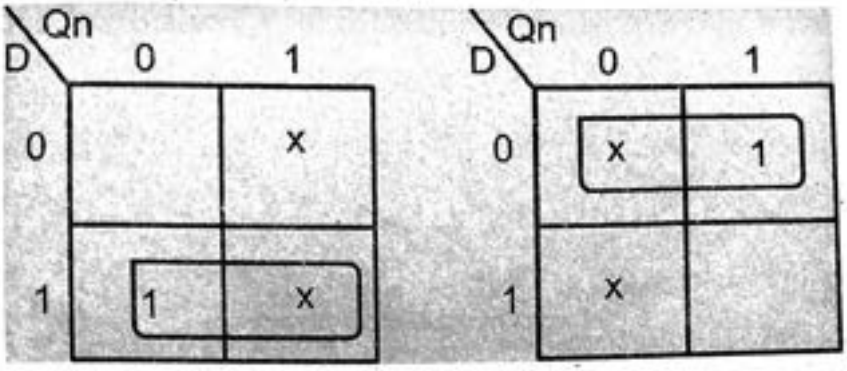
D flip-flop Input	Present State	Next State	SR flip-flop Inputs	
D	Q_t	Q_{t+1}+1	S	R
0	0	0	0	x
0	1	0	0	1
1	0	1	1	0
1	1	1	x	0

Based on the provided table, we can express the Boolean functions for each input as follows :

$S = m_2 + d_3$

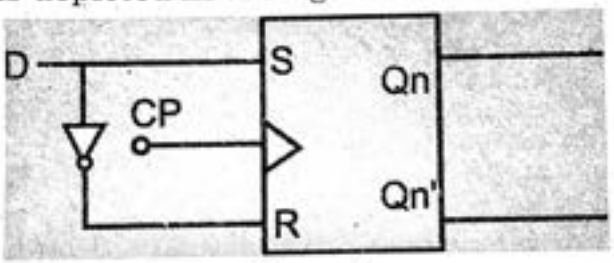
$R = m_1 + d_0$

We can utilize two-variable K-Maps to obtain simplified expressions for these inputs. The K-Maps for S and R are illustrated below.



(Figure)

After simplification, we obtained $S = D$ and $R = D'$ as the simplified expressions. The circuit diagram of the D flip-flop is depicted in the figure below.



(Figure)

6. ◆ ***Differentiate synchronous and asynchronous circuits.*** **(2019)**

◆ ***Design a MUX for the function of time variable :***

$F(A, B, C) = \sum (1, 3, 5, 6)$

Draw the implementation table. **(2018)**

◆ ***Draw the block diagram of sequential circuit and discuss it. Implement the function F (A, B, C) = $\sum$ (1, 3, 5, 6) with multiplexer.*** **(2015)**

Sequential Circuit

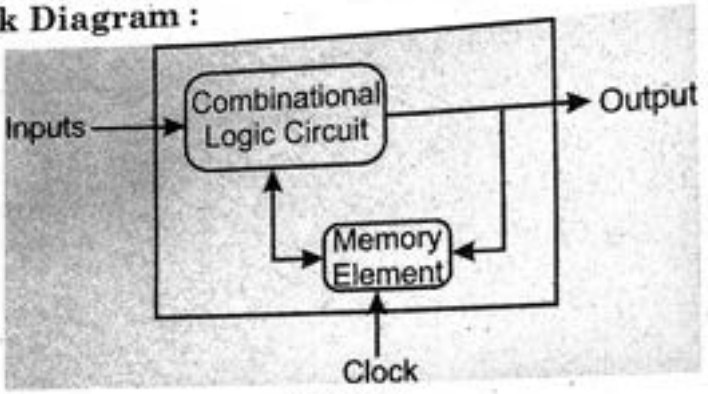
The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input.

This type of circuits uses previous input, output, clock and a memory element.

There are two types of input to the combinational logic; External inputs which come from outside the circuit design and are not controlled by the circuit; Internal inputs which functions of previous output states are.

The internal inputs and outputs are referred to as "secondaries". Secondary inputs are state variables produced by the storage elements, where as secondary outputs are excitations for the storage elements.

Block Diagram :



(Figure)

Types of Sequential Circuits

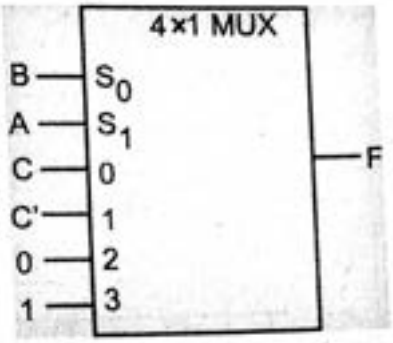
There are two types of sequential circuits, synchronous and asynchronous.

(1) **Synchronous Sequential Circuits :** Synchronous types use pulsed or level inputs and a clock input to drive the circuit (with restrictions on pulse width and circuit propagation).

(2) **Asynchronous Sequential Circuits :** Asynchronous sequential circuits do not use a clock signal as synchronous circuits do. Instead the circuit is driven by the pulses of the inputs.

Implementation of Function with Multiplexer

A	B	C	F	
0	0	0	0	F = C
0	0	1	1	
0	1	0	1	F = C'
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	



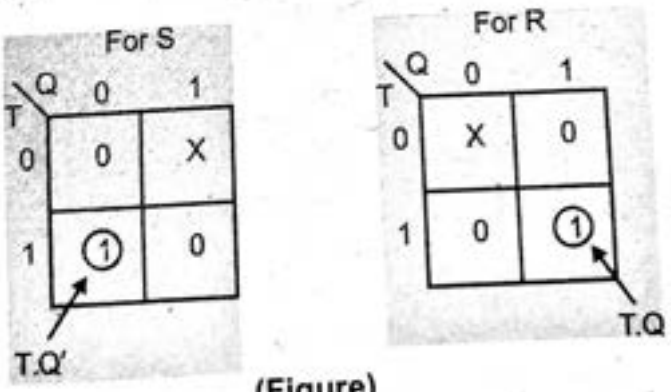
(Figure)

7. *Convert RS Flip – Flop to T Flip – Flop using excitation table.* **(2016, 2017)**

For conversion of SR flip flop to T flip at first we have to make combine truth table for SR flip flop and T flip flop. In below see the combine truth table of SR flip flop and T flip flop.

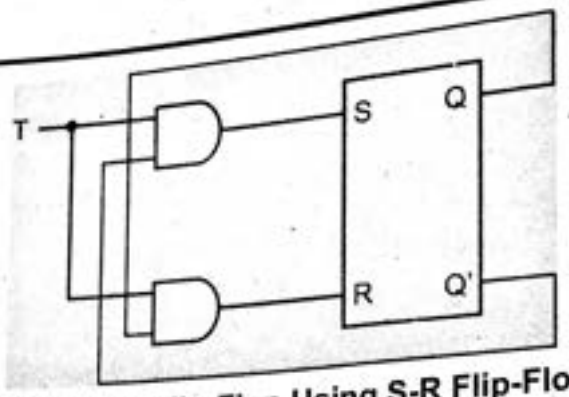
FF Data Inputs	Output	SR FF Inputs	
T	Q	S	R
0	0	0	X
1	0	1	0
1	1	0	1
0	1	X	0

From above truth table we can understand that what are those different inputs of T flip flop and SR flip flop, we need to get the output Q. Now from above truth table we can draw the Karnaugh map for input S and R. Then we can easily get the relation between SR with T. For that see below



(Figure)

Now from this above Karnaugh map we get the relation $S = TQ'$ and $R = TQ$. Now as per the relation of SR with T flip flop from karnaugh map we can easily build T flip flop using SR flip flop.



(Figure : A T Flip-Flop Using S-R Flip-Flop)

8. *Find the minimum number of D flip – flops needed to design a mode 258 counter.* **(2015)**

An *n-bit* binary counter consists of *n*-flip flops can count in binary from 0 to $2^n - 1$. So to count from 0 to *n* we need at least

$\log (258 - 1) = \log 257 = 9$

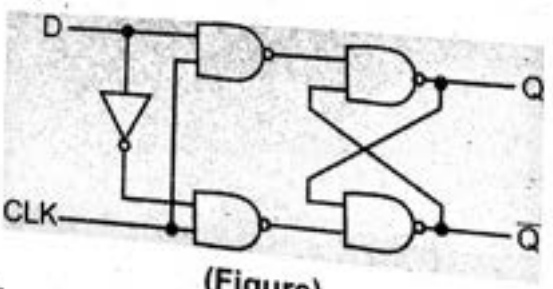
9. *Write short note on Race condition in RS flip – flop.* **(2015)**

Race Condition in RS Flip-Flop

The race condition is that, from a 00 input state, one input changes to 0, and the second one also changes to 0 before the effect of the first change has settled. Now the effects of the two changes are 'racing' for priority.

The explanation stated is for a simple Set-Reset FF (or latch).

A level-triggered circuit (Latch) can be thought of as a RS-FF with both inputs gated by the enable input (CLK in this diagram) :



(Figure)

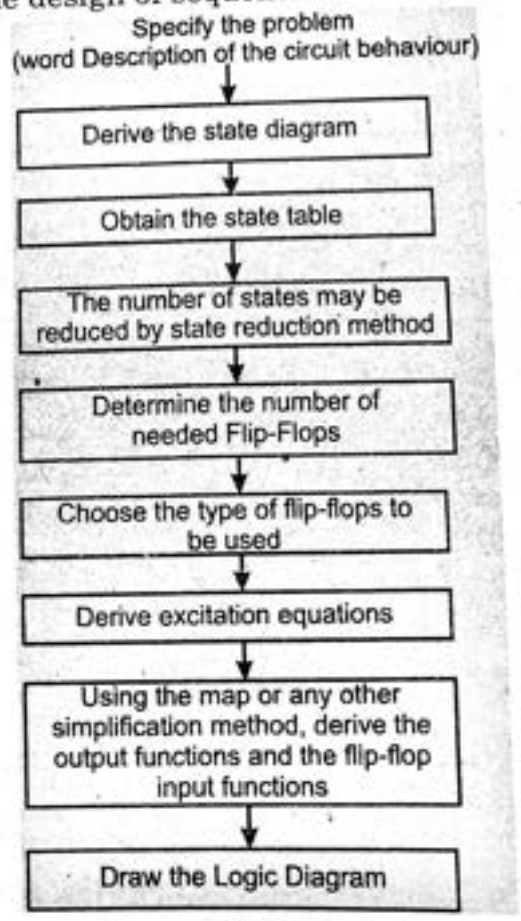
In this circuit, a simultaneous 00 → 11 transition of the hidden 'inputs' of the cross-coupled NANDs still causes a race condition. Such a transition can occur (due to the delay caused by the inverter) when the D input changes simultaneously with the CLK input changing from 1 to 0.

10. *Write short note on Design procedure of sequential circuits.* **(2015)**

The design of a synchronous sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of boolean functions from which a logic diagram can be obtained. In contrast to a combinational logic, which is fully specified by a truth table, a sequential circuit requires a state table for its specification. The first step in the design of sequential circuits is to obtain a state table or an equivalence representation, such as a state diagram.

A synchronous sequential circuit is made up of flip-flops and combinational gates. The design of the circuit consists of choosing the flip-flops and then finding the combinational structure which, together with the flip-flops, produces a circuit that fulfils the required specifications.

The number of flip-flops is determined from the number of states needed in the circuit. The recommended steps for the design of sequential circuits are set out below.

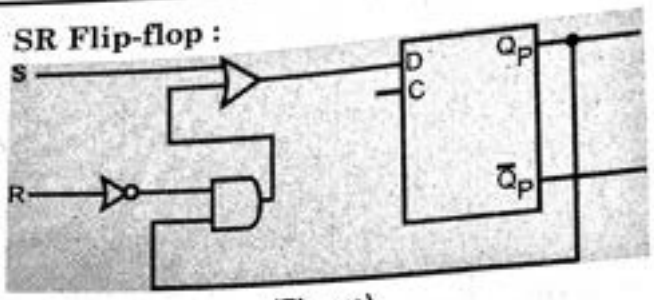


(Figure)

[C.14] (2014)

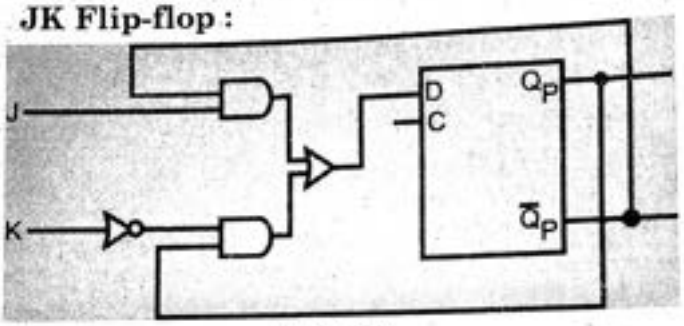
11. ***Construct a D flip-flop using :***
(1) *SR flip-flop*
(2) *JK flip-flop*

(1) SR Flip-flop :



(Figure)

(2) JK Flip-flop :



(Figure)

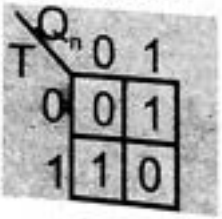
12. ***Construct a T flip-flop using :*** **(May 2012)**
(1) *D-flip-flop*
(2) *JK flip-flop*
Use block diagram.

(1) D-Flip-Flop to T Flip-Flop : The excitation table for above conversion is as shown in the Table.

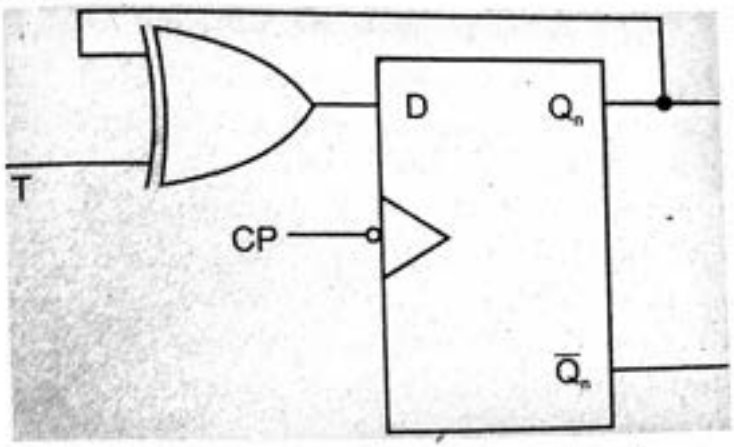
Input	Present State	Next State	Flip-flop Input
T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Table

K-map Simplification : For D



Logic Diagram :



(Figure : D to T Flip-Flop Conversion)

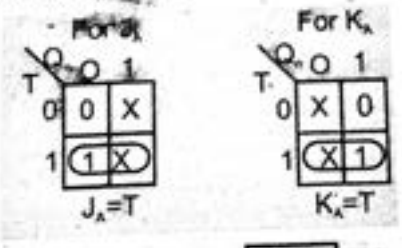
$$D = \overline{T}Q_n + T\overline{Q}_n = T \oplus Q_n$$

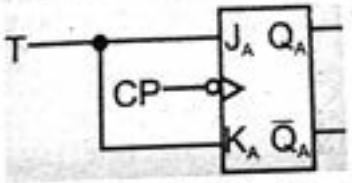
(2) JK Flip-Flop to T Flip-Flop : The excitation table for above conversion is as shown in table

Input	Present State	Next State	Flip-Flop Inputs	
T	Q_n	Q_{n+1}	J_A	K_A
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

Table

K-Map Simplification :





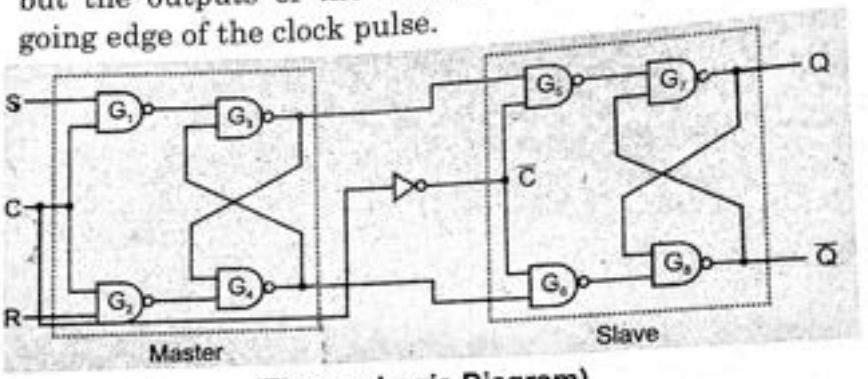
(Figure : JK to T Flip-Flop Conversion)

13. *Write short note on Master slave flip-flop.* **(May 2012)**

The Master-Slave (Pulse-Triggered) S-R Flip-Flop

Figure shows the logic diagram and the truth table of a master-slave, S-R flip-flop. The truth table operation is

the same as that for the edge-triggered S-R flip-flop except for the way it is clocked-internally though the master-slave type is quite different. The external control inputs S and R are applied to the master section. The master section is basically a gated S-R latch, and responds to the external S-R inputs applied to it at the positive-going edge of the clock signal. The slave section is the same as the master section except that it is clocked on the inverted clock pulse and thus responds to its control inputs (which are nothing but the outputs of the master flip-flop) at the negative-going edge of the clock pulse.



(Figure : Logic Diagram)

Thus, the master section assumes the state determined by the S and R inputs at the positive-going edge of the clock pulse and the slave section copies the state of the master section at the negative-going edge of the clock pulse. The state of the slave then immediately appears on its Q and $\overline{Q}$ outputs.

Inputs			Output	Comment
S	R	CLK	Q	
0	0	⎍	Q_0	No Change
0	1	⎍	0	RESET
1	0	⎍	1	SET
1	1	⎍	?	Invalid

Truth Table

(Figure : The Master-Slave S-R Flip-Flop)

14. *What is triggering? What are its types?*

Triggering is the process of making a circuit active. Triggering allows the circuit to take input and generate output. The circuit will become active at certain states of clock pulse depending upon the triggering methods we used.

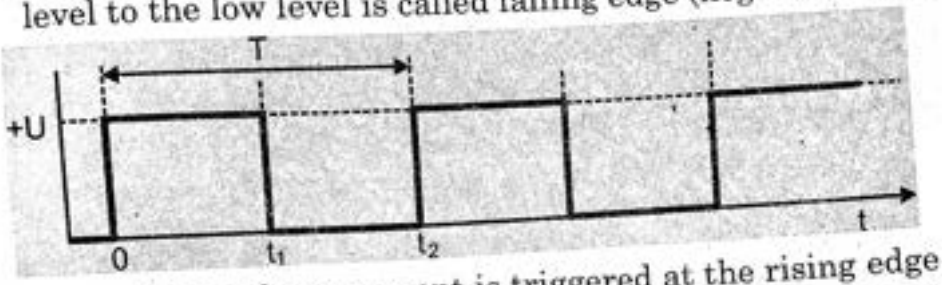
Types of Triggering

There are two types of triggering

(1) Edge triggering

(2) Level triggering

Edge Triggering : In a sequential circuit, if the output changes when the signal transits from a high level to a low level or from a low level to a high level, we call it edge triggering. Here, the edge that changes the voltage from low level to the high level is called rising edge (positive edge). And, the edge that changes the voltage from high level to the low level is called falling edge (negative edge).



Thus, when an event is triggered at the rising edge or falling edge, we call it edge triggering. For example, assume lighting an LED according to the edge triggering. In this scenario, the LED lights on every time the signal transits from low voltage to high voltage. Taking some examples; S-R flip flop, J-K flip flop and D flip flop are some common examples for flip flops with edge triggering.

Level Triggering : In the sequential circuit, if the output changes during the high voltage period or low voltage period, it is called level triggering. In other words, the output changes during either high voltage or low voltage period- not during the edges like in edge triggering.

Thus, when an event triggers at the clock level, we call it level triggering. Assume lighting an LED according to level triggering. LED can turn on at any time during the high voltage. In other words, the event is triggered whenever a clock level is encountered. Considering examples; SR latch and D latch are some examples for latches with level triggering.

15. *What is the Difference between Edge and Level Triggering?*

The main difference between edge and level triggering is that in edge triggering, the output of the

sequential circuit changes during the high voltage period or low voltage period while, in level triggering, the output of the sequential circuit changes during transits from the high voltage to low voltage or low voltage to high voltage.

In a sequential circuit, the output changes depending on the triggering. There are two types of triggering as edge and level triggering. There are two levels in a clock pulse or a signal. One is a high voltage (VH), and the other is low voltage (VL). Furthermore, these voltage levels help to determine the triggering type.

Difference between Edge and Level Triggering

Definition : Edge triggering is a type of triggering that allows a circuit to become active at the positive edge or the negative edge of the clock signal. In contrast, level triggering is a type of triggering that allows a circuit to become active when the clock pulse is on a particular level.

Functionality : In edge triggering, an event occurs at the rising edge or falling edge whereas, in level triggering, an event occurs during the high voltage level or low voltage level. Thus, this is the main difference between edge and level triggering.

Applications : Furthermore, another difference between edge and level triggering is that the flip flops work according to edge triggering, whereas Latches work according to level triggering.

Conclusion : In brief, there are two types of triggering in sequential circuits. The triggering results can change the output of the circuit. The main difference between edge and level triggering is that, in edge triggering, the output of the sequential circuit changes during the high voltage period or low voltage period while in level triggering, the output of the sequential circuit changes during transits from the high voltage to low voltage or low voltage to high voltage.

❑❑

REGISTERS

1. ◆ ***What do you understand by shift register?*** **(2023-24)**
   ◆ ***Explain types of shift register.*** **(2023-24)**
   ◆ ***Define Shift Register. Explain different types of Shift Register.*** **(2022-23)**
   ◆ ***What do you mean by shift Register? Show the design of a 4 – bit right shift register and its operations.*** **(2019)**
   ◆ ***Write short note on Shift register.*** **(2014, 2015, 2016)**
   ◆ ***Explain the shift registers and its operations.***

A register is a device which is capable of storing a *bit*. The data can be serial or parallel. A register can convert a data from serial to parallel and parallel to serial. Shifting the data from left or right is an important aspect of arithmetic operation.

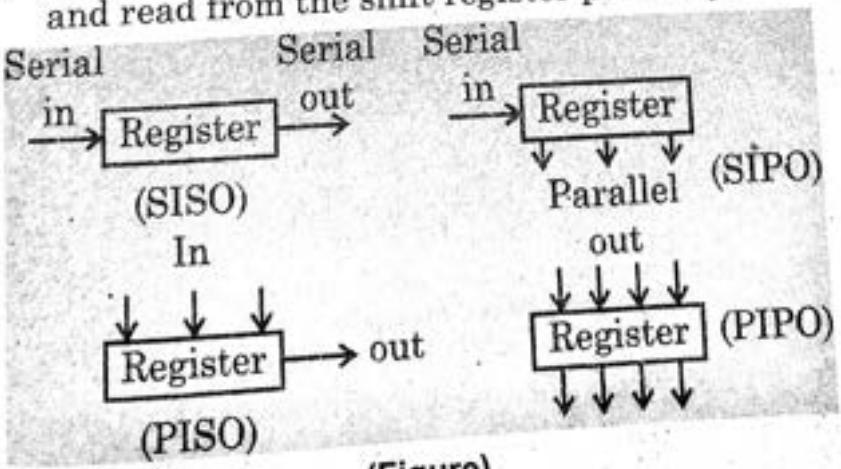
Mainly four types of operations are there :

(1) Serial In – Serial Out : The data is loaded into and read from the shift register serially.

(2) Serial In – Parallel Out : The data is loaded serially but read in parallel.

(3) Parallel In – Serial Out : The data is loaded in parallel i.e., the bits 0 are entered simultaneously in their respective stages and read serially.

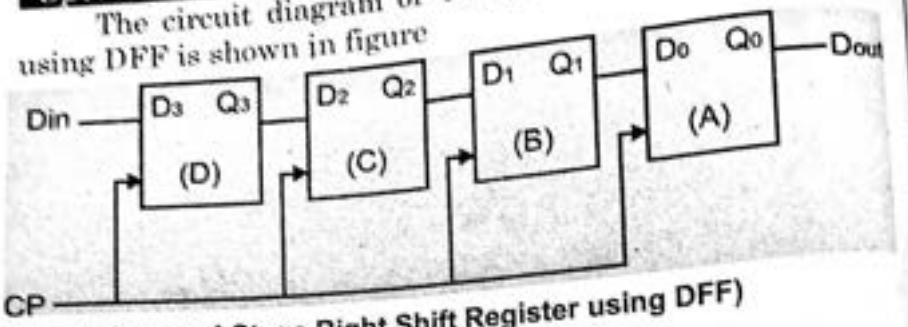
(4) Parallel In – Parallel Out : The data is loaded into and read from the shift register parallely.



(Figure)

Design of a 4 – Bit Right Shift Register and Its Operations

The circuit diagram of 4 stage shift – right register using DFF is shown in figure



(Figure : 4 Stage Right Shift Register using DFF)

Operation : Initially if Din = 1 and

	D	C	B	A
Q =	0	0	0	0

The striking of the first rising clock edge will set the FFD and the stored word becomes as below :

	D	C	B	A
Q =	1	1	0	0

Now when this word appears, D_3 and D_2 are both 1 and thus the second +Ve clock edge gives the stored word as below :

	D	C	B	A
Q =	1	0	0	0

The third +ve clock edge gives the output :

	D	C	B	A
Q =	1	1	1	0

The fourth rising clock edge makes the contents of register as

	D	C	B	A
Q =	1	1	1	1

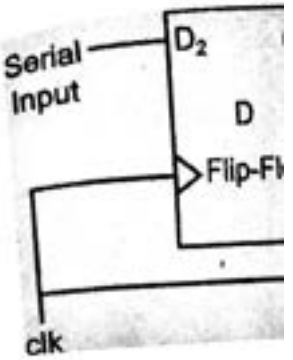
Thus word is stored and remains unchanged so long as Din = 1. It is a shift right register, because it is shifting 1 towards right side.

Different Types of Shift Register

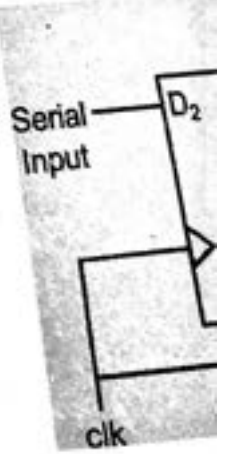
There are several types of shift registers, each with its own characteristics and applications. The main types of shift registers are :

(1) Serial-In, Serial-Out (SISO) Shift Register : This type of shift register has a single data input (serial-in) and a single data output (serial-out). It

shifts da
The inp
flip-flop
stages.
the regi

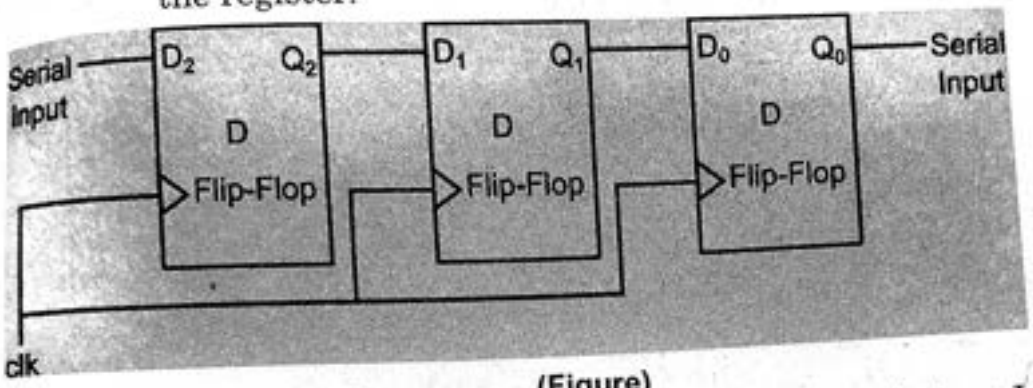


(2) Serial
this ty
but th
serial
data
when
acces



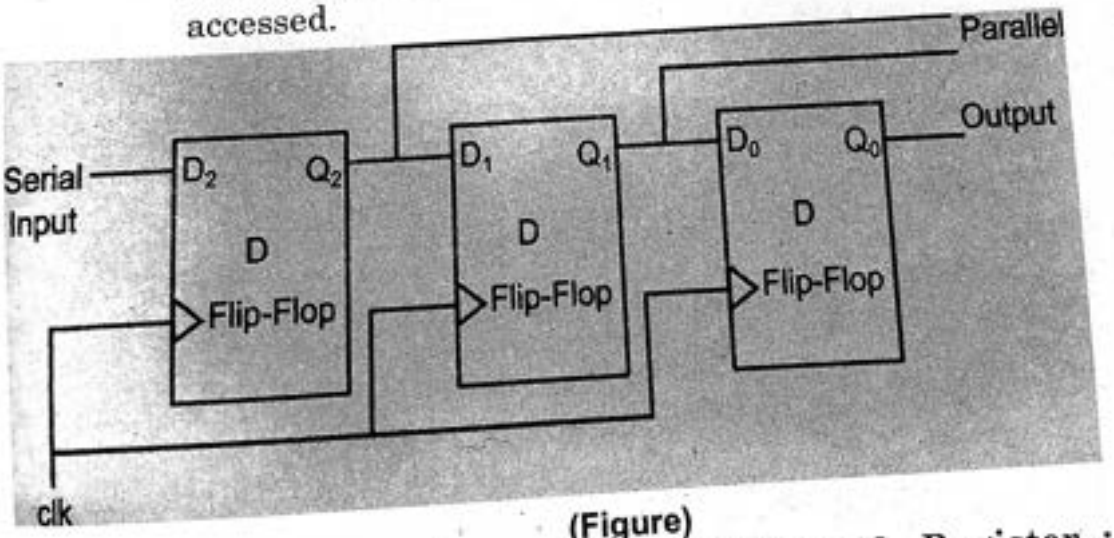
(3)

shifts data through the register one bit at a time. The input data is serially loaded into the first flip-flop and then shifted through the remaining stages. The output is taken from the last flip-flop in the register.



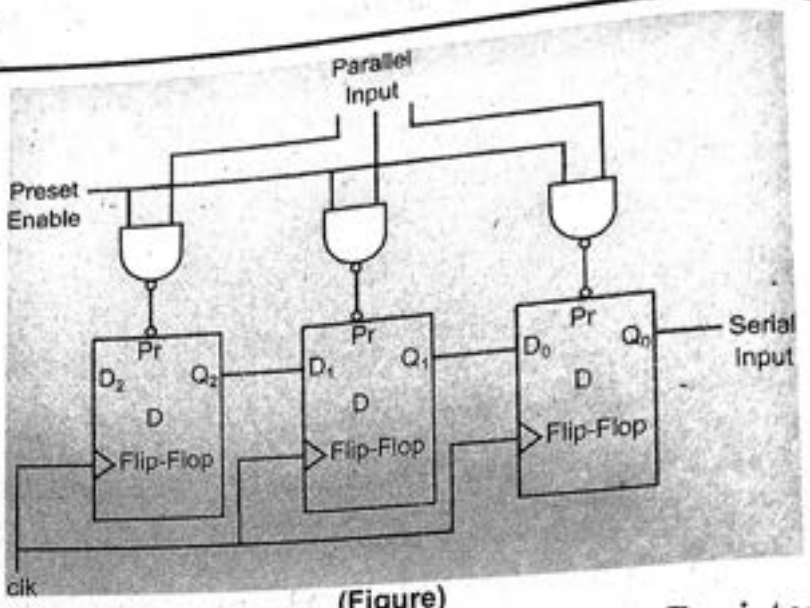
(Figure)

(2) Serial-In, Parallel-Out (SIPO) Shift Register : In this type, data is inputted serially (one bit at a time), but the output is obtained in parallel form. It has a serial input and multiple parallel outputs. The input data is loaded sequentially into the flip-flops and, when required, all the outputs can be simultaneously accessed.



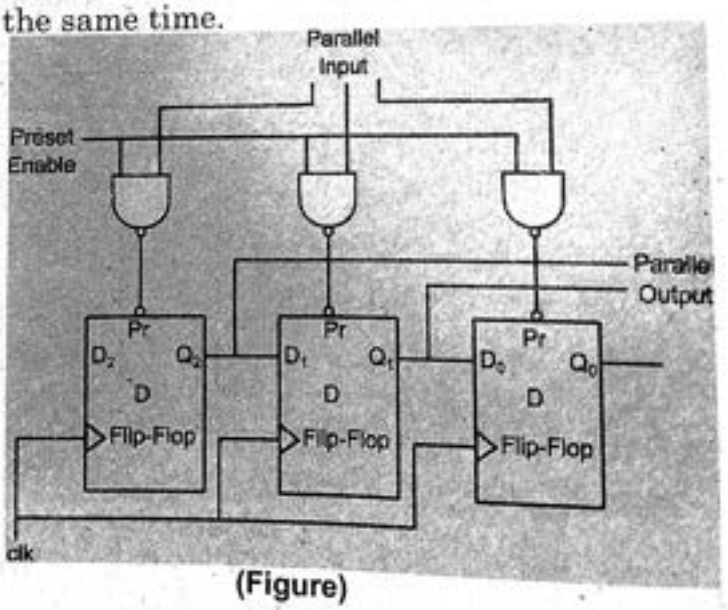
(Figure)

(3) Parallel-In, Serial-Out (PISO) Shift Register : This type of shift register accepts data in parallel form (multiple bits at a time) and outputs it serially (one bit at a time). It has multiple parallel inputs and a single serial output. The parallel input is loaded into the flip-flops simultaneously, and the data is shifted out serially bit by bit.



(Figure)

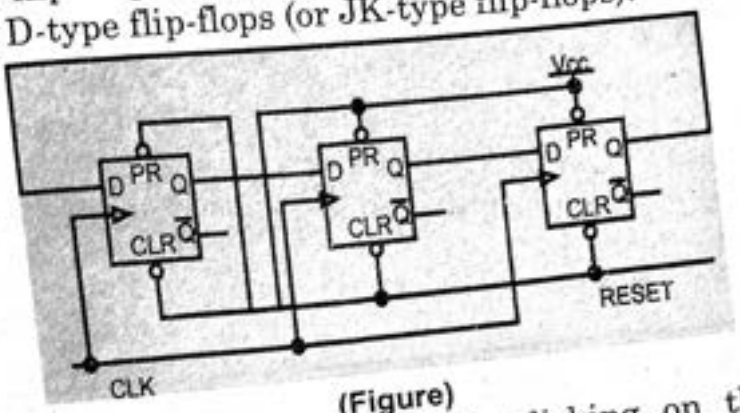
(4) Parallel-In, Parallel-Out (PIPO) Shift Register : This shift register allows parallel loading of data into the flip-flops and parallel outputting of the data. It has multiple parallel inputs and multiple parallel outputs. The data is loaded into the flip-flops simultaneously, and all the outputs can be accessed at the same time.



(Figure)

2. ◆ ***What do you understand by ring counter?***
   ◆ ***What is ring counter? Explain its working.*** **(2023-24)**
   ◆ ***What is a ring counter? Draw its circuit and explain its working.*** **(2014)**

A ring counter is a Shift Register (a cascade connection of flip-flops) with the output of the last flip flop connected to the input of the first. It is initialized such that only one of the flip flop output is 1 while the remainder is 0. The 1 bit is circulated so the state repeats every n clock cycles if n flip-flops are used. The "MOD" or "MODULUS" of a counter is the number of unique states. The MOD of the *n* flip flop ring counter is *n*. It can be implemented using D-type flip-flops (or JK-type flip-flops).



(Figure)

(1) Enable the flips flops by clicking on the RESET (Green) switch. The RESET switch is a on/off switch (similar to a room light switch).

(2) Click on CLK (Red) switch and observe the changes in the outputs of the flip flops. The CLK switch is a momentary switch (similar to a door bell switch - normally off).

(3) The D flip flop clock has a rising edge CLK input. For example Q_1 behaves as follows :

(a) The D input value just before the CLK rising edge is noted (Q_0).

(b) When CLK rising edge occurs, Q_1 is assigned the previously noted D value (Q_0).

3. ◆ *Differentiate between synchronous asynchronous counter.* (2023-24)

◆ *What is the difference between synchronous and asynchronous counter?* (2022-23)

(1) **Synchronous Counter :** A synchronous counter is also known as a parallel counter. In a synchronous counter, all flip-flops are clocked simultaneously by the same clock signal. The outputs of the flip-flops are connected to the next flip-flop's clock input, forming a

chain. The clock signal ensures that all flip-flops change their state at the same time.

(2) **Asynchronous Counter :** An asynchronous counter is also known as a ripple counter or a serial counter. In an asynchronous counter, each flip-flop is clocked by the output of the previous flip-flop in the chain. The clock signal ripples through the flip-flops sequentially, causing a delay between the changes in state of each flip-flop.

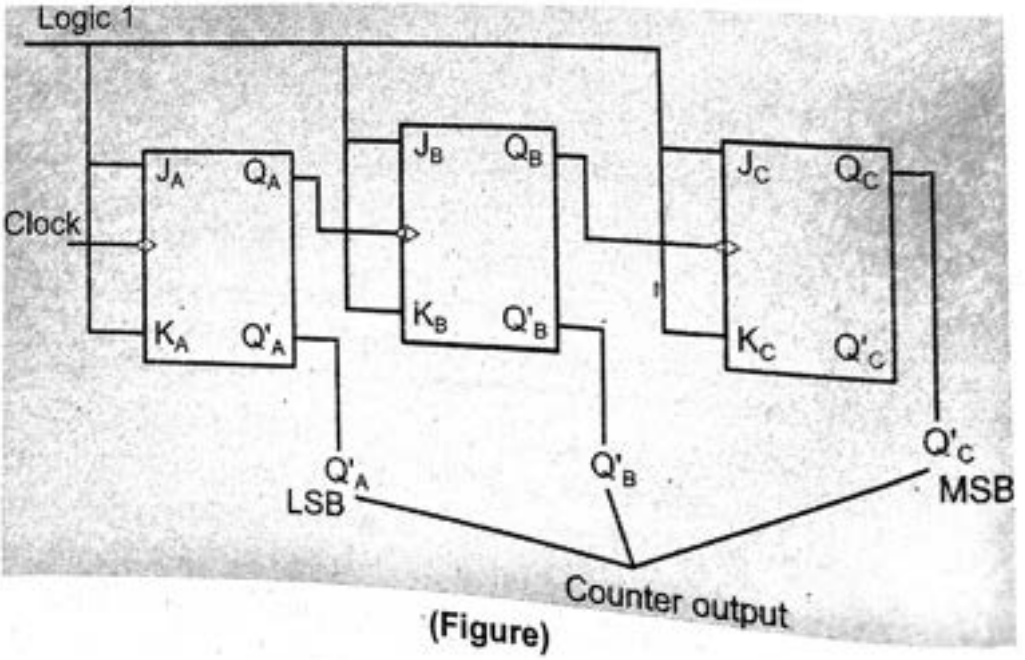
The key difference between synchronous and asynchronous counters lies in the clocking mechanism and the timing relationship between the input clock signal and the output states. Synchronous counters are clocked simultaneously, ensuring precise timing, while asynchronous counters have a ripple effect, causing delays and potentially less precise timing.

4. *Design 3 – bit (MOD - 8) asynchronous counter with state and Logic diagram.* **(2023-24)**

Design 3-bit (MOD-8) asynchronous counter with state and logic diagram

In the asynchronous counter, an external clock pulse is provided for only the first flip flop, thereafter the output of the 1st FF acts as a clock pulse for the second FF and so on. In the case of synchronous FFs, all the flip flops are triggered simultaneously by an external clock pulse.

3 Bit Asynchronous Down Counter :



(Figure)

For the 3 bit counter, we require 3 flip flops and we can generate 23 = 8 state and count(111 110 ... 000).

We can generate down counting states in an asynchronous down counter by two ways.

Method 1 :

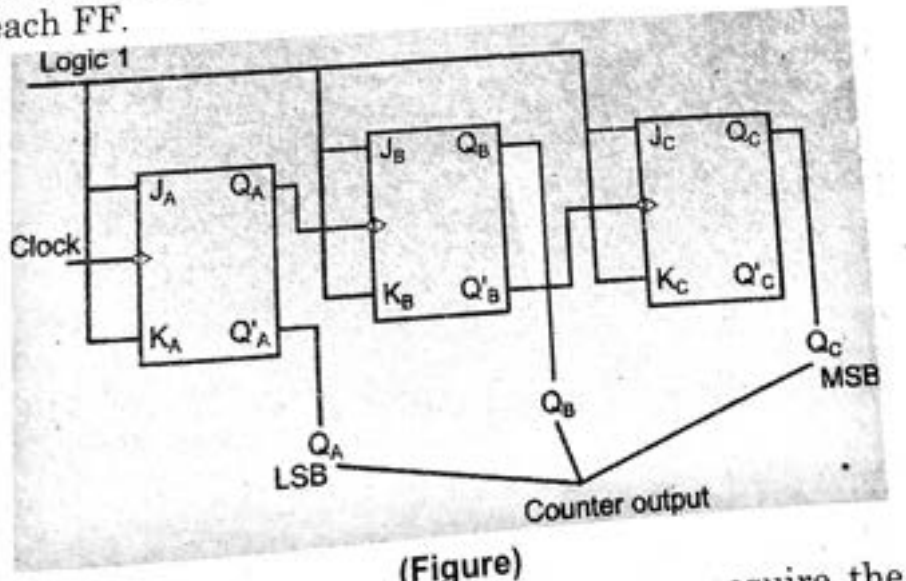
In this implementation, the clock pulse(of 50% duty cycle) is given to only the first FF. Thereafter, the output of the first FF is feed as a clock to second FF and the output of the second FF is feed as the clock for the third FF. But the complemented output is taken from each FF(i.e. same as Up counter but output states are complemented). Here QA is LSB and QC is MSB.

State Table :

Clock	Q_C	Q_B	Q_A	Q'_C	Q'_B	Q'_A
Initially	0	0	0	1	1	1
1st	0	0	1	1	1	0
2nd	0	1	0	1	0	1
3rd	0	1	1	1	0	0
4th	1	0	0	0	1	1
5th	1	0	1	0	1	0
6th	1	1	0	0	0	1
7th	1	1	1	0	0	0

Method 2 :

In this implementation, the clock pulse is given to only the first FF. Thereafter, the complemented output of the first *FF*(*Q'A*) is feed as a clock to the second FF and complemented output(i.e. *Q'B*) of the second FF is feed as a clock for the third FF. But the output(*Q*) is taken from each FF.



(Figure)

By both implementations, we can acquire the same counting states.

Timing Diagram : The working of counter can be easily understood by the timing diagrams.

Explanation : Here the -ve edge clock pulse is used (i.e. the counter state transition can occur only at falling edge of the clock pulse) , therefore toggling will take place.

All the counter states are generated by the frequency division.

Initially $QA = 0$ $QB = 0$ and $QC = 0$.

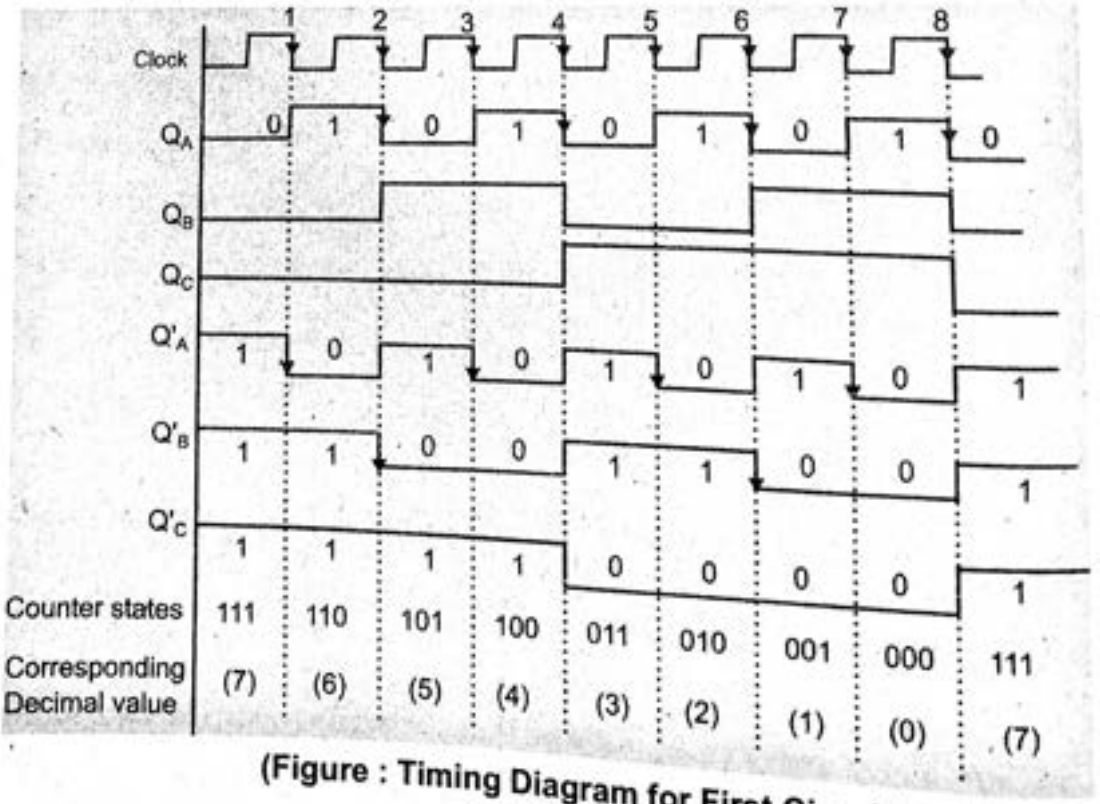
First Circuit : The -ve edge clock pulse is provided to 1st counter. Therefore, the output state of the first counter (i.e. QA) will be toggled at every falling edge of the clock pulse.

As the QA is feed as a clock to the second FF, therefore the output state (i.e. QB) will be toggled at every falling edge of QA.

In the same manner, the QB acts as clock for the third FF, therefore the output state (QC) of the third FF will be toggled for every falling edge of QB.

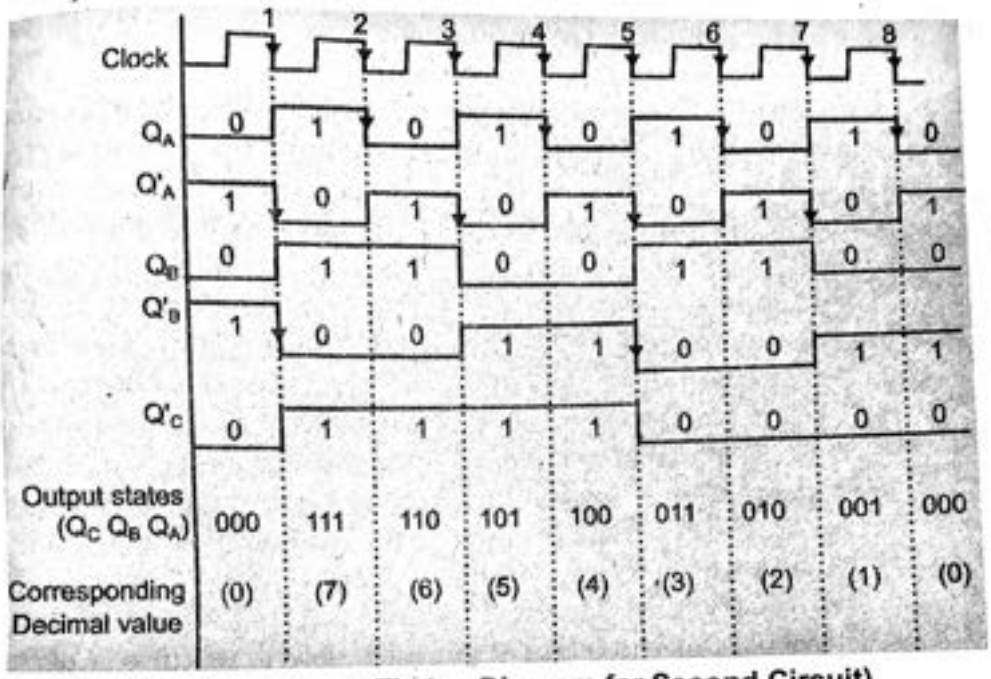
As we know, this is the working of UP counter, but here the output is taken as in complemented form (i.e. $Q'C$ $Q'B$ $Q'A$), therefore we get the complemented outputs (i.e. down counting 111 to 000)

After the 8th falling edge of the external clock pulse, the counter is reset to 000.



(Figure : Timing Diagram for First Circuit)

Second Circuit : The -ve edge clock pulse is provided to 1st counter. Therefore, the output state of the first counter (i.e. *QA*) will be toggled at every falling edge of the clock pulse.



(Figure : Timing Diagram for Second Circuit)

As the complemented output(*Q'A*) is feed as a clock to second FF, therefore the output state (i.e. *QB*) will be toggled at every falling edge of *Q'A*.

In the same manner, the *Q'B* acts as a clock for the third *FF*, therefore the output state (*QC*) of the third *FF* will be toggled for every falling edge of *Q'B*.

In this case, the outputs from the three FF are taken as (*QC QB QA*).

After the 8th falling edge of the external clock pulse, the counter is reset to 000.

Here, the down counting states can be acquired after first -ve edge clock pulse(i.e. after first -ve pulse the counting output becomes 111).

5. *Explain the various methods for converting asynchronous UP counter into asynchronous DOWN counter with example.* **(2022-23)**

An asynchronous up counter is a digital circuit that counts upwards by incrementing its output value with each clock pulse. On the other hand, an asynchronous down

counter is a circuit that counts downwards by decrementing its output value with each clock pulse. Converting an asynchronous up counter into an asynchronous down counter can be achieved using various methods, including the following :

(1) **Reverse the Counting Sequence** : One straightforward method is to reverse the counting sequence of the counter. In an up counter, the sequence starts from 0 and goes up to the maximum count value. To convert it into a down counter, reverse the counting sequence, starting from the maximum count value and decrementing down to 0.

(2) **Complement the Output** : Another approach is to complement the output of the counter. In an up counter, the output bits represent the count value directly. To convert it into a down counter, complement all the output bits using logic gates such as XOR or NOT gates. This will invert the count value, effectively making it count down.

(3) **Modify the Input Signals** : Adjusting the input signals of the counter can also convert it into a down counter. Typically, an up counter increments its count value when the clock signal transitions from low to high. In a down counter, the count value should decrement on the rising edge of the clock signal. By modifying the clock signal or using additional control signals, you can change the counter's behavior accordingly.

(4) **Use a Downward Count Enable Signal** : Some counters have an additional input signal called "Downward Count Enable" (DCE) or "Count Down" (CD). This signal allows you to control the counting direction of the counter explicitly. By activating the DCE or CD signal, the counter will decrement instead of incrementing with each clock pulse, effectively transforming it into a down counter.

6. *What are Registers? Explain the general purpose registers used in the CPU.* **(2015)**

The internal processor memory of a CPU is served by its registers. One of the key differences among various computers is the difference in their register sets. Some

computers have very large while some has smaller sets. But one the whole, from a user's point of view the register set can be classified under two basic categories :

(1) **Programmer Visible Register :** These registers can be used by machine or assembly language programmers to minimize the references to main memory.

(2) **Status Control and Register :** These registers can be used by the programmers but are used to control the CPU or the execution of a program.

Programmer Visible Registers

These registers can be accessed using machine language. In general, we encounter four types of programmer visible registers :

(1) General Purpose Registers
(2) Data Registers
(3) Address Registers
(4) Condition Codes Register

(1) **General Purpose Registers :** The general purpose registers are used for various functions desired by the processor. A true general purpose register can contain operand or can be used for calculations of address of operand for any operator code of an instruction. But trends in today's machines show drift towards dedicated registers, for examples, some registers may be dedicated to floating point operations. In some machines, there is a distinct separation between data and address registers.

(2) **Data Registers :** The data registers are used only for storing intermediate results or data. These data registers are not used for calculation of address of the operand. An address register may be a general purpose register, but some dedicated address register are also used in several machines. Examples of dedicated address registers can be :

Segment Pointer : Used to point out a segment of memory.

Index Register : These are used for index addressing scheme.

Stack Pointer : (When programmer visible stack addressing is used).

(3) **Address Registers :** The issue related to the register set design is the number of general purpose

registers or data and address registers to be provided in a micro-processor. The number of registers also affects the instruction design as the number of registers determines the number of bits needed in an instruction to specify a register reference. In general, it has been found that optimum number of registers in a CPU is in the range 8 to 32. In case registers fall below the range then more memory reference per instruction on an average will be needed, as some of the intermediate result then has to be stored in the memory. On the other hand, if the number of registers goes above 32, then there is no appreciable reduction in memory references. However, in some computers hundreds of registers are used. These systems have special characteristics. Reduced Instruction Set Computers (RISC) exhibits this property. What is the importance of having less memory references? As the time required for memory references results in slower execution of a program.

Register Length : Another important characteristic related to registers is the length of a register. Normally, the length of a register is dependent on its use. For example, a register which is used to calculate address must be long enough to hold the maximum possible address. Similarly, the length of data register should be long enough to hold the data type it is supposed to hold. In certain cases, two consecutive registers may be used to hold data whose length is double of the register length.

(4) **Condition Code Registers :** Condition code registers may only be partially available to the programmers. These registers contain conditions codes which are also known as flags. These flags are set by the CPU hardware while performing an operation. For example, an addition operation may set the overflow flag or on a division by 0 the overflow flag can be set etc. These condition codes are collected in one or more registers. RISC machines have several set of conditional code bits. In these machines an instruction specifies the set of condition codes which is to be used. Independent sets of conditions code enable the provisions of having parallelism within the instruction execution unit.

Status and Control Register

For control of various operations several registers are used. These registers cannot be used in data manipulations; however, the content of some of these registers can be used by the programmer. Some of the control register for Von Neumann machine can be the Program Counter (PC), Memory Address Register (MAR) and Data Register (DR).

Almost all the CPUs have status register, a part of which may be programmer visible. A register which may be formed by conditions codes is called conditions code register. Some of the commonly used flags or condition codes in such a register may be :

Sign Flag : This indicates whether the sign of previous arithmetic operations was positive (0) or negative (1).

Zero Flag : This flag bit will be set if the result of last arithmetic operations was zero.

Carry Flag : This is set, if a carry results from the addition of the highest order bits or a borrow is taken on subtraction of highest order bit.

Equal Flag : This bit flag will be set if a logic comparison operation finds out that both of its operands are equal.

Interrupt : This flag is used for enabling or disabling interrupts.

Flag Supervisor

This flag is used in certain computers to determine whether the CPU is flag executing in supervisor or user mode. In case, the CPU is in supervisor mode it will be allowed to execute certain privileged instructions.

In most CPUs, on encountering a subroutine call or interrupt handling routine, it is desired that the status information such as conditional codes and other register information be stored and restored on initiation and end of these routines respectively. The register often known as Program Status Word (PSW) contains conditions code plus other status information. There can be several other status and control registers such as interrupt vector registers in the machines using vectored interrupt, stack pointer if a stack is used to implement subroutine calls, etc.

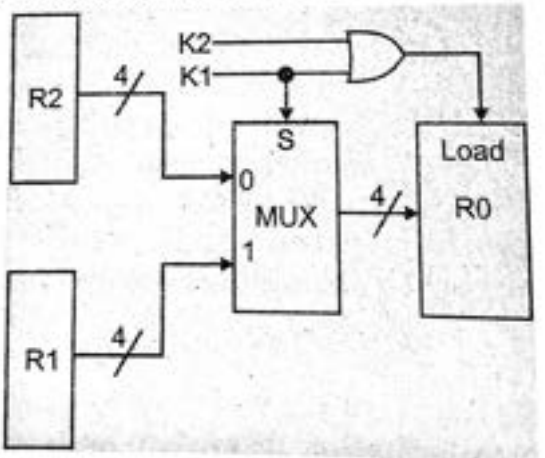
The status and control register design is also dependent on the Operating Systems (OS) support. The functional understanding of OS helps in tailoring the register organization. In fact, some control information is only specific use to the operating system.

One major decision to be taken for designing status and control registers organization is : How to allocate control information between registers and the memory. Generally, first few hundred or thousands words of memory are allocated for storing control information. It is the responsibility of the designer to determine how much control information should be in registers and how much in memory. This in fact is a tradeoff between the cost and the speed.

7. ◆ ***What is register? What kind of flip – flop (FF) is used in register?*** **(2019)**
◆ ***Differentiate between Register and Latch.*** **(2017)**
◆ ***What is a Register? Draw a block diagram of 4- bit register.*** **(May 2012)**

Difference between Register and Latch

Register : A register includes a set of flip-flops. Since each flip-flop is capable of storing one bit of information, an *n-bit* register, including *n* flip-flops, is capable of storing *n bits* of binary information. By the broadest definition, a register consists of a set of flip-flops, together with gates that implement their state transitions.



(Figure : Block Diagram)

Latch : A flip-flop or latch is a circuit that has two stable states and can be used to store state information. A flip-flop is a bistable multivibrator. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. It is the basic storage element in sequential logic. Flip-flops and latches are fundamental building blocks of digital

electronics systems used in computers, communications, and many other types of systems.

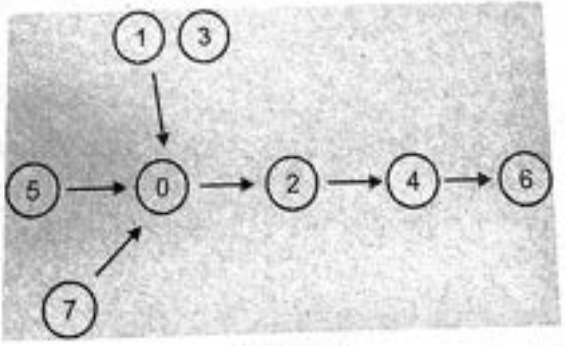
Flip-flops and latches are used as data storage elements. A flip-flop stores a single bit (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of state, and such a circuit is described as sequential logic. When used in a finite-state machine, the output and next state depend not only on its current input, but also on its current state (and hence, previous inputs). It can also be used for counting of pulses, and for synchronizing variably-timed input signals to some reference timing signal.

Kind of Flip – Flop Used in Register : There are following flip – flop used in register :

(1) SRFF
(2) DFF
(3) J.K FF

In general we use D flip - flop in register

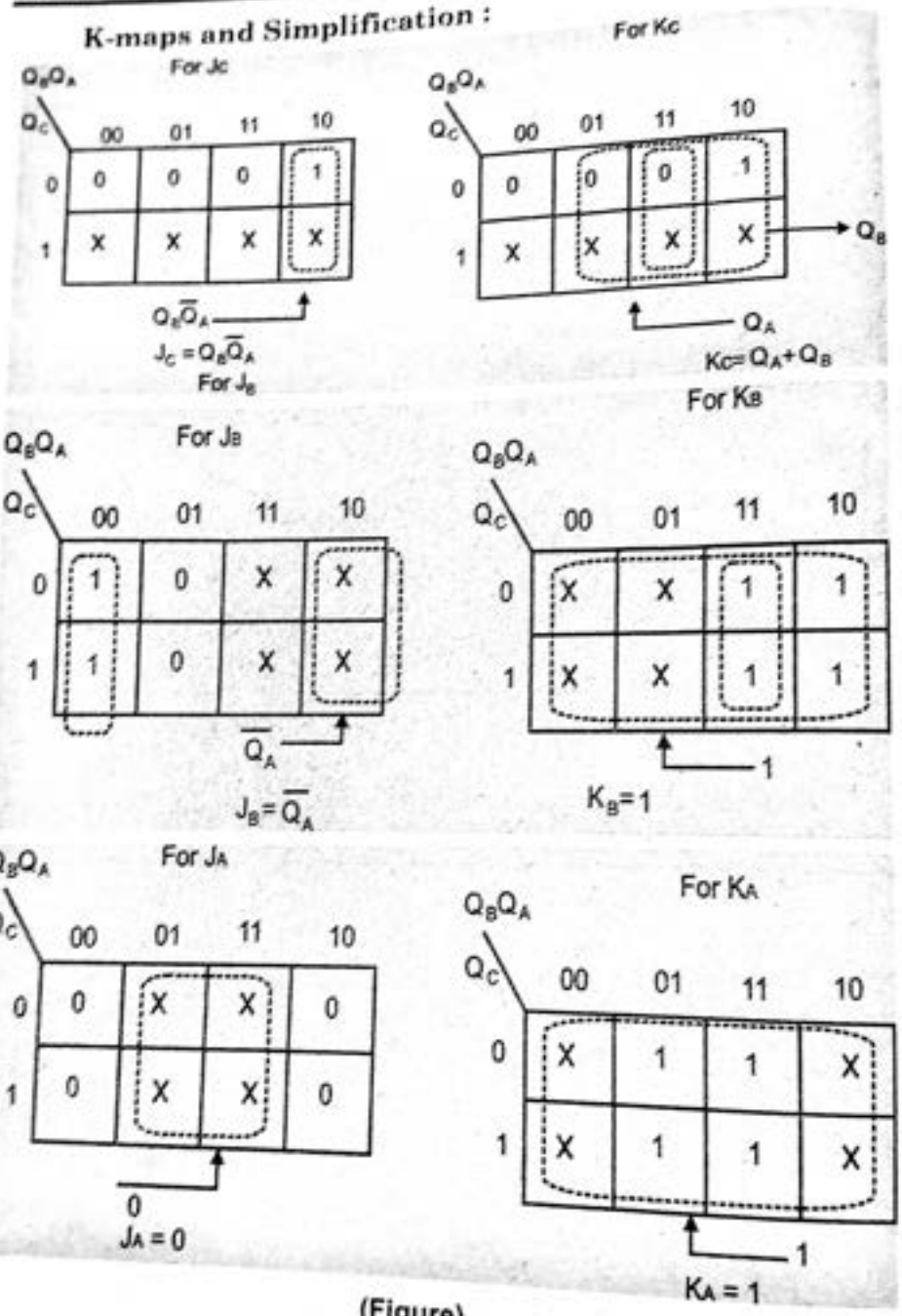
8. ***Design a synchronous counter having repeated sequence of 0, 2, 4, 6 using J – K FF. For remaining unused states, assume that the next state goes to 0.*** **(2019)**



(Figure)

Present State			Next State			Flip Flop Inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_K	K_C	J_B	K_B	J_A	K_A
0	0	0	0	1	0	0	×	1	×	0	×
0	0	1	0	0	0	0	×	0	×	×	1
0	1	0	1	0	0	1	×	×	1	0	×
0	1	1	0	0	0	0	×	×	1	×	1
1	0	0	1	1	0	×	0	1	×	1	×
1	0	1	0	0	0	×	1	0	×	×	1
1	1	0	0	0	0	×	1	×	1	1	×
1	1	1	0	0	0	×	1	×	1	×	1

K-maps and Simplification :



(Figure)

9. ***A 4 bit binary up / down counter is in binary state of zero. Find the next state in down mode.*** **(2017)**

1111.

10. ◆ *What*
that
4, 5, 6
◆ *Desig*
of six
K flip

Counter

A Cou
incremented
register, wh
upon the ap
An inp
When the v
the increme
count the
operations i

Numerico

Counter D
counter th
sequences
Step 1 :
table

Present S	
A	B
0	0
0	0
0	1
1	0
1	0
1	

the
are
Ste
K –
fun

10. ◆ ***What is counter? Explain. Design a counter that has repeated sequence of six states 0, 1, 2, 4, 5, 6 using JK flip- flop.*** **(2018)**

◆ ***Design a counter that has repeated sequence of six State : 0, 1, 2, 4, 5, 6 and repeat using J – K flip flop.*** **(2017)**

Counter

A Counter is a sequential circuit whose value is incremented by one, on the occurrence of some event. It is a register, which goes into series of predetermined states upon the application of inputs.

An input may be clock pulse or provided externally. When the value stored reaches, its maximum, then after the increment, its value becomes zero. Counters are used to count the number of line an event occurs and other operations in digital computers.

Numerical Solution

Counter Design : Using J – K flip flops to design a counter that repeatedly counts the following binary sequences : 0, 1, 2, 4, 5 and 6.

Step 1 : Derive the state table and the circuit excitation table

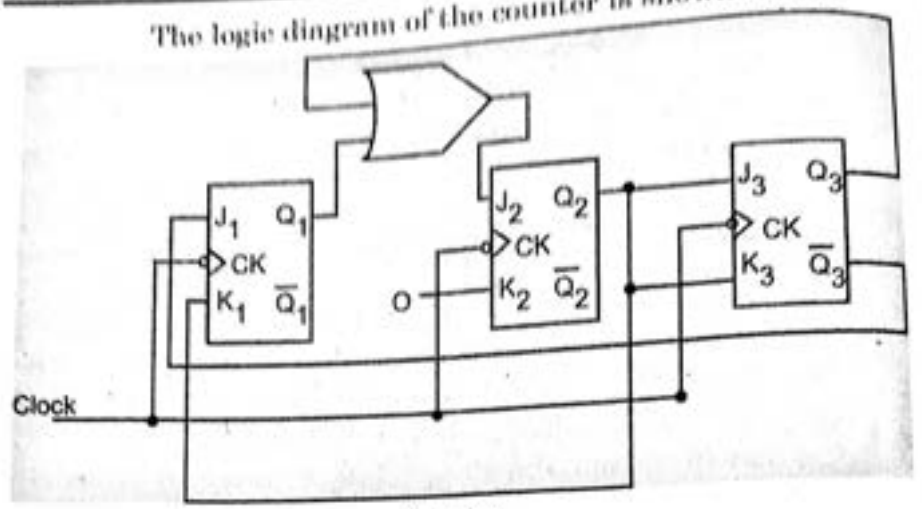
Present State			Next State			Flip – Flop Inputs					
A	B	C	A	B	C	JA	KA	JB	KB	JC	KC
0	0	0	0	0	1	0	x	0	x	1	x
0	0	1	0	1	0	0	x	1	x	x	1
0	1	0	1	0	0	1	x	x	1	0	x
1	0	0	1	0	1	x	0	0	x	1	x
1	0	1	1	1	0	x	0	1	x	x	1
1	1	0	0	0	0	x	1	x	1	0	x

Note that two states, 011 and 111 are not included in the count and hence their corresponding J and K inputs are don' care.

Step 2 : Derive the circuit output functions. By use of K – maps (with those don't cares figured in), the simplified functions are

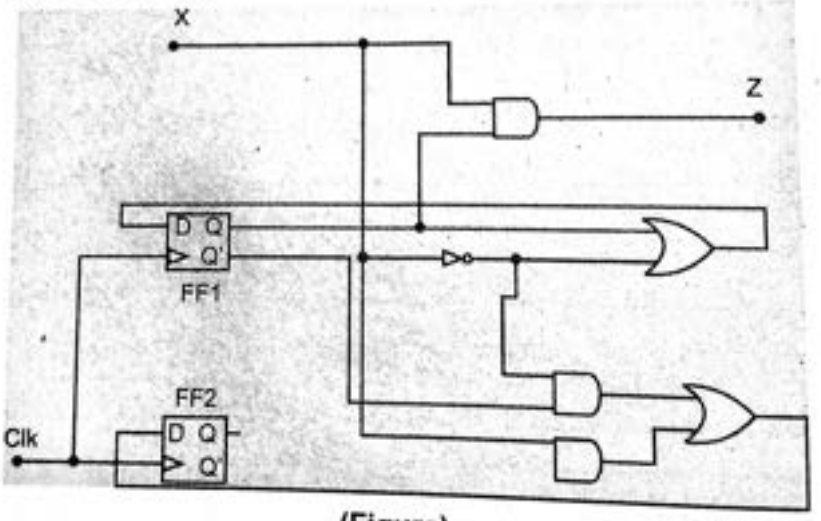
$$JA = B \quad KA = B$$
$$JB = C \quad KB = 1$$
$$JC = \overline{B} \quad KC = 1$$

The logic diagram of the counter is shown below :



(Figure)

Consider a sequential circuit shown in the following figure. It has one input x, one output Z and two state variables Q_1Q_2 (thus having four possible present states 00, 01, 10, 11)



(Figure)

The behavior of the circuit is determined by the following Boolean expressions :

$$Z = x \times Q1$$

$$D1 = x' + Q1$$

$$D2 = x \times Q2' + x' \times Q1'$$

These equations can be used to form the state table.

Table : State Table for the Sequential Circuit in figure

Presen
Q1
0
0
1
1

The s

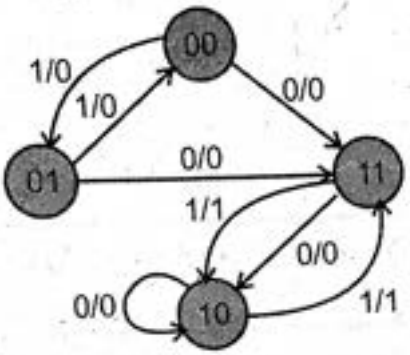
11. ◆ Wr
Pa
◆ Dr
po
◆ D
p

T
seriall
D. It
ente
(1)

sto
(2

Present State		Next State		Flip – Flop Inputs	
Q1	Q2	x = 0	x = 1	x = 0	x = 1
0	0	11	01	0	0
0	1	11	00	0	0
1	0	10	11	0	1
1	1	10	10	0	1

The state diagram for the sequential circuit in figure



(Figure)

11. ◆ ***Write short note on 4 Bit Register with Parallel load.*** **(2017)**

◆ ***Draw the block diagram of 4 bit register with parallel load.*** **(May 2013)**

◆ ***Describe with diagram the working of parallel in serial out shift register.***

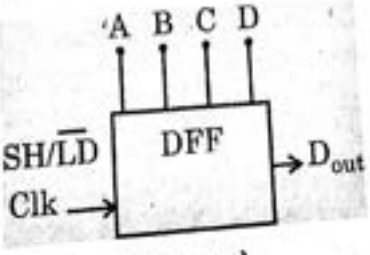
The data bits are entered parallel and taken out serially. It uses D-flip-flop and 4 data input times *A*, *B*, *C*, *D*. It has a shift/$\overline{\text{load}}$ input which allows 4 *bit* of data to be entered into the register simultaneously.

(1) When shift/$\overline{\text{load}}$ = 0

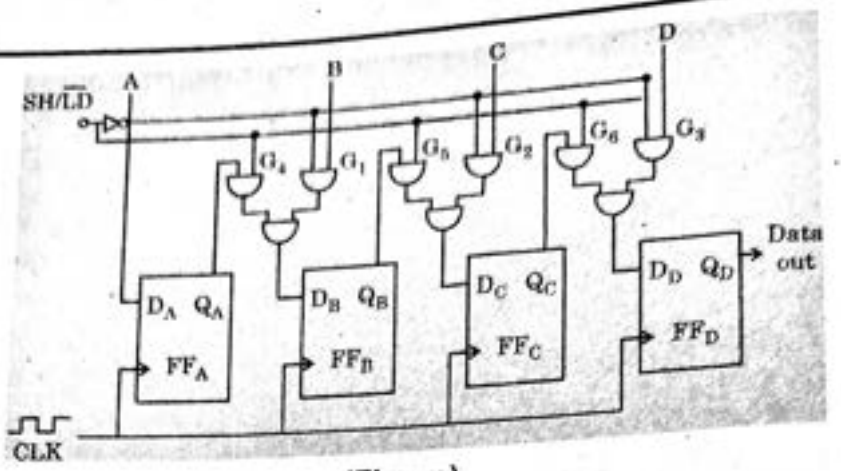
G_1, G_2, G_3 are enabled and data entered and stored.

(2) When shift/$\overline{\text{load}}$ = 1

G_4, G_5, G_6 are enabled and data get shifted.



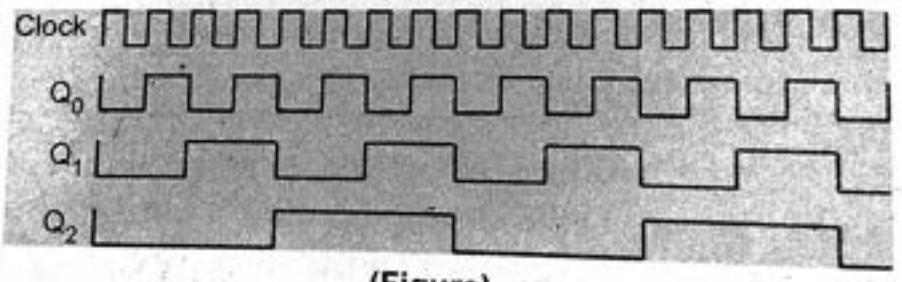
(Figure)



(Figure)

12. *Write short note on Divide-by-N ripple counter.* **(2016)**

Divide by N counter implies that it divides the input clock frequency by N. For Example : If you cascade four flip-flops then, the output of every stage is divided by 2, if you are taking the output from the 4th flip-flop, then its output frequency is clock frequency by 16(2^4). Example : Take a look at the timing diagram of a 3-bit counter.



(Figure)

A modulo *N* counter does the same thing as above and is also used as alternate definition which implies that the counter counts *N* states.

❑❑

MEMORY ORGANIZATION

1. *Differentiate between static & dynamic RAM.* **(2023-24)**

Difference between SRAM and DRAM

SRAM	DRAM
It stores information as long as the power is supplied.	It stores information as long as the power is supplied or a few milliseconds when the power is switched off.
Transistors are used to store information in SRAM.	Capacitors are used to store data in DRAM.
Capacitors are not used hence no refreshing is required.	To store information for a longer time, the contents of the capacitor need to be refreshed periodically.
SRAM is faster compared to DRAM.	DRAM provides slow access speeds.
It does not have a refreshing unit.	It has a refreshing unit.
These are expensive.	These are cheaper.
SRAMs are low-density devices.	DRAMs are high-density devices.
In this bits are stored in voltage form.	In this bits are stored in the form of electric energy.
These are used in cache memories.	These are used in main memories.
Consumes less power and generates less heat.	Uses more power and generates more heat.
SRAMs has lower latency	DRAM has more latency than SRAM
SRAMs are more resistant to radiation than DRAM	DRAMs are less resistant to radiation than SRAMs
SRAM has higher data transfer rate	DRAM has lower data transfer rate
SRAM is used in high-speed cache memory	DRAM is used in lower-speed main memory
SRAM is used in high performance applications	DRAM is used in general purpose applications

2. *What do you understand by maxterm & minterm?* **(2023-24)**

Minterm : Minterm is the product of various different literals in which each literal occurs exactly once. The output result of the minterm functions is 1. It is represented by m. To represent a function, we perform a sum of minterms also called the Sum Of Products (SOP).

Example of SOP : $A'B + AC + BC$

Maxterm : Maxterm is the sum of various different literals in which each literal occurs exactly once. The output result of the maxterm functions is 0. It is represented by M. To represent a function, we perform product of maxterms also called Product of Sum (POS).

Example of POS : $(A + B).(C + D)$

Minterm VS Maxterm :

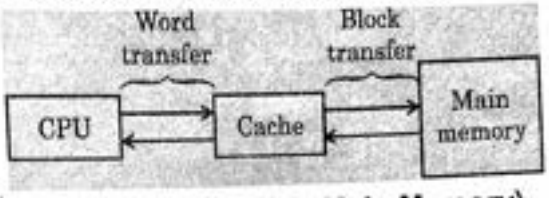
Minterm	Maxterm
Minterm is the term with the product of N literals occurring exactly once.	Maxterm is the term with the sum of N literals occurring exactly once.
It is represented by m.	It is represented by M.
It is logical AND of distinct literals.	It is logical OR of distinct literals.
The sum of minterms forms SOP (Sum of Product) functions.	The product of maxterms forms POS (Product of Sum) functions.
The output result of minterm function is 1.	The output result of maxterm function is 0.
It works on active high.	It works on active low.
Example: AB + A'B'	Example: (A+B). (A'+B')

3. ◆ ***Explain cache memory in detail.*** **(2023-24)**
   ◆ ***What is Cache memory? Design 32 × 8 RAM structure.*** **(2018)**
   ◆ ***Write short note on Cache memory.*** **(2014, 2017)**
   ◆ ***Explain the basic read operation in cache memory organization.*** **(2016)**

Cache Memory

Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memories. The concept is illustrated in figure. There is a relatively large and slow main memory together with a smaller, faster cache memory. The cache contains a copy of portions of main memory (when the processor attempts to read a word of memory) a check is made to determine if the word is in

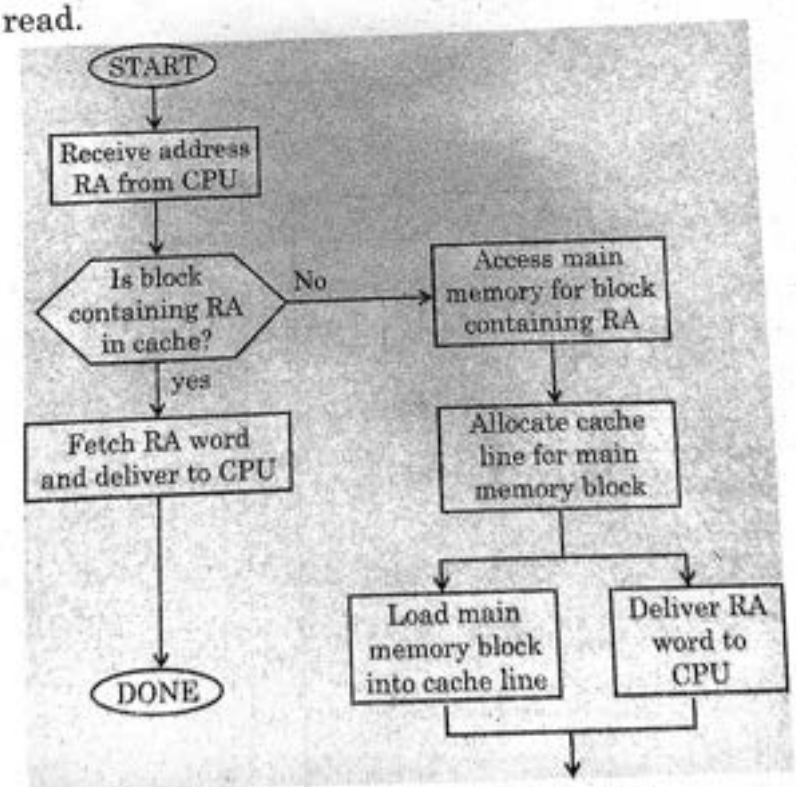
the cache. If so, the word is delivered to the processor. If not, a block of main memory consisting of some fixed number of words is read into the cache and then the word is delivered to the processor. Because of the phenomenon of locality of reference, when a block of data is fetched into the cache to satisfy a single memory reference, it is likely that there will be future references to that same memory location or to other words in the block.



(Figure : Cache and Main Memory)

Cache Memory Operation

The processor generates the address, RA, of a word to be read.

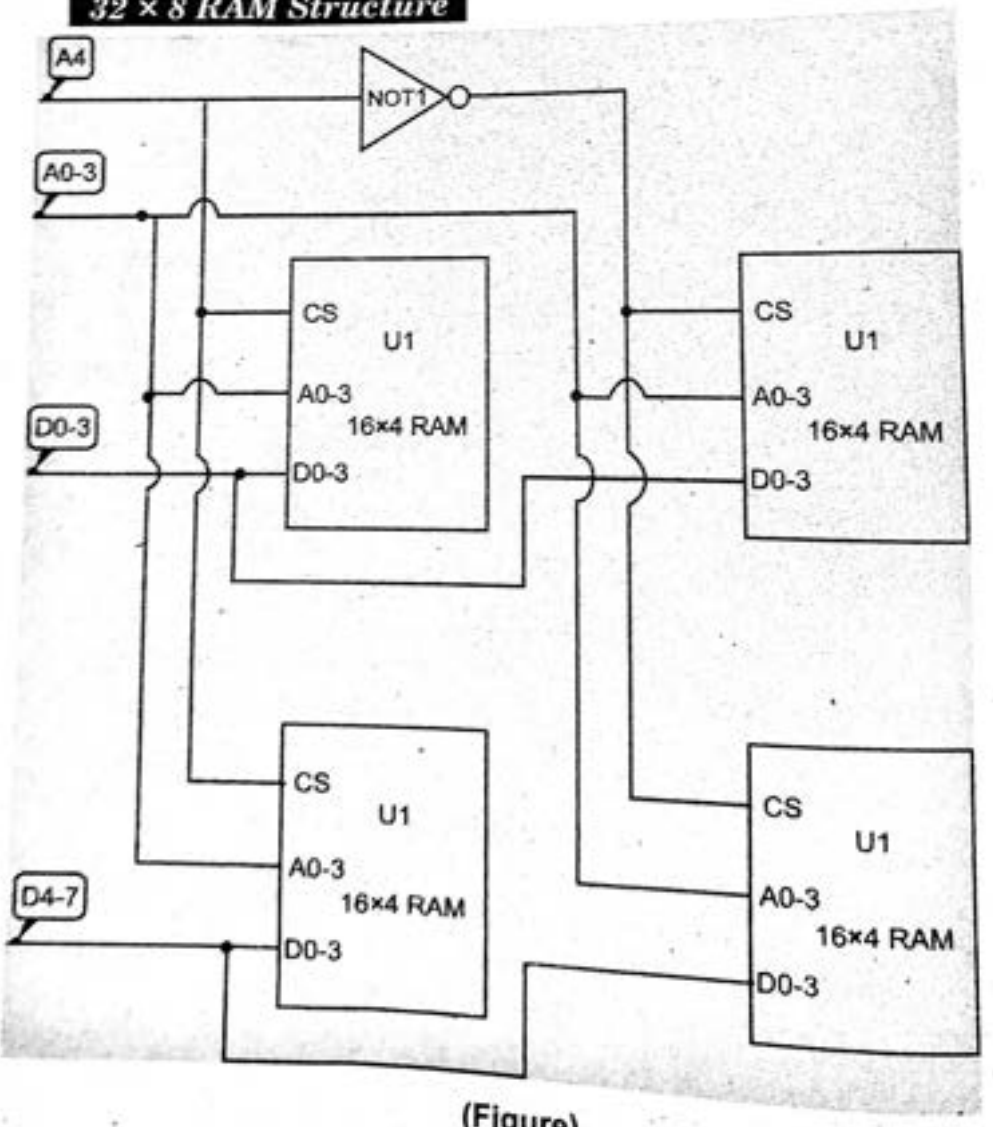


(Figure : Cache Read Operation)

If the word is contained in the cache, it is delivered to the processor. Otherwise, the block containing that word is loaded into the cache, and the word is delivered to the processor. Figure shows these last two operations occurring in parallel and reflects the organization shown in figure, which is typical of contemporary cache organizations.

In this organization, the cache connects to the processor via data, control and address lines. The data and address lines also attach to data and address buffers, which attach to a system bus from which main memory is reached. When a cache hit occurs, the data and address buffers are disabled and communication is only between processor and cache, with no system bus traffic. When a cache miss occurs, the desired address is loaded onto the system bus and the data are returned through the data buffer to both the cache and the processor. In other organizations, the cache is physically interposed between the processor and the main memory for all data, address and control lines. In this case, for a cache miss the desired word is first read into the cache and then transferred from cache to processor.

32 × 8 RAM Structure



(Figure)

4. ◆ ***Define virtual memory in detail.*** **(2023-24)**
◆ ***What is virtual memory?*** **(2022-23)**
◆ ***Write short note on Virtual memory and its implementation.*** **(2019)**
◆ ***Write short note on Virtual memory.*** **(2012, 2014, 2017, 2018)**
◆ ***What is virtual memory? Discuss the technique to manage a virtual memory organization in detail.*** **(2016)**

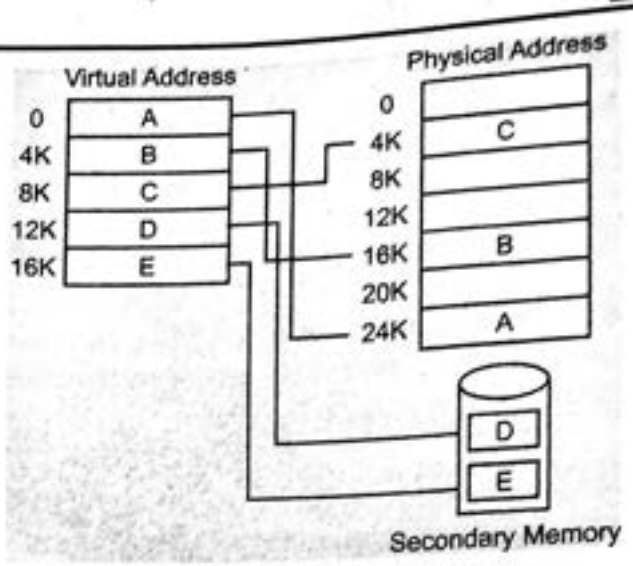
Virtual Memory

The main memory is considered the physical memory in which many programs want to reside. However, the limited-size physical memory cannot load in all programs simultaneously. The virtual memory concept was introduced to alleviate this problem. The idea is to expend the use of the physical memory among many programs with the help of an auxiliary (backup) memory such as disk arrays.

Only active programs or portions of them become residents of the physical memory at one time. The vast majority of programs or inactive programs are stored on disk. All programs can be loaded in and out of the physical memory dynamically under the coordination of the operating system. To the users, virtual memory provides them with almost unbounded memory space to work with. Without virtual memory, it would have been impossible to develop the multi – programmed or time – sharing computer systems that are in use today.

Technique to Manage a Virtual Memory Organization

Modern microprocessors intended for general purpose use, a memory management unit, or MMU is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below :



(Figure)

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

5. *Explain the various page replacement algorithm in brief.* **(2022-23)**

There are three types of Page Replacement Algorithms. They are :

(1) Optimal Page Replacement Algorithm
(2) First In First Out Page Replacement Algorithm
(3) Least Recently Used (LRU) Page Replacement Algorithm

(1) **First In First Out (FIFO) :** This is the simplest page replacement algorithm. This algorithm evicts the oldest page in memory. It treats memory as a circular queue, where the page that was loaded first is the first one to be replaced. However, FIFO may suffer from the "Belady's Anomaly" where increasing the number of allocated frames can result in more page faults.

(2) **Optimal Page Replacement :** This algorithm selects the page for eviction that will be accessed furthest in the future. It requires knowledge of future memory references, which is usually not practical in real-time systems. OPT serves as a theoretical benchmark for other algorithms.

(3) Least Recently Used : This algorithm replaces the page that has not been accessed for the longest period of time. It assumes that pages that have been accessed recently are more likely to be accessed again in the near future.

Example : Let's consider a scenario where we have a system with a total of 4 page frames available in memory, and the following sequence of page references is received :

5, 2, 3, 1, 4, 2, 5, 1, 6, 2

FIFO :

				1	4	4	5	5	6	2
			3	3	1	1	4	4	5	6
		2	2	2	3	3	1	1	4	5
	5	5	5	5	2	2	3	3	1	4
Incoming →	**5**	**2**	**3**	**1**	**4**	**2**	**5**	**1**	**6**	**2**
PF →	**Y**	**Y**	**Y**	**Y**	**Y**	**N**	**Y**	**N**	**Y**	**Y**

Total Page Faults = 8

Optimal :

				1	1	1	1	1	6	6
			3	3	4	4	4	4	4	4
		2	2	2	2	2	2	2	2	2
	5	5	5	5	5	5	5	5	5	5
Incoming →	**5**	**2**	**3**	**1**	**4**	**2**	**5**	**1**	**6**	**2**
PF →	**Y**	**Y**	**Y**	**Y**	**Y**	**N**	**N**	**N**	**Y**	**N**

Total Page Faults = 6

LRU :

				1	1	1	1	1	1	1
			3	3	3	3	5	5	5	5
		2	2	2	2	2	2	2	2	2
	5	5	5	5	4	4	4	4	6	6
Incoming →	**5**	**2**	**3**	**1**	**4**	**2**	**5**	**1**	**6**	**2**
PF →	**Y**	**Y**	**Y**	**Y**	**Y**	**N**	**Y**	**N**	**Y**	**N**

Total Page Faults = 7

6. *What do you understand by static memory?*

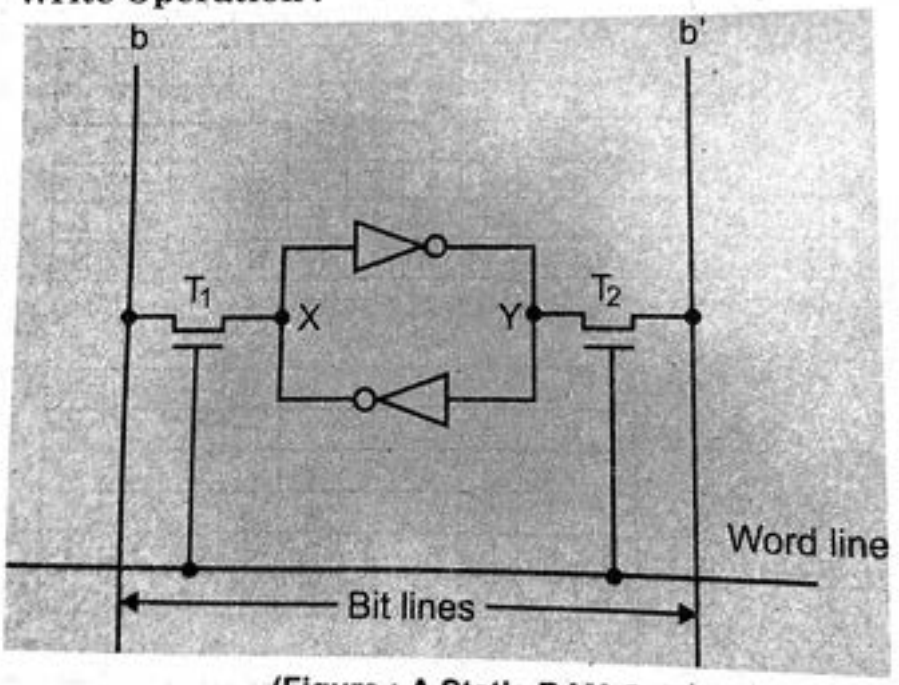
Static Memories

Static memories are the Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memories. Figure illustrates how a static RAM (SRAM) cell may be implemented. Two inverters are cross-connected to form a latch. The latch is

connected to two bit lines by transistors $T1$ and $T2$. These transistors act as switches that can be opened or closed under control of the word line. When the word line is at ground level, the transistors are turned off and the latch retains its state. For example, let us assume that the cell is in state 1 if the logic value at point X is 1 and at point Y is 0. This state is maintained as long as the signal on the word line is at ground level.

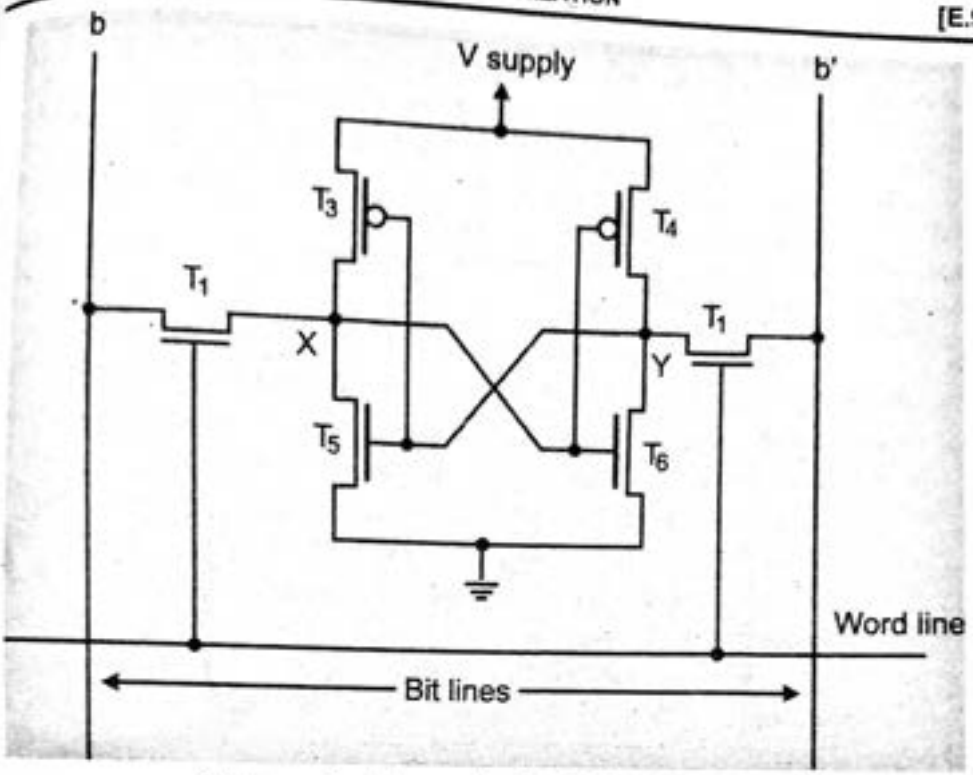
Read Operation : In order to read the state of the SRAM cell, the word line is activated to close switches $T1$ and $T2$. If the cell is in state 1, the signal on bit line b is high and the signal on bit line b' is low. The opposite is true if the cell is in state 0. Thus, b and b' are complements of each other. Sense/Write circuits at the end of the bit lines monitor the state of b and b' and set the output accordingly.

Write Operation :



(Figure : A Static RAM Cell)

The state of the cell is set by placing the appropriate value on bit line b and its complement on b', and then activating the word line. This forces the cell into the corresponding state. The required signals on the bit lines are generated by the Sense/Write circuit. if T_1 and T_2 are turned on (closed), if lines b and b' will have high and low signals, respectively.



(Figure : An Example of a CMOS Memory Cell)

The power supply voltage, V supply, is 5 V in older CMOS SRAMs or 3.3 V in new low voltage versions. Note that "continuous power is needed for the cell to retain its state. If power is interrupted, the cell's contents will be lost. When power is restored, the latch will settle into a stable state, but it will not necessarily be the same state the cell was in before the interruption. Hence, SRAMs are said to be volatile memories because their contents are lost when power is interrupted.

A major advantage of CMOS SRAMs is their very low power consumption because current flows in the cell only when the cell is being accessed. Otherwise, T_1, T_2 and one transistor in each inverter are turned off, ensuring that there is no active path between V supply and ground. Static RAMs can be accessed very quickly. Access times of just a few nanoseconds are found in commercially available chips. SRAMs are used in applications where speed is of critical concern.

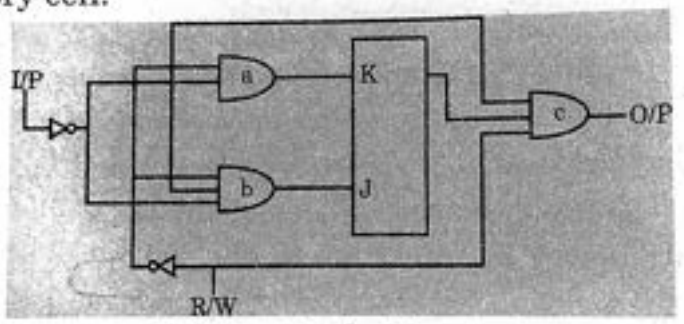
7. *Discuss the RAM Chip Design. A computer uses RAM chip of 1024×1 capacity.*

(1) *How many chips are needed, and how should their address lines be connected to provide a memory capacity of 1024 bytes × 8?*
(2) *How many chips are needed to provide a memory capacity of 16KB?*

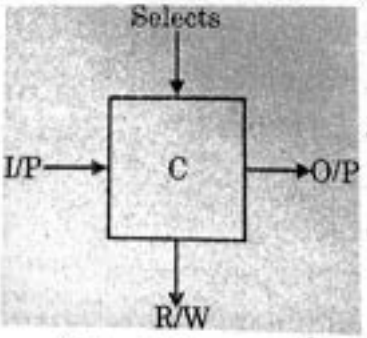
RAM Chip Design

RAM is the main memory that is the randomly accessed memory. It is normally organized as words of fixed length. These memory words have an independent address and each have same number of bits.

The total number of words in memory is some power of 2. It is both type readable and writable and R/W type. The access time and cycle time in RAMs are constant and independent of the accessed location. In RAM, the bits are stored using a sequential circuit of flip-flop or binary cell or memory cell.



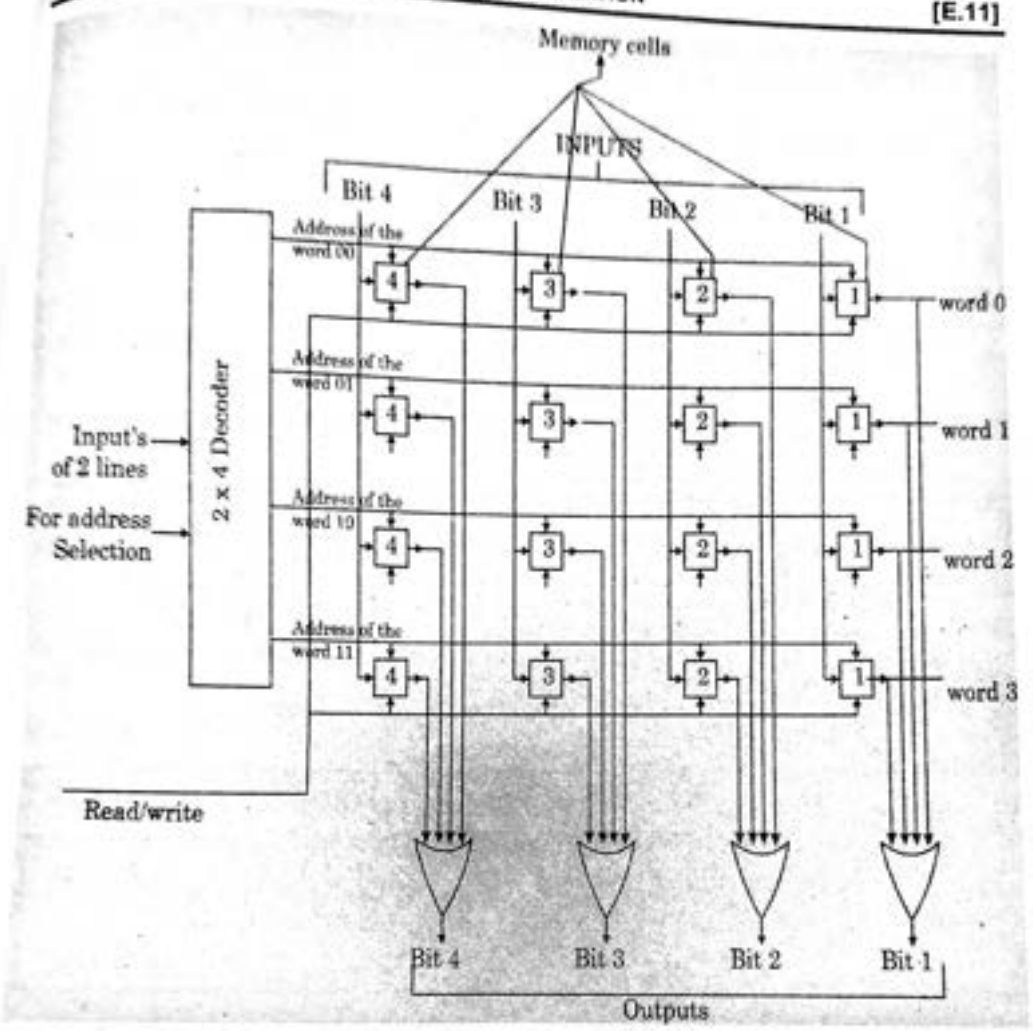
(Figure : Logic Diagram)



(Block Diagram)

In this RAM organization, the input is fed in complemented form to AND gate '*a*'. The read/ write signal have a value 1 if it is a read operation.

Therefore, during the read operation only the AND gate '*c*' becomes active. Since AND '*a*' and '*b*' have 0 inputs, and if the select is 1, i.e. this cell is currently being selected, then the output will become equal to the state of flip-flop, means the data value has been read.



(Figure : Logic Diagram of 4 × 4 RAM Organization)

In write operation, only '*a*' and '*b*' gates become active and they set or clear the J-K flip-flop depending on the input value, when the input is 0 flip-flop will go to clear state and if input is 1, The flip-flop go to set.

In effect, the input data is reflected in the state of the flip-flop. Thus say that the input data stored in flip-flop or binary cell or memory cell. The RAM Organization given by 4 × 4 decoder is the extension of binary cells, to a IC RAM circuit, where 2 × 4 decoder is used for inputs and each decoder output is connected to a 4 bit word and the r/w signal, and supplied to binary cell.

The output is derived using OR gate and all non selected cells will produce a zero output, and selected word will determine the overall output.

Numerical Solution : A Computer uses RAM chips of 1024×1 capacity :

(1) Number of RAM chips required $= \frac{\cancel{1024}}{\cancel{1024}} = 1$

(2) Number of RAM chips required $= \frac{16K}{1024} = \frac{16 \times \cancel{1024}}{\cancel{1024}} = 16$

$16K = 2^4 \times 2^{10} = 2^{14}$, therefore 10 lines are required to address each chip. Remaining $14 - 10 = 4$ lines are required to decoder for selecting 16 chips. Therefore, size of decoder 4×16 decoder.

8. ***The application program in a computer system with cache uses 1400 instruction acquisition bus cycle from cache memory and 100 from main memory. What is the hit rate?***

$$\text{Hit rate} = \frac{\text{Number of hits}}{\text{Number of read/wirte bus cycles}} \times 100\%$$

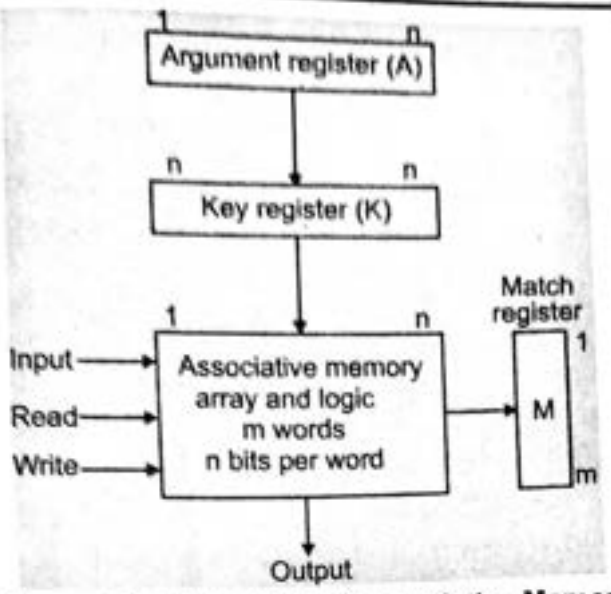
$$= \frac{1400}{1400+100} \times 100 = 93.3333\% \quad \textbf{(Ans.)}$$

9. ◆ ***What is Associative Memory? Explain the logic behind its operation through its organization schematically.*** **(2019)**

◆ ***Write short note on Associative memory.*** **(May 2012)**

Associative Memory

Many data-processing applications require the search of items in a table stored in memory. They use object names or number to identify the location of the named or numbered object within a memory space. For example, an account number may be searched in a file to determine the holder's name and account status. To search an object, the number of accesses to memory depends on the location of the object and the efficiency of the search algorithm. The time required to find an object stored in memory can be reduced considerably if objects are selected based on their contents, not on their locations. A memory unit accessed by the content is called an associative memory or content addressable memory (CAM). This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.



(Figure : Block Diagram of Associative Memory)

10. ***How many 256 × 8 RAM chips are needed to provide a memory capacity of 2048 bytes? Also find the number of address lines and data lines.*** **(2018)**

$$\text{Number of chips required} = \frac{2048 \times 8}{256 \times 8}$$

$$= 8 \text{ Chips}$$

Total memory is of 2048 bytes i.e. 2^{11} bytes.

Hence, 11 lines of address bus is required to access 2048 bytes.

RAM chip is of $256 = 2^8$ bytes

So 8 lines out of 11 are common to all chips.

11. ***How many 32 K × 1 RAM chips are needed to provide a memory capacity of 256 K bytes?*** **(May 2013, 2015)**

Each IC gives 1 *bit*. Hence to form a byte 8 Ic's are needed. Since each IC then 32 *K* location, eight IC give 32 *KB* capacity.

Hence to obtain 256 *KB* capacity, we need

$$\left(\frac{256\,KB}{32\,KB} = 8\right) 8 \times 8 = 64 \; IC's$$ **(Ans.)**

12. ***A RAM chip has a capacity of 1024 words of 8 bits each (1 k × 8). Find the number of 2 × 4 decoders with enable lines needed to construct a 16 k × 16 RAM from 1 k × 8 RAM.*** **(2015)**

RAM chip size = 1 k × 8 (1024 words of 8 *bits* each)

RAM to construct = 16k × 16

No. of chips required = $\frac{16k \times 16}{1k \times 8} = 16k \times 2$

16 chips vertically with each having 2 chips horizontally. So to select one chip out of 16 vertical chips, we need 4 × 16 decoder. Available decoder is 2 × 4 decoder. To be constructed is 4 × 16 decoder. Hence 4 + 1 = 5 decoders are required.

13. ◆ ***How many 128 × 8 RAM chip are needed to provide a memory capacity of 2048 bytes?*** **(2017)**

◆ (1) ***How many 128 × 8 RAM chips are needed to provide a memory capacity of 2048 bytes?***

(2) ***How many lines of the address bus must be used to access 2048 bytes of memory? How many of these lines will be common to all chips?*** **(May 2012)**

(1) Chips required are = $\frac{2048}{128} = 16$

(2) $2^{11} = 2048$

∴ address lines required are = 11

as $2^7 = 128$

lines common are = 7

and four lines are used to select chip.

❑❑

BCA

(SEM. III) EXAMINATION, 2022-23

BCA – 3004 : DIGITAL ELECTRONICS AND COMPUTER ORGANIZATION

Time : 2 Hours *Maximum Marks : 75*

Note : This paper consists of three Sections A, B and C. Carefully read the instructions of each Section in solving the question paper. Candidates have to write their answers in the given answer-copy only. No separate answer copy (B-Copy) Will be provided.

SECTION – A
(Short Answer Type Questions)

Note : All questions are compulsory. Answer the following questions as short answer type questions. Each question carries 5 marks.

1. (A) Define min terms and max terms.
   (B) How can AND OR circuit converted to NAND logic?
   (C) Design half adder and full adder using only NAND gates.
   (D) What is virtual memory?
   (E) Differentiate between combinational and sequential circuit.
   (F) Express the following function :

   $F_1 = A\overline{C} + AB + BC$ in canonical S.O.P.

   $F_2 = (A + B)(B + C)(A + C)$ in canonical P.O.S.

   (G) What is multiplexer? Draw the logic diagram of 8 x 1 multiplexer.
   (H) What are the advantages of JK flip – flap over an SR flip – flop?
   (I) What is the difference between synchronous and asynchronous counter?

SECTION – B
(Long Answer Type Questions)

Note : This section contains four questions from which one question is to be answered as long question. Each question carries 15 marks.

2. Design a code converter that converts Binary code to Gray code.

Or

3. Simplify the Boolean expression using

$$F(A, B, C, D) = \prod M(4, 5, 6, 7, 8, 12).$$
$$d(0, 1, 2, 3, 4, 11, 14)$$

SCAN AnyScanner

Or

4. What is Encoder? Explain in brief about octal to Binary Encoder.

5. (a) Implement the following function using 8 : 1 MUX :
$F(A, B, C, D) = \Sigma m(0, 1, 4, 8, 12, 14, 15)$

(b) Design a 5 x 32 decoder using one 2 x 4 and four 3 x 8 decoder IC's.

SECTION – C
(Long Answer Type Questions)

Note : This section contains four questions from which one question is to be answered as long question. Each question carries 15 marks.

6. Explain the procedure to convert one flip – flop to another flip – flop.

Or

7. Define Shift Register. Explain different types of Shift Register.

Or

8. Explain the various methods for converting asynchronous UP counter into asynchronous DOWN counter with example.

Or

9. Explain the various page replacement algorithm in brief.

❑❑

BCA

(SEM. III) EXAMINATION, 2023-24
BCA – 3004 : DIGITAL ELECTRONICS & COMPUTER ORGANIZATION

Time : Two Hours *Maximum Marks : 75*

Note : This paper consists of three Section A, B and C. Carefully read the instructions of each Section in solving the question paper. Candidates have to write their answers in the given answer-copy only. No separate answer – copy (B copy) will be provided.

SECTION – A
(Short Answer Type Questions)

Note : *All* questions are compulsory. Answer the following questions as short answer type questions. Each question carries 5 marks.

1. (A) Convert the given decimal number to equivalent octal number –
$(125)_{10} \rightarrow (?)_8$

(B) Simplify the following Boolean expression using Boolean algebra-
B (A + B') (B + C)

(C) Explain half adder with block diagram and truth table.

(D) Differentiate between combinational & sequential circuit.

(E) What do you understand by maxterm & minterm?

(F) Differentiate between static & dynamic RAM.

(G) What do you understand by shift register?

(H) What do you understand by ring counter?

(I) What is flip flop?

SECTION – B
(Long Answer Type Questions)

Note : This section contains four questions from which one question is to be answered as long question. Each question carries 15 marks.

2. (a) Explain all Logic gates with diagram & truth table.

(b) How many 32 K×1 RAM chips are needed to provide a memory capacity of 256 K bytes?

Or

3. Simplify the POS using K-map :
$F(A, B, C, D) = \Pi M (1,3,5,7,8, 9, 10, 11, 12, 13, 14, 15)$

Or

4. What is Multiplexer? Construct a 16×1 line multiplexer with two 8×1 and one 2×1 line multiplexer.

AnyScanner

Or

5. Define full adder with logic circuit diagram & truth table.

SECTION – C
(Long Answer Type Questions)

Note : This section contains four questions from which one question is to be answered as long question. Each question carries 15 marks.

6. Explain working of SR & JK flip flop with logic diagram and characteristics table.

Or

7. (a) Explain cache memory in detail.
   (b) Define virtual memory in detail.

Or

8. (a) Differentiate between synchronous asynchronous counter.
   (b) Design 3 – bit (MOD - 8) asynchronous counter with state ar d Logic diagram.

Or

9. Explain types of shift register.

AIR-05
AIR-41
AIR-10
AnyScanner