LONG ANSWER QUESTIONS:

1. Draw and explain the components of a Digital Computer.

A computer is a machine that can perform computation. A computation involves the following three components:

- Input: The user gives a set of input data.
- Processing: The input data is processed by a well-defined and finite sequence of steps.
- Output: Some data available from the processing step are output to the user.

A *digital computer* accepts, processes and outputs data in digitized forms (as opposed to analog forms).

The basic components of a digital computer

In order that a digital computer can solve problems, it should be equipped with the following components:

- **Input devices**
  These are the devices using which the user provides input instances. In a programmable computer, input devices are also used to input programs. Examples: keyboard, mouse.

- **Output devices**
  These devices notify the user about the outputs of a computation. Example: screen, printer.

- **Processing unit**
  The central processing unit (CPU) is the brain of the computing device and performs the basic processing steps. A CPU typically consists of:

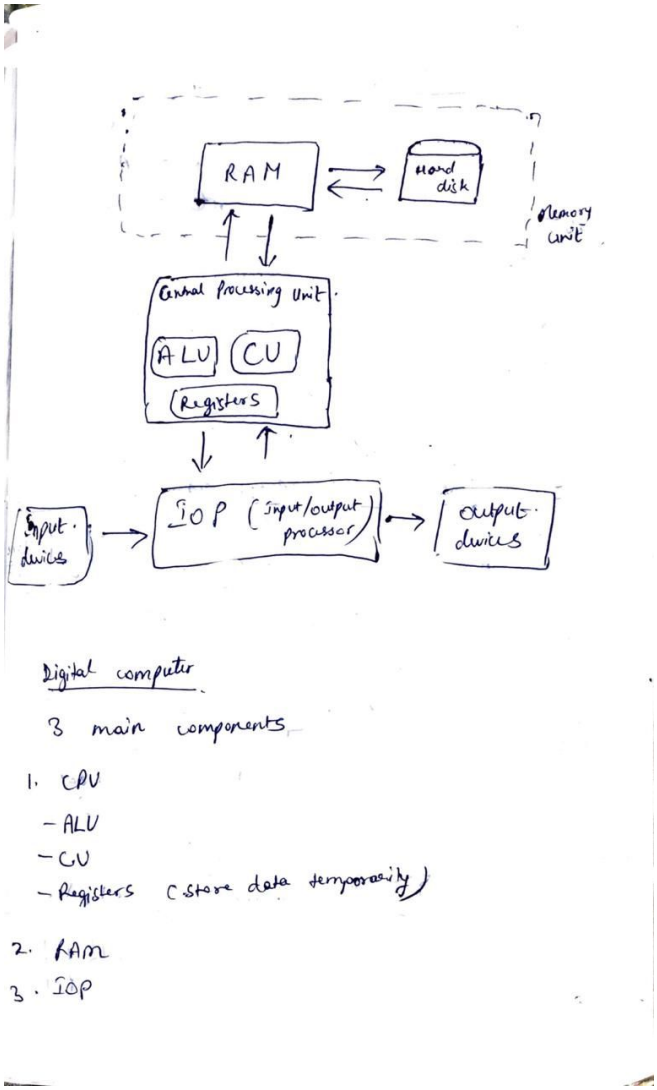
➔ **An arithmetic and logical unit (ALU):** This provides the basic operational units of the CPU. It is made up of units (like adders, multipliers) that perform arithmetic operations on integers and real numbers, and of units that perform logical operations (logical and bitwise AND, OR etc.).
➔ **A control unit:** This unit is responsible for controlling flow of data and instructions.
➔ **General purpose registers:** A CPU usually consists of a finite number of memory cells that work as scratch locations for storing intermediate results and values.

- **External memory**
  The amount of memory (registers) resident in the CPU is typically very small and is inadequate to accommodate programs and data even of small sizes.
  Out-of-the-processor memory provides the desired storage space. External memory is classified into two categories:

➔ **Main (or primary) memory:** This is a high-speed memory that stays close to the CPU. Programs are first loaded in the main memory and then executed. Usually main memory is volatile, i.e., its contents are lost after power-down.

➔ **Secondary memory:** This is relatively inexpensive, bigger and low-speed memory. It is normally meant for off-line storage, i.e., storage of programs and data for future processing. One requires secondary storage to be permanent, i.e., its contents should last even after shut-down. Examples of secondary storage include floppy disks, hard disks and CDROM disks.

- **Buses**
  A bus is a set of wires that connect the above components. Buses are responsible for movement of data from input devices, to output devices and from/to CPU and memory.



2. Discuss about the principle of Von Neumann Architecture.

ANS) Historically there have been 2 types of Computers:

1. **Fixed Program Computers –** Their function is very specific and they couldn't be programmed, e.g. Calculators.
2. **Stored Program Computers –** These can be programmed to carry out many different tasks, applications are stored on them, hence the name.

The modern computers are based on a stored-program concept introduced by John Von Neumann. In this stored-program concept, programs and data are stored in a separate storage unit called memories and are treated the same. This novel idea meant that a computer built with this architecture would be much easier to reprogram.

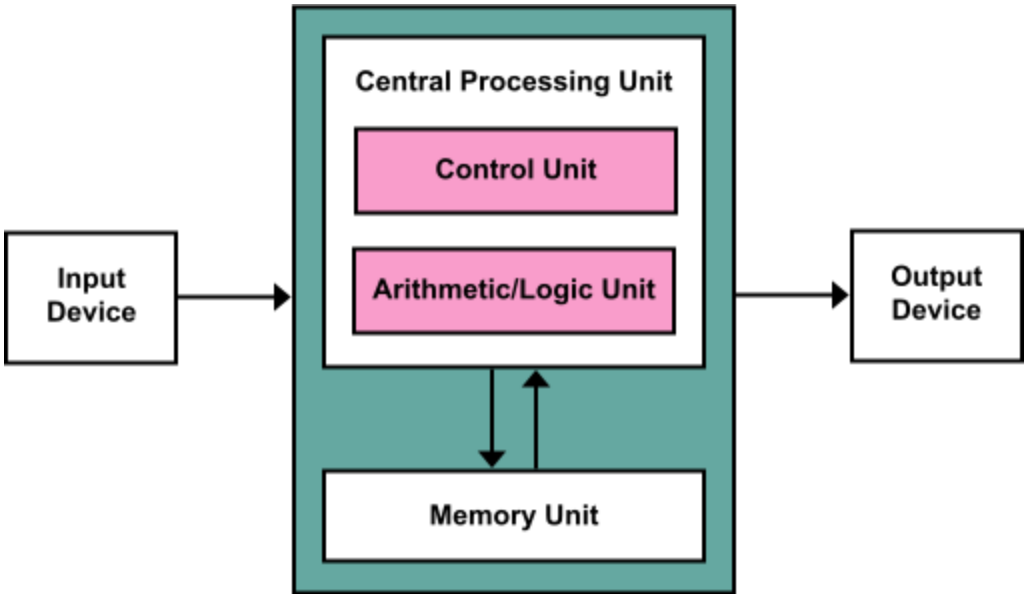
von Neumann architecture

The **von Neumann architecture** describes a general framework, or structure, that a computer's hardware, programming, and data should follow. Although other structures for computing have been devised and implemented, the vast majority of computers in use today operate according to the von Neumann architecture.

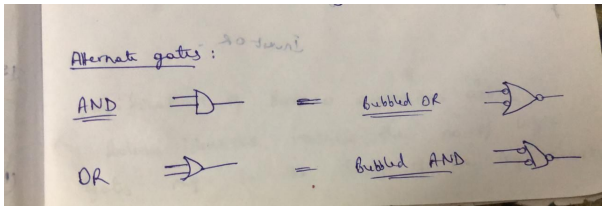
von Neumann envisioned the structure of a computer system as being composed of the following components:

1. **ALU:** The **Arithmetic-Logic unit** that performs the computer's computational and logical functions. e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic Operation.
2. **RAM:** Memory; more specifically, the computer's main, or fast, memory, also known as **Random Access Memory(RAM)**.
3. **Control Unit**: This is a component that directs other components of the computer to perform certain actions, such as directing the fetching of data or instructions from memory to be processed by the ALU; and
4. **Input/Output Devices –** Program or data is read into main memory from the *input device* or secondary storage under the control of CPU input instruction.

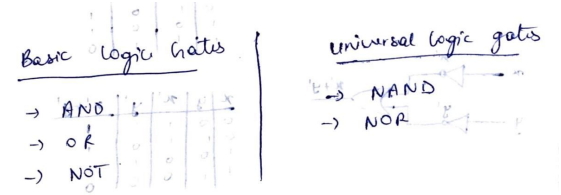
Output devices are used to output the information from a computer. If some results are evaluated by computer and it is stored in the computer, then with the help of output devices, we can present it to the user.



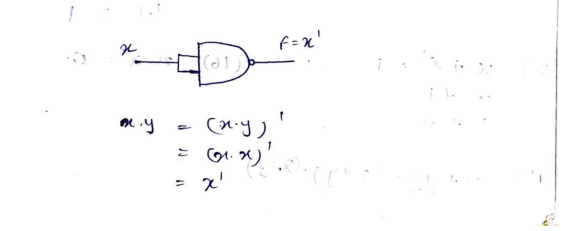
An example of computer architecture base on the von Neumann architecture is the desktop **personal computer**.



NAND REALIZATION(DERIVING OTHER GATES USING NAND):

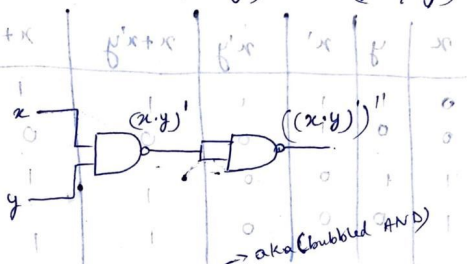


Realisation of ~~NAND~~ NOT gate using NAND



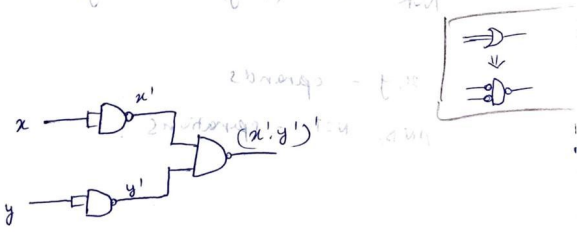
Realisation of AND using NAND

$x \cdot y = ((x \cdot y)')' = (x' + y')'$

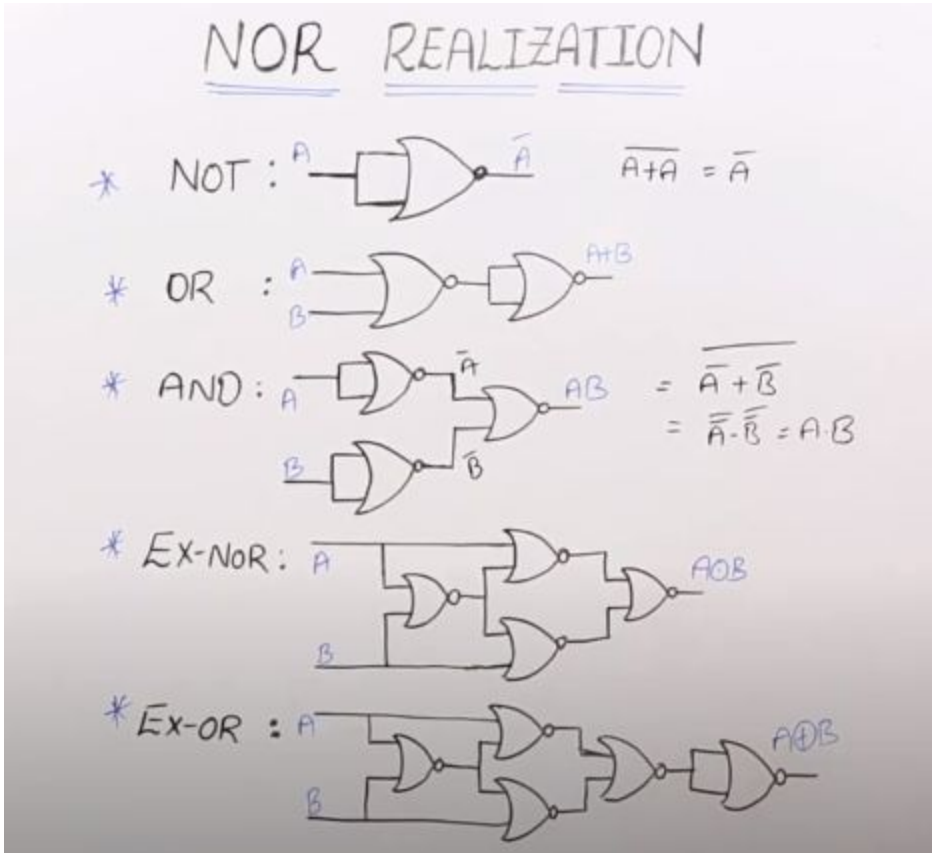


Realisation of OR using NAND

$x + y = ((x+y)')' = (x' \cdot y')'$



NOR REALIZATION(DERIVING OTHER GATES USING NOR):



3. List the Boolean Identities. (Or) State the laws of Boolean algebra.

1.	Law of Identity	$A = A$ $\overline{A} = \overline{A}$
2.	Commutative Law	$A \cdot B = B \cdot A$ $A + B = B + A$
3.	Associative Law	$A \cdot (B \cdot C) = A \cdot B \cdot C$ $A + (B + C) = A + B + C$
4.	Idempotent Law	$A \cdot A = A$ $A + A = A$
5.	Double Negative Law	$\overline{\overline{A}} = A$
6.	Complementary Law	$A \cdot \overline{A} = 0$ $A + \overline{A} = 1$
7.	Law of Intersection	$A \cdot 1 = A$ $A \cdot 0 = 0$
8.	Law of Union	$A + 1 = 1$ $A + 0 = A$
9.	DeMorgan's Theorem	$\overline{AB} = \overline{A} + \overline{B}$ $\overline{A + B} = \overline{A}\,\overline{B}$
10.	Distributive Law	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ $A + (BC) = (A + B) \cdot (A + C)$
11.	Law of Absorption	$A \cdot (A + B) = A$ $A + (AB) = A$
12.	Law of Common Identities	$A \cdot (\overline{A} + B) = AB$ $A + (\overline{A}B) = A + B$

(13. X+X' Y = X+Y)

4. Simplify the following Boolean Expression using Boolean Identities and draw the logic diagram for the simplified expression.

a. A+BC'+ABD'+ABCD

b. A'BC+AC

c. A'B+ABC'+ABC

d. (BC'+A'D) (AB'+CD')

(a) $A + BC' + ABD' + ABCD$.

$\Rightarrow A + B(C' + AD' + ACD)$

$\Rightarrow A + B(C' + A(D' + CD))$ $[x + x'y = x + y]$

$\Rightarrow A + B(C' + A(D' + C))$

$\Rightarrow A + B(C' + AD' + AC)$ $[x + x'y = x + y]$

$\Rightarrow A + B(C' + A + AD')$

$\Rightarrow A + B(C' + A(1 + D'))$ $[1 + A = 1]$

$\Rightarrow A + B(C' + A)$

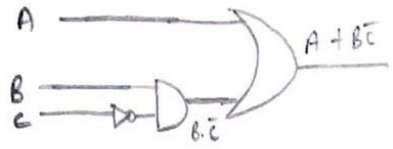
$\Rightarrow AC' + A + BC' + AB.$

$A(1 + C') + BC' + AB.$ $[1 + A = 1]$

$A + BC' + AB$

$A(1 + B) + BC'$ $[1 + A = 1]$

$A + BC'$

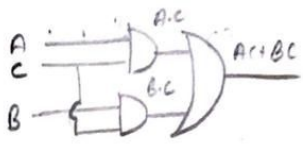


(b) $A'BC + AC$ $[x+x'y = x+y]$

$\Rightarrow C(A'B + A)$

$\Rightarrow C(A + B)$

$\Rightarrow AC + BC$



(c) $A'B + ABC' + ABC$

$\Rightarrow B(A' + AC' + AC)$

$\Rightarrow B(A' + A(C+C'))$ $[A + \bar{A} = 1]$

$\Rightarrow B(A + A')$ $[A + \bar{A} = 1]$

$\Rightarrow B$

B

(d) $(BC' + A'D)(AB' + CD')$

$\Rightarrow ABB'C' + BCC'D' + AA'B'D + A'CDD'$

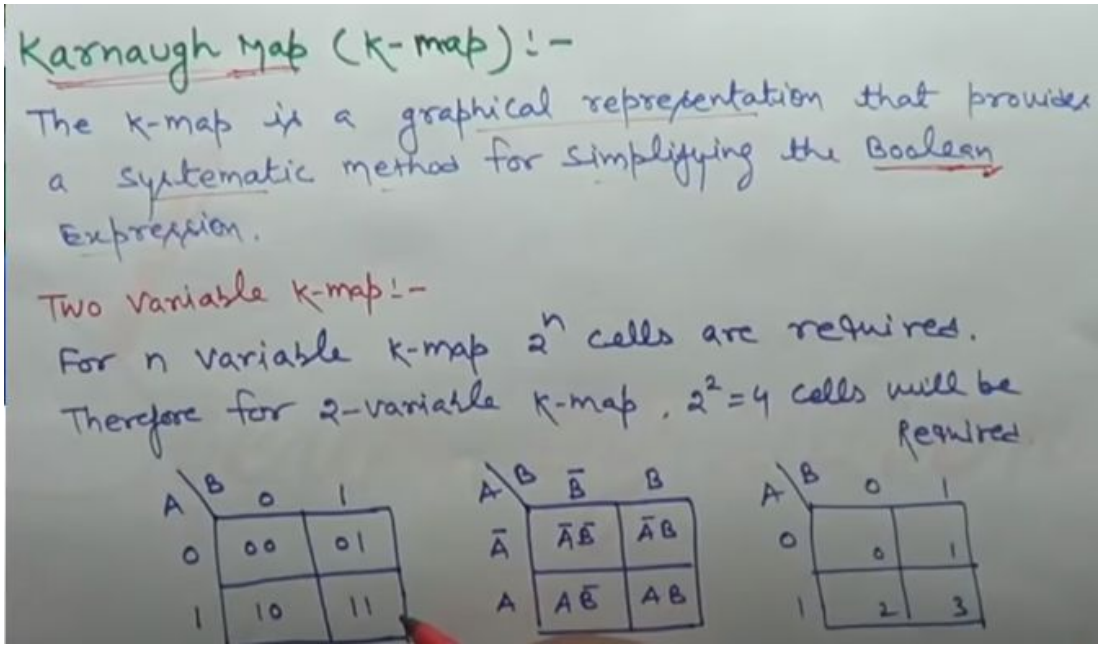
$\Rightarrow 0 + 0 + 0 + 0$

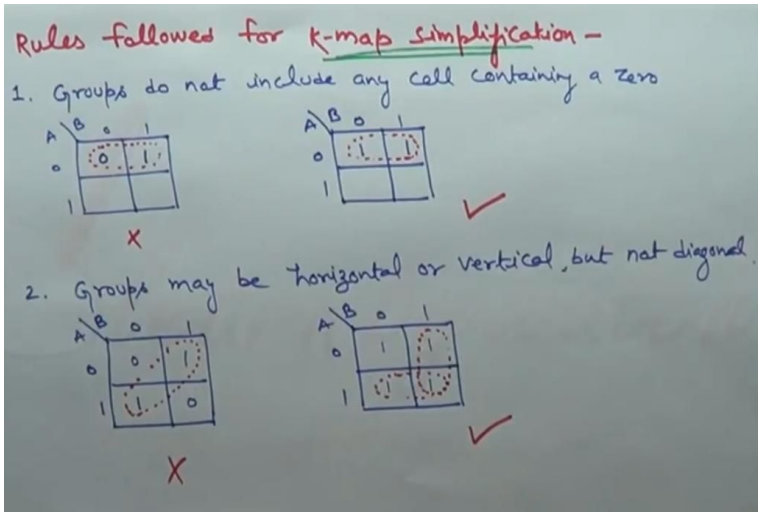
$\Rightarrow 0.$

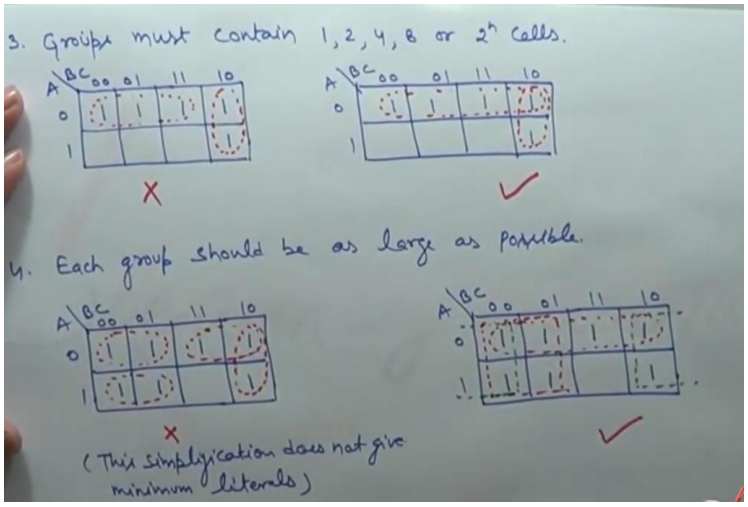
5. Write the rules for simplifying the given Boolean expression into Sum-of-Products (SoP) form using K-Maps.

6. Write the rules for simplifying the given Boolean expression into Product-of-Sums (PoS) form using K-Maps.

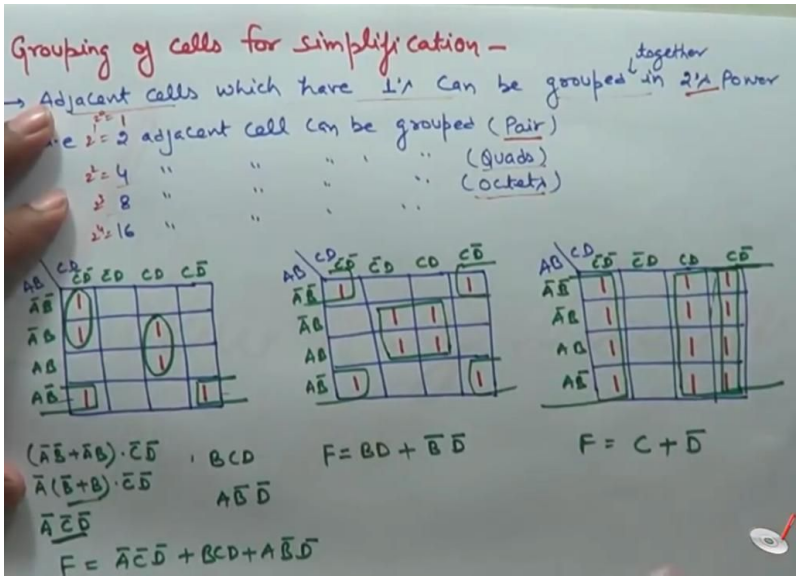
K-MAPS:



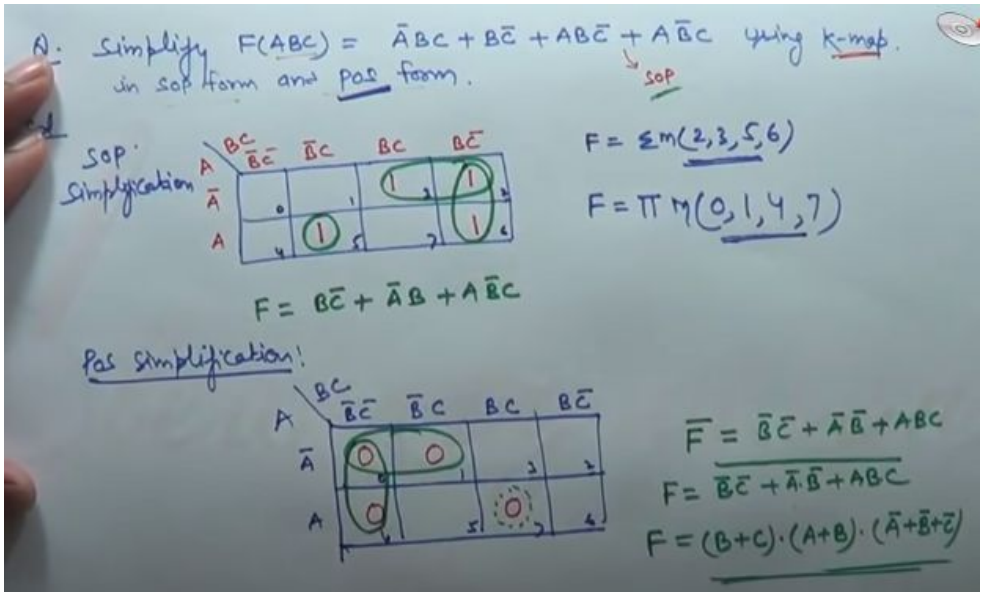




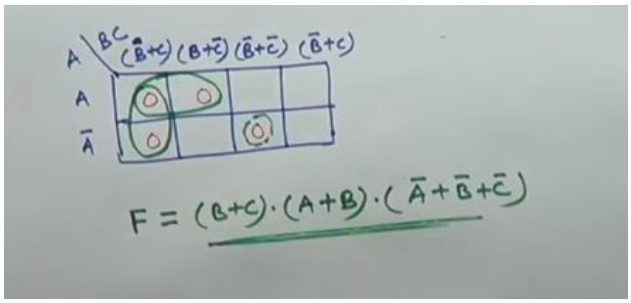
SUM OF PRODUCTS:

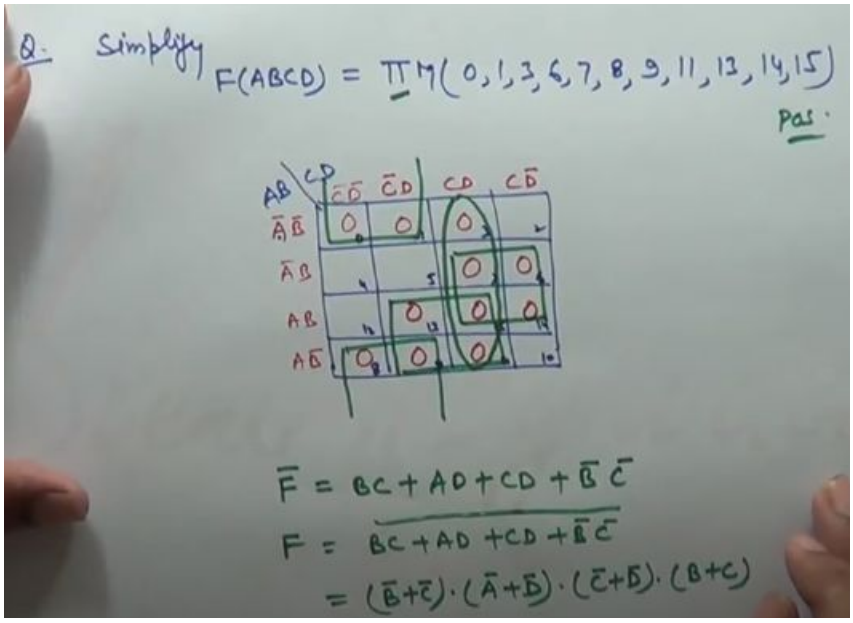


PRODUCT OF SUMS:



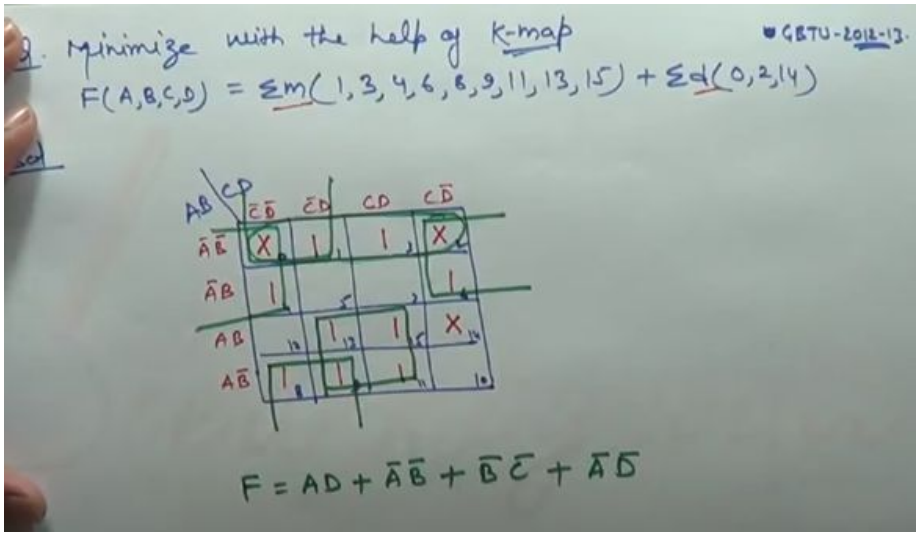
(OR)





DON'T CARE CONDITION:

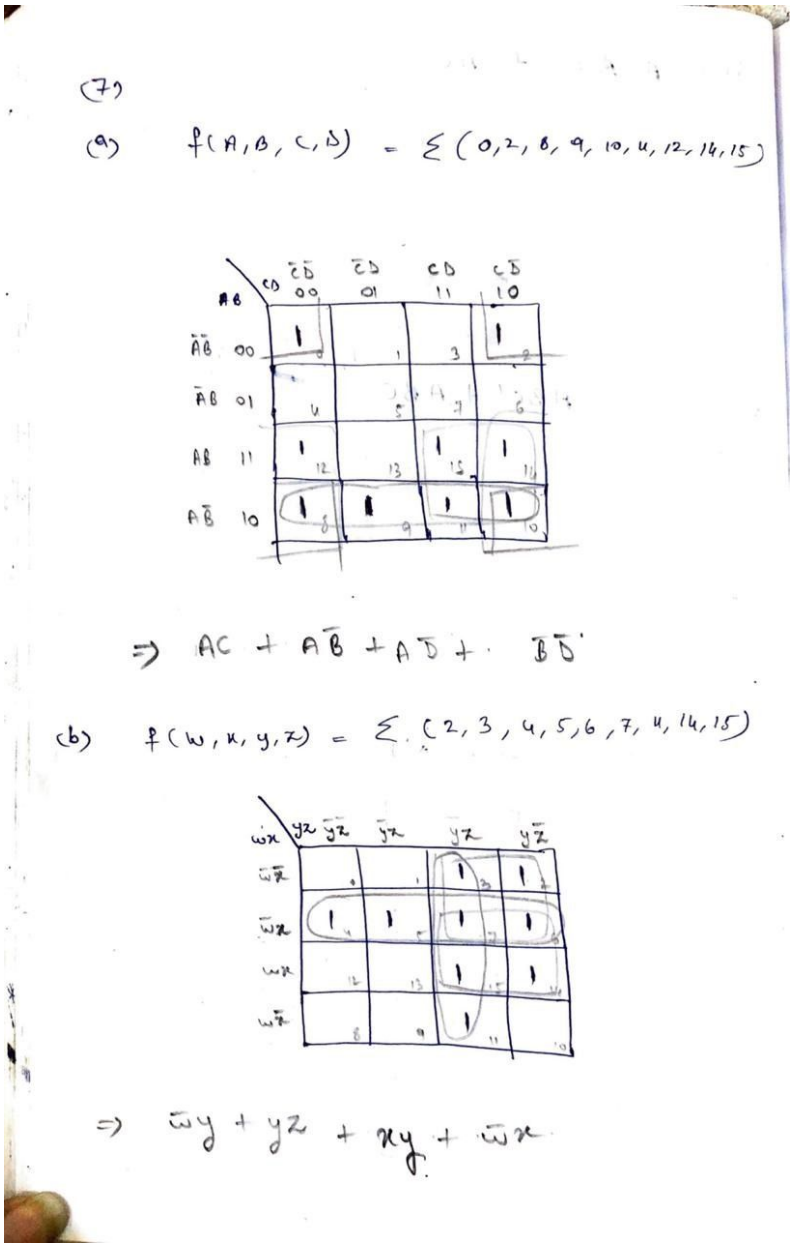
Don't care is represented as x. It can be grouped with either 0 (POS) or 1 (SOP). It is not necessary to use it, only use it if the group becomes bigger.



7. Simplify the following function into Sum-of-Products form and draw the corresponding Logic Diagram.

a. f (A, B, C, D)= ∑ (0,2,8,9,10,11,12,14,15)

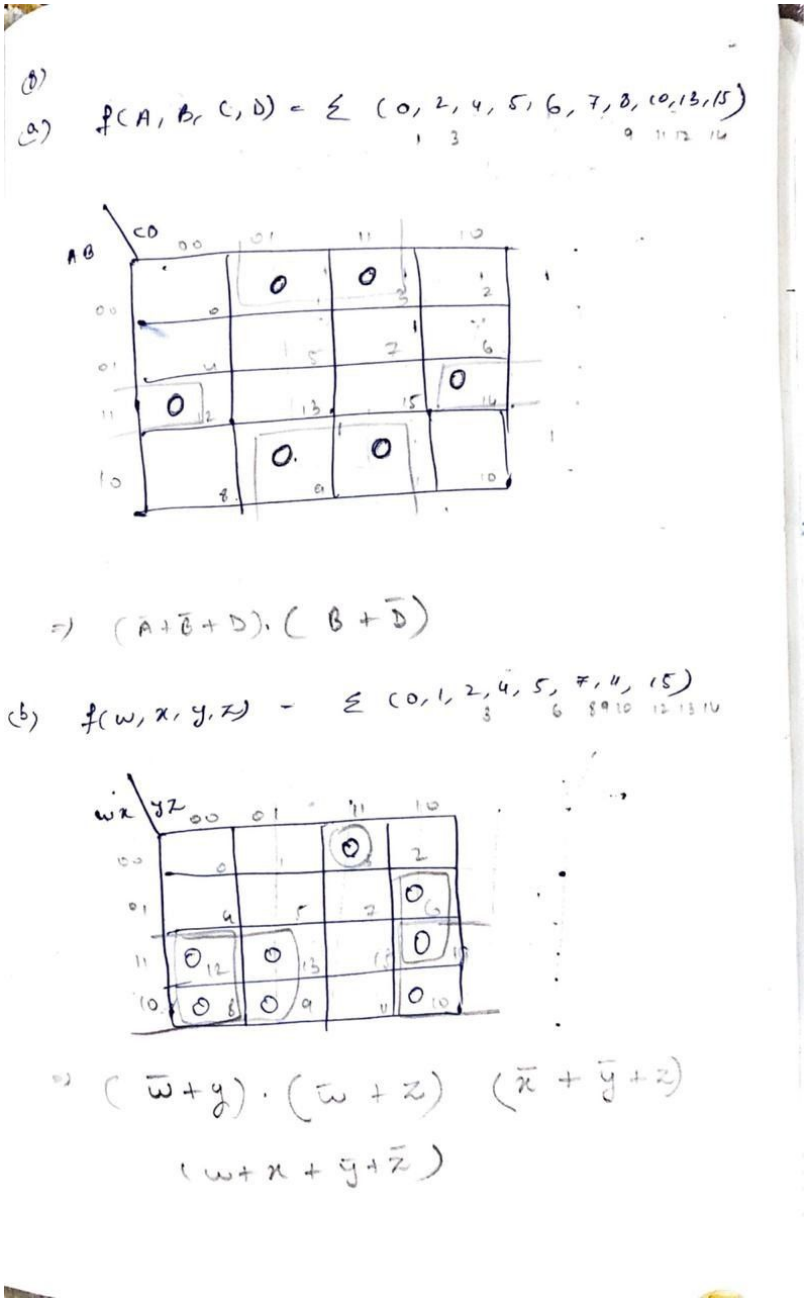
b. f (w, x, y, z)= ∑ (2,3,4,5,6,7,11,14,15)



8. Simplify the following function into Product-of-Sums form and draw the corresponding Logic Diagram.

a. f (A, B, C, D)= ∑ (0, 2, 4, 5, 6, 7, 8, 10, 13, 15)

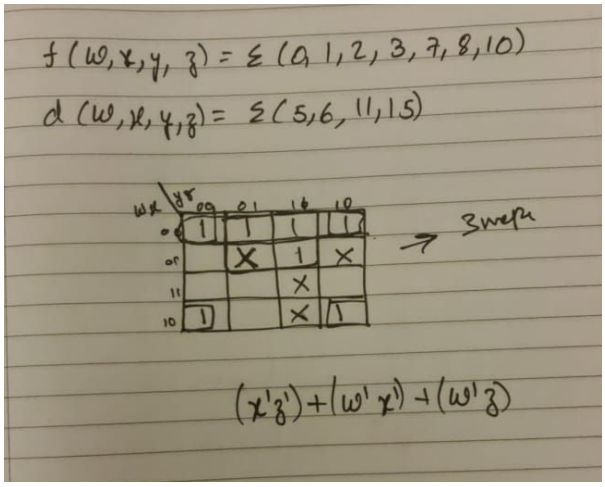
b. f (w, x, y, z)= ∑ (0, 1, 2, 4, 5, 7, 11, 15)

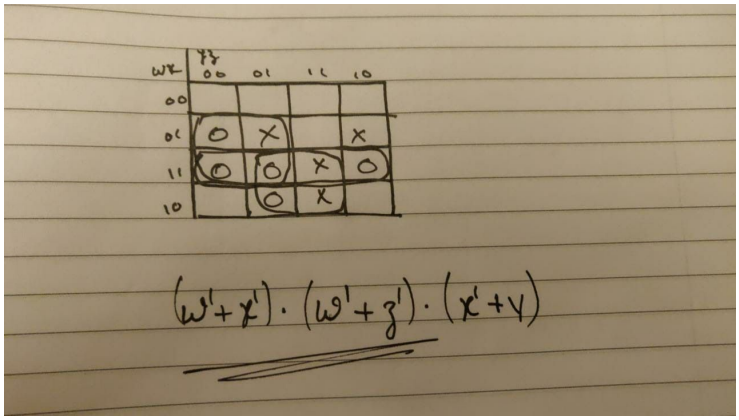


9. Simplify the following function into Sum-of-Products form and Product-of-Sums form using Don't-Care Conditions.

f (w, x, y, z)= ∑ (0, 1, 2, 3, 7, 8, 10)

d (w, x, y, z)= ∑ (5, 6, 11, 15)





10. Write the procedure for designing a combinational circuit.

Combinational circuits.

The inter connection of various logic gates to achieve certain functionality.

Features of combinational Circuits.

1) combinational circuits take 'm' no. of inputs & 'n' no. of outputs
2) The output doesn't depend upon the previous state of the circuit
3) They don't have memory & the circuits can have one or multiple outputs.

Analysis

Analysis of a combinational circuit involves deriving a simplified boolean expression from the given logic diagram.

Design:

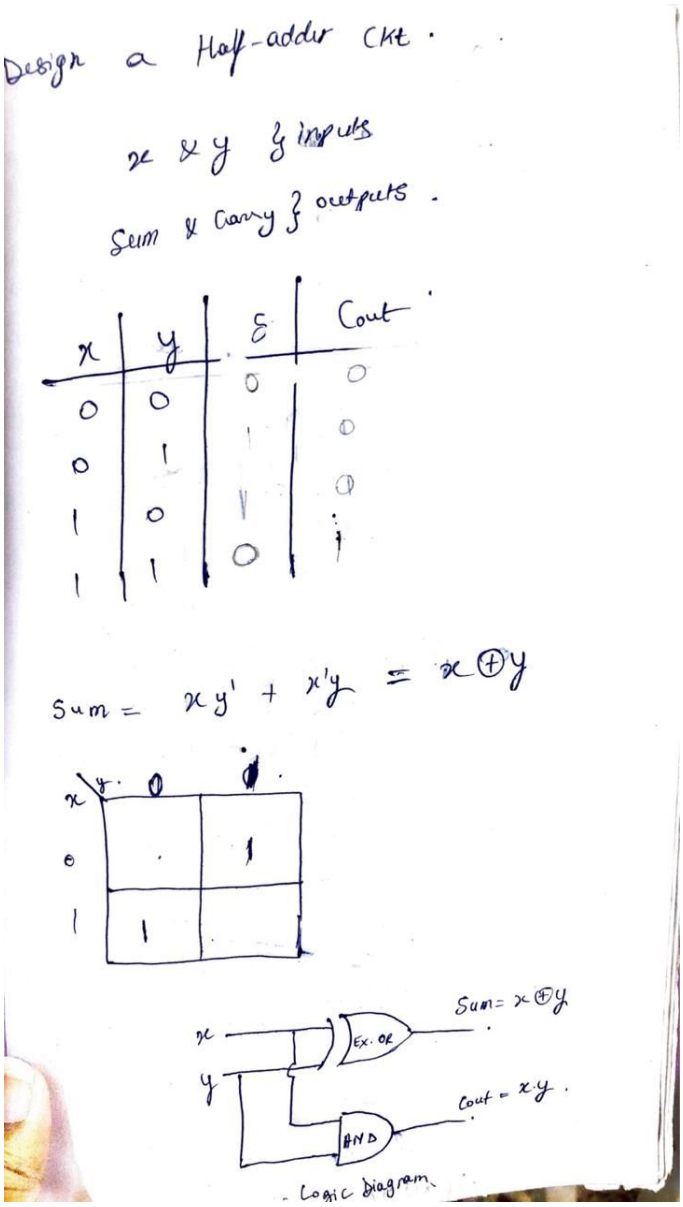
Design is the process of expressing the given task in the form of a logic diagram.

Design Process for a combinational ckt

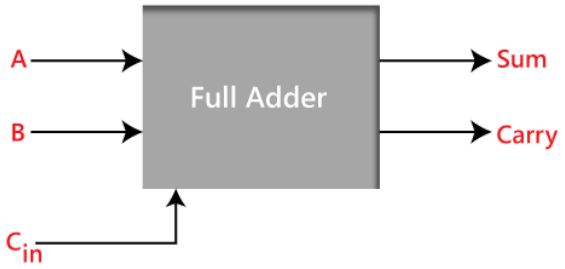
1) Analyse/understand the problem statement
2) Take the required no. of variables & name them.
3) Draw the truth table by taking these variables as inputs & represent the outputs depending on the task or functionality.
4) Simplify the truth table using either k-maps or boolean laws.

11. Design the combinational circuit for

a. Half-Adder - The Half-Adder is a combinational circuit of adding two numbers as two inputs and produce out two outputs. The adder is used to perform OR operation of two single bit binary numbers. The x and y bits are two input states, and 'carry' cout and 'sum' s are two output states of the half adder.



b. Full-Adder - The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers x, y, and carry Cin. The full adder has three input states and two output states i.e., sum s and carry cout.

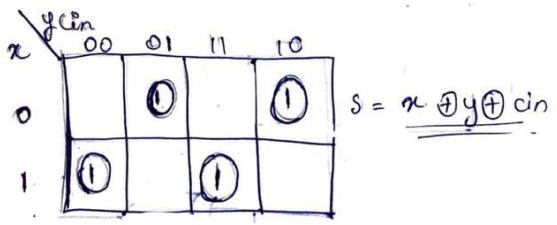


Full adder

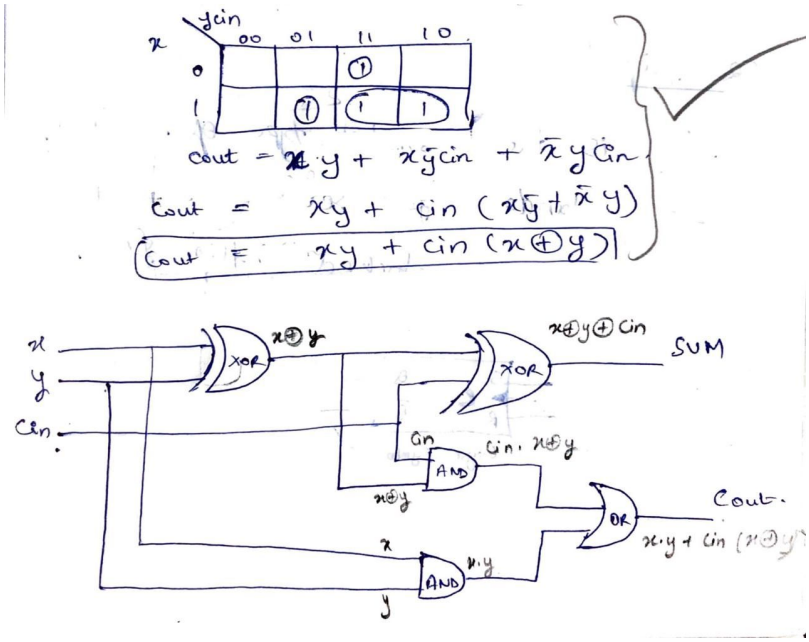
x, y, cin } i/p

S, Cout } o/p

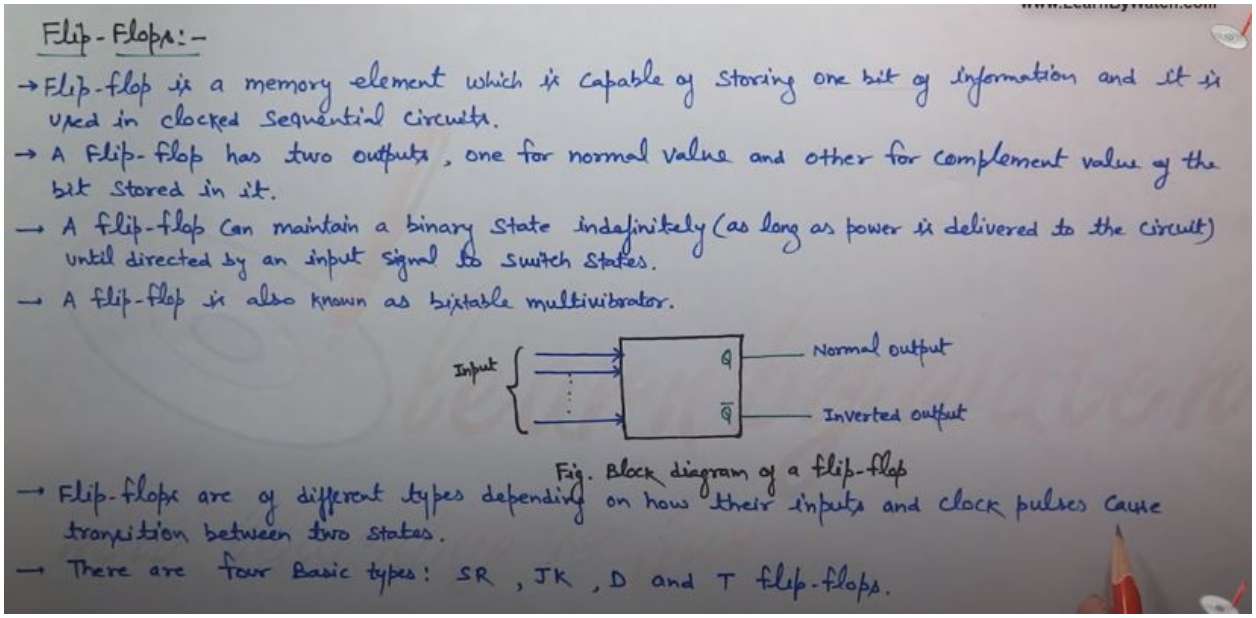
x	y	cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$S = x \oplus y \oplus cin$



12. Explain the operation of the following flip-flops.



Flip Flops

→ A flip flop is a binary cell capable of storing one (1) bit of info.

→ It has 2 o/p's, one for normal value & other for complem

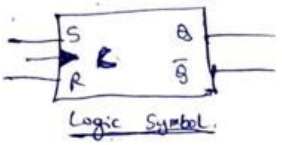
→ A clock signal directs the change of state of flip-flop.

- SR flip flop
- D flip-flop
- JK flip-flop
- T flip-flop.

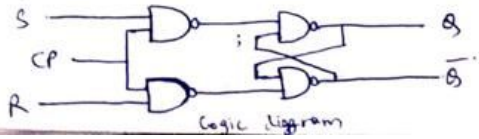
a. SR-Flip-flop

SR flip-flop

* It is the simplest type of flip flop.
* It stands for set Reset flip flop.
* it is a clocked flip flop.



Logic Symbol.



Logic Diagram

Truth table / characteristics

I/P			O/P.
S	R	Q_n (present state)	Q_{n+1} (Next state)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	indeterminate
1	1	1	indeterminate

→ if $S = 0$ & $R = 0$, the next state of SR flip-flop is same as present state.

→ if $S = 0$ & $R = 1$, the next state will be 0.

→ if $S = 1$ & $R = 0$, next state will be 1

→ when both $S = R = 1$, the next state will be indeterminate.

Reduced truth table / characteristic table

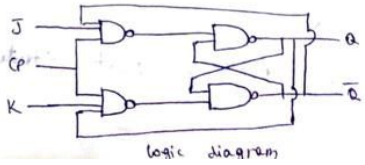
I/P			O/P	Remarks
S	R	Q_n	Q_{n+1}	States & conditions
0	0	X	Q_n	Hold state condition $S = R = 0$
0	1	X	0	Reset state condition $S = 0$, $R = 1$
1	0	X	1	Set state condition $S = 1$, $R = 0$
1	1	X	indeterminate	Indeterminate state condition $S = R = 1$

b. JK- Flip-flop

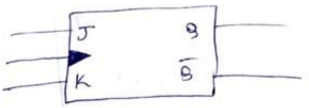
JK flip-flop

→ Refine & improved version of SR flip flop.

→ introduced to solve the problem of indeterminate state in SR flip flop when both S = R = 1.



logic diagram



logic symbol

I/P			O/P
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

→ when J = K = 0, the next state is same as present state of the flip-flop.

→ when J = 0 & K = 1, the next state is 0.

→ when J = 1 & K = 0, $Q_{n+1} = 1$

→ when J = K = 1, $Q_{n+1} = \overline{Q_n}$

I/P			O/P	Remark
J	K	Q_n	Q_{n+1}	States & Condition
0	0	X	Q_n	Hold state condition. J=K=0
0	1	X	0	Reset State condition J=0, K=1
1	0	X	1	Set state condition J=1, K=0.
1	1	X	Q'_n	

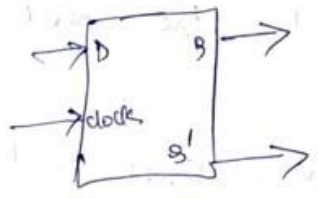
c. D- Flip-flop

D flip-flop

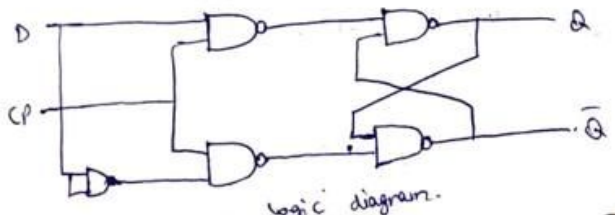
→ D (Data) flip flop.

→ A slight modification of SR flip-flop

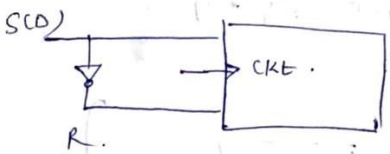
→ Inserting an inverter b/w S & R



D	Q(t+1)	
0	0	Clear to 0
1	1	Set to 1



Logic diagram.



S	R
1	0
0	1

It is same as SR flip flop, except, S & R are connected by not gate.

Truth Table:-

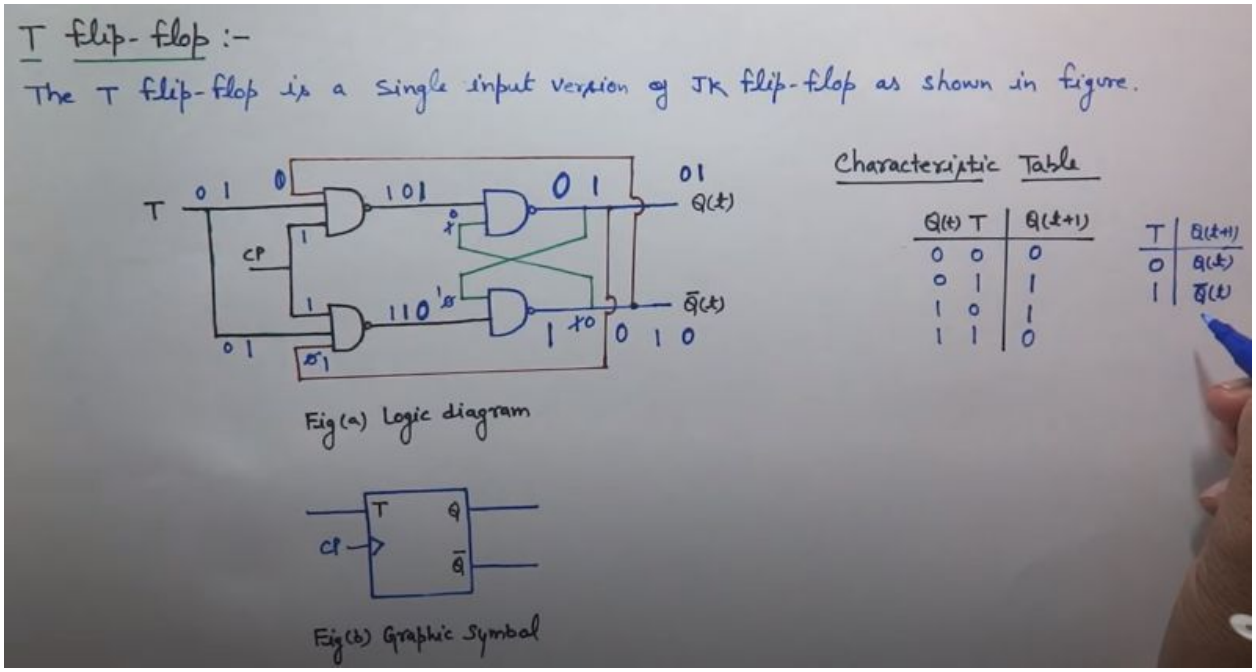
Clk	D	Q_{n+1}
0	x	Q_n
1	0	0
1	1	1

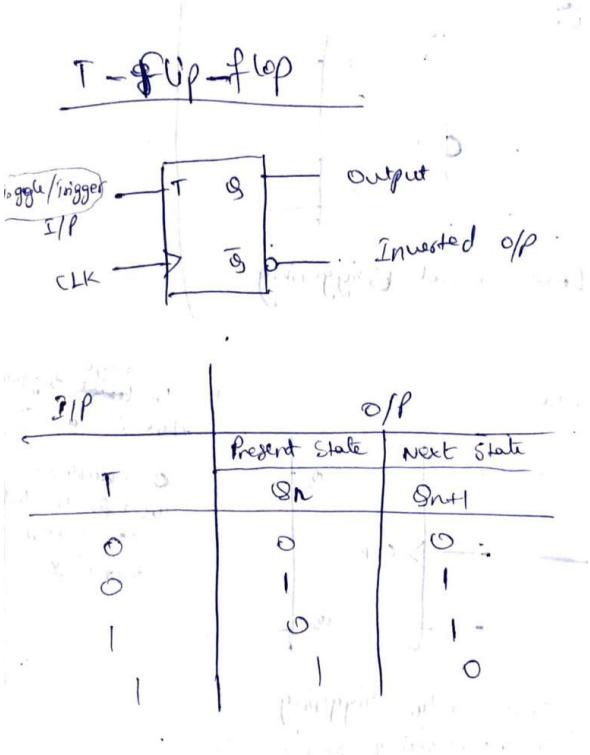
Char. Table:-

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

$Q_{n+1} = D$

d. T- Flip-flop


T flip-flop :-
The T flip-flop is a single input version of JK flip-flop as shown in figure.
T
CP
Q(t)
$\bar{Q}(t)$
Characteristic Table
Q(t) T | Q(t+1)
0 0 | 0
0 1 | 1
1 0 | 1
1 1 | 0
T | Q(t+1)
0 | Q(t)
1 | $\bar{Q}(t)$
Fig(a) Logic diagram
T
CP
Q
$\bar{Q}$
Fig(b) Graphic Symbol


T - flip-flop
Toggle/trigger I/P
CLK
T
Q
$\bar{Q}$
Output
Inverted o/p
I/P
T
O/P
Present State
Q_n
Next State
Q_{n+1}
0 0 0
0 1 1
1 0 1
1 1 0

13. Draw and explain the excitation tables of the following flip-flops.

e. SR-Flip-flop

f. JK- Flip-flop

g. T- Flip-flop

h. D- Flip-flop

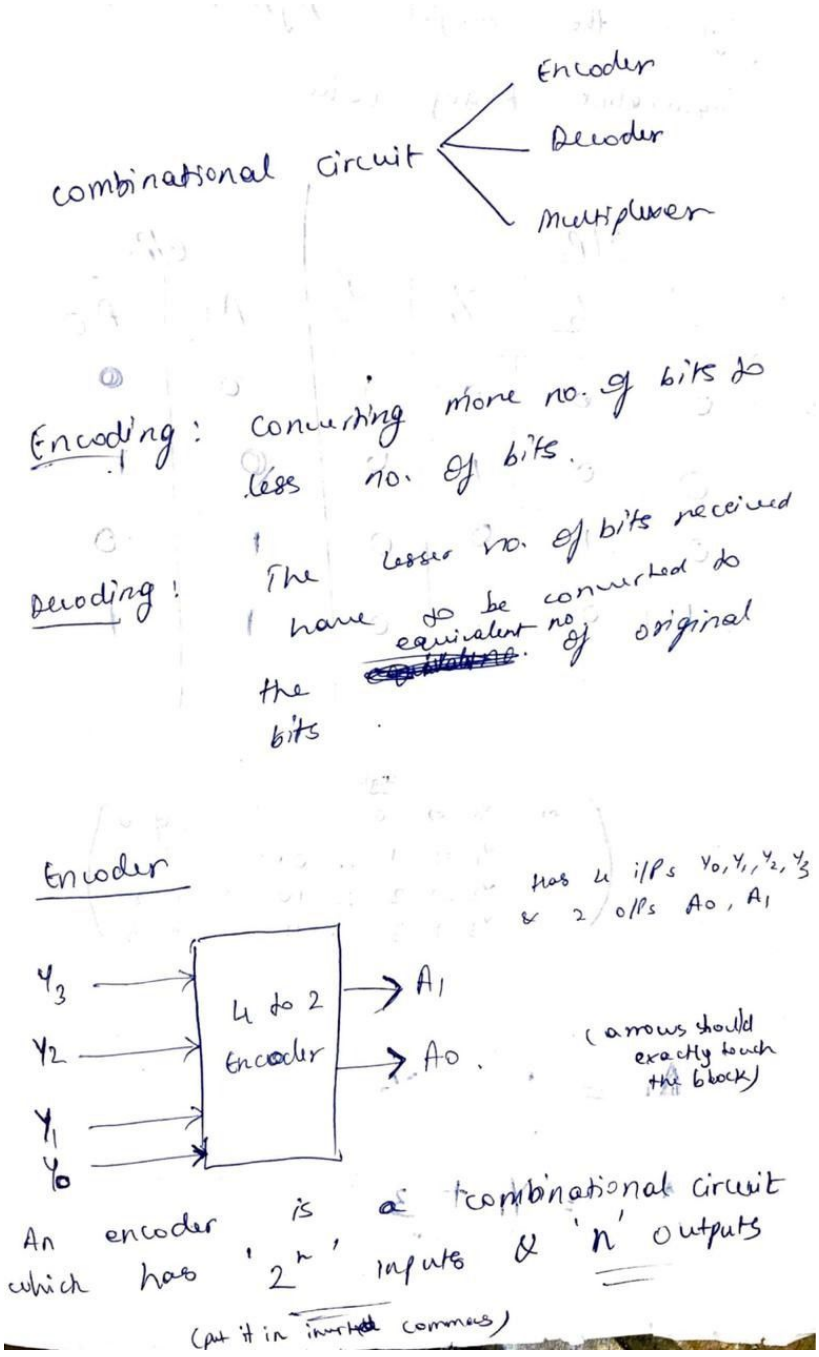
SR Flip-flop			
Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

D Flip-flop		
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

JK flip-flop			
Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

T flip-flop		
Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

ENCODER:


combinational circuit
Encoder
Decoder
Multiplexer
Encoding : Converting more no. of bits to less no. of bits.
Decoding : The lesser no. of bits received have to be converted to equivalent no. of original the bits
Encoder
has 4 i/Ps Y_0, Y_1, Y_2, Y_3 & 2 o/Ps A_0, A_1
Y_3
Y_2
Y_1
Y_0
4 to 2 Encoder
A_1
A_0.
(arrows should exactly touch the block)
An encoder is a combinational circuit which has '2^n' inputs & 'n' outputs
(put it in inverted commas)

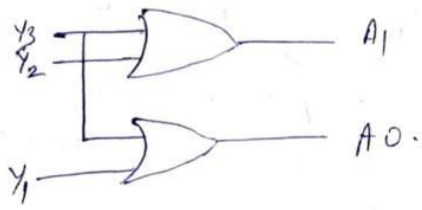
Depending on which i/p is active high. the output represents the equivalent binary value.

I/Ps				O/Ps	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

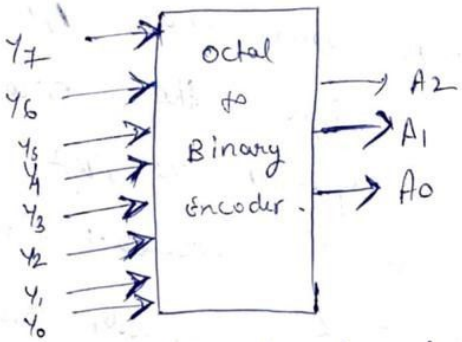
(For Y_0 => 0 (High) ∴ 00 → binary of 0
Y_1 => 1 ∴ 01 " " 1
Y_2 => 2 ∴ 10 " " 2
Y_3 => 3 ∴ 11 " " 3)

$$A_1 = Y_2 + Y_3$$

$$A_0 = Y_1 + Y_3$$



Octal to binary encoder.



Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A
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	0	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

DRAWBACKS OF ENCODER:

Drawbacks of encoder.

(If none of the inputs are high, o/p's are zeroes)

→ When all outputs are zeroes, there is an ambiguity with encoder whether the 0th i/p is active high or all the i/p are zeroes

→ If more more than one i/p is active high, the encoder doesn't represent the proper output.

To overcome this drawback there is another type of encoder known as priority encoder which allocates assigning priorities to the i/ps.

If both Y6 & Y7 are active high, the o/p is going to be 110 when Y6 is given higher priority.

combination ckt → uses logic gates

Sequential ckt → flip flops

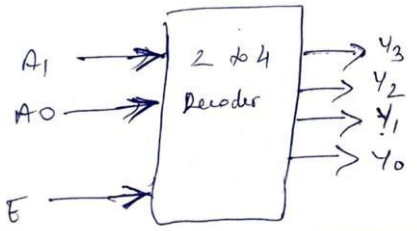
DECODER:

Decoder.

→ It is a combinational ckt. that performs the reverse operation of encoder.

→ It has 'n' input lines & max. of '2^n' output lines.

→ One of these outputs will be active high based on the combination of inputs present, when the decoder is enabled



Enable	I/P s		Outputs.			
E	A1	A0	Y3	Y2	Y1	Y0.
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0.
1	1	1	1	0	0	0

$Y_0 = EA_1'A_0'$

$Y_1 = EA_1'A_0$

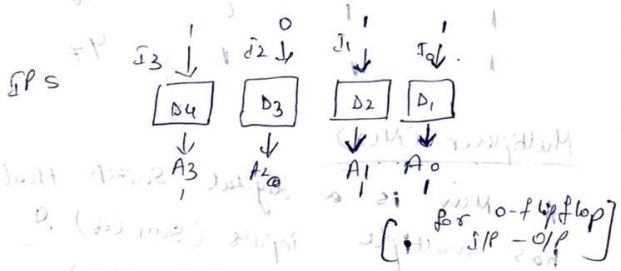
$Y_2 = EA_1A_0$

$Y_3 = EA_1A_0$.

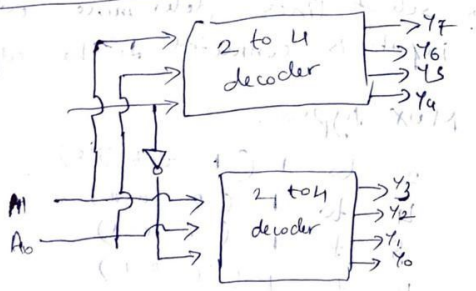
→ For decoder if there are n inputs, max no. of outputs are 2^n.

$Y_0 = E A_1' A_0'$ $\quad Y_1 = E A_1' A_0$

$Y_2 = E A_0' A_1$ $\quad Y_3 = E A_1 A_0$



construction of octal to binary decoded.



$A_2 = E$	A_1	A_0	O/P
0	0	0	Y_0
0	0	1	Y_1
0	1	0	Y_2
0	1	1	Y_3
1	0	0	Y_4
1	0	1	Y_5
1	1	0	Y_6
1	1	1	Y_7

MULTIPLEXER:

Multiplexer (MUX)

A MUX is a digital switch that has multiple inputs (sources) & a single output (destination)

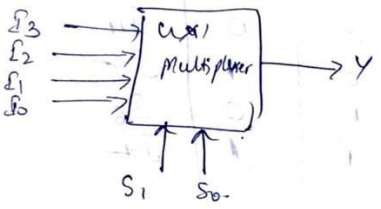
The select lines determine which input is connected to the output

→ MUX types.

2 to 1 (1 select line)
4 to 1 (2 " ")
8 to 1 (3 " ")
10 to 1 (4 " ")

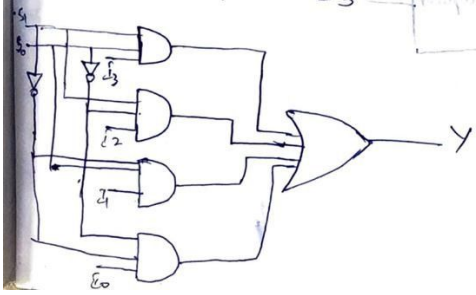
→ It is a combinational ckt that has max. of 2^n data inputs, 'n' selection lines & single o/p line

4×1 multiplexer.



$Y = S_1' S_0' I_0 + S_1' S_0 I_1 + S_1 S_0' I_2 + S_1 S_0 I_3$

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



8×1 Multiplexer

S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7