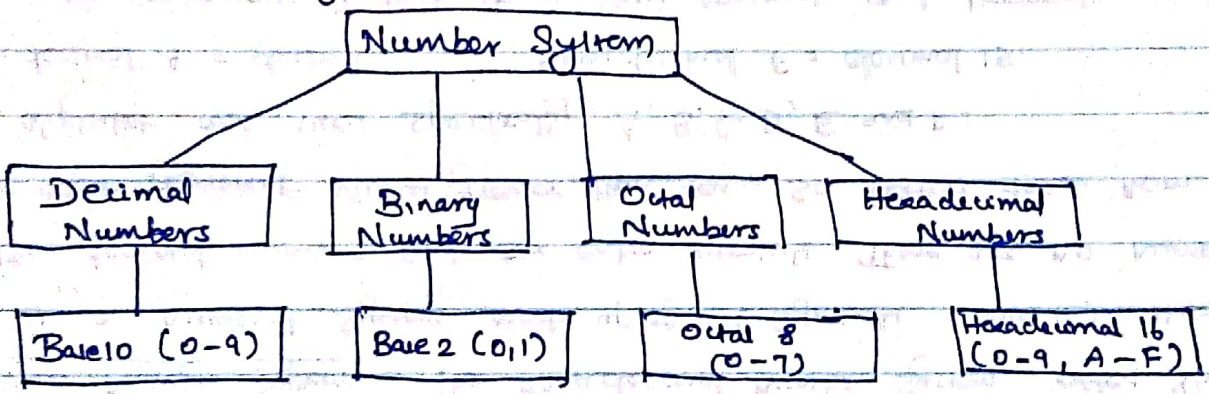
Digital Systems :- Digital Systems are designed to store, process and communicate information in digital form.

Digital system design is a process of designing or developing systems which represent information using a binary system. It is easier to store, reproduce, transmit and manipulate digital data & cheaper / easier to design such systems. (eg). Microcontrollers, Microprocessors, Memory Chips, FPGA are eg. of digital IC design. Systems like PC can be built using these components is called Digital systems.

Number System :- In digital electronics, the number system is used for representing the information. The number system has different bases and the most common of them are the decimal, binary, octal and hexadecimal.

→ The base or radix of the number system is the total number of ~~the digit~~ numerals used in the number system.



Decimal Number System :- → 10 numerals from 0 to 9.

→ Each decimal numeral in the decimal number has some value which depends upon its position.

→ That is each numeral is multiplied by weighting factor 10^x, where x is the position of the numeral from the right most number or from the decimal point.

(eg) 5432 has to be written as, $5\times10^3 + 4\times10^2 + 3\times10^1 + 2\times10^0$

→ Similarly, 5432.167 has to be written as $5\times10^3 + 4\times10^2 + 3\times10^1 + 2\times10^0 + 1\times10^{-1} + 6\times10^{-2} + 7\times10^{-3}$.

→ Each numeral in the right side of the decimal point is multiplied by 10^{-x}, where x is the position of the numeral from the decimal point towards the right side.

Octal Number System:- The octal numeral System, or oct for short is the base-8 number System, and uses the digits 0 to 7. Octal numerals can be made from binary numerals by grouping consecutive binary digits into groups of three (starting from the right)

→ The main advantage of using octal number System is that it can be converted directly to binary in a very easy manner. (and also the vice versa)

→ because computer understands only the binary System.

→ The major disadvantage of the octal number System is that the computer doesn't understand the octal number System. Hence additional Circuit is required for the digital systems which convert the octal number to binary number. ~~This is~~ The octal number System is used in a minicomputer.

Hexadecimal Number System:- The Hexadecimal number System, often Shortened to "hex" is a numeral System made up of 16 Symbols. (base 16). Hexadecimal uses the decimal numbers and six extra symbols. There are no numerical Symbols that represent values greater than ten., So letters taken from the English alphabet are used specifically A, B, C, D, E and F.

Hexadecimal A = decimal 10, Hexadecimal F = decimal 15.

→ The main advantage is that it is very compact and ~~by using a base of~~ the number of digits used to represent a given number is usually less than in binary or decimal. Also it is quick and easy to convert between hexadecimal numbers and binary.

→ Hexadecimal numbers are sometimes written with a "h" after the number. For eg. 63h means 63 hexadecimal. Software developers quite often use 0x before the number. (0x63)

→ Initially only decimal number system. Computers however only have on and off called a binary digit. In earlier days such as 1960's, ~~the experts~~ grouped 3 bits. Three bits each being on and off can represent the eight numbers. This is called octal. As computer got bigger, it was more convenient to group bits by four instead of three. ~~This doubles the number~~ This is called hexadecimal. Four bits is called a nibble (nybble). A nibble is one hexadecimal digit and it is written using a Symbol 0–9 or A–F. Two nibbles is a

byte. (8 bits). Most computer operations use the byte, or multiple of the byte (16 bits, 24, 32, 64 etc), Hexa decimal makes it easier to write these large binary numbers.]

Decimal to octal conversion:- 1) convert the decimal number $(243)_{10}$ into octal number. (repeated division and remainder algorithm)

1) 8|243
8|30 – 3 LSB
MSB 3 – 6

$(243)_{10} = (363)_8$.

2) $(0.625)_{10}$

Integral part. → 8|0
0 – 0

So Integral part

$(0)_{10} = (0)_8$.

$0.625 \times 8 = 5.0$

$0.0 \times 8 = 0.0$ $(0.50)_8$.

Octal equivalent of $(0.625)_{10} = (0.50)_8$.

(3) 0.015625 $(0)_{10} = (0)_8$.

$0.015625 \times 8 = 0.125$

Integral part = 0 fractional part = 0.125.

As, fractional part is not equal to 0, go to next step.

Step 2 $0.125 \times 8 = 1.000$

Integral part : 1 fractional part = 0.

Here fractional part is 0. hence stop here.

step 1 = 0

Step 2 = 1 $\therefore (0.015625)_{10} = (0.01)_8$.

(4) convert decimal number 7.16 into octal form.

8|7
0 – 7 1st remainder is 7.

$0.16 \times 8 = 1.28$.

Integral part 1 ✓

Step 2 $0.28 \times 8 = 2.24$.

Integral part = 2. ✓

Step 3 $0.24 \times 8 = 1.92$ ✓

Integral part = 1.

Step 4 $0.92 \times 8 = 7.36$. ✓

Integral part = 7.

Step 5 $0.36 \times 8 = 2.88$ ✓

$0.88 \times 8 = 7.04$.

Integral part = 7. ✓

fractional part .04.

In this case, we have 5 digits as answer and the fractional part is still not 0 so, we stop here.

$(7)_{10} = (7)_8$

$(0.16)_{10} = (0.12172..)_8$

$(7.12172)_8$.

$\therefore (7.16)_{10} = (7.12172\ldots)_8$.

(or) $(7.16)_{10} = (7.12172)_8$ (approx)

(Eg) 1) $(98)_{10}$ (Ans : $(142)_8$)

2) $(210)_{10}$ (Ans : $(322)_8$)

3) $(0.140869140625)_{10}$ (Ans : $(0.11010)_8$)

4) $(153.513)_{10} = (231.4065)_8$

8|210
8|26 – 2 LSB
MSB 3 – 2

$(322)_8$.

8|153
8|19 – 1
2 – 3

Convert octal number to decimal number.

1) $(363)_8$

3 LSB
6
3 MSB.

$3 \times 8^0 = 3 \times 1 = 3$

$6 \times 8^1 = 6 \times 8 = 48$

$3 \times 8^2 = 3 \times 64 = 192$

243

$(363)_8 = (243)_{10}$.

2) $(0.01)_8$

$1 \times 8^{-2} = 0.015625$

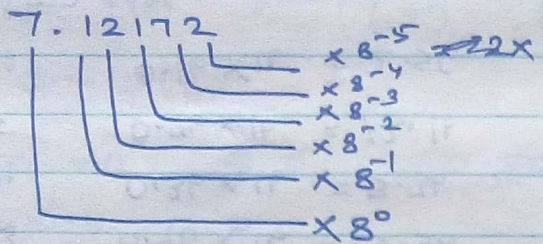
$0 \times 8^{-1} = 0$

$0 \times 8^0 = 0$

0.015625

$(0.01)_8 = (0.015625)_{10}$

3) $(7.12172)_8$. Convert into decimal form.



$= 7\times8^{0} + 1\times8^{-1} + 2\times8^{-2} + 1\times8^{-3} + 7\times8^{-4} + 2\times8^{-5}$

$= 7 + 0.125 + 0.03125 + 0.001953125 + 0.001703984375 +$
$0.000061035156 24$

$= 7.15997314454$

$= (7.16)$ approx. 7.16 approx

$\therefore (7.12172)_8 = (7.16)_{10}$. $(7.16)_{10}$ approx.

(eg:1) $(0.50)_8 = (0.625)_{10}$

2) $(26.24)_8 = (22.3125)_{10}$

DECIMAL TO HEXA DECIMAL

1) $(243)_{10}$ - convert to hexadecimal number.

16|243

16|15 – 3 LSB.

F = 15. ∴ F3

$(243)_{10} = (F3)_{16}$.

16×5 = 80
16×1 = 48
128
1×1 = 16
144

7×16 = 112

2) $(14)_{10}$ 14 = E.

$(14)_{10} = (E)_{16}$

3) $(80)_{10}$. 16|80
5 – 0

$(50)_{16}$.

4) $(3000)_{10}$ 16|3000
16|187 – 8
11 – 11

$(3000)_{10} = (BB8)_{16}$

A – 10
B – 11

Decimal Fractional to Hexa decimal:

1) $(0.625)_{10}$ $\quad 0.625 \times 16 = 10.0 \simeq A$.

$(0)_{10} = (0)_{16}$. $\quad 0 \times 16 = 0$

$(0.A0)_{16}$

2) $(0.00390625)_{10}$

Integral part $(0)_{10} = (0)_{16}$

Fractional part. $0.00390625 \times 16 = 0.0625$

Integral part = 0

Step 2

$0.0625 \times 16 = 1.0$

Integral part = 1

Fractional part 0. Hence stop here.

Fractional part = $(0.01)_{16}$. / $(0.00390625)_{10} = (0.01)_{16}$

3) $(10.16)_{10}$ to Hexadecimal form.

16|10 = A. Integral part = A. $(10)_{10} = (A)_{16}$. —— (1)

Fractional part.

$0.16 \times 16 = 2.56$ Integral part = 2

Step 2 $0.56 \times 16 = 8.96$ " = 8

Step 3 $0.96 \times 16 = 15.36$ " = F

Step 4 $0.36 \times 16 = 5.76$ " = 5

Step 5 $0.76 \times 16 = 12.16$ " = C

Step 6 $0.16 \times 16 = 2.56$ " = 2

Stop here.

Fractional part $(0.16)_{10} = (0.28F5C2)_{16}$ —— (2)

Combining (1) & (2) $(10.16)_{10} = (A.28F5C2)_{16}$

4) $(62.500)_{10}$

16 | 62
3 - 14 (E)

$(62)_{10} = (3E)_{16}$.

$0.500 \times 16 = 8.0$ $\quad (0.500)_{10} = (.8)_{16}$

$0 \times 16 = 0$.

$\therefore (62.500)_{10} = (3E.8)_{16}$.

Hexa decimal to decimal.

1) $(F3)_{16}$ ⇒ decimal number.

F3
- $3 \times 16^0 = 3 \times 1 = 3$
- $15 \times 16^1 = 15 \times 16 = 240$

243

$(F3)_{16} = (243)_{10}$.

2) $(64)_{16}$

64
- $4 \times 16^0 = 4 \times 1 = 4$
- $6 \times 16^1 = 6 \times 16 = 96$

100

$(64)_{16} = (100)_{10}$.

3) Hexa fraction number. $(0.A0)_{16}$ to decimal.

0.A0
- $0 \times 16^{-2} = 0 = 0$
- $A \times 16^{-1} = 10 \times \frac{1}{16} = 0.625$
- $0 \times 16^0 = 0 = 0$

0.625

$(0.A0)_{16} = (0.625)_{10}$

4) $(16.5)_{16}$

16.5

$5 \times 16^{-1} = 5 \times 0.0625 = 0.3125$

$6 \times 16^{0} = 6 \times 1 = 6.0$

$1 \times 16^{1} = 1 \times 16 = 16.0$

22.3125

$(16.5)_{16} = (22.3125)_{10}$

5) $(FAFA.B)_{16}$

FAFA.B

$11 \times 16^{-1} = 11 \times 0.0625 = 0.6875$

$10 \times 16^{0} = 10 \times 1 = 10.0$

$15 \times 16^{1} = 15 \times 16 = 240.0$

$10 \times 16^{2} = 10 \times 256 = 2560.0$

$15 \times 16^{3} = 15 \times 4096 = 61440.0$

64250.6875

$(FAFA.B)_{16} = (64250.6875)_{10}$

Binary Number System:

→ Binary number system has only two numerals (0 and 1)

→ The base is 2

→ Any number greater than 1 will be represented by the combination of 0 and 1

→ This number system is used in all digital systems including computers.

→ Each binary numeral is called a bit. A group of 4-bit is called a nibble and a group of 8-bit is called byte.

~~Need for~~ Why Binary systems... used in computers?

→ For a decimal system we have 10 ~~level~~ numerals, to represent a electrical system using a decimal number system, we require 10 levels of voltages and current.

→ It is very difficult to design such system with reliable tolerance.

→ But, Binary number System can be easily and reliably represented electrically since it requires only two states.

→ Eg) $(1011)_2$ can be easily represented by a group of four wires. In this case 3 wires will be at a potential of 5V and one wire will be at a Potential of .0V.

→ Transistors and switching devices can be easily and reliably operated in two states. That is either ON ~~or~~ state or OFF state. An ON state can be assumed as a 1 and OFF state can be assumed as a Zero.

→ These two states, will allow a wide tolerance to design parameters. These tolerances produce highly reliable digital devices.

Need for octal and Hexadecimal Systems.

→ Digital computers and digital systems use binary numbers.

→ The conversion between Octal to binary and vice versa can be done by inspection. This is applicable for Hexadecimal to binary and vice versa.

→ Therefore, any hardware Engineer can use Hexadecimal system ~~and~~ or Octal system for programming and can perform required conversion by inspection while communicate with the digital System.

DECIMAL to binary conversion

1) $(243)_{10}$

2	243	
2	121	−1 LSB
2	60	−1
2	30	−0
2	15	−0
2	7	−1
2	3	−1
	MSB. 1	−1

$(243)_{10} = (11110011)_2$

2) $(396)_{10}$

2 | 396
2 | 198 - 0 LSB.
2 | 99 - 0
2 | 49 - 1
2 | 24 - 1
2 | 12 - 0
2 | 6 - 0
2 | 3 - 0
MSB 1 - 1

$(396)_{10} = (110001100)_2$

3) $(4096)_{10}$

2 | 4096
2 | 2048 - 0 LSB.
2 | 1024 - 0
2 | 512 - 0
2 | 256 - 0
2 | 128 - 0
2 | 64 - 0
2 | 32 - 0
2 | 16 - 0
2 | 8 - 0
2 | 4 - 0
2 | 2 - 0
MSB 1 - 0

$(4096)_{10} = (1000000000000)_2$

1) $(0.625)_{10}$

$(0)_{10} = (0)_2$ → Integral part.

$0.625 \times 2 = 1.25$ Integral part = 1

Step 2 $0.25 \times 2 = 0.5$ Integ. part = 0

Step 3 $0.5 \times 2 = 1.0$ Integral : 1
$0 \times 2 = 0$: 0

Fractional zero. Stop Here.

$(0.625)_{10} = (0.1010)_2$

2) $(22.3125)_{10}$.

2 | 22
2 | 11 - 0
2 | 5 - 1
2 | 2 - 1
1 - 0

$(22)_{10} = (10110)_2$.

$0.3125 \times 2 = 0.625$ 2P = 0
$0.625 \times 2 = 1.25$ " = 1
$0.25 \times 2 = 0.5$ " = 0 $(0.1010)_2$
$0.5 \times 2 = 1.0$ R = 1
$0 \times 2 = 0$ 4 = 0.

Stop here.

Combining $(22.3125)_{10} = (10110.01010)_2$

Binary to decimal conversion

1) $(11110011)_2$.

1 1 1 1 0 0 1 1

$1 \times 2^0 = 1 \times 1 = 1$
$1 \times 2^1 = 1 \times 2 = 2$
$0 \times 2^2 = 0 = 0$
$0 \times 2^3 = 0 = 0$
$1 \times 2^4 = 1 \times 16 = 16$
$1 \times 2^5 = 1 \times 32 = 32$
$1 \times 2^6 = 1 \times 64 = 64$
$1 \times 2^7 = 1 \times 128 = 128$

243

$= (243)_{10}$

2) $(0.101)_2$ to decimal.

0.101

$1 \times 2^{-3} = 1 \times 0.125 = 0.125$

$0 \times 2^{-2} = 0 = 0.0$

$1 \times 2^{-1} = 1 \times 0.5 = 0.5$

$0 \times 2^{0} = 0 = 0.0$

0.625

3) $(11100011.1010)_2 = (243.625)_{10}$.

4) $(10110.0101)_2 = (22.3125)_{10}$

5) $(1010.1010)_2 = (10.625)_{10}$.

Binary to octal conversion.

1) $(1111\,0011.1010)_2$

1111 0011 . 1010

011 110 011 . 101 000

3 6 3 . 5 0 $= (363.50)_8$

2) $(363)_8$ to binary.

3 6 3

011 110 011 011110011

discard 0 $(11110011)_2$

Binary to hexadecimal.

1) 11110011.1010.

1111 0011 . 1010

F 3 . A.

$(F3.A)_{16}$

2) $(1010.10)_{16}$ to binary

1 0 1 0 . 1 0

0001 0000 0001 0000 . 0001 0000

$(0001\,0000\,0001\,0000.0001\,0000)_2$.

3) convert the above to octal.

0001 0000 0001 0000 . 0001 0000

0 1 0 0 2 0 . 0 4 0

$(010020.040)_8$

4) $(1.10010)_2$ to decimal. and hexa decimal.

$(1.11010)_2 = (1.8125)_{10}$

$(1.11010)_2 = (1.D)_{16}$

5) $(1110.10)_2$ to decimal and hexadecimal.

$(1110.10)_2 = (14.5)_{10}$

$(1110.10)_2 = (E.8)_{16}$

6) $(1011101.10101)_2$ to decimal number.

$(93.65625)_{10}$

7) $(1100011)_2$ to hexadecimal and octal.

<u>Ans</u> $(73)_{16}$ & $(163)_8$

Representation of numbers in various number system.

Decimal	Octal	Hexadecimal	Binary
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	10	8	1000
9	11	9	1001
10	12	A	1010
11	13	B	1011
12	14	C	1100
13	15	D	1101
14	16	E	1110
15	17	F	1111
16	20	10	10000

Complement of a Number

$(32320)_{10}$. r's complement = $r^n - N$

r = 10 (base)

n = 5 (number of integer numerals)

N = 32320 (given number)

∴ 10's complement $= 10^5 - 32320$

$= (67680)_{10}$.

2) $(0.5267)_{10}$ $= 10^0 - 0.5267 = (0.4733)_{10}$

3) $(52.749)_{10} = 10^2 - 52.749 = (47.251)_{10}$

In mathematics and computing, the method of complements is a technique to encode a symmetric range of positive and negative integers in a way that they can use the same algorithm (& hardware) for addition throughout the whole range. For a given number of places half of the pellible representations of number encode the positive numbers, the other half represents their respective additive inverses. The pairs of mutually additive inverse numbers are called complements. Thus ~~the~~ subtraction of any number is implemented by adding its complement.

→ complement of a number is natural negation of that number. Let r be the base of a number N and n be the number of numerals of the integer part. Then r^{th} complement of the number is $r^n - N$.

(r-1)'s complement of a number

(r-1)'s complement is defined as $(r^n - 1) + (1 - 10^{-m}) - N$.

where, r - base of the number system, n - number of numerals of integer part, m a the number of numerals of fractional part

N - given number.

It is nothing but subtracting each numeral from 9.

(eg) Find 9's complement of $(32320)_{10}$.

$(r^n-1) + (1-10^{-m}) - N$.

$r = 10 \quad n = 5 \quad m = 0 \quad N = 32320$

$\therefore$ 9's complement $= (10^5 - 1) + (1-1) - 32320$

$= 99999 - 32320$

$= 67679$

```
  9
32320
  9
-----
67679
-----
```

2) $(0.5267)_{10}$ $\quad r = 10$

$n = 0 \quad m = 4 \quad N = 0.5267.$

9's complement $= (10^0 - 1) + (1-10^{-4}) - 0.5267$

$= 0.9999 - 0.5267$

$= (0.4732)_{10}$

3) $(52.749)_{10} = (27.250)_{10}$.

→ From the above eg, we can note that 9's complement is subtracting each numeral from 9. The resulting number would be 9's complement. 10's complement can be obtained by adding 1 to the 9's complement.

4) Find 10's complement of the following numbers after obtaining the 9's complement.

(a) $(32320)_{10}$ — AN 67680

b) $(0.5267)_{10}$ — 0.4733

(c) $(52.749)_{10}$ — 47.251

2's complement and 1's complement

r's complement and (r-1)'s complement for binary number is 2's complement and 1's complement respectively.

① Find the 2's complement and 1's complement of $(1011011)_2$.

To find 1's complement subtract each bit from 1

```
  1 1 1 1 1 1 1
(-) 1 0 1 0 1 1
-----------------
  0 1 0 1 0 0
```

1's complement

2's complement (+) 0 0 0 0 0 1

0 1 0 1 0 1

Note! The 1's complement of the given number is bit-wise inversion of a given number. Addition of 1 to the 1's complement is 2's complement of given number.

Another way $\longrightarrow (110101)_2$

$r^n - N$.

$r = 2 \quad n = 6 \quad N = (110101)_2$

2's complement : $2^6 - 110101$

$= (64)_{10} - (110101)_2$

$(1000000)_2 - (110101)_2$

$= (001011)_2$

```
  0 0 0 0 0 0
+ 1 1 1 1 1 1
+ 0 0 0 0 0 1
-------------
  1 1 0 1 0 1
-------------
  1 1 0 1 0 1
-------------
```

```
  1 0 0 0 0 0 0
- 0 0 0 1 0 1 1
---------------
  0 0 1 0 1 1
```

1's complement = $r^n - r - m - N$

$m = 0$.

1's complement $= 2^n - 1 - 110101 = 2^6 - 1 - 110101$

$= (1000000)_2 - (1)_2 - (110101)_2$

$= (001010)_2$

② Obtain 1's and 2's complement of the following binary numbers.

		1's	2's
1)	11101010	00010101	00010110
2)	01111110	10000001	01111111
3)	00000001	11111110	11111111
4)	10000000		
5)	00000000	11111111	00000000

Subtraction 6678 - 2

= 6676.

Another way '2' complement. 9998.

6678 + 9998 = 16676

Ignore the overflow, that is 1

= 6676 //

1) ~~18 - 24~~ Perform 9's complement and 10's complement Subtraction between 18 and 24.

Ans : 18 - 24

9's complement of 24 = 75

9's complement Subtraction 18+75 = 93, This is 9's complement of -6

10's complement of 24 = 75+1 = 76.

10's complement subtraction = 18+76 = 94. This is 10's of -6

Binary Arithmetic

→ Binary arithmetic has to be performed by all digital systems and computers. Hence it is necessary for an embedded system Engineer.

→ Binary addition, subtraction, multiplication and division.

Binary Addition :- There are four possible cases in the binary bit-wise addition

A+B	Carry	Sum
0+0	0	0
0+1	0	1
1+0	0	1
1+1	1	0

Addition of three bits.

0+0+0 = 00 (carry = 0, sum = 0)

0+0+1 = 01 (carry = 0, sum = 1)

0 + 1 + 1 = 10 (carry = 1, sum = 0)

1 + 1 + 1 = 11 (carry = 1, sum = 1)

(9.) Add the following binary number A and B.

1) A = 1111 B = 1111

```
  1111
  1111
 -----
 11110
 -----
```

2) A = 1000 B = 100

```
  1000
   100
 ------
  1100
 ------
```

c) A = 1111 B = 111

```
  1111
   111
 -----
 10110
 -----
```

d) A = 1110 B = 100

```
  1110
   100
 -----
 10010
 -----
```

Note! The carry of addition of first columns is added with the second column of numerals. While adding second column, the number of bits to be added becomes three including the carry bit. This is true for the third and fourth columns also.

Binary Subtraction: The four operation for subtraction is possible.

0 − 0	0 − 0
0 − 1	1 − 1
1 − 0	1 − 0
1 − 1	0 − 1

→ In the first three case, we have to subtract a number from a bigger or equal number but in the fourth case we have to subtract a bigger number from a smaller number.

```
  0        1        1        0        1
- 0      - 1      - 0      - 1      - 1
 ---      ---      ---      ---      ---
  0        0        1      Not possible  1
 ---      ---      ---                  ---
```

to (1 is borrowed from L.C.)

(9) Do Subtraction operation:-

(a) 111 − 011 b) 111 − 10 c) 110 − 011

```
  111
  011
 ----
  100
 ----
```

```
  111
  010
 ----
  100
 ----
```

```
  110
  011
 ----
  011
 ----
```

Subtrahend
Minuend.

Binary multiplication

Four basic operations

0 × 0 = 0

0 × 1 = 0

1 × 0 = 0

1 × 1 = 1

(a) 1010100 × 0111100

(b) 1000011 × 0101011

(a) 1010100 × 0111100 + carry if any = 0010001

(b) 1000011 × 0101011 + carry if any = 1101110

(a) x = 1010100

Y = 1000011

a) x − Y

b) Y − x using 1's complement

a) 0010001

b) 1101110

Multi-bit multiplication of binary numbers is identical to multiplication in decimal numbers. Partial products for each numeral will be obtained and added together after doing successive shift left operations as explained for binary number multiplication in the following example.

1) 11 × 11

```
  11
 x11
 ---
  11
 11
 ---
 1001
 ---
```

2) 111 × 101 = 100011

```
  111
 x101
 ----
  111
 000
111
----
100011
----
```

Subtraction

```
 111101
-100101
--------
 011000
--------
```

```
 1100 1010
 1001 1011
-----------
 0010 1111
-----------
```

```
  1001
    10
  ----
  1111
```

```
  9
- 2
---
  7
---
```

Binary division - Binary division is identical to decimal division.

(1) 1111 / 11

(2) 1100 / 100

(f) 1111 / 11

7)
$$\begin{array}{r} 101 \\ 11\overline{)\,1111} \\ 11 \\ \hline 011 \\ 11 \\ \hline 0 \end{array}$$

Ans = 101

Note that

$(1111)_2 = (15)_{10}$ &

$(11)_2 = (3)_{10}$

$\frac{15}{3} = (5)_{10} = (101)_2$.

8)
$$\begin{array}{r} 11 \\ 10\overline{)\,110} \\ 10 \\ \hline 10 \\ 10 \\ \hline 0 \end{array}$$

Ans = 11

Hint

$(110)_2 = (6)_{10}$

$(10)_2 = 2$

$\frac{6}{2} = (3)_{10} = (11)_2$.

1's complement Subtraction :-

→ Subtraction of a binary number from another binary number is similar to addition of binary number with 1's complement of the binary number to be subtracted.

1's complement method.

Step 1 : Obtain 1's complement of subtrahend.

2 : Add with minuend

3 : If carry comes add with result

4 : If bigger number is subtracted from smaller number then there is no carry and the result will be in 1's complement form (negative number).

(Eg) Subtract $(1001)_2$ from $(1110)_2$ using 1's complement method. Hence compare the ~~method~~ result with conventional method.

$$\begin{array}{r} 1110 \\ (-)\ 1001 \\ \hline 0101 \end{array}$$

Conventional method

① $\begin{array}{r} 1110 \\ 0110 \text{ 1's comp} \\ \hline 10100 \end{array}$

Carry is added to the ~~MSB~~ LSB.

$\begin{array}{r} 0100 \\ 1 \\ \hline 0101 \end{array}$ Ans.

$\begin{array}{r} 1110 \\ 1001 \\ \hline 0101 \end{array}$

$\begin{array}{r} 110101 - 100101 \\ 110101 \\ 011010 \\ \hline 001111 \\ 1 \\ \hline 010000 \end{array}$ Ans

$\begin{array}{r} 101011 - 111001 \\ 101011 \\ 000110 \\ \hline 110001 \\ \text{No carry over} \\ -001110 \end{array}$

If we subtract a bigger number from a Smaller number then the result will be a negative number. In 1's complement method, the answer will be in 1's complement and there is no carry at all. Subtracting a bigger number from a smaller number is illustrated in the following example.

2) Subtract $(1110)_2$ from $(1001)_2$ using 1's complement method. Hence compare the result with conventional method.

1's complement method.

```
  1001
+ 0001  (1's comp)
------
  1010  (There is no carry, therefore it is in 1's complement)
```

Note that it is 1's complement of 0101

conventional method:-

```
  1001        1001
  1110        1110
------      ------
  1001       -1010
```

1) 1011.001 − 110.10

Ans 100.101

2) 11010.10 − 00101.01

Ans − 100.01

Signed binary Numbers:-

A binary number which is either positive or negative is known as signed binary number. Three representation schemes had been proposed for signed integers.

1. Sign magnitude representation
2. 1's complement representation
3. 2's complement representation

Signed magnitude representation:-

Signed magnitude representation is the representation system for signed binary numbers in which the MSB represent the sign

of number and the remaining bits represents the magnitude of the number.

What is a sign bit?

The MSB of the signed binary number is called sign bit. If it is zero (0) the number is positive. When it is one (1) then the number is negative.

Sign bit	magnitude
(MSB) 1	(0)

4 bit signed binary representation

Now, In decimal number system plus sign is used to present a positive number while negative sign shows a negative number. ~~Since~~

→ Since digital circuit can understand two numbers 0 and 1, So we must have the same symbols to indicate signed number. So an additional bit is used to express sign of a number known as sign bit. and it is placed at the most significant bit, where 0 represent positive number and 1 represent a negative number. [Since magnitude of number zero (0) is always 0, So there can be two representation of number (0) positive (+0) and negative (-0) which depends on value of sign bit]

for eg:- In an 4 bit signed number representation, 0111 represents a positive number and its magnitude is 7 (Left most bit MSB is 0 it represent that it is a positive number and the remaining bits 111 shows its magnitude).

On the other hand, 1111 represent a negative number and its magnitude is 7. ('1' in the left most MSB indicates that the number is negative and the other remaining bits represent is value (magnitude) Hence these representations are ambiguous generally because of two representation of number zero (0)

1's complement form Since 1's complement of a number is obtained by inverting each bit of given number. So we represent positive numbers in binary form and negative numbers in 1's complement form. There is a extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and you have to take 1's complement of given binary number. You can get negative number by 1's complement of a positive number. and positive number by ~~using 1's complement of a negative number.~~ Simple binary representation.

Therefore, in this representation Zero (0) can have two representation,

that is why 1's complement form is also ambiguous form.
The range of 1's complement form is from $(2^{(n-1)}-1)$ to $(2^{(n-1)}-1)$

(For eg) range of 6 bit 1's complement form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (ie 1 00000) to maximum value +31 (ie 0 11111) and Zero (0) has two representation, -0 (ie 1 11111) and +0 ie (0 00000)

(eg) Signed magnitude form For (eg) range of 6 bit Sign-Magnitude form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (ie 1 11111) to maximum value +31 (ie 0 11111). And zero (0) has two representation, -0 (ie 1 00000) and +0 (ie 0 00000)

2's complement form :- Since, 2's complement of a number is obtained by inverting each bit of given number plus 1 to least significant bit (LSB). So, we represent positive numbers in binary form and negative numbers in 2s complement form. There is extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and you have to take 2's complement of given binary number. You can get negative number by 2's complement of a positive number and positive numbers by directly using simple binary representation. If value of most significant bit (MSB) is 1, then take 2's complement from, else not. Therefore, in this representation, zero (0) has only one (unique) representation which is always positive. The range of 2's complement form is from (2^{n-1}) to $(2^{n-1}-1)$

For example, range of 6 bit 2's complement form binary number is from (2^5) to (2^5-1) which is equal from

minimum value -32 (ie 1 00000) to maximum value + 31 (ie 0 11111). ~~And two representation, 0 (ie 1.1111) and to 0 000000~~

Sign bit	1's comp & actual mag

Decimal	Signed 2's complement		signed 1's complement		Signed magnitude
+7	0111		0111		0111
+6	0110	1010 / 1011	0110	-1100 / -1011 / 1100	0110
+5	0101		0101		0101
+4	0100	-(0001) 1110 / 1111	0100	1010 / 1001 / 1000	0100
+3	0011		0011		0011
+2	0010	-(1000) 0111 / 1000	0010	1000 / 0111 / 1000	0010
+1	0001		0001		0001
+0	0000	1101 / 10	0000	-(0000) 1111 / 0	0000
-0	-	1111 / 10000	1111		1000
-1	-(0001) 1111	1000 / 0111 / 1	1110		1001
-2	-(0010) -(0011) 1110		1101		1010
-3	1101	1011 / 1100	1100		1011
-4	-(0100) 1100	11000 / 1001	1011		1100
-5	-(0101) 1011	1001 / 1010	1010		1101
-6	-(0110) 1010	1111 / 1010	1001		1110
-7	-(0111) 1001	10000	1000		1111
-8	-(1000) 1000		-		-

Zero is represented as -zero and +zero in both signed magnitude and 1's complement form. But there is no two types of zero representation in 2's complement form. 2's complement of zero is again zero. That is why 2's complement is most commonly used representation of negative numbers. Another advantage of 2's complement representation is sign bit also part of number. That is why with the help of four bits, 2's complement representation represents upto -8 where as 1's complement can represent only upto -7 only.

To increase the size of a signed positive number any number of MSB

bits can be filled with zeros - similarly to increase size of a signed negative number any number of MSB bits can be filled with one's

Signed binary addition

(a) Both are positive numbers.

1) Let us find the sum of $(01110000)_2$ and $(00000101)_2$. both are positive numbers. To find the sum add corresponding bits including Sign bit.

$$\begin{array}{r} 01110000 \\ 00000101 \\ \hline 01110101 \end{array}$$

Addition of two +ve numbers is again a positive number.

2) Both are negative nos.

Find the sum of $(10110111)_2$ and $(11010000)_2$. Note that both numbers are negative and are in 2's complement form. The decimal equivalent are $(-73)_{10}$ and $(-48)_{10}$ respectively. The sum of them is $(-121)_{10}$. To get the sum, add the corresponding bits including sign bit and discard the carry.

$$\begin{array}{rl} 1\ 01\ 0111 & (-73) \\ 1\ 1\ 01\ 0000 & (-48) \\ \hline 1]\ 1\ 0000111 & (-121) \end{array}$$

Discard the carry and final answer is 1 0000111. This is equivalent to -121 in decimal.

Addition of $(10110111)_2$ and $(11010000)_2$ gives negative number. Since the Sign bit is one for both numbers. Note that we added Sign bit also as if it is part of number. Automatically we get correct Sign bit. To Know the magnitude of the 2's complement answer (negative number) take again 2's complement as shown below.

1000 0111 → Answer:

0111 1000 → 1's complement

$\overset{1+}{\underline{0111\ 1001}}$ → 2's complement of the answer. This is equal to $(121)_{10}$

(iii) Addition of Smaller positive and bigger negative number.

Add the signed binary numbers equal to $(+18)_{10}$ and $(-104)_{10}$ respectively.

$+18 = 0\ 0010\ 010$

$-104 = 1\ 0011\ 000$

$-86 = 1\ 1010\ 010$

2's complement of $+18$ is $1\ 1101\ 110$

$+1$

$1\ 1101\ 111$

2's complement of -104 is $0\ 1100\ 111 + 1 = 0\ 1101\ 000$

$0\ 1101\ 000$

$0\ 0010\ 111$

$0\ 0010\ 010$ $(+18)$

$1\ 0011\ 000$ (-104)

$1\ 1010\ 010$ (-86)

2's complement of $+104$ is $1\ 0010\ 111 + 1 = 1\ 0011\ 000$

2's complement of -18 is $0\ 0010\ 010$

Addition of $+18$ and -104 is a negative number. That is $\underline{0111^{0}}$ the resultant sign bit equal to 1. Note that we added sign bit also as if it a part of a number. Automatically we get correct sign bit. To find the magnitude of the answer take 2's complement of the answer neglecting sign bit. The decimal equivalent of the answer is $(-86)_{10}$.

Case-4 Addition of bigger positive number and smaller negative number.

Add the signed binary numbers equal to $(+104)_{10}$ and $(-18)_{10}$ respectively.

$+104 = 0\ 1101000$

$-18 = 1\ 1101110$

$+86\ \ 1\ 0\ 1010110$ Ignore the carry.

Signed Binary Subtraction :-

Subtraction of subtrahend from the minuend is adding 2's complement of subtrahend with minuend. Similar to addition, in subtraction also four possible cases.

① Both are positive numbers :- Subtrahend is smaller than the minuend.
Subtract binary equivalent of $(+13)_{10}$ from binary equivalent of $(+86)_{10}$.

+86	0	1010110	
−(+13)	1	1110011	(2's comp)
	10	1001000	
(+73)	0	1001000	

Sign bits are considered as part of numbers and carry (9th bit) is ignored.

$$2\overline{)13}\quad 2\overline{)6}-1\quad 2\overline{)3}-0\quad 1-1$$

0001101

1110010

1110011

Case II :- Both positive numbers. Subtrahend is bigger than the minuend.

Sub $(+86)_{10}$ from $(+13)$

+13	0	0001101	
−(86)	1	1010101	(2's comp)
−73	1	0110111	(In 2's comp. form)

→ The answer is negative number and it is in 2's complement form.
→ The sign bit correctly indicate the negative sign.
→ Sign bits are considered as a part of numbers while doing subtraction.

Case III : positive minuend and negative subtrahend.

Subtract $(-28)_{10}$ from $(+67)_{10}$

+67	0	1000011	
−(−28)	0	0011100	(negative of negative becomes +ve)
+95	0	1011111	

Case IV Negative minuend and positive subtrahend.

Subtract $(+18)_{10}$ from $(-104)_{10}$.

−104	= 1	0011000	(2's comp)
−(+18)	= 1	1101110	(")
−122	= 1	0001010	Ignore carry and sign bit is 1

The Signed binary addition and Signed binary subtraction (the above 4+4 calci) is working well without any error. This addition will fail when the resultant number exceeds 7-bit binary value. The range is from $(-128)_{10}$ to $(+128)_{10}$. If the result (answer) exceeds this range, there will be a problem in the Sign bit. This is called overflow problem. Whenever Overflow occurs then the sign bit is not correct and the results becomes invalid.

Case 1 Addition of two positive numbers.

$+104 = 0\ 1100101$ The answer must be $(+152)_{10}$.

$+51 = 0\ 0110011$

$+86 = 1\ 0011000$ → Sign bit is not correct.

Hence the result is invalid due to Overflow.

Case 2 Addition of two negative numbers.

$-86 = 1\ 0101010$

$-98 = 1\ 0011110$

$-184 = 10\ 1001000$ Sign bit is incorrect and the carry can be ignored.

Hence the result is invalid due to overflow.

* This situation may also occurs during subtraction. While designing, overflow has to be monitored through software to have a reliable result. However, overflow problem can be solved by selecting 16 bit signed number instead of 8 bit signed number. In that case the 16 bit will be the sign bit and the remaining 15 bits will be magnitude.

Floating point numbers :-

The numbers which has both an integer and fractional part is called as floating point number. The floating point number can be used to represent large as well as small fraction or mixed number. Generally floating point represents an method of approximation to real number in such a way that they can support a wide range of values.

General ~~number~~ form of a number can be represented as,

$N = \pm m \times b^{\pm e}$

m - mantissa or Significand.
b - base of number
e - exponent.

In the whole expression the first number part is called as mantissa which is signed fixed point number and the second part is called as exponent that is either decimal or binary. ~~special pattern~~

Disadvantages:- 1) Floating point number suffers from the loss of precision when it is represented with a fixed number of bit (eg) 32 or 64 bit.

2) Integer arithmetic is very much efficient than floating number arithmetic.

Adv
* To represent a small number or large number, this floating point representation requires only fewer number of bits in the computer memory.

eg) $0.000000000005 = 0.5 \times 10^{-10}$

$500000000000 = 5 \times 10^{10}$

To have single fixed representation of a number in a floating point form, we have to follow some norms or rules. They are said to be normalization of floating pt numbers.

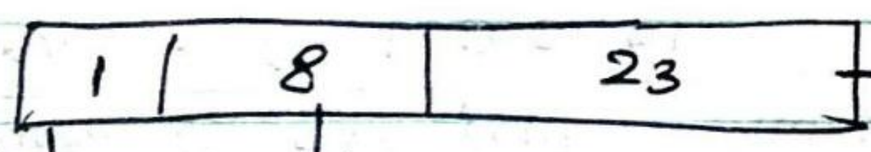
Normalization of floating point numbers:-

1. The integer part should be zero.
2. $0.d_1d_2 \ldots d_n \times B^{\pm E}$ then $d_1 \neq 0$
   and all d_i, $i \geq 2$, ≥ 0.

eg) 0.123×10^4 , 0.0123×10^5 , 123×10^3.

Two representation: FPN can be represented by two ways.

1. Single precision (32 bit)
2. Double precision (64 bit)



Bias - allows floating point numbers to be compared and stored correctly even when interpreting them as integers

1023

(Q1) -53.5

$-(110101.1)_2$

0.1101011×2^6

m = 1101011

e = 6

b = 2

2|53 .5

2|26 -1 1×2^{-1}

2|13 -0 $0.5 \times 2 = 1.0$

2|6 -1

2|3 -0 2^{8-1}

1 -1 2^7 $2 \times 2 = 4$

$= 4$

$= 44$

IEEE floating point Number Representation.

IEEE (Institute of Electrical and Electronics Engineers) Rel standardized floating point representation representation

$(-1)^s (1+m) \times 2^{(e-Bias)}$

Binary Code

Binary Codes:-

Binary number system is the most natural system for a computer, but most of us are more accustomed to the decimal system. One way to resolve the difference is to convert decimal number to binary, perform all arithmetic calculation in binary, and then convert the binary results back to decimal. This method requires that we store decimal numbers in the computer so that they can be converted to binary. Since the computer can accept only binary values, we must represent the decimal digits by means of a code that contains 1's and 0's. It is also possible to perform the arithmetic operations directly on decimal numbers when they are stored in the computer in coded form.

[Even though the digital system and computers work in binary number system, the interface (key board) between man and machine generally uses decimal number system. The logic circuits can accept two valued signals (binary) only. ∴ Decimal numbers must be coded in terms of binary signals.]

Codes are used to establish an interface between human and machine. The binary code represented by the number as well as alphanumeric letter. They are classified into weighted code and non-weighted code.

If each position of numeral has some weight, then the code is called weighted code. If there is no weight on the numeral position then it is called non-weighted code. 8 4 -2 -1

(Eg) weighted code :- BCD or 8421, 2421 code, 5421 code, ~~Excess-3 code~~.

non-weighted code :- Gray code, Excess-3 code.

③ Binary coded decimal ④ Alphanumeric codes, ⑤ Error detating code ⑥ Error correcting codes

BCD code: [In this code, each decimal digit is replaced by its binary equivalent. In this code, each bit of a BCD number is represented by a 4-bit binary number] This is also known as 8421 code because each of the four bits is given a 'weighting' according to its column value in the binary system. A number with K decimal digit will require 4K bits in BCD.

Eg) Decimal 396 is represented by 12 bits.

0011 1001 0110

A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 to 9. A BCD number greater than 10 looks different from its equivalent binary number, even though both contain 1's and 0's. ~~Hence the representation~~ The remaining six combinations are invalid in BCD.

Def A binary coded decimal (BCD) is a type of binary representation for decimal values where each digit is represented by a fixed number of binary bits usually between 4 and 8. The norm is four bits, which effectively represent decimal values from 0 to 9. This writing format system is used because there is no limit to the size of a number. Four bits can simply be added as another decimal digit, versus real binary representation, which is limited to the usual powers of two, such as 16, 32, or 64 bits.

Decimal	BCD	Excess 3 [BCD+0011]	Gray	2421	84-2-1
5421					
0000 0	0000	0011	0000	0000	0000
0001 1	0001	0100	0001	0001	0111
0010 2	0010	0101	0011	0010	0110
0011 3	0011	0110	0010	0011	0101
0100 4	0100	0111	0110	0100	0100
1000 5	0101	1000	0111	1011	1011
1001 6	0110	1001	0101	1100	1010
1010 7	0111	1010	0100	[illegible]	1001
1011 8	1000	1011	1100	[illegible]	1000
1100 9	1001	1100	1101	[illegible]	1111
- 10	-	-	1111	-	-
- 11	-	-	1110	-	-

12	-	-	1010	-	-	
13	-	-	1011	-	-	
14	-	-	1001	-	-	
15	-	-	1000	-	-	

Advantages of BCD code:- 1) It is very similar to decimal system.
2) we need to remember binary equivalent of decimal numbers from 0 to 9.

Disadvantages :- 1) The addition and subtraction of BCD have different rules.
2) The BCD arithmetic is little more complicated.
3) BCD needs more no of bits than binary to represent the decimal number.
So BCD is less efficient than binary.

Advantages of Binary code:-
→ Binary codes are suitable for the computer applications
→ Binary codes are suitable for digital communications
→ Binary codes make the analysis and designing of digital circuits if we use the binary codes.
→ Since only 0 to 1 are being used, implementation becomes easy.

A code is said to be reflective if codes for 0 to 9 are equal to the corresponding 1's complement of 0 to 9. in the reverse order.

Excess-3 code:-
The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding $(0011)_2$ or $(3)_{10}$ to each code word in 8421. The excess-3 codes are obtained as follows:-

Decimal Number → 8421 BCD $\xrightarrow[0011]{\text{Add}}$ Excess-3

It is a self complementing code. It is reflective and sequential.

Gray code: It is the non weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position. It has a very special feature that, only one bit will change each time the decimal number is incremented. ~~as shown.~~ As only one bit changes at a time, the gray code is called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operations

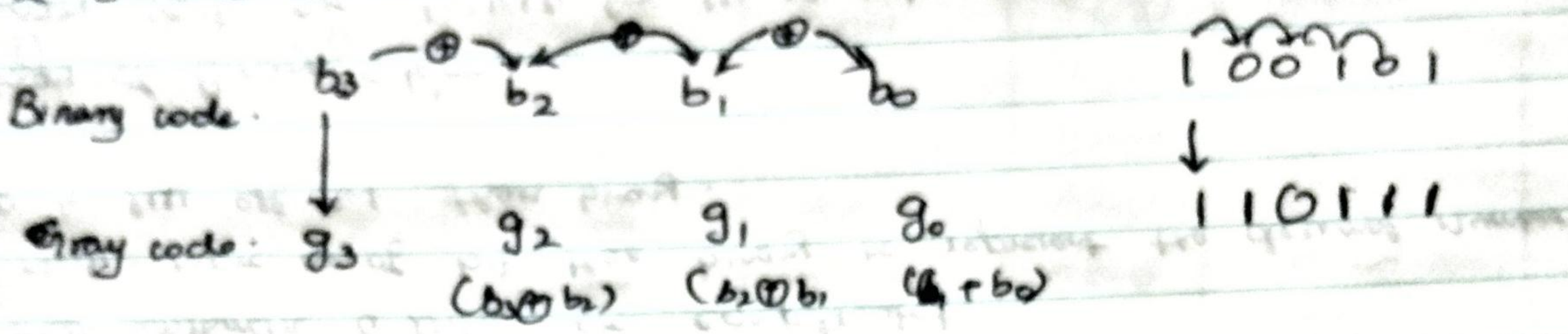
Application of Gray code :-

→ Gray code is popularly used in the shaft position encoders.

→ A shaft position encoder produces a code word which represents the angular position of the shaft.

Binary to Gray code conversion:-

Let Binary code be $b_3\ b_2\ b_1\ b_0$. Then the respective Gray code can be obtained as follows:



i.e. $g_3 = b_3$

$g_2 = b_3 \oplus b_2$

$g_1 = b_2 \oplus b_1$

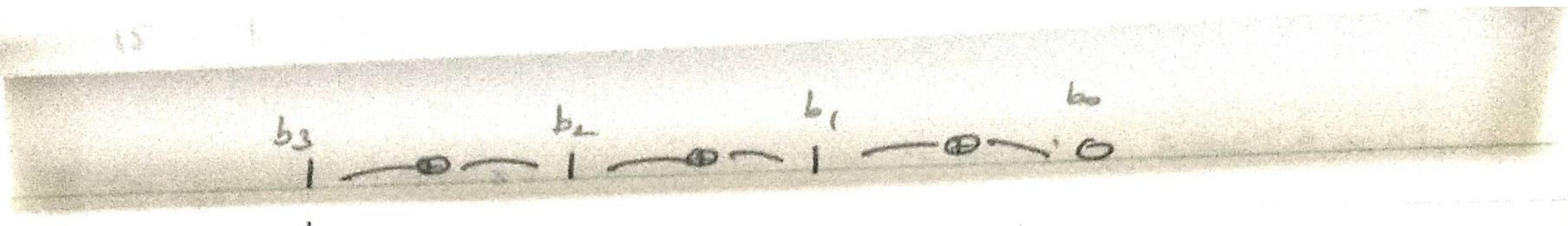
$g_0 = b_1 \oplus b_0$

Excess-3 code

A code is said to be sequential if the binary value of the code is in ascending order and each succeeding code is incremented by one.

Example: Binary code $b_3\ b_2\ b_1\ b_0 = 1110$

Gray code: $g_3\ g_2\ g_1\ g_0$



0 0 1

g_3 (1)

Final gray code is 1001

Conversion from Gray code to Binary code :- Let Gray code be $g_3 g_2 g_1 g_0$ Then the respective Binary code can be obtained as follows.

Gray code : g_3 g_2 g_1 g_0

Binary code b_3 b_2 b_1 b_0

The number's code and its 9's complement's code should have complementary relationship.

(e) $b_3 = g_3$

$b_2 = b_3 \oplus g_2$

$b_1 = b_2 \oplus g_1$

$b_0 = b_1 \oplus g_0$

Self complementing code :- For a Self complementary code, 9's complement of a decimal number is obtained directly by changing '1' to '0' and '0' to '1' (ie by complementing)

(eg) $6 \xrightarrow{E-3} 1001$ $\xrightarrow{9's\ comp}$ $(r^n-1)+(1-10^{-m})-N$. r-base. n - . m - .

$(10^1-1)+(1-10^{-0})-6$

$10-1+0-6$

$10-7=3$.

Excess 3 = 0110

(eg) Gray code 1001 then Binary code is 1110

Binary code to Gray code conversion

Follow these steps for converting a binary code into its equivalent Gray code.

→ Consider the given binary code and place a zero to the left of MSB.

→ Compare the successive two bits starting from zero. If the 2 bits are same, then the output is Zero. Otherwise, output is One.

→ Repeat the above step till the LSB of Gray code is obtained.

(eg) Binary code 1000

Gray code 1100.

Alphanumeric codes:- Digital Systems and computers have to handle alphabetic characters, symbols and numerals. But the computers and digital systems cannot handle alphabetic characters as such. Hence digital codes are required for alphabetic characters, symbols and numerals. A set of standard codes for alphabets, symbols and numerals are called alphanumeric codes. The alphanumeric code includes codes for 10 numerals from 0 to 9, 26 alphabets and 28 other symbols. There are many standard available for alphanumeric codes.

commonly used
~~They are~~ 1) The ASCII codes
alphanumeric codes are, ➡

→ It is widely accepted and used in computers and digital systems.

→ Most of the computer keyboards are made using ASCII.

→ It is a 7 bit code.

→ Obviously, There are 128 characters and Symbols.

→ Even though it is a 7 bit code, it can be considered as 8 bit code with MSB 0.

→ The list of ASCII characters is given in the table.

→ The seven bits of the code are designated by b_1 to b_7 with b_7 the most significant bit.

→ The letter A, for example, is represented in ASCII as 1000001 (column 100, row 0001)

→ The ASCII code also contains 94 graphic characters and 34 non printing characters used for various control functions.

→ The graphic characters consists of the 26 uppercase letters (A to Z), the 26 lowercase letters (a to z), the 10 numerals (0 to 9) and 32 special printable characters such as %, * and $.

→ 34 control characters are shown in the table. The control characters are used for routing data and arranging the printed text into prescribed format.

→ There are three types of control characters : format effectors, information separators & communication - control characters.

→ Format effectors are characters that control the layout of printing. They include the familiar word processor and type writer controls such as backspace (BS), horizontal tabulation (HT) and carriage return (CR)

→ Information separators are used to separate the data into divisions such as paragraphs and pages. They include characters such as record separator (RS) and file separator (FS)

→ The communication - control characters are useful during the transmission of text between remote terminals. (eg) STX (start of text) and ETX (end of text) which are used to frame a text message transmitted through telephone wires.

→ Even though it is a 7bit code, ~~most computers manipulate an eight bit~~ it can be considered as 8 bit code with MSB 0. An additional 128 eight bit characters with the most significant bit set 1 are also ~~used~~ introduced by IBM called as Extended ASCII characters. This is also accepted as universal standard. It has codes for non English characters, currency symbols for various nations, Greek characters, mathematical symbols, graphical characters etc.

~~BCD code~~ Error detecting and Error correcting code :-

In digital systems, during communication and data transfer, if one bit is unintentionally changed from 0 to 1 or 1 to 0, then the data becomes error (corrupted). A ~~detection of~~ mechanism is required to check ~~and correct~~ the error in the transmission. Such error detecting mechanism is called error detecting codes. (eg) parity code.

Ility, to correct the error during the data transmission, Error correcting mechanism is adopted. That is called error correcting code (eg) Hamming code.

BCP code Binary coded Pentary code.

BCP code is a base 5 number system. It is also similar to BCD. In BCD, (number system) each bit of a BCD number is represented by a 4 bit binary number. But here in BCP, each bit will be represented by three bit binary.

(eg) $(123)_5$ will be represented by $(001\ 010\ 011)_{BCP}$.

Boolean Algebra:

Boolean Algebra is the branch of algebra in which the values of the variables are the truth values true and false, usually denoted (time or false)
1 and 0 respectively.

→ In 1847, George Boole developed Boolean algebra ⇒ deductive logic problems

→ In 1938, After a century Claude Shannon of Bell Laboratories, (open circuit/close ckt) used this Boolean algebra in the field of multi contact telephone networks.

→ Now the entire digital computing world is based on this Boolean Algebra. (0 and 1)

→ It is a tool to design logic system mainly used by engineers and mathematicians:

Boolean Postulates and theorems:

Based on Logical operations on the Boolean variables on 0 and 1, there are Seven postulates.

Postulate 1: A Boolean variable will have two values either 0 or 1 ie $X=0$ or $X=1$. The next three postulates are based on Boolean addition - And the results are taken as granted.

P2: $0+0=0$

P3: $0+1=1+0=1$

P4: $1+1=1$

The remaining three postulates are based on Boolean multiplication The results are taken as granted.

P5: $0 \cdot 0 = 0$

P6 : 0.1 = 1.0 = 0

P7 : 1.1 = 1

There are ten theorems based on the above said postulates. The theorems have fundamental relation between the Boolean variables. These theorems will be useful to simplify and manipulate logical circuits. The operators used in the Boolean algebra are AND(.), OR(+) and Negation (NOT) (bar or prime)

Theorem 1 commutative law :- The property is true for both AND and OR operation

ie $A+B = B+A$

$A \cdot B = B \cdot A$

Theorem 2. Associative law :- This property also is true for both AND and OR operation.

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

Only $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

A	B	C
0	0	0
0	0	1
0	1	0

Theorem 3 Distributive law :- This property is combination of AND and OR operation.

$A \cdot (B+C) = A \cdot B + A \cdot C$

Only $A+(B \cdot C) = (A+B) \cdot (A+C)$

Theorem 4 Identity law :- This property is true for both AND and OR operation.

$A \cdot A = A$

Only $A+A = A$

Theorem 5 Negative law :- $\bar{\bar{A}} = A$.

That is inverse of inverse of A is A itself

Theorem 6 Redundance law,

1) $A + A \cdot B = A$

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

Proof (1) LHS $= A + A \cdot B = A \cdot 1 + A \cdot B = A(1+B)$

$= A \cdot$

$=$ RHS

(2) LHS $= A \cdot (A+B) = A \cdot A + AB = A + AB$

$= A \cdot 1 + AB = A(1+B)$

$= A \cdot$

$=$ RHS.

Theorem 7

1) $0 + A = A$

2) $0 \cdot A = 0$

3) $1 \cdot A = A$

4) $1 + A = 1$

Theorem 8

(1) $\bar{A} + A = 1$

(2) $\bar{A} \cdot A = 0$

Theorem 9 :- (1) $A + \bar{A} \cdot B = A + B$

(2) $A \cdot (\bar{A} + B) = A \cdot B$.

Proof

(1) LHS $A + \bar{A} \cdot B$

$= A(1+B) + \bar{A}B$ since $1+B=1$

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

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

$= A + (1)B$

$= AB = A + B =$ RHS.

2) $A \cdot (\bar{A} + B) = A \cdot B$

LHS $= A \cdot (\bar{A} + B)$

$= A\bar{A} + AB$

$= 0 + AB$

$= AB =$ RHS.

Theorem 10 De Morgan theorems :-

De Morgan's 1st Theorem :- Complement of sum is equal to product of complements. (Break the line and change the sign)

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

De Morgan's Second theorem :- Complement of product is equal to sum of complements. (Break the line and change the sign)

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

Proof : 1st theorem (using truth table (perfect Induction method)

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

Output of LHS expression $= \overline{A+B}$

I/P		O/P	
A	B	A+B	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Output of RHS expression $= \overline{A} \cdot \overline{B}$

I/P :	$\overline{A}$	$\overline{B}$	Output
A	B		$\overline{A} \cdot \overline{B}$
0	0	1	1
0	1	1	0
1	0	0	0
1	1	0	0

For all possible values of A and B, value of expression in LHS is equal to the corresponding value of expression in RHS. Hence De Morgan's first theorem is proved.

Proof of De Morgan's Second theorem :- $\overline{A} \cdot \overline{B} = \overline{A} + \overline{B}$

O/P of LHS expression $= (\overline{A \cdot B})$

Input		Output	
A	B	A·B	$\overline{A \cdot B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

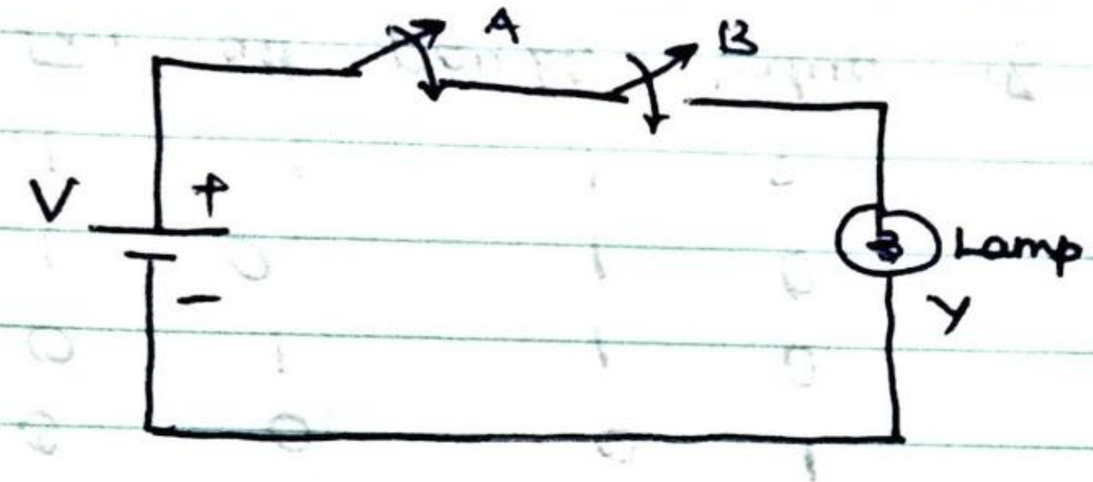
Output of RHS expression $= \overline{A} + \overline{B}$

Input		O/P		
A	B	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

For all possible values of A and B, the values of expression in the LHS are equal to the corresponding values of expression in RHS. Hence, DeMorgan's second theorem is proved.

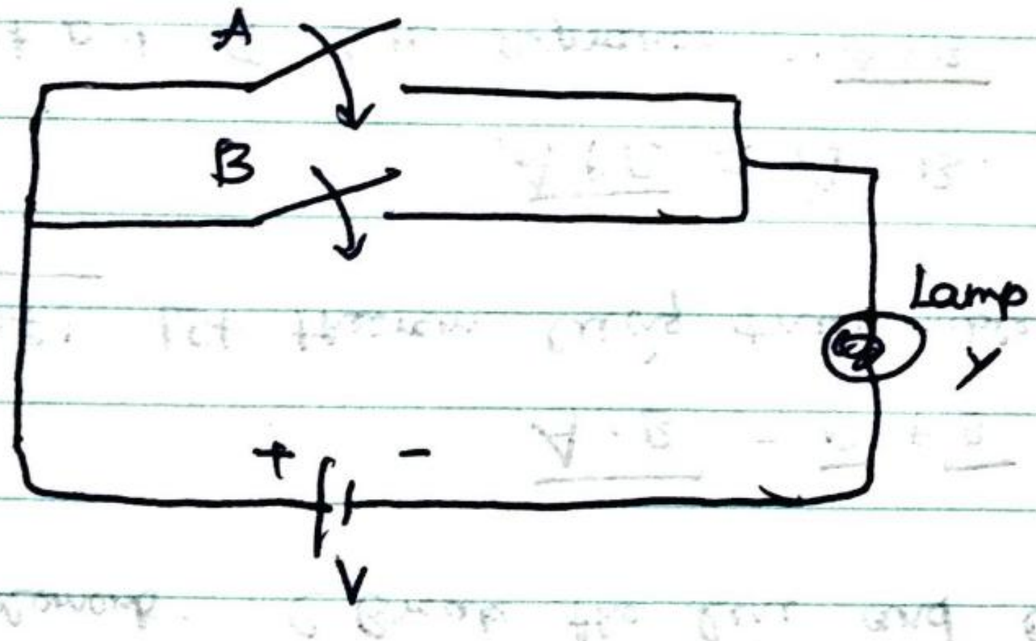
Truth table:- A truth table provides the o/p of expressions (mathematical expression of theorem) for all possible input combination. This method of proving a theorem is called perfect induction method. It is one of the ways to prove the statement of theorems. This is the use of the truth table.

Electric circuit demonstrate AND Operation :-



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

Electric circuit for OR operation :-



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

Principle of duality :- Theorem 7 and 8 are called principle of duality. Any expression in these theorems can be obtained from the other dual expression simply by replacing 0 by 1, 1 by 0 and '+' by '.' and '.' (a dot) by '+' (a plus)

Duality function:

$0+A \Rightarrow A \Leftrightarrow 1 \cdot A = A$

$1+A \Rightarrow 1 \Leftrightarrow 0 \cdot A = 0$

$A + \bar{A} = 1 \Leftrightarrow A \cdot \bar{A} = 0$

Consensus theorem:- This theorem States that

$$xy + \bar{x}z + yz = xy + \bar{x}z$$

proof

$$\text{LHS} = xy + \bar{x}z + yz = xy + \bar{x}z + (x+\bar{x}) \cdot yz$$
$$= xy + \bar{x}z + xyz + \bar{x}yz$$
$$= xy(1+z) + \bar{x}z(1+y)$$
$$= xy + \bar{x}z = \text{RHS}. \quad [\text{since } (1+z) = (1+y) = 1]$$

Transposition theorem:-

(1) $x \cdot y + \bar{x} \cdot z = (x+z)(\bar{x}+y)$

(2) $(x+y)(\bar{x}+z) = x \cdot z + \bar{x} \cdot y.$

proof by perfect induction method.

(Q) Simplify $A \cdot (A \cdot B + C)$

1) $A \cdot (A \cdot B + C) = A \cdot A \cdot B + A \cdot C$
$= (A \cdot A) \cdot B + A \cdot C$ (Since $A \cdot A = A$)
$= A \cdot B + A \cdot C$ (Theorem 3)
$= A \cdot (B + C)$

Simplify

2) $XYZ + \bar{X}Y + XY\bar{Z}$

$XYZ + \bar{X}Y + XY\bar{Z} = Y \cdot (XZ + \bar{X} + X\bar{Z})$ (Theorem 3)
$= Y \cdot (X(Z+\bar{Z}) + \bar{X})$ (Theorem 3)
$= Y(X \cdot 1 + \bar{X})$ (Theorem 8)
$= Y \cdot (X + \bar{X})$ (Theorem 8)
$= Y.$

Complement of a function:-

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

dual of this $\overline{(ABC)} = \bar{A} + \bar{B} + \bar{C}.$

MINTERMS & MAX TERMS.

Let A be a binary variable (0 or 1) and $\bar{A}$ be the complement of A. Let us consider two binary variables A and B. All possible Input combination for AND operation are called minterms or standard products.

Minterms for two variable input

Binary values of A and B	Minterms	Notation	Decimal code.
00	$\bar{A}\bar{B}$	m_0	0
01	$\bar{A}B$	m_1	1
10	$A\bar{B}$	m_2	2
11	AB	m_3	3.

Minterms for three variable I/P: →

MAX TERMS:-

All possible combination of Input variables for OR operation is called max terms or standard sums.

MAX terms for two variable I/P.

Binary values of A and B	Max terms	Notation	Decimal code
00	$\bar{A}+\bar{B}$	M_0	0
01	$\bar{A}+B$	M_1	1
10	$A+\bar{B}$	M_2	2
11	$A+B$	M_3	3

Preparing Logic diagrams :-

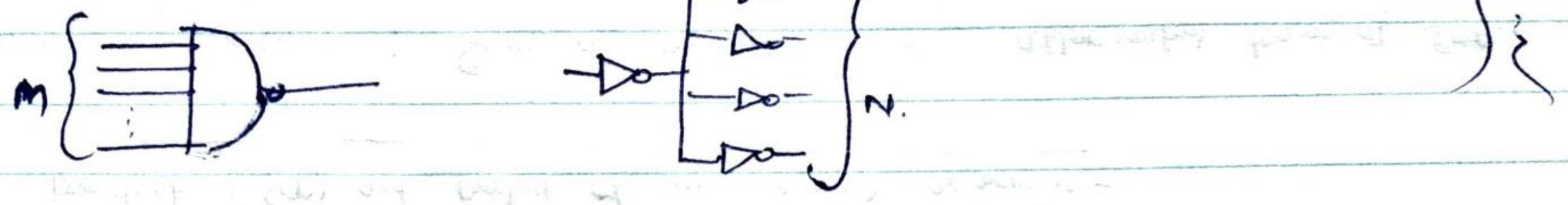
Extension of multi-inputs :-

The logic gates so far we have seen, has only two inputs. But there is no limits of numbers of inputs in a logic gate. So a gate have 'n' number of inputs.

Fan out & Fan in:-

Fan out: It is the maximum number of inputs of same type of gates that can be connected to one output of logic gate. Gates with Larger fanout are slower.

Fan in: It is the maximum number of inputs that a single gate can accept. gates with large fan in are bigger and slower.



High Impedance state: It is the state of output of a logic gate. If the impedance of the output of the gate is very high then output of the gate will be electrically disconnected with the next stage input of the gates. High impedance state can be used to electrically isolate input and o/p devices. It is equal to physically disconnecting the two devices. In otherwords mechanically connected but electrically disconnected.

Buffer: Buffer is a gate used to increase the fanout of a gate.

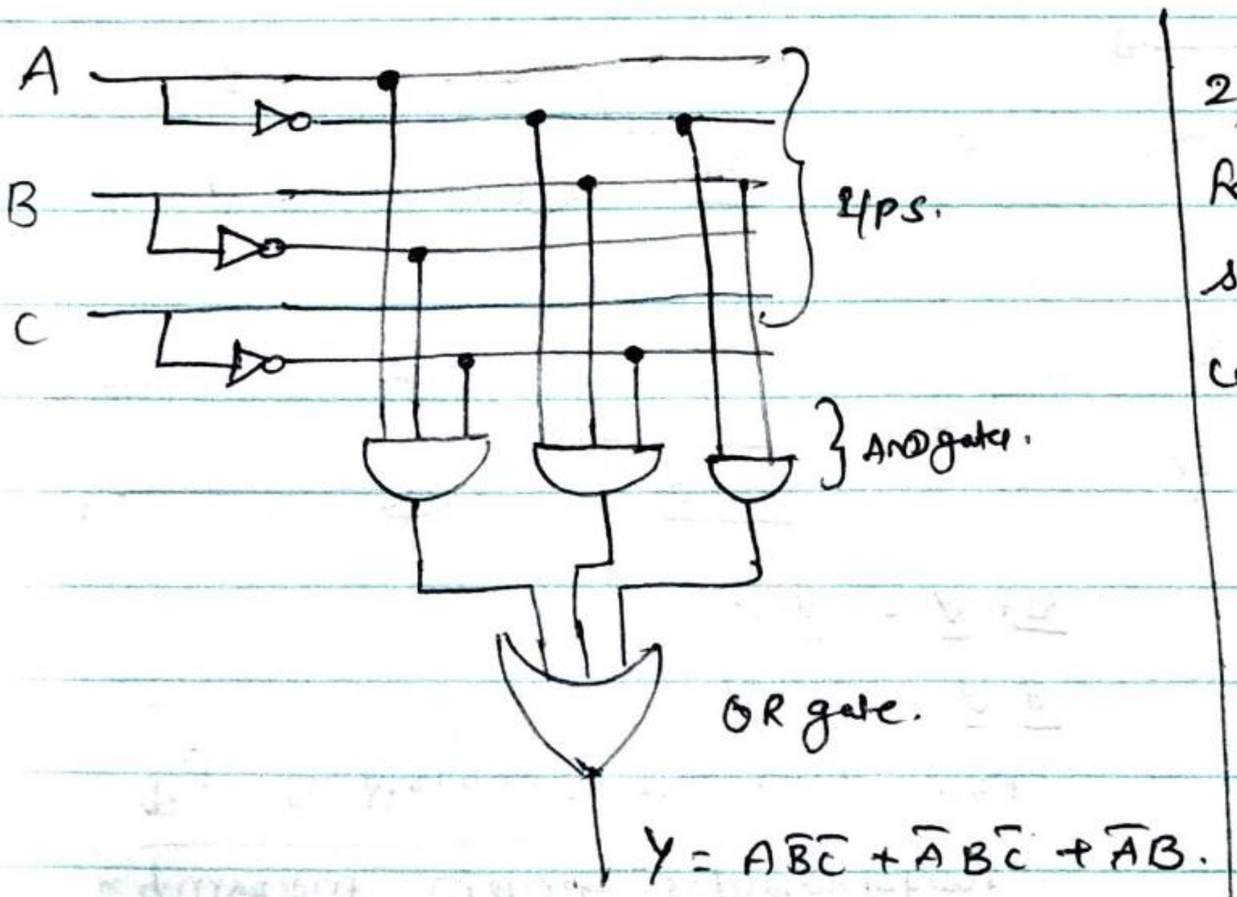
Achive high tristate buffer, Achive high tristate inverter
Active low " Active low tristate inverter.

(Eg) 1) Implement the following logic expression as it it is given using logic gates. (do not simplify the expression)

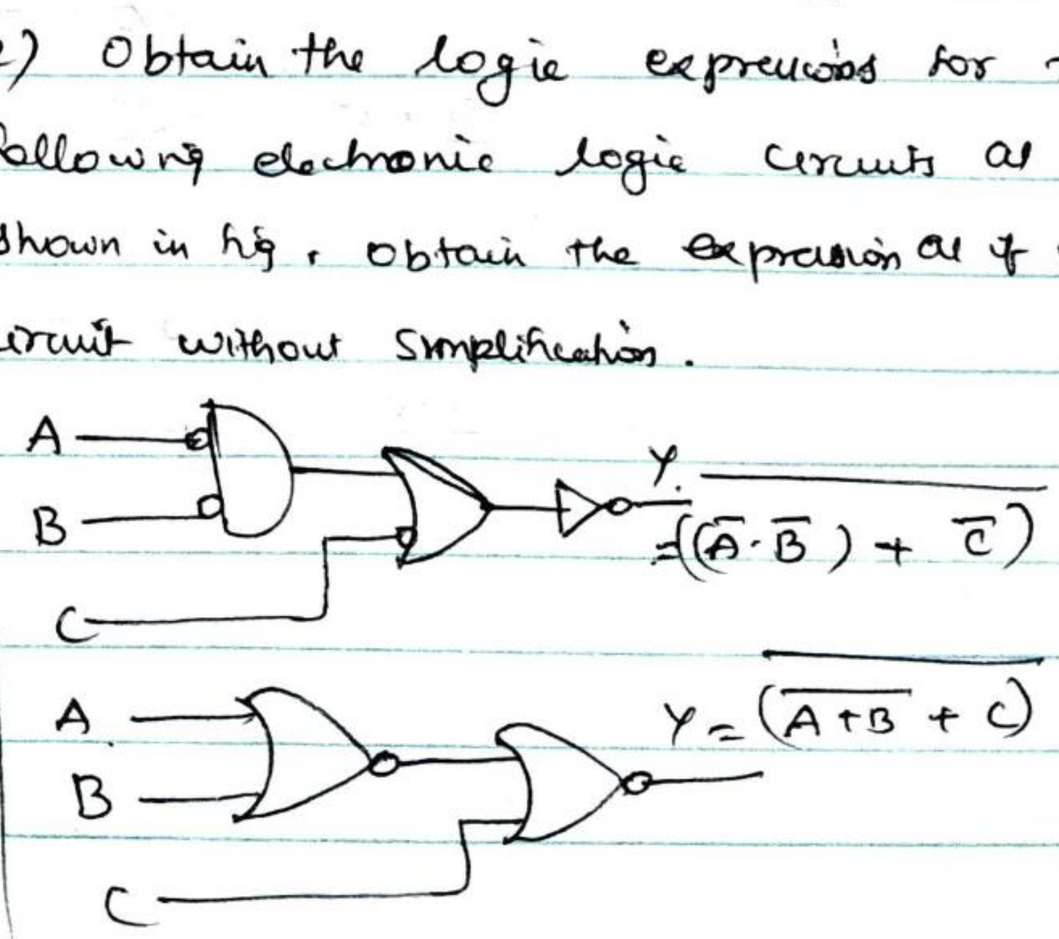
(a) $Y = A\bar{B}\bar{C} + \bar{A}BC + \bar{A}B$.

(b) $Y = A\bar{B}(C+\bar{B}) + \bar{A}C$ — new.

(c) $Y = [A(B+\bar{C}) + A\bar{B}]C$



2) Obtain the logic expressions for the following electronic logic circuits as shown in fig. Obtain the expression as it is in the circuit without simplification.

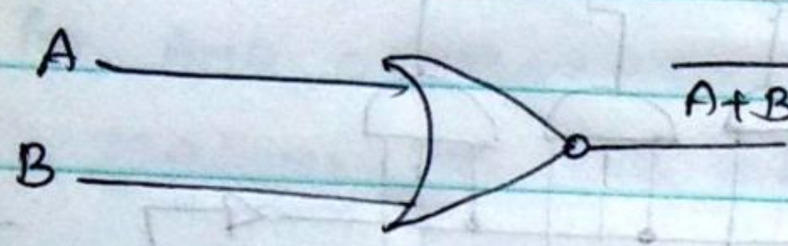


Equivalent circuit representations:

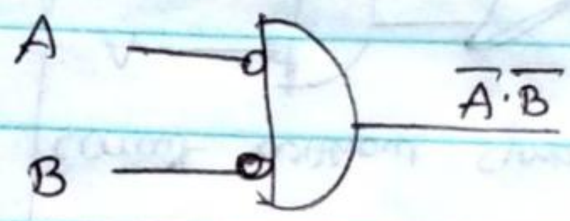
By DeMorgan's theorem, we get,

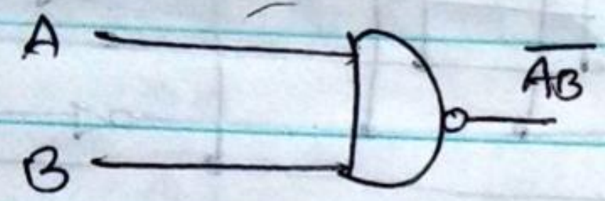
$$\overline{A+B} = \bar{A}\cdot\bar{B}$$

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

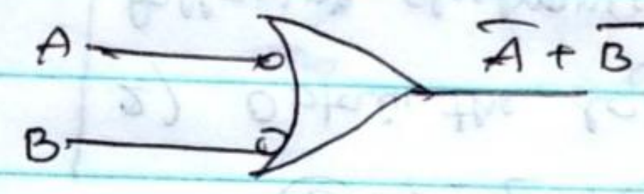


=



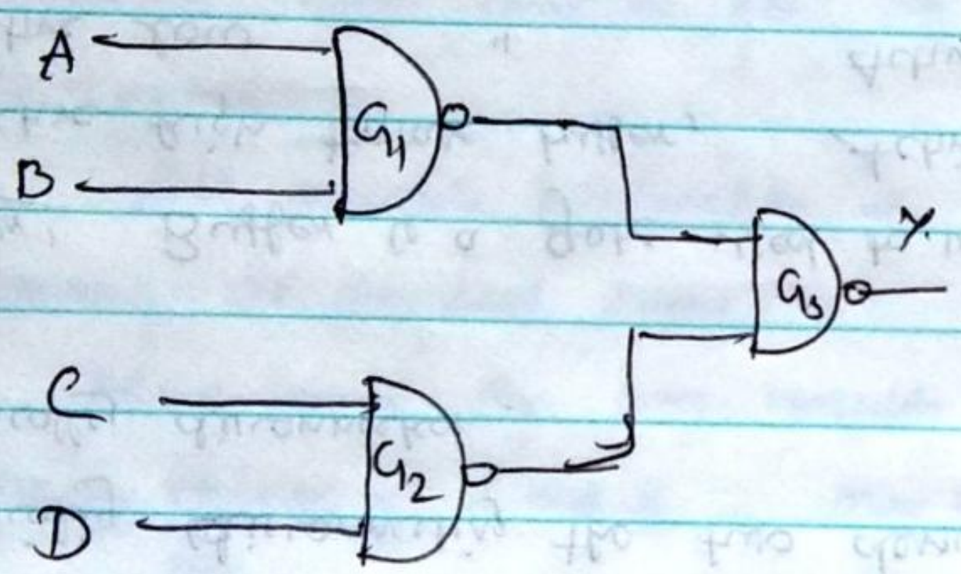


=

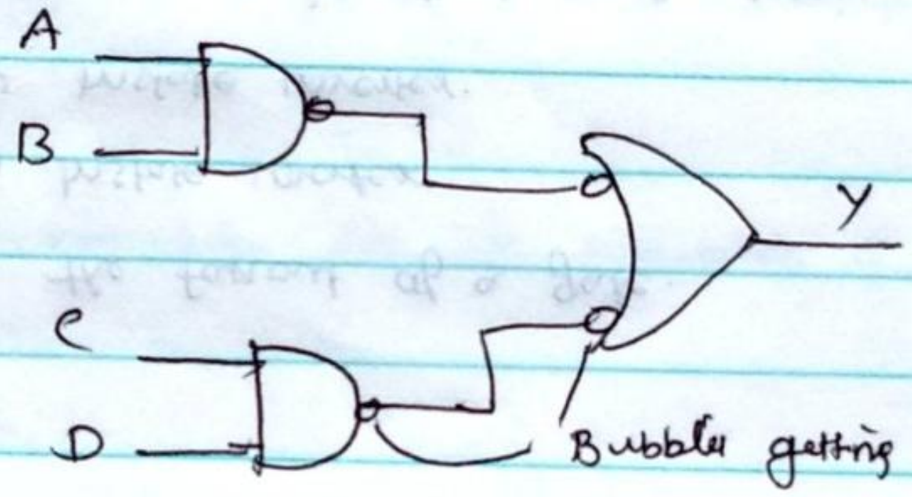


(9) Obtain the equivalent logic expression for the circuit shown,

Given circuit:

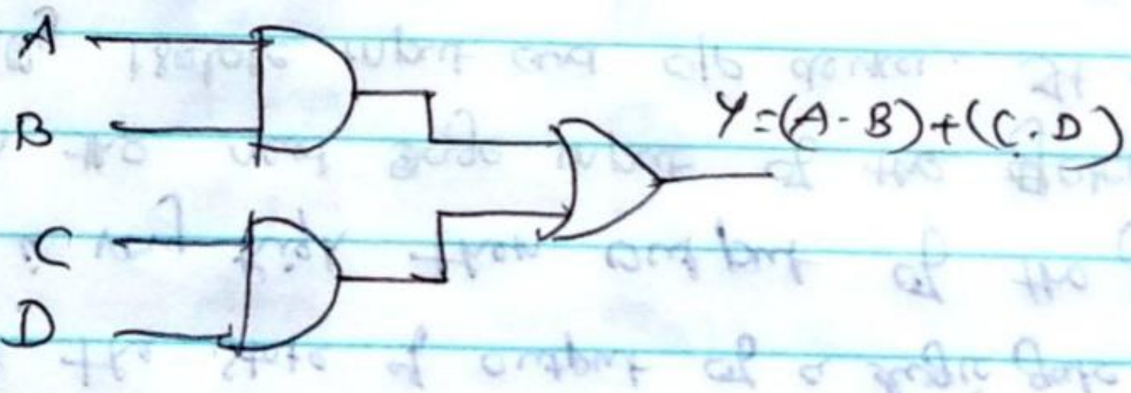


Replacing G_3.



Bubbles getting cancelled.

The equivalent circuit is,



Sum of products (SOP) and product of sums (POS) expression:-

Sum of products (SOP): Sum of product is the abbreviated form of SOP. Sum of product form is a form of expression in Boolean algebra in which different product term of inputs are being summed together. This product is not arithmetical multiply but it is Boolean logical AND and the Sum is Boolean logical OR.

Types of SOP form

There are two different forms of sum of product

* Canonical SOP form
* Non-Canonical SOP form
* minimal SOP form

Canonical SOP form:-

This is standard form of SOP. It is formed by the minterms of the function for which the output is true. This is also known as sum of min terms or canonical disjunctive normal form (CDNF). It is just a fancy name.
[Canonical means - standardized and disjunctive means logical OR Union]. Canonical SOP expression is represented by summation sign Σ and minterms in the braces for which the output is true.

(for eg)

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

for this function, the canonical SOP expression is

$F = \Sigma(m_1, m_2, m_3, m_5)$

which means that the function is true for the minterms $\{1, 2, 3, 5\}$

By expanding the summation we get,

$F = m_1 + m_2 + m_3 + m_5$

now, putting min terms in the expression

$F = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C$.

Non-Canonical SOP form:-

As the name suggests, this form is the non-standardized form of SOP expression. The product terms are not the min terms but they are simplified. Let's take the above function.

$F = \bar{A}\bar{B}C + \bar{A}B(\bar{C}+C) + A\bar{B}C. = \bar{A}\bar{B}C + \bar{A}B(1) + A\bar{B}C$

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

This expression is still in SOP but it is not non-canonical or non-standardized form.

Minmal SOP form:-

This form is most simplified form. This is also non-canonical form. Simplification is by Boolean algebra or K-map. It is preferred because minimum no. of gates and input lines. It is commercially beneficial because compact size, fast speed, and low fabrication cost.

Product of Sum :- The POS form is a form, in which products of different sum terms of inputs are taken. These are not arithmetic product and sum but they are logical Boolean AND and OR respectively.

Types

1) Canonical POS form
2) Non - canonical POS form
3) Minimal POS form.

Canonical form :- It is also known as product of Max term. or Canonical conjunctive normal form (CCNF) canonical means standard and conjunctive means intersection. In this form, Max terms are AND together for which the output is false,

Canonical POS expression is represented by Π and Max terms for which output is false in brackets as shown in the example given below.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$F = \Pi (M_0, M_4, M_6, M_7)$

Expanding the product,

$F = M_0 . M_4 . M_6 . M_7$

putting max terms,

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

Non- canonical form :- The product of Sum expression that is not in standard form is called non - canonical form.

above eq, $F = (A+B+C)(\bar{A}+B+C)(\bar{A}+\bar{B}+C)(\bar{A}+\bar{B}+\bar{C})$

$= \underline{(.B+C)}$ (final Simplification).

This expression is still in product of sum form but it is non - canonical form.

Minimal POS form :-

This is the most simplified and optimized form of a POS expression which is non canonical.

Logic design

→ Logic design is a procedure to obtain a logic circuit that gives the desired output for a particular combination of I/ps.

Step 1 clearly state the problem and define input and output logic levels.

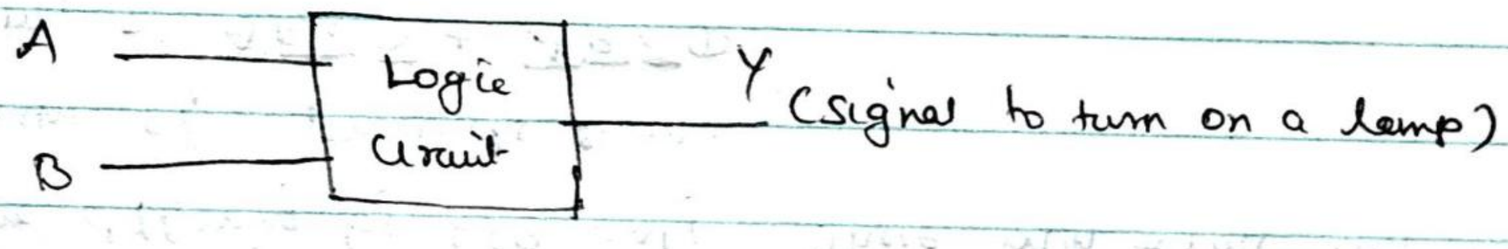
Step 2 Prepare a truth table for the statement of a problem

Step 3 Obtain logic expression from the truth table.

Step 4 Simplify the logic expression

Step 5 Draw the logic circuit diagram for the simplified expression.

(eg) Design a two bit comparator logic circuit that will provide a signal to turn on a lamp if two input signals are identical.



Solution

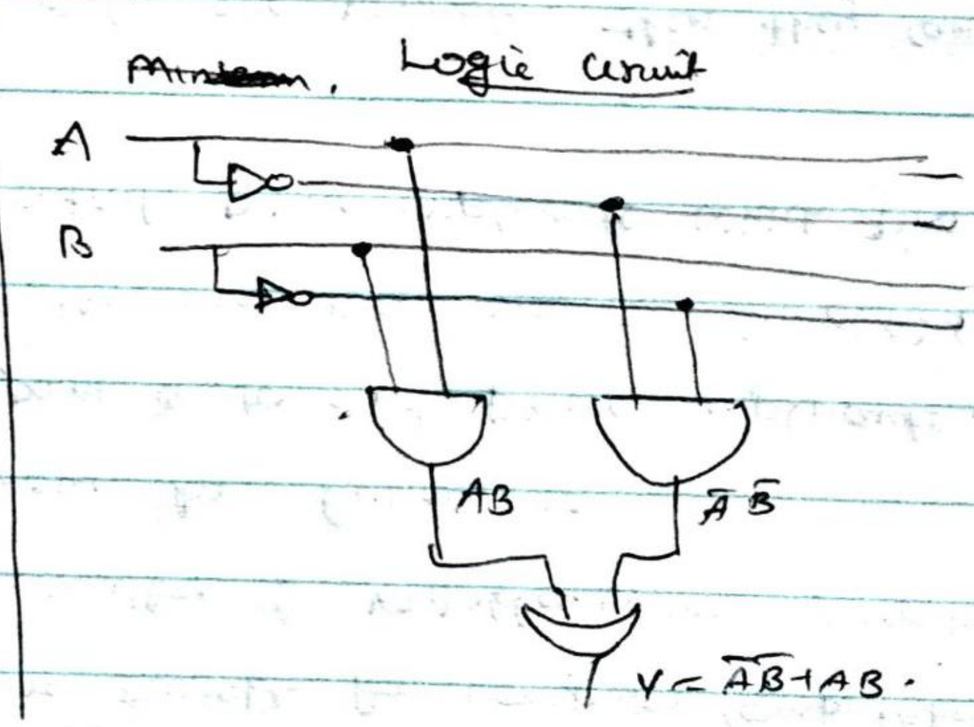
Step 1 The inputs are A and B and the output is Y. If A and B are equal then the output is one.

Step 2 The truth table for the comparator is shown.

Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Step 3 $Y = \bar{A}\bar{B} + AB$.

Step 4 Logic circuit



Karnaugh Map :→ It is a method of Simplifying Boolean algebra expression. Maurice Karnaugh introduced it in 1953 as a refinement of Edward Veitch's 1952 Veitch chart.

→ It was actually rediscovery of Allan Marquand's Marquand diagram in 1881.
∴ Veitch charts are also known as Marquand - Veitch diagram.
and Karnaugh maps as Karnaugh - Veitch maps. (KV maps).

→ The required Boolean results are transferred from a truth table onto a two dimensional grid, Karnaugh Map, the cells are ordered in Gray code...

→ Each cell position represents one combination of input conditions while each cell value represents the corresponding output value.

→ Optimal groups of 1s or 0s are identified, which represent the terms of a canonical form of the logic in the original truth table.

→ These terms can be used to write a minimal Boolean expression representing the required logic.

→ [illegible] 1950 - 1980 → K-map are mostly used

Rules for K-map Simplification:- (SOP)

1. Groups may not contain Zero.
2. We can group 1, 2, 4, 8 (or) 2^n cells.
3. Each group should be as large as possible.
4. Cells containing 1 must be grouped.
5. Groups may overlap
6. Opposite grouping and corner grouping is allowed.
7. There should be as few groups as possible.

[8. When All the cells are having '1', then the value of K-map is 1]

2-variable K-map

3-variable K-map

4-variable K-map

5-variable K-map

Don't care:-

for any system, out of possible inputs, some of them may not occur. For eg, in binary to BCD encoder, the input is a 4-bit binary number. There are 16 possible input combinations available. But we have only 9 ~~outputs~~ input combinations. So No need to bother about the o/p of the system for these never occuring inputs.

→ These o/p can be either assumed as 0 or 1 according to our convenience (Because they never occur)

→ ~~This is also known as incomp~~ The functions with don't care inputs are also known as ~~incomp~~ Incompletely Specified functions. or functions with optional ~~optional~~ combinations.

→ It is generally represented as 'X'. Some authors represent them as 'd'

→ ~~Can be incomp,~~ Inclusion of don't care condition will reduce cost and size of the hardware.

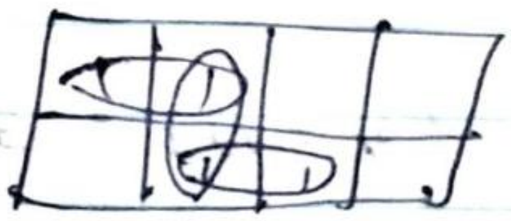
Implicants:- Implicant is a product/min term in SOP or Sum/max term in POS of a Boolean function.

(Eg) Consider a Boolean function $F = AB + ABC + BC$.

Then the Implicants are AB, ABC and BC

Prime Implicants:-

A group of square or rectangle made up of bunch of adjacent minterms which is allowed by def. of K-map are called prime implicants i.e all possible groups formed in K-map.



No. of PI = 3,

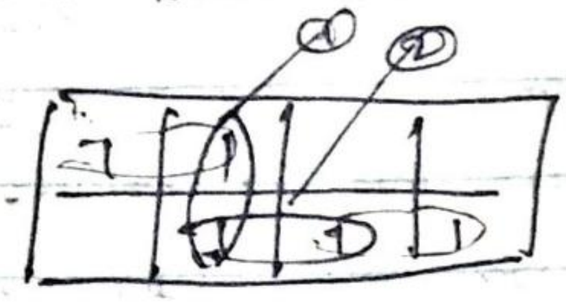
Essential PI:-

These are those subcubes (groups) which cover atleast one minterm that can't be covered by any other PI. EPI are those PI which always appear in the final solution. (above eq) EPI = 2.

Redundant PI The prime implicants for which each of its minterm is covered by some essential prime implicant are redundant prime implicants (RPI) This PI never appear in final solution.

RPI = 1.

Selective PI :- The PIs for which are neither essential nor redundant prime implicants are called selective prime implicants. (SPI). They are also known as non-essential prime implicants. They may appear in some solution or may not appear in some solution.



SPI = 2

(Q). $F = \Sigma(1, 5, 6, 7, 11, 12, 13, 15)$ find no. of PI, EPI, RPI, SPI

Implicant

PI, Implicant = 8, PI = 5, EPI = 4, RPI = 1, SPI = 0.

POS

In SOP, we considered cells for minterms. (the input terms for which the output is 1). Hence we can simplify the given Boolean function in POS form. For POS form, we have to consider all input terms for which the output is zero (Maxterms). In other words, set of all maxterms has to be considered for POS form.

(Q) Simplify the Boolean function $F(A, B, C, D) = \Sigma m(1, 3, 7, 11, 15)$ in POS form. Hence find the sum of products (SOP) form and compare the hardware requirement for both cases.

$$\overline{Y} = G_1 + G_2 + G_3$$
$$= \bar{D} + B\bar{C} + A\bar{C}$$
$$Y = (D)(\bar{B}+C)(\bar{A}+C)$$

SOP $G_1 = CD$ $Y = CD + \bar{A}\bar{B}D$.

$G_2 = \bar{A}\bar{B}D$

The no. of hardwares required in both form are equal. In POS form two NOT gates, two OR gates and one AND are required. Whereas in SOP form two NOT gates, two AND gate and one OR gate are required.

Quine McCluskey or Tabulation method of Minimization of Logic functions

Developed by William V. Quine and extended by Edward J. McCluskey.

→ Karnaugh Map method of simplification is simple and convenient as long as no. of variables is with in five or six.

→ If no. of variables exceed six then it is difficult to select the adjacent Squares.

→ The main disadvantage of the K-map is that it is a trial and error procedure. depends upon the ability of the person and may be poor for six or more variables.

→ Quine McCluskey is a step by step procedure suitable for computer computation. Hence this method can be implemented for 'n' number of variables using computer. Because of monotonous procedure, it is tedious for human use.

→ Tabular method is two fold. The first part is to find prime implicants. The prime implicants are terms (expression) that they are members of simplified function. The second part is selection of essential prime implicants which give an expression with minimum number of literals.

If two minterms are differing by only one variable, then they can be simplified as follows:

For eg) $m_0 = 0000 = \bar{A}\bar{B}\bar{C}\bar{D}$

$m_1 = 0001 = \bar{A}\bar{B}\bar{C}D$.

m_0 and m_1 are differing by LSB only. These min terms can further be simplified for sum of products form.

ie $m_0 + m_1 = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$

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

The minterms m_0 and m_1 are differing by only one bit and hence we can eliminate one variable. The implicants obtained in this stage can be further simplified by repeating the above proces. If there is no simplification is possible then the set of minterms are called prime implicants.

Step by Step Procedure:-

Step 1: Represent the minterms in binary code. Simplify the minterms by repeatedly combining the minterms which are differ by only one bit. If there is no further simplification possible then the resultant implicants become prime implicants.

Step 2 Tabulate the prime implicants Vs minterms.

Step 3 Encircle the prime implicants which alone covers particular minterms. Such prime implicants are called essential prime implicants.

Step 4 Essential prime implicants are part of minimal sum.

Step 5 If the minimal sum obtained in step 4 are covering all minterms of the function then the problem is solved. otherwise proceed to step 6.

Step 6 Delete the essential Prime implicants from the table.

Step 7 Remove dominating Rows and dominating subset columns.

Step 8 Again find the (secondary) essential prime implicants and include the function along with EPI obtained in Step 4

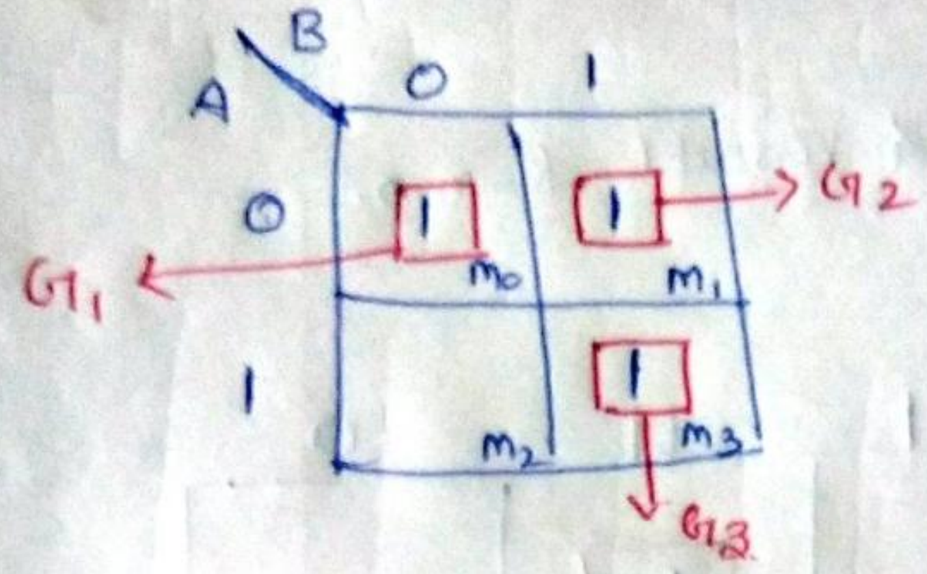
Step 9 Repeat the Steps 6 to 8 till the minimal function covers all minterms.

(Pg P6) Simplify the following Boolean function by using the Quine-Mccluskey tabulation method $F = \sum m(0, 1, 2, 4, 5, 8, 10, 11)$. Hence, check the result by K-map method.

(1) 1st reduction (2)

$F(A,B) = \Sigma m(0,1,3)$

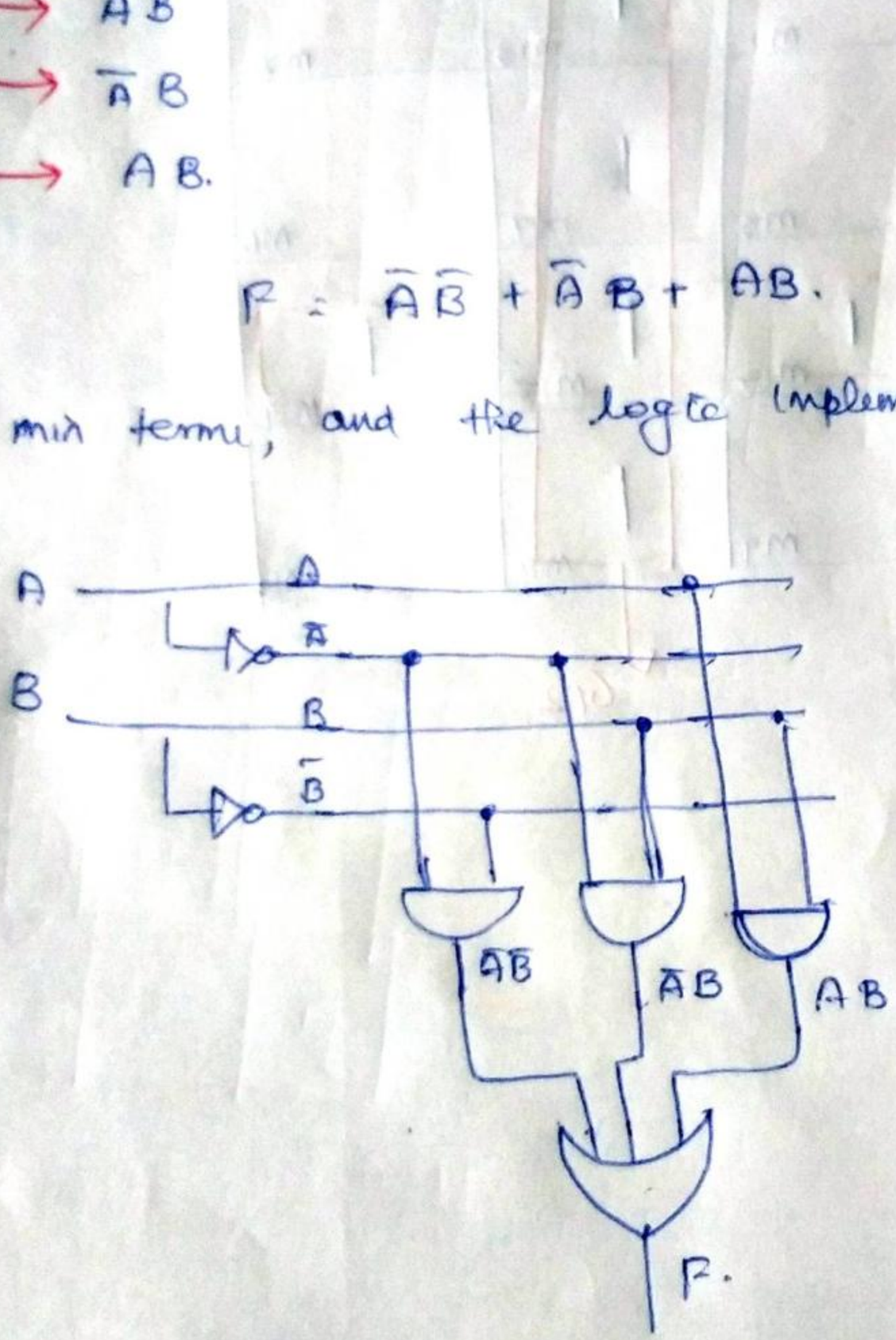
2 I/ps ; hence,
2 variable K-map



$G_1 \rightarrow \bar{A}\bar{B}$
$G_2 \rightarrow \bar{A}B$
$G_3 \rightarrow AB.$

$$F = \bar{A}\bar{B} + \bar{A}B + AB.$$

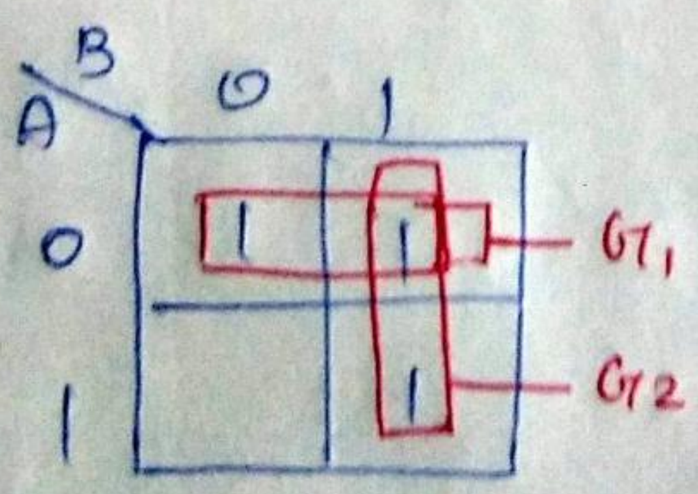
→ Three min terms, and the logic implementation.



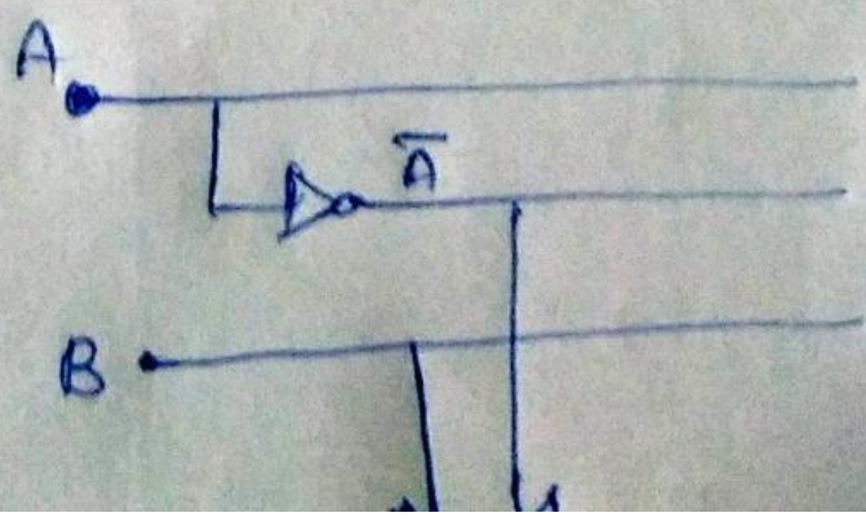
6 gates required.

If we simplify using grouping, that is only minimum no. of groups should be present.

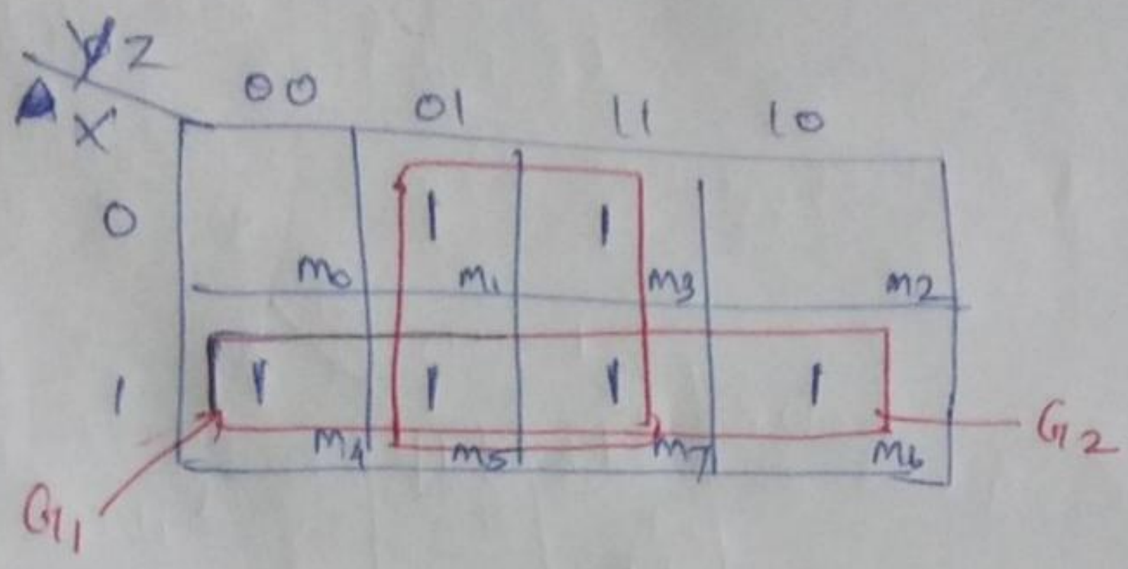
∴



$G_1 \rightarrow \bar{A}$



2) $F(x,y,z) = \Sigma m(1,3,4,5,6,7)$



$G_1 \rightarrow x$

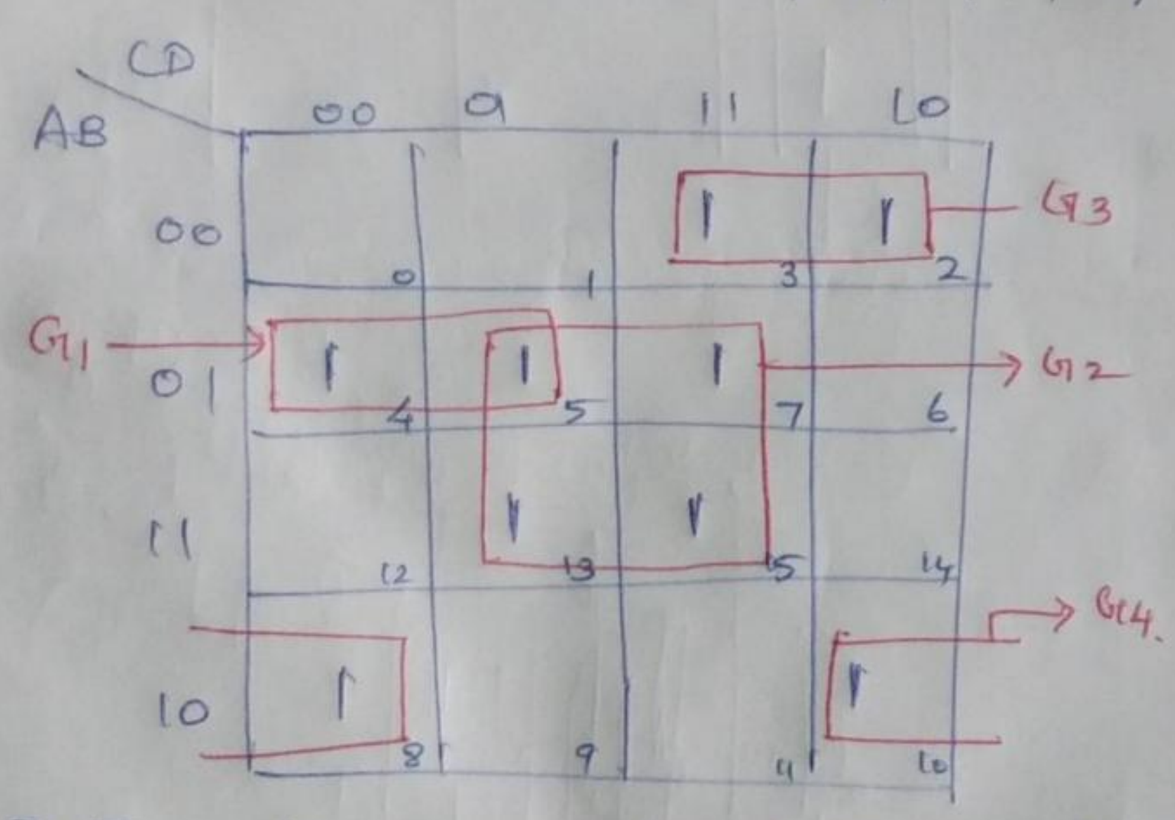
$G_2 \rightarrow z$

$F = x + z$

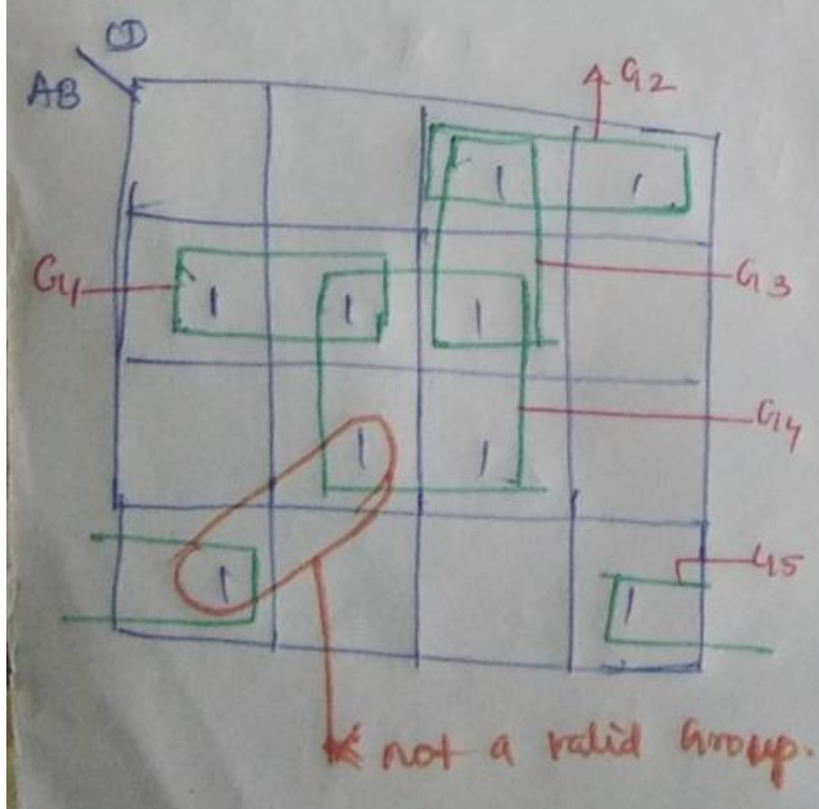
3) 4 variable K-map.

$f(A,B,C,D) = \Sigma m(2,3,4,5,7,8,10,13,15)$

K-map ①



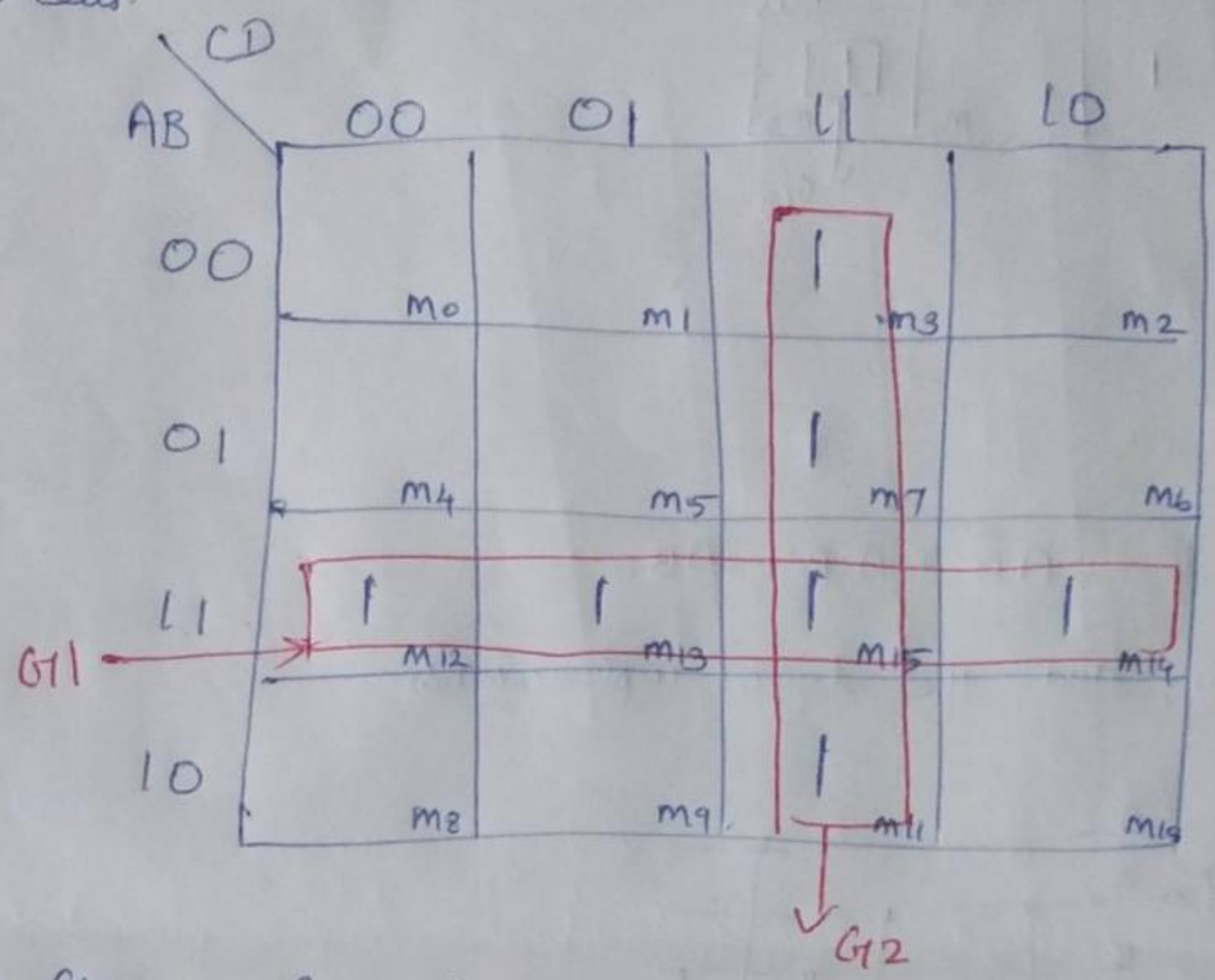
$G_1 \rightarrow \bar{A}B\bar{C}$ $G_2 \rightarrow BD$ $G_3 \rightarrow \bar{A}\bar{B}C$ $G_4 \rightarrow A\bar{B}\bar{D}$



1) Minimise the following Boolean function using Sum of products (SOP):

$$f(A,B,C,D) = \Sigma m(3,7,11,12,13,14,15)$$

1) Identify I/Ps and o/ps.
2) Four I/p variable. hence 4 variable K-map.
3) 16 cells.

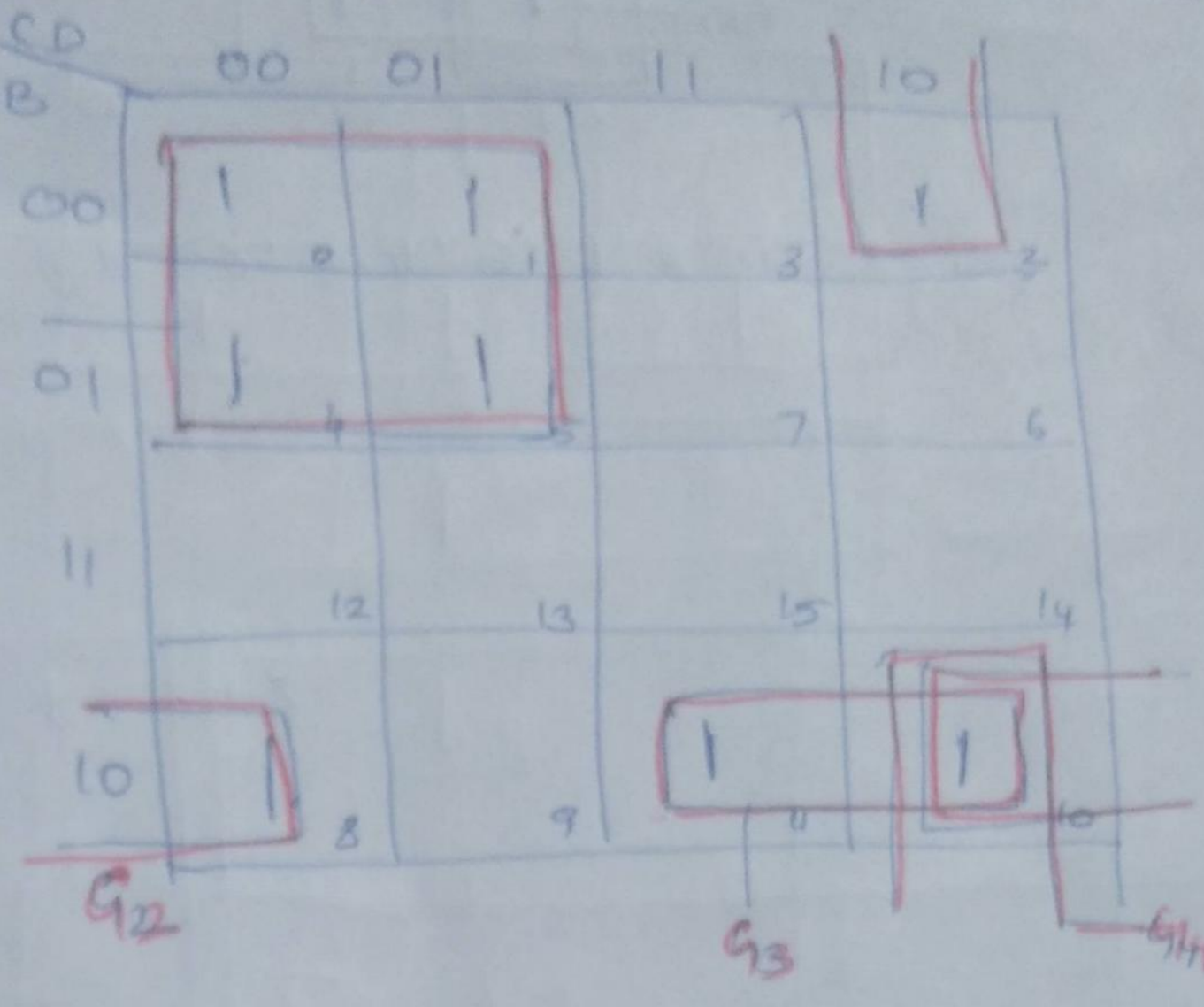


Two Groups. G_1 & G_2.

$$F = AB + CD$$

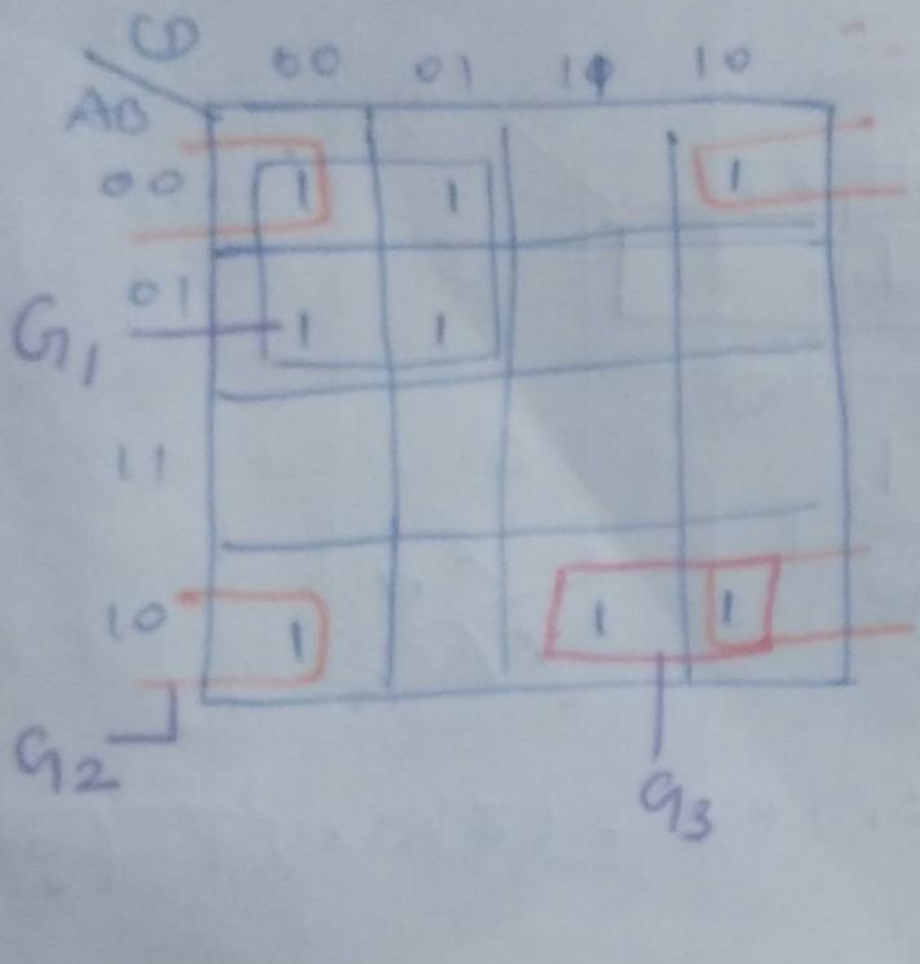
$(0,1,4,5)$

$(0,2,8,10)$



G_1 G_2 G_3

$F = \bar{A}\bar{C} + A\bar{B}\bar{D} + A\bar{B}C + \bar{B}C\bar{D}$ G_4



$G_1 = \bar{A}\bar{C}$

$G_2 = \bar{B}\bar{D}$

$G_3 = A\bar{B}C$

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

a) $Y(A,B,C,D) = \Sigma m(0,1,3,7,8,9,11,15)$

0, 1, 8, 3, 9, 7, 11, 15

Step 1 Table 1

Group	Minterm	Binary Representation A B C D
0	m_0	0 0 0 0
1	m_1	0 0 0 1
	m_8	1 0 0 0
2	m_3	0 0 1 1
	m_9	1 0 0 1
3	m_7	0 1 1 1
	m_{11}	1 0 1 1
4	m_{15}	1 1 1 1

Table 2

Group	Matched pairs	Binary Rep A B C D
0	$m_0 - m_1$	0 0 0 -
	$m_0 - m_8$	- 0 0 0
1	$m_1 - m_3$	0 0 - 1
	$m_1 - m_9$	- 0 0 1
	$m_8 - m_9$	1 0 0 -
2	$m_3 - m_7$	0 - 1 1
	$m_3 - m_{11}$	- 0 1 1
	$m_9 - m_{11}$	1 0 - 1
3	$m_7 - m_{15}$	- 1 1 1
	$m_{11} - m_{15}$	1 - 1 1

Table 3

Step 3

Group	M.P	Binary Rep. A B C D	
0	$m_0 - m_1 - m_8 - m_9$	- 0 0 -	$\bar{B}\bar{C}$
	$m_0 - m_8 - m_1 - m_9$	- 0 0 -	
1	$m_1 - m_3 - m_9 - m_{11}$	- 0 - 1	$\bar{B}D$
	$m_1 - m_9 - m_8 - m_{11}$	- 0 - 1	
2	$m_3 - m_{11} - m_7 - m_{15}$	- - 1 1	CD
	$m_3 - m_{11} - m_7 - m_{15}$	- - 1 1	

PI

Step 2

EPI

PI	Minterms involved	0	1	3	7	8	9	11	15
$\bar{B}\bar{C}$	0, 1, 8, 9	X	X			X	X		
$\bar{B}D$	1, 3, 9, 11		X	X			X	X	
CD	3, 7, 11, 15			X	X			X	X

$F = \bar{B}\bar{C} + CD$

Step 1 Ascending order (1's present)

0, 1, 2, 4, 8, 5, 10, 11

I

Groups	Minterms	Binary equivalent
GA1	0 ✓	0000
GA2	1 ✓	0001
	2 ✓	0010
	4 ✓	0100
	8 ✓	1000
GA3	5 ✓	0101
	10 ✓	1010
GA4	11 ✓	1011

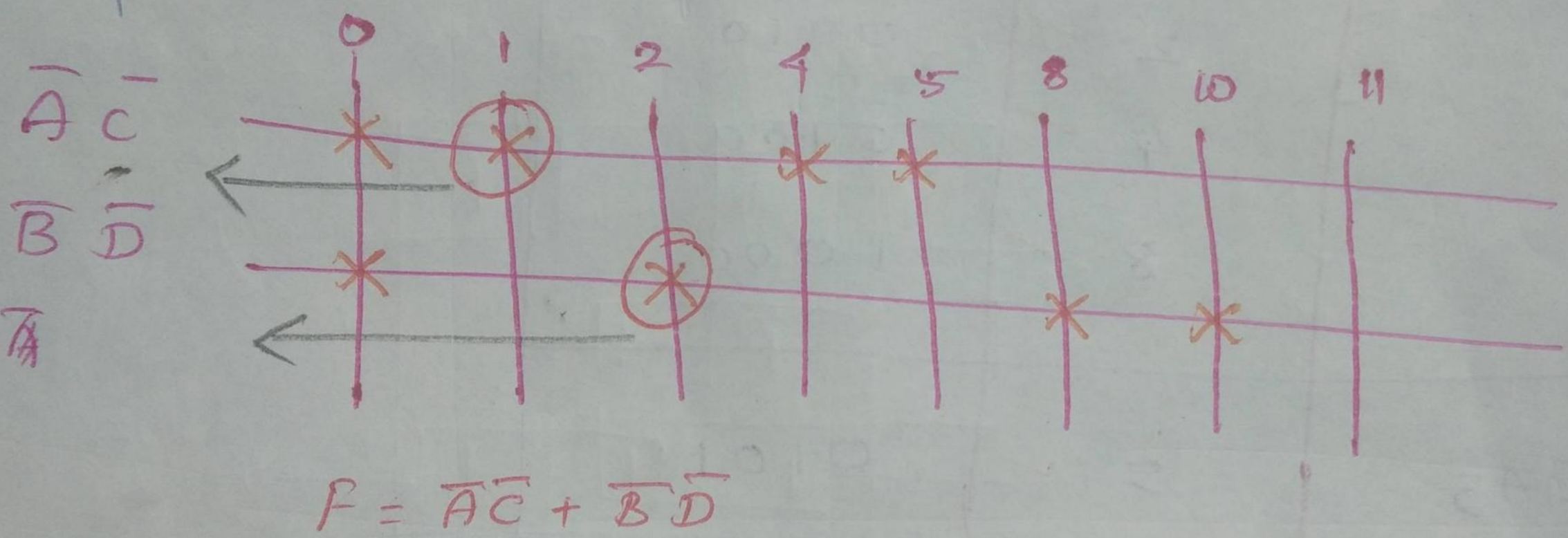
II

Groups	Minterms	Binary Equivalent
GB1	(0,1) ✓	000-
	(0,2) ✓	00-0
	(0,4) ✓	0-00
	(0,8) ✓	-000
GB2	(1,5) ✓	0-01
	(2,10) ✓	-010
	(4,5) ✓	010-
	(8,10) ✓	10-0
GB3	(10,11)	101-

(11) Groups | Minterms | Binary Equivalent

Groups	Minterms	Binary Equivalent A B C D'	
GC1	(0,1,4,5)	0 - 0 -	$\bar{A}\bar{C}$
	(0,2,8,10)	- 0 - 0	$\bar{B}\bar{D}$
	(0,4,1,5)	0 - 0 -	$\bar{A}\bar{C}$
	(0,8,2,10)	- 0 - 0	$\bar{B}\bar{D}$

PI / minterms



$F = AC + B\bar{D}$

Unit - III

Synchronous Sequential logic

→ Combinational circuits, the outputs are entirely dependent on the current inputs.

→ Sequential circuits is nothing but a combinational circuit but with an storage element. The storage elements are connected to the combinational circuit to form a feed back path. The storage elements are devices capable of storing binary information.

→ The binary information stored at any given time defines the state of the Sequential circuit at that time.

Block diagram

Q difference between combinational and sequential unit

The Sequential circuit receives binary information from external inputs that, together with the present state of the storage elements, determine the binary value of the outputs.

→ The o/p is not only the function of the present inputs, but also the function of present state of the storage elements.

Types
1) Synchronous Sequential circuit
2) Asynchronous "

Synchronous Sequential circuit :-

If all the outputs of a Sequential circuit change (affect) with respect to active transition of a clock signal ~~then the circuit~~. That means, all the outputs of Synchronous Sequential circuits change at the same time. Therefore the outputs of a synchronous Sequential circuits are in synchronous with either only positive edges or only negative edges of clock signal. Eg) flip flops.

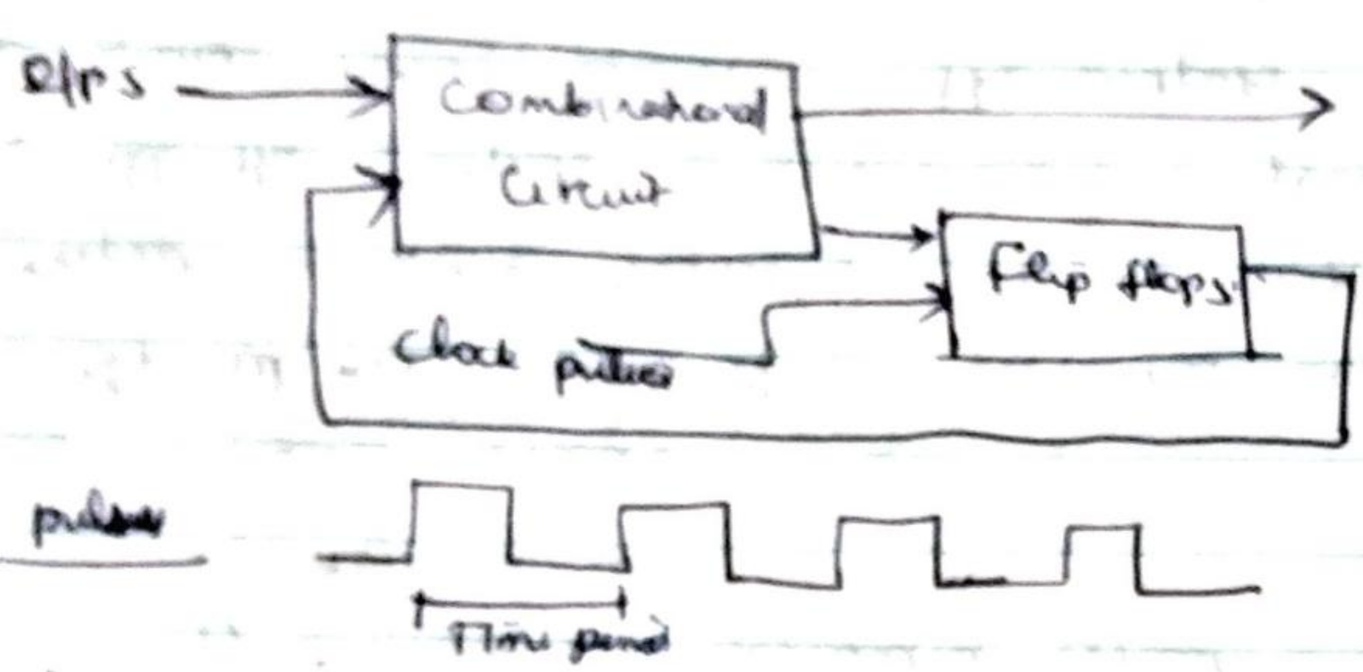
Asynchronous Sequential Circuit :-

If some or all the outputs of a sequential circuit do not change with respect to active transition of clock signal. Therefore the O/Ps are not in synchronous with ~~either~~ the positive edges or negative edges of clocks.

→ The storage elements in Asynchronous circuits are time delay circuits or latches

→ In Synchronous → they are flip flops.

Synchronous clocked sequential circuits

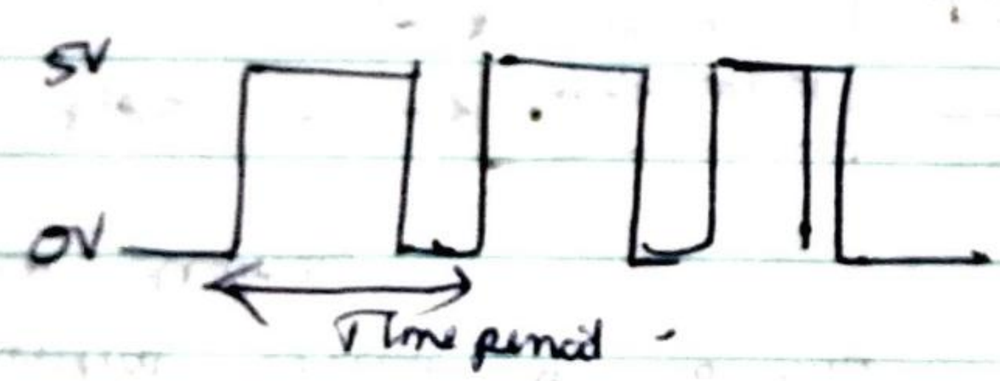


Clock pulses

→ It is a periodic signal and its ON time and OFF time need not be the same. We can represent the clock signal as a square wave, when both its ON time and OFF time are same.

→ This signal stays at logic High for some time and stays at logic low for equal amount of time. In this case, time period will be equal to either twice of ON time or twice of OFF time.

→ We can represent the clock signal as train of pulses, when ON time and OFF time are not same.



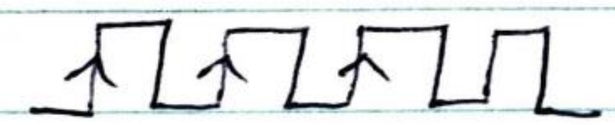
→ The reciprocal of the time period of clock signal is known as the frequency of the clock signal.

Type of triggering

Two possible types of triggering that are used in sequential cir
① Level triggering
② Edge triggering

Level triggering :- two types
1) positive level triggering 
2) Negative level triggering 
↳ The sequential circuit is operated with the clock signal when it is in Logic High. *
2) The sequential circuit is operated with the clock signal when it is in Logic Low.

Edge triggering :- two types of transitions that occur in clock signal. that means, the clock signal transition either from Logic low to Logic high or Logic high to Logic low

two types
① Positive edge triggering 
2) Negative edge triggering 

① The sequential circuit is operated with the clock signal that is transitioning from logic Low to logic High.
2) The sequential circuit is operated with the clock signal, that is transitioning from Logic High to logic low.

Storage elements :- (Memory elements)

A storage element in a digital circuit can maintain a binary state indefinitely (as long as power delivered to the circuit), until directed by an input signal to switch states.

→ The major difference among storage devices are, no. of inputs and in which the inputs affects the binary state.

→ Storage elements that operates with signal levels (rather than signal transitions) are referred to as latches and those controlled by a clock transition are flip-flops.

→ Latches are said to level sensitive devices whereas the flip flops are edge sensitive devices.

→ The two types of related to each other because the latches are the basic circuits from which all flip flops are constructed.

→ latches are used to store binary information in asynchronous sequential circuit.

Latches

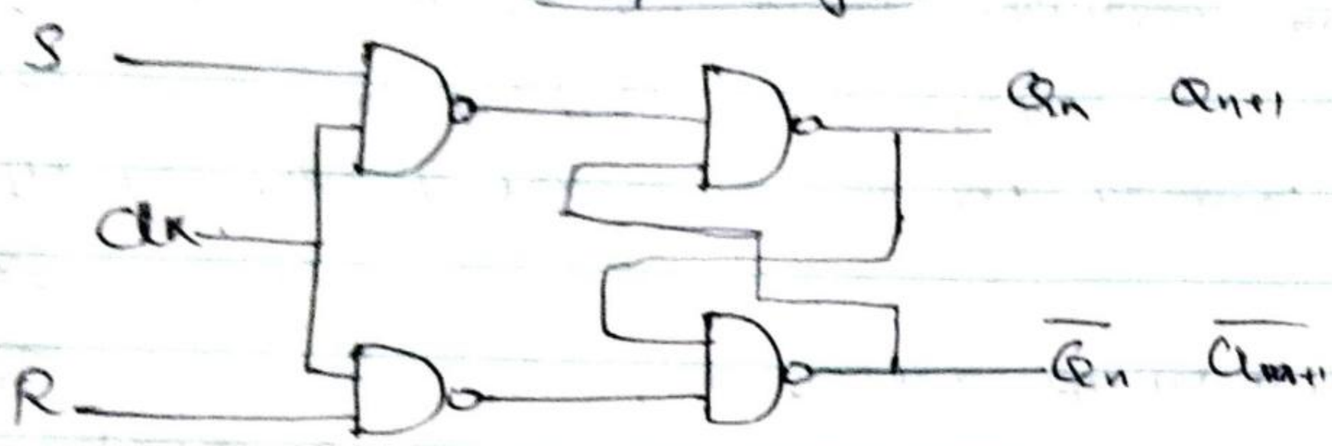
SR Latch → design using NOR gate and Nand gate.

D Latch.

Flip flops : Flip-flop is a circuit that has two stable states and can be used to store state information. Single bit storage devices.

SR Flip flop Set and Reset flipflop

Using nand gate



S, R - I/Ps.

Q_n ⇒ present state

O/P → Q_{n+1}.

Logic Symbol.

PS NS

Truth table

Clk	S	R	Q_n	Q_{n+1}	
↑	0	0	x	Q_n	(No change)
↑	0	1	x	0	Reset condition.
↑	1	0	x	1	Set condition.
↑	1	1	x.	I.D.	

Condition (i) S=0, R=0.

clk is on then clk=1 (to trigger the flip flop)

$$Q_{n+1} = \overline{1 \cdot \bar{Q}_n} = Q_n.$$

(ii) S=0, R=1

$$Q_{n+1} = \overline{1 \cdot \bar{Q}_n} = Q_n.$$

$$\overline{Q_{n+1}} = \overline{0 \cdot \bar{Q}_n} = \bar{0} = 1$$

$$\therefore \Rightarrow Q_{n+1} = 0$$

(iii) S=1, R=1

$$Q_{n+1} = \overline{0 \cdot \bar{Q}_n} = \bar{0} = 1$$

(iv) S=1, R=1

$$Q_{n+1} = \overline{0 \cdot \bar{Q}_n} = 1$$
$$\overline{Q_{n+1}} = \overline{0 \cdot \bar{Q}_n} = 1$$

} not possible. Inter determinate state.

SR Latch

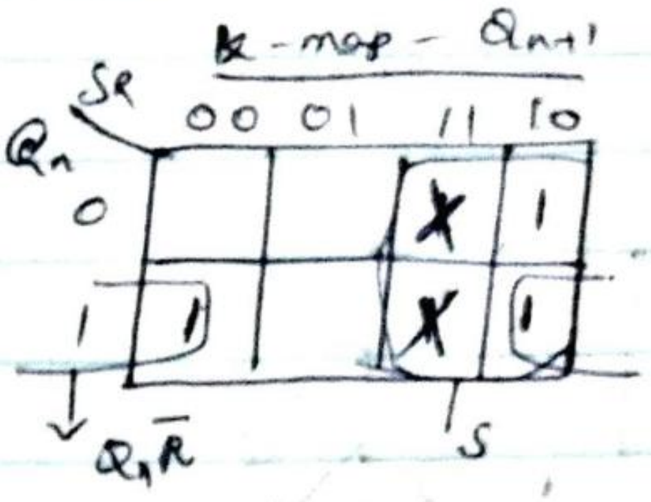
Equation for SR flip flop:

S	R	Q_n	Q_{n+1}	
0	0	0	0	
0	0	1	1	
0	1	0	0	
0	1	1	0	
1	0	0	1	
1	0	1	1	
1	1	0	X	don't care
1	1	1	X	don't care

→ we have to draw K-map for next state.

To get the equation, expand the truth table

K-map - Q_{n+1}



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

SR flop flop

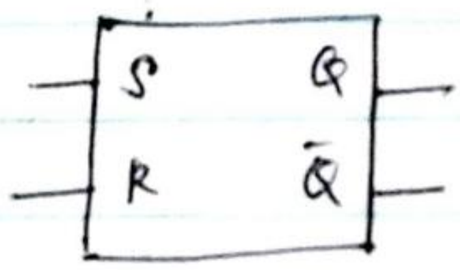
Excitation table

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

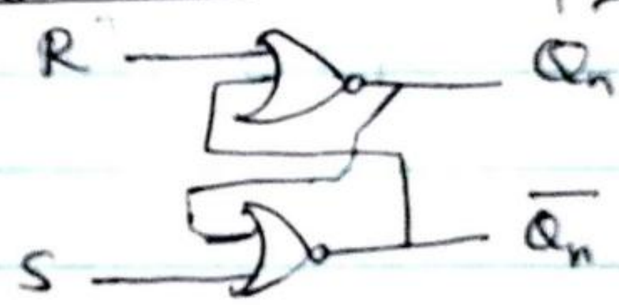
SR Latch

Logic symbol

NOR gate



Logic circuit



Truth table

S	R	Q_n	Q_{n+1}	
0	0	0	0	no change
0	0	1	1	
0	1	0	0	Reset state
0	1	1	0	
1	0	0	1	Set state
1	0	1	1	
1	1	0	?D	?D.
1	1	1	?D	

S	R	Q_n	Q_{n+1}
0	0	x	Q_n
0	1	x	0
1	0	x	1
1	1	x	?D

(i) $S=0 \quad R=0$

$Q_{n+1} = \overline{0 + \bar{Q}_n} = Q_n$

(ii) $S=0 \quad R=1$

$Q_{n+1} = \overline{1 + \bar{Q}_n} = \bar{1} = 0$

(iii) $S=1 \quad R=0$

$Q_{n+1} = \overline{0 + \bar{Q}_n} = Q_n$

$\overline{Q_{n+1}} = \overline{1 + Q_n} = 0$

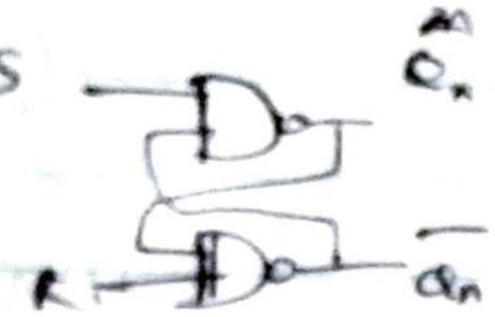
$\Rightarrow Q_n = 1$

(iv) $S=1 \quad R=1$

$Q_{n+1} = \overline{1 + \bar{Q}_n} = 0$

$Q_{n+1} = \overline{1 + Q_n} = 0$

NAND gate



S	R	Q	$\bar{Q}$
0	0	x	ID
0	1	1	Set
1	0	0	Reset
1	1	x	no change

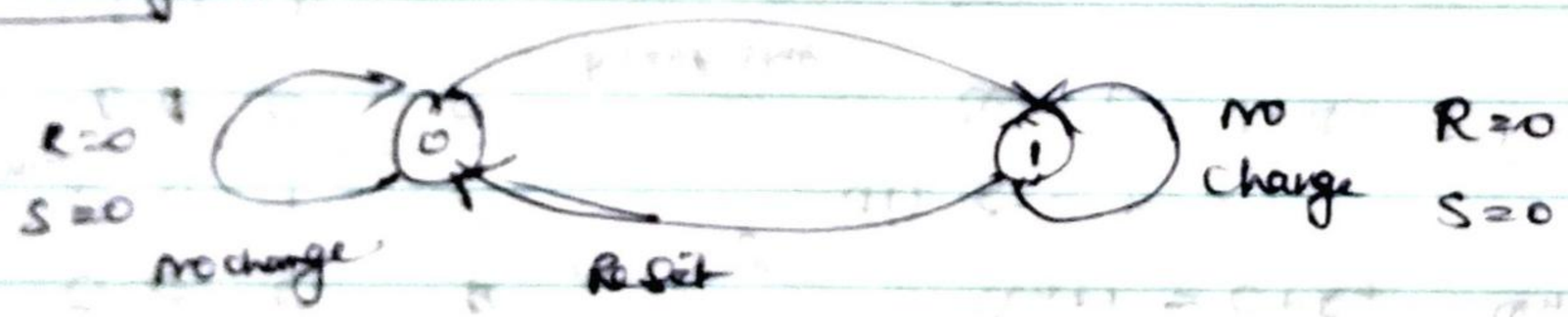
State table

The relationship that exists among the I/Ps, O/Ps, present states and next states can be specified by either state table or state diagram.

→

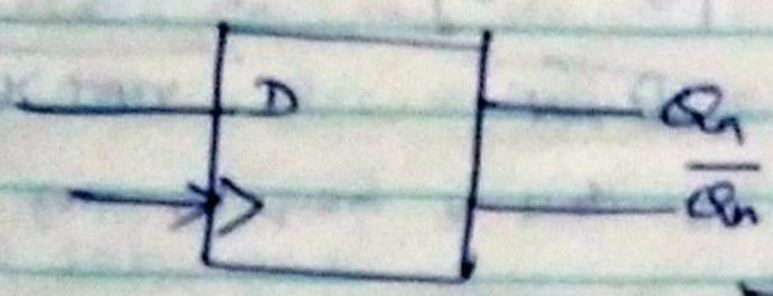
State diagram

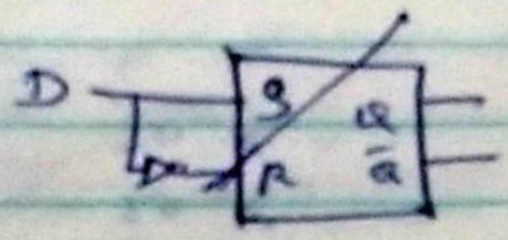
R=0 S=1
Set



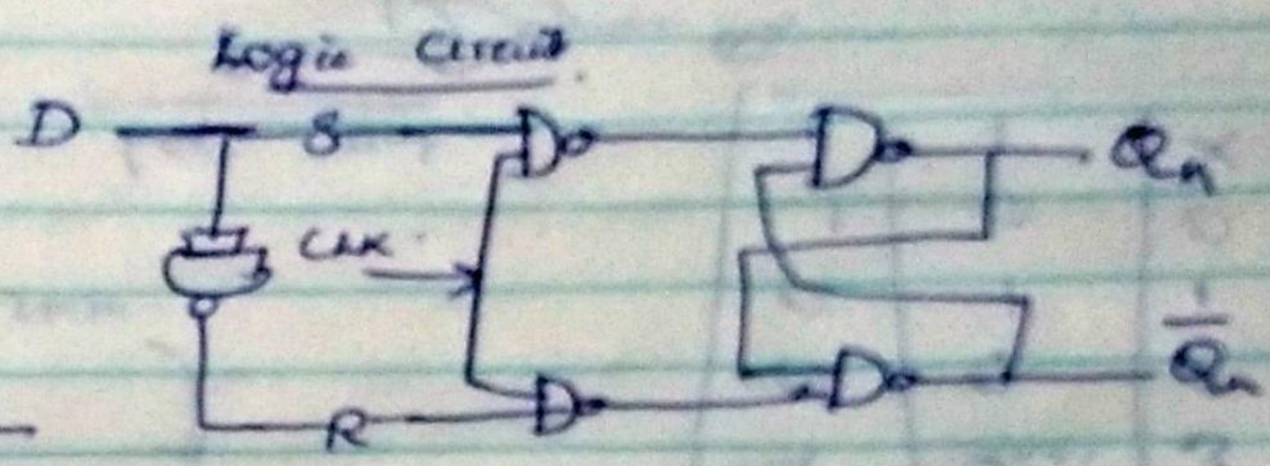
D Flipflop using NAND gate

Block diagram





Logic circuit



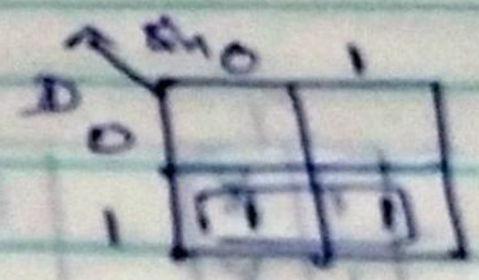
Truth table

clk	D	Q_n	Q_{n+1}
↑	0	x	0
↑	1	x	1
0	x	x	No change

State table

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

Characteristic equation :



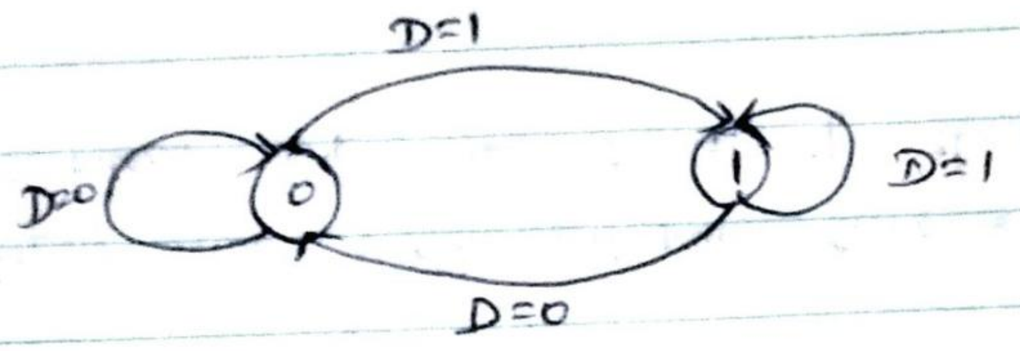
$Q_{n+1} = D$.

∴ Transparent flipflop

D flipflop is delay flipflop
or ~~Data flipflop~~.

It is also called as DATA flipflop as it stores a bit of data. ie the input data applied at the input D, it changes the output state according to input and remains in the same state until the input changes.

State diagram.



Excitation table.

PS. Q_n	NS. Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

D → It also known as delay flip flop. because it can be used to introduce a delay in the digital circuit by changing the propagation delay of the flipflop.

Excitation table : Excitation table shows the minimum inputs that are necessary to generate a particular next state. That is to excite it to the next state. when the current state is known.

→ It is also used to convert one flip flop to another.

Application // Level triggered D Flip flop The output changes
when the clock level is high and it remains in the
~~JK flipflop~~ same state when the clock level goes low. This is called
D Latch and it is not normally used configuration

Edge triggered D Flip flop : The output changes when the clock goes from

low to high, or high to low. If the flip flop triggers for the clock low to high transition then it is called positive edge triggering and if it triggers for the clock high to low transition then it is called negative edge triggering.

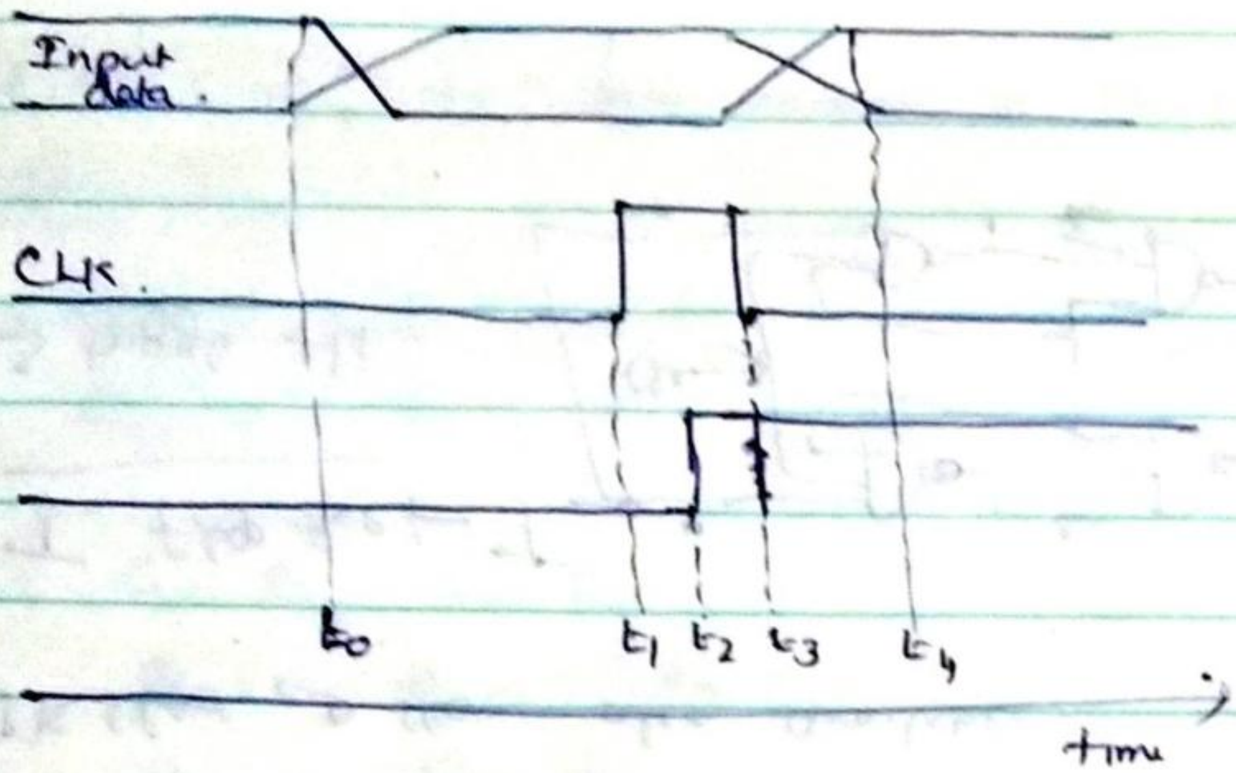
Propagation delay :- It is the duration of time required for an output of flip flop to change its states after getting changes in the input That is turning ON or OFF time leads to delay in the output of flip flops called propagation delay. This is very small duration and it will be in nano seconds.

(eg) In data sheet of any flip flop, it will be written as propagation delay t_p is 10ns.

→ means the flip flop will take 10ns to change its output after getting changes in the input.

Set up time :- The input terminals take some time to stabilise at the terminal due to stray capacitances and other factors. The applied voltage appears across the input terminal after stabilisation. This time taken to stabilise the input signal is called set up time.

Hold time :- Hold time is the minimum amount of time that the input data must be available after the arrival of clock pulse.



Setup time t_s = $t_1 - t_0$

Hold time $t_h = t_3 - t_1$

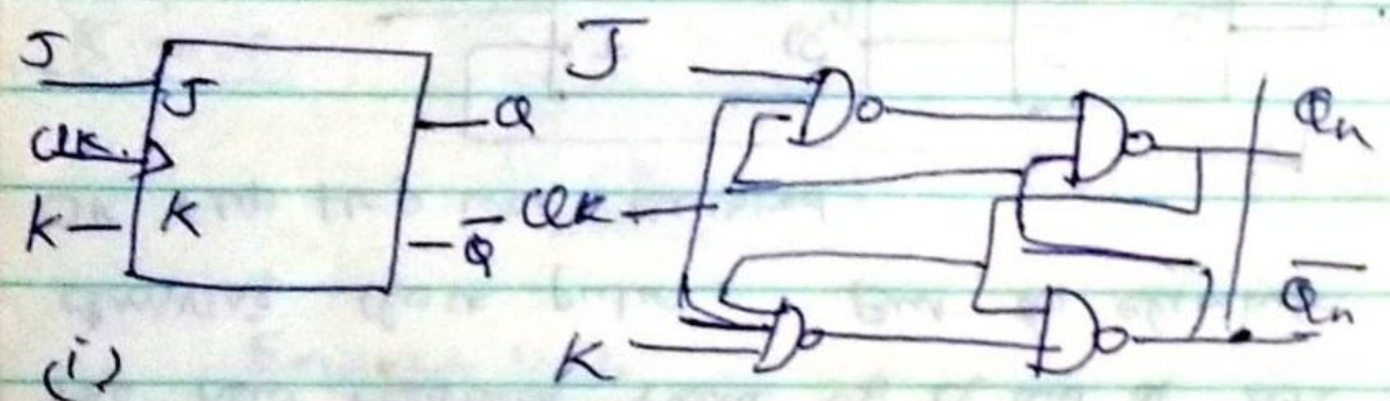
Propagation delay $t_p = t_2 - t_0$

Next data in at t_4.

JK Flipflop:

→ One time I going to write. To we this 2D state, we are going for JK flop flop.

→



J	K	PS Q_n	NS Q_{n+1}	
0	0	x	Q_n.	no change
0	1	x	0	Reset
1	0	x	1	Set
1	1	x.	$\bar{Q}_n$	— toggle condition.

(i)

$J=0$ $K=0$

$Q_{n+1} = 1 \cdot \bar{Q}_n = Q_n$

$\overline{Q_{n+1}} = \overline{1 \cdot Q_n} = \bar{Q}_n$

(i) $J=0$ $K=1$

$Q_{n+1} = 1 \cdot \bar{Q}_n = Q_n$

$\overline{Q_{n+1}} = \overline{\bar{Q}_n \cdot Q_n} = \bar{0} = 1$

(ii) $J=1$ $K=0$

$Q_{n+1} = \overline{Q_n \cdot \bar{Q}_n} = \bar{0} = 1$

$\overline{Q_{n+1}} = 0$

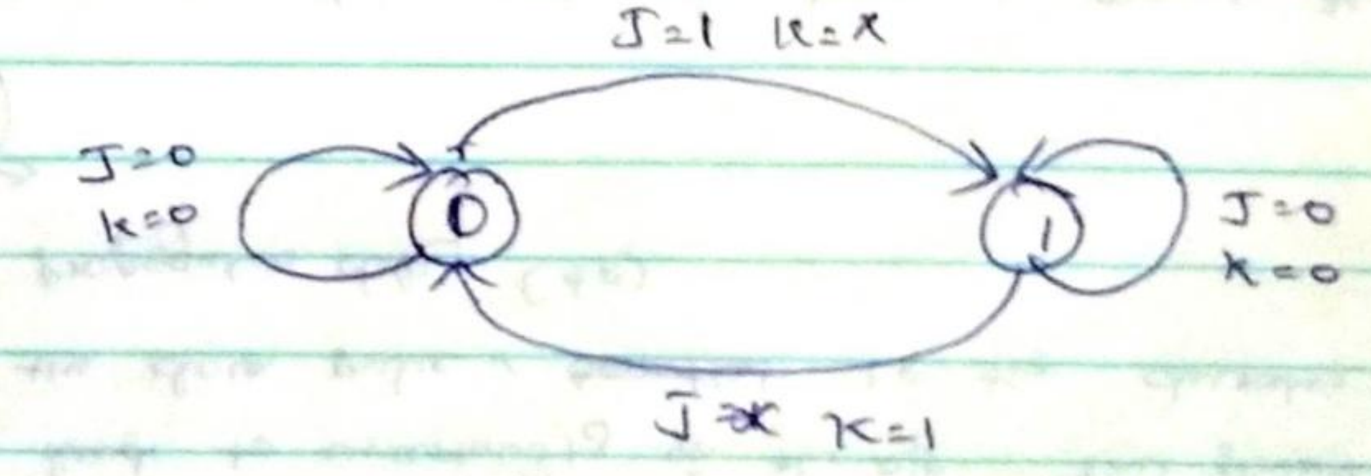
(iv) $J=1$, $K=1$

$Q_{n+1} = \overline{Q_n \cdot \bar{Q}_n} = \bar{0} = 1$

$\overline{Q_{n+1}} = \overline{Q_n \cdot \bar{Q}_n} = \bar{0} = 1$

$Q_{n+1} = Q_n \cdot \bar{1} = \bar{Q}_n$

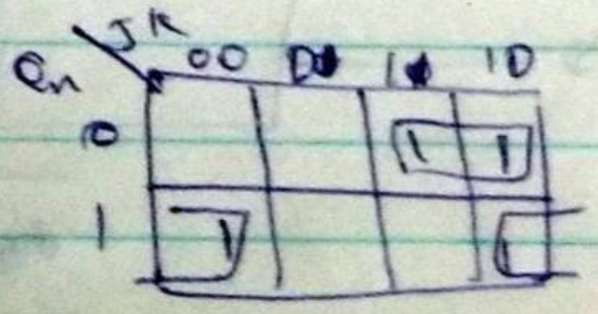
State diagram



Excitation table

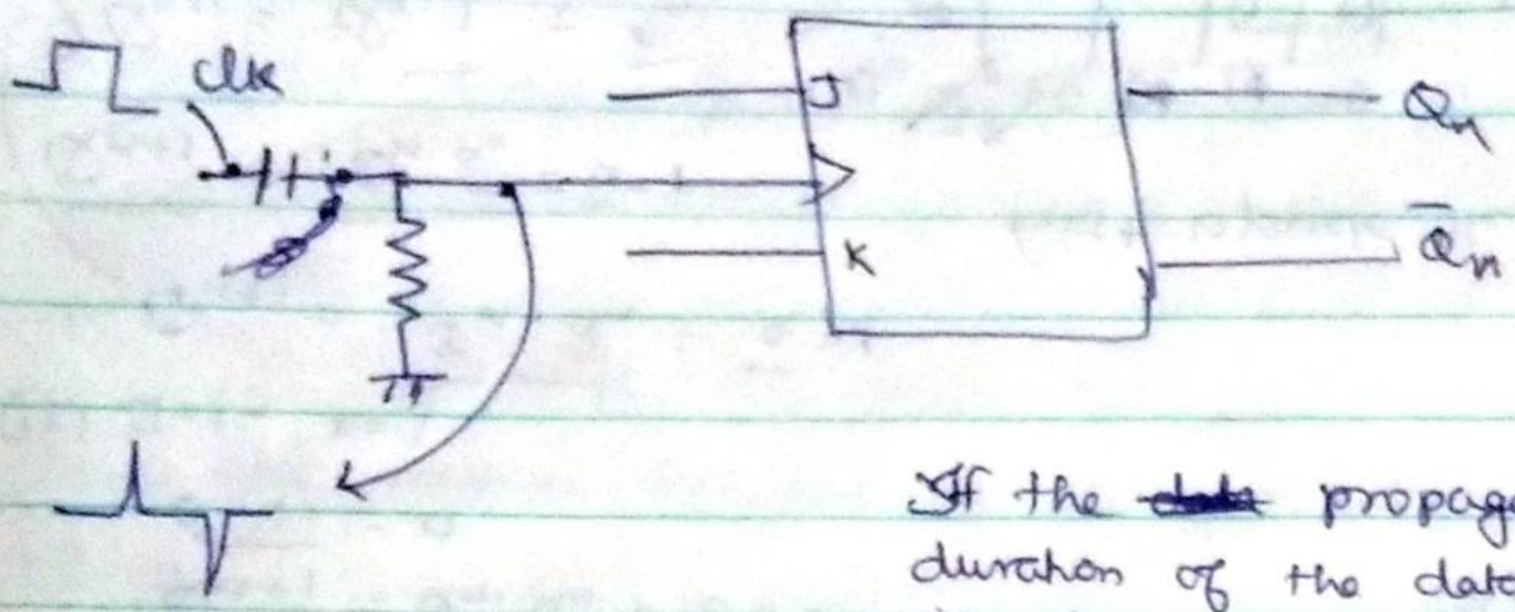
Q_n	Q_{n+1}	J	K.
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

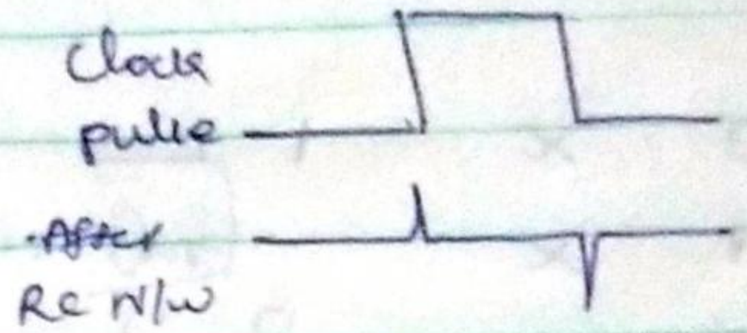
first operate this Q_{n+1} gate.



$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$.

Edge Triggered JK flipflop

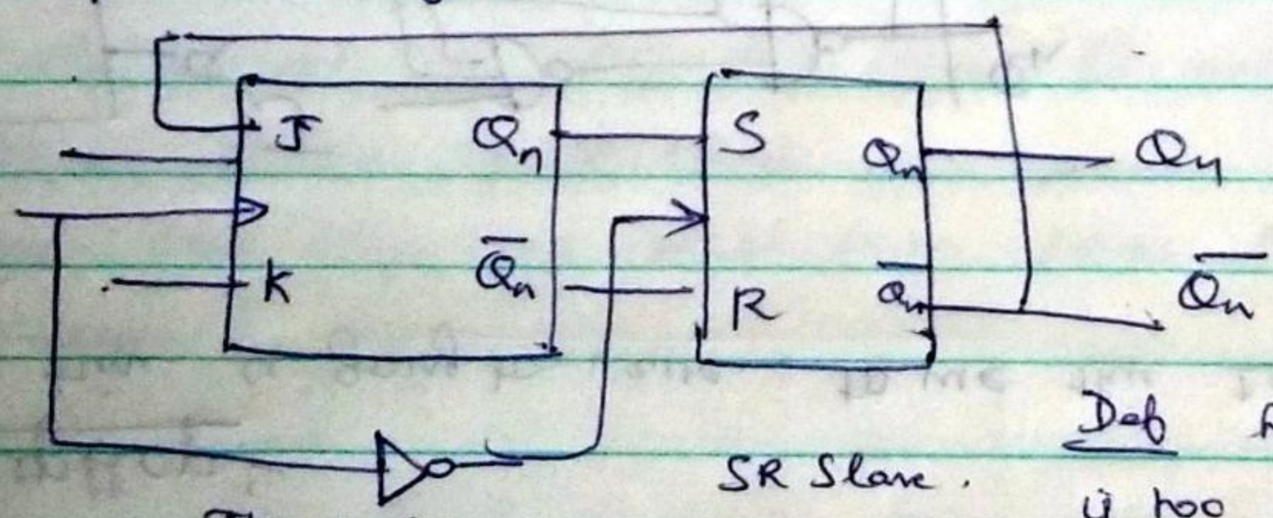


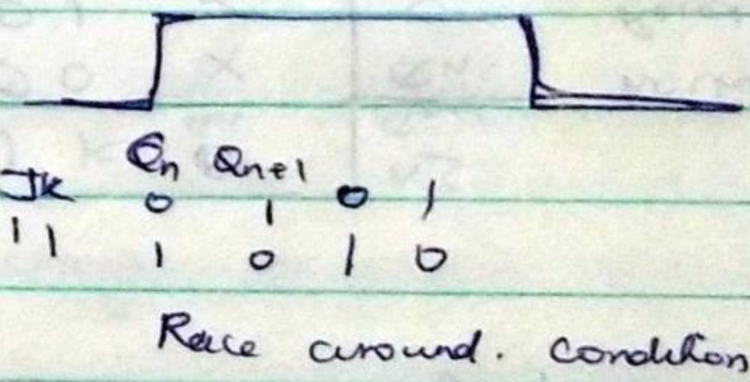


If the ~~data~~ propagation delay is less than the duration of the data available at the input, than there will be one more toggling effect occurs in the JK f/f. This leads to uncertainty in the o/p. This problem ~~Master Slave flipflop.~~ can be eliminated by narrow down the clock pulse, ~~that~~ i.e the duration of the clock pulse is less than the propagation delay (t_{pd})

Racing or Race around condition:-

The proper combination of propagation delay and edge triggering prevents the racing problem. Suppose the output change after the +ve clock edge arrived, then the new feedback value of Q and $\bar{Q}$ are too late to coincide with positive driving clock pulse. But to eliminate this racing completely, Master slave JK flip flop was designed.



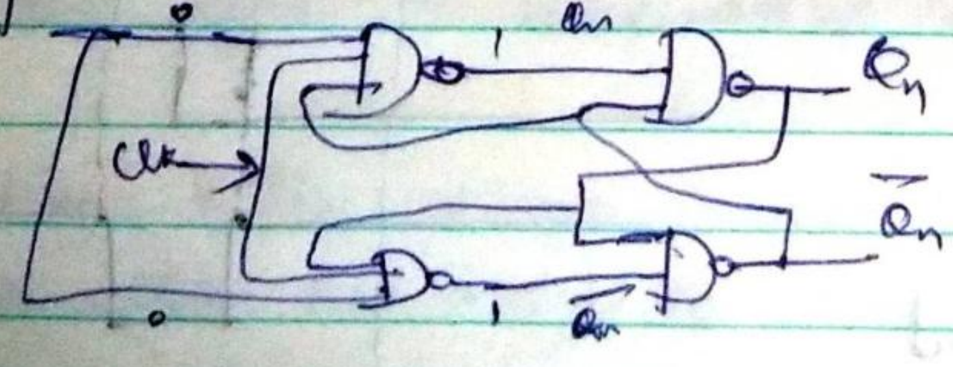


Race around condition.

Def for JK f/f if J=K=1 & if clk is too long, then state of f/f keep on toggle which leads to uncertainty in determining output state of f/f. This problem is called Race around condition.

JK Slave, D slave also available.

T flip flop ← T



⇒ toggle f/f.

If we want only to toggle condition in JK flf.

Condition $T=0$

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

$T=1$ $\quad G_1 = Q_n \quad G_2 = \overline{Q_n}$

$Q_n = 0 \quad Q_{n+1} = \overline{0 \cdot \overline{Q_n}} = 1$

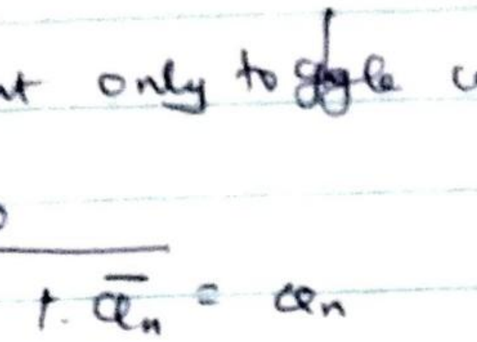
$Q_n = 1 \quad 1 \cdot \overline{Q_n} = 0$

G_4 o/p $\Rightarrow \overline{Q_{n+1}} = \overline{0 \cdot Q_n} = \overline{0} = 1$

$\Rightarrow Q_{n+1} = 0$

Characteristic equation of T flf

K-map



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

Excitation table

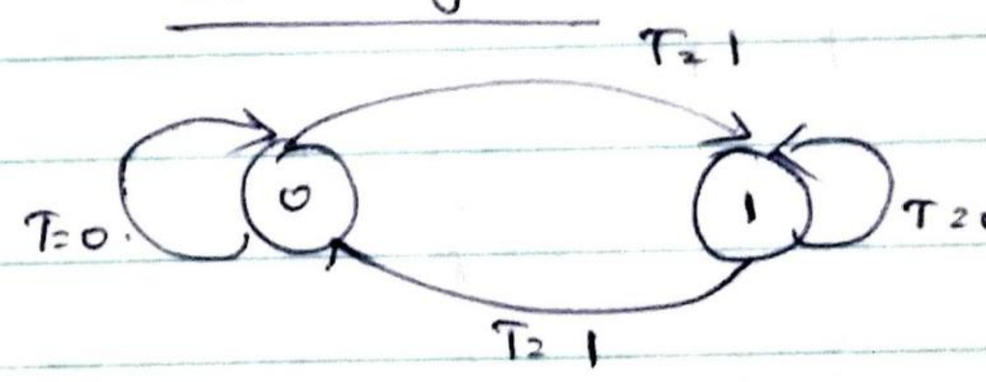
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

T	PS Q_n	NS Q_{n+1}	
0	x	Q_n	no change
1	x	$\overline{Q_n}$	

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

State Diagram



State Reduction

1) Reduce the number of states in the following state table and tabulate the reduced state table. (ii) Starting from state a, and input sequence ~~0111010101~~ 0111001001 1, determine the output sequence for the given and reduced state table.

P.S	Next state. x=0	x=1	output x=0	x=1
a	f	b	0	0 ✓
b	d	c	0	0 ✓
c	f	e	0	0 ✓
d	g	a ←	1	0 *
e	d	c	0	0 ✓
f	f	b	1	1
g	g	h	0	1
h	g	a ←	1	0 *

e = b
h = d

From the given table,
e = b , h = d.

P.S	N.S x=0	x=1	o/p x=0	x=1
a	f	b	0	0 ✓
b	d	c	0	0
c	f	b	0	0 ✓
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

⇓

C = a

Replace c by a, then,

P.S	N.S x=0	x=1	o/p x=0	x=1
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

↓ Reduced State table.

o/p sequence for the ~~reduced~~ given State table.

Input sequence	State transition	output
0	a → f	0
1	f → b	1
1	b → c	0
1	c → e	0
0	e → d	0
0	d → g	1
1	g → h	1
0	h → g	1
0	g → g	0
1	g → h	1
1	h → a	0

O/p sequence for reduced state table.

I/p sequence	State transition	O/p
0	a → f	0
1	f → b	1
1	b → a	0
1	a → b	0
0	b → d	0
0	d → g	1
1	g → d	1
0	d → g	1
0	g → g	0
1	g → d	1
1	d → a	0

State Reduction

1) Reduce the number of states in the following state table and tabulate the reduced state table. (ii) Starting from state a, and input sequence ~~0111010101~~ 0111001001 1, determine the output sequence for the given and reduced state table.

P.S	Next state. x=0	x=1	output x=0	x=1
a	f	b	0	0 ✓
b	d	c	0	0 ✓
c	f	e	0	0 ✓
d	g	a ←	1	0 *
e	d	c	0	0 ✓
f	f	b	1	1
g	g	h	0	1
h	g	a ←	1	0 *

e = b
h = d

From the given table,
e = b , h = d.

P.S	N.S x=0	x=1	o/p x=0	x=1
a	f	b	0	0 ✓
b	d	c	0	0
c	f	b	0	0 ✓
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

⇓

C = a

Replace c by a, then,

P.S	N.S x=0	x=1	o/p x=0	x=1
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

↓ Reduced State table.

o/p sequence for the ~~reduced~~ given State table.

Input sequence	State transition	output
0	a → f	0
1	f → b	1
1	b → c	0
1	c → e	0
0	e → d	0
0	d → g	1
1	g → h	1
0	h → g	1
0	g → g	0
1	g → h	1
1	h → a	0

O/p sequence for reduced state table.

I/p sequence	State transition	O/p
0	a → f	0
1	f → b	1
1	b → a	0
1	a → b	0
0	b → d	0
0	d → g	1
1	g → d	1
0	d → g	1
0	g → g	0
1	g → d	1
1	d → a	0

Counters

→ Counters a a digital device used to count number of pulse & it can also be used as frequency divider.

→ counter can count in two ways ie.

1. Up count (0, 1, 2, ... N) [used in EVM]

2) Down count (N, N-1, ... 1, 0) [Space related application]

→ Present count of the counter represents state of counters

→ counter contains set of flip-flops, A n-bit counter requires n-flip flops & 2^n states.

→ Each state frequency = $\frac{\text{total frequency}}{2^n}$

Classification → 1) Asynchronous counters
[according to clock pulse] 2) Synchronous Counters.

Asynchronous Counters

→ Asynchronous refers to states that doesn't have a fixed time relationship with each other.

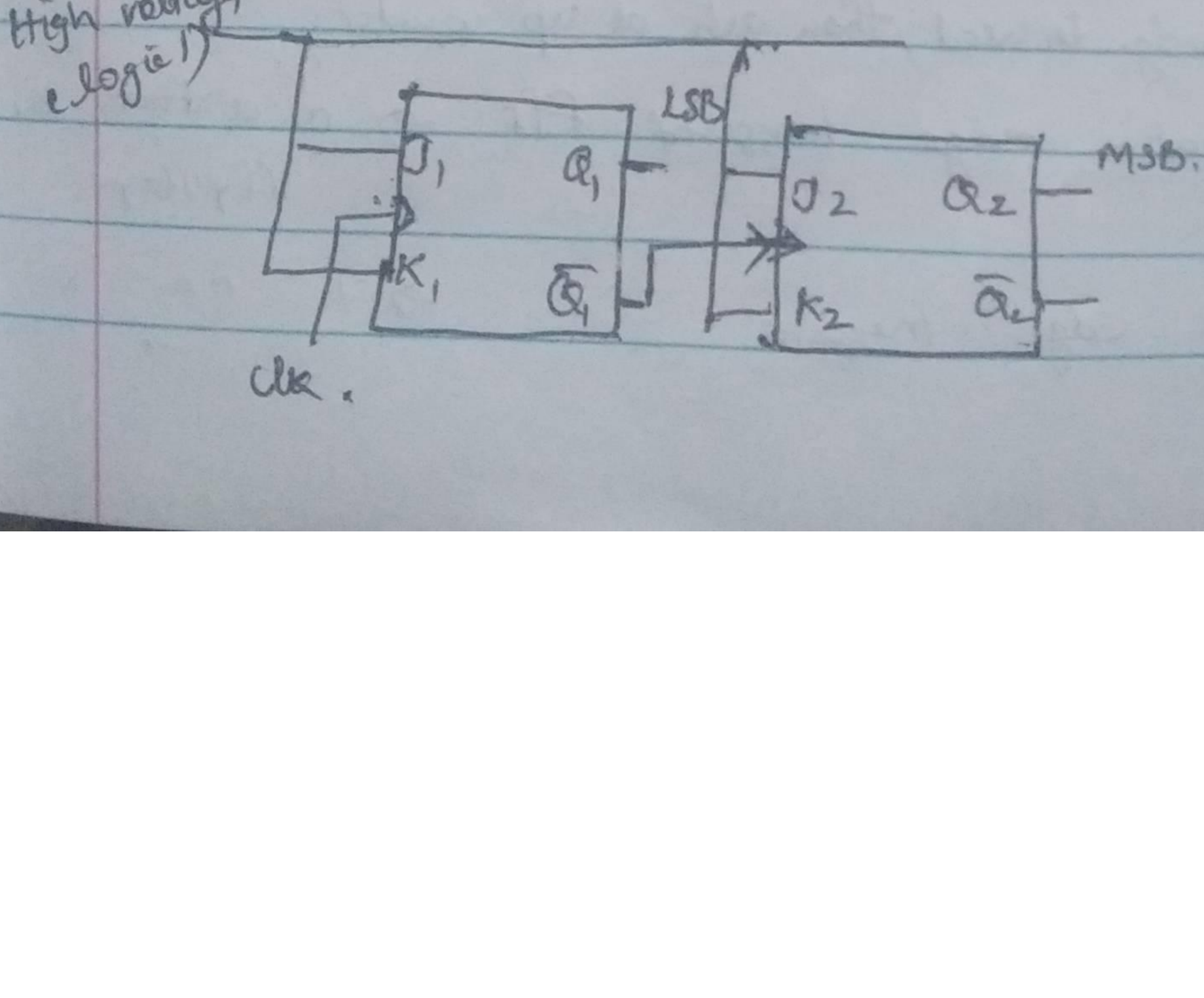
→ In asynchronous counters flip-flops doesn't have a common clock pulse. So their states doesn't change exactly at same time.

→ 1'st flip flop ka clock, output of each flip-flop will acts as clock to the next flip-flop in the counter.

→ eg. Ripple counter.

→ Flip-flops must be in toggle condition. (JK & T flip flop only).

Ripple - up- counter 2-bit Ripple-up-counter.



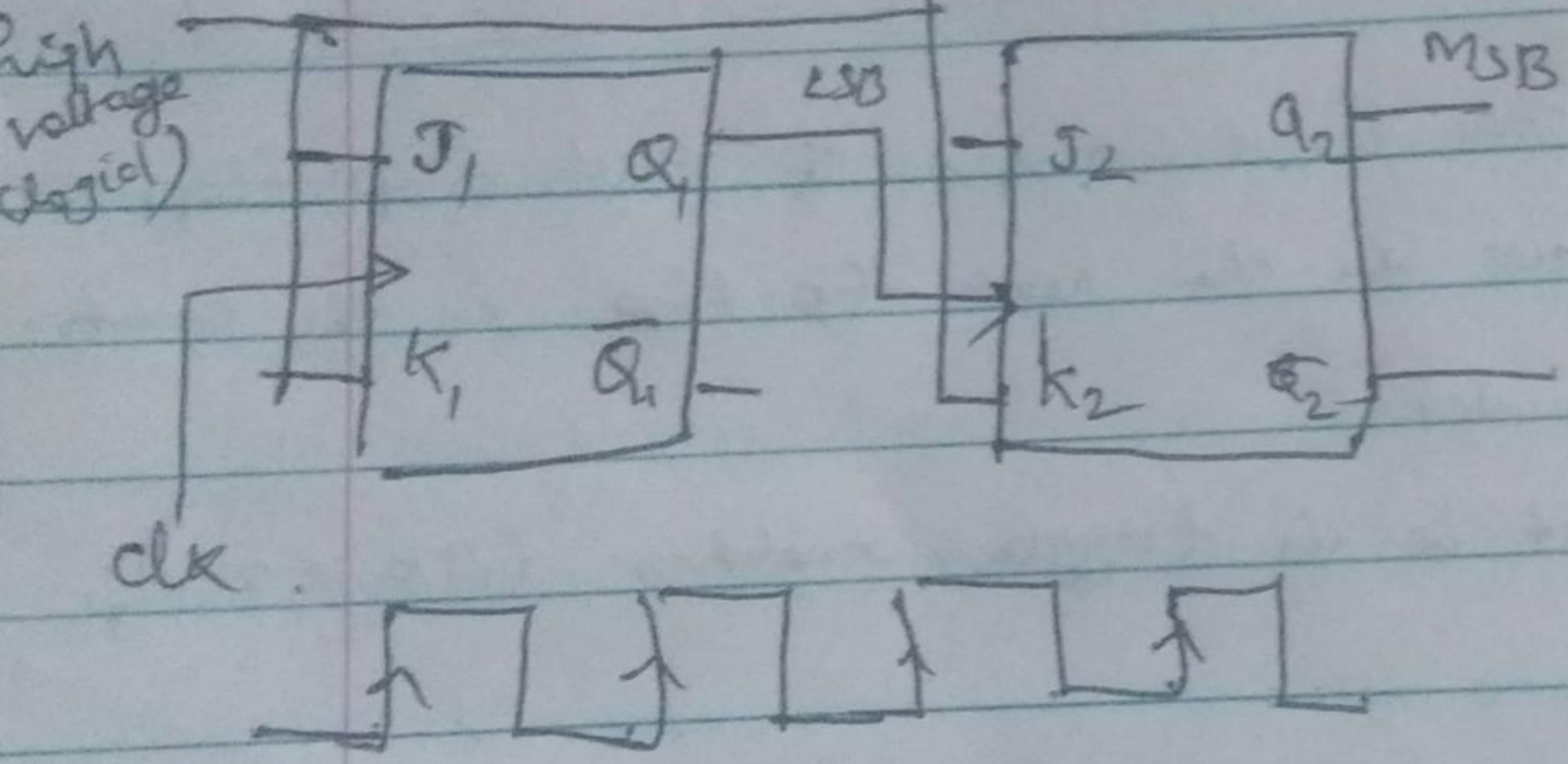
→ Asy

Truth table

	Q_2	Q_1	
0	0	0	
1	0	1	
2	1	0	
3	1	1	
4	0	0	

→ counter resets, shows the completion of counter

Ripple down counter



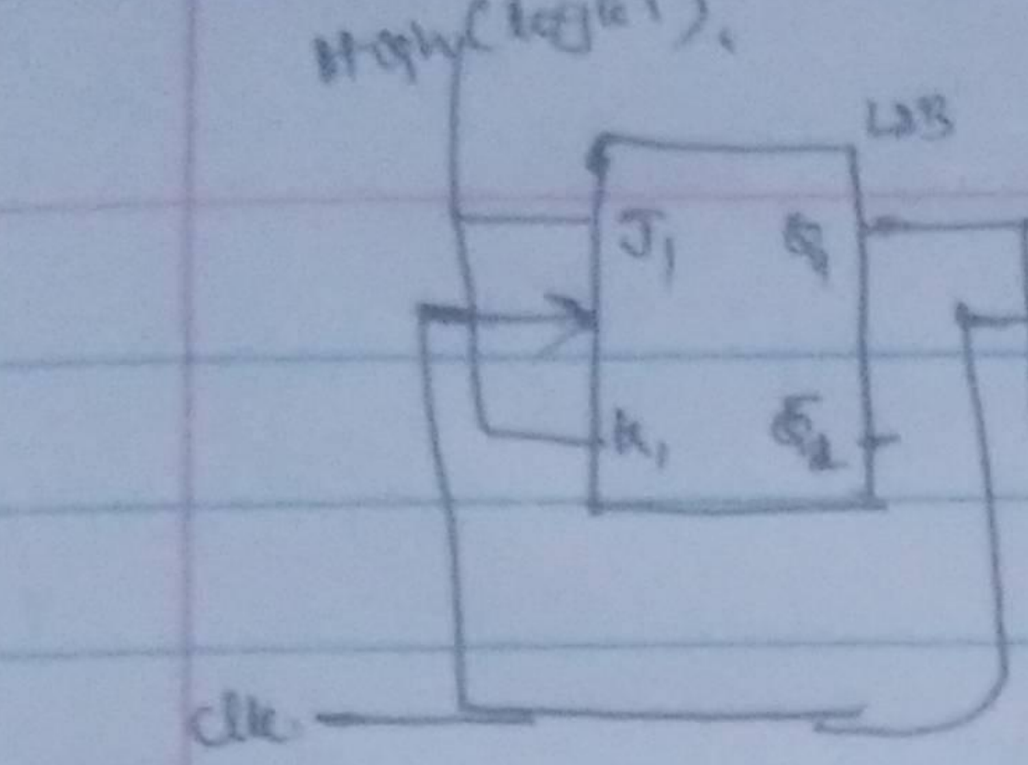
Q_2	Q_1		
0	0	0	
1	1	1	3
2	1	0	2
3	0	1	1
4	0	0	0

If negative edge triggered, then acts as up counter.

→ Negative edge triggered flf → 0 activate the flipflop.

+ve edge triggers → 1

Synchronous counter



clk	Q_2	Q_1
0	0	0
1	0	1
2	1	0
3	1	1
4	0	0

Same circuit for down counting, change the clock to negative edge triggered clock.

(or) with the same clock, give the o/p from $\overline{Q_1}$.