

Digital Circuits :

Number System :

Number System is used to represent the numbers.

A collection of digits makes a number which has two parts integer and fractional, set apart by a radix point.

$$N_b = \underbrace{d_{n-1} d_{n-2} \dots d_1}_{\text{Integer}} \cdot \underbrace{d_0}_{\text{Radix point}} \underbrace{d_{-1} d_{-2} \dots d_{-m}}_{\text{fractional}} \quad \text{MSB} \quad \text{LSB}$$

Where N = a Number

b = radix or base of the number system (r)
= maximum number of digits that can be used in a particular Number System.

n = Number of digit in integer position

m = Number of digit in fractional position

Computer Number System :

① Binary Number System :

→ Base (r) = 2

→ No. of digits used 0 to $r-1$. i.e. (0, 1)

→ digit position has weight in binary

$$[2^n \dots 2^3 2^2 2^1 2^0 2^{-1} 2^{-2} \dots 2^{-n}]$$

② Octal Number System :

→ Base (r) = 8

→ No. of digits used 0 to $r-1$ i.e. (0, 1, ..., 7)

→ digits position has weight

$$[8^n \dots 8^2 8^1 8^0 8^{-1} 8^{-2} \dots 8^{-n}]$$

③ Decimal number System

→ Base(r) = 10

→ No. of digits used 0 to r-1 i.e (0, 1, 2, ..., 8, 9)

→ digit position has weight

$$10^n \dots 10^2 10^1 10^0 \cdot 10^{-1} 10^{-2} \dots 10^{-n}$$

④ Hexadecimal Number System :

→ Base(r) = 16

→ No. of digit used 0 to r-1 i.e (0, 1, ..., E, F), 0 to 15

→ digit position has weight $[16^n \dots 16^2 16^1 16^0 16^{-1}, 16^{-2} \dots 16^{-n}]$

→ $\begin{matrix} 10 & 11 & \dots & 15 \\ \downarrow & \downarrow & & \downarrow \\ A & B & & F \end{matrix}$

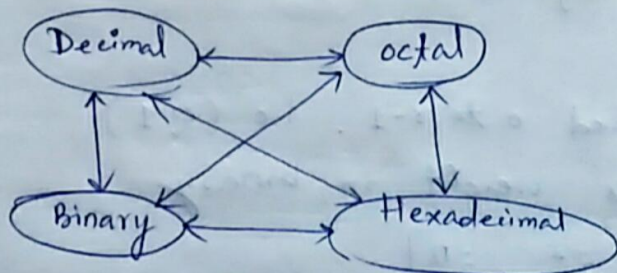
Notes : (1) Group of 4 bits binary Number System called nibble.

(2) Group of 8bit Binary Number system called byte.

(3) Decimal Number System used by humans
Binary Number System used in computers.

Number System Conversion :

Conversion among Bases :



Decimal to Binary Conversion and Vice versa :—

① converts $(13.6875)_{10} = ()_2$

$(11111.10101)_2 = ()_{10}$

Solⁿ

Consider the integer $(13)_{10}$

	13	
2	6	1
2	3	0
2	1	1
	0	1

Diagram labels: "past quotient" points to the 13; "remainder" points to the 1 in the first row; "LSB" points to the 1 in the first row; "MSB" points to the 1 in the last row.

→ first 13 is divided by 2 to give an quotient of 6 and remainder 1.

→ The quotient is again divided by 2 to give a new quotient and remainder.

→ This process is continued until the integer quotient become '0'.

→ write from bottom to top the remainders, we will get the binary equivalent numbers.

$0.6875 \times 2 = 1.3750$	— 1	Top
$0.3750 \times 2 = 0.7500$	— 0	
$0.7500 \times 2 = 1.5000$	— 1	
$1.5000 \times 2 = 1.0000$	— 1	Bottom

→ 0.6875 is multiplied by 2 to give an integer and a fraction.

→ The new fraction is multiplied by 2 to give a new integer and a new fraction.

→ This process is continued until the fraction becomes '0', or until the number of digits have sufficient accuracy.

→ writing integer from top to bottom we get the answers.

→ Considering all the integer part we get the results.

$$(13.6875)_{10} = (1101.1011)_2$$

Reverse Operation :

Binary to decimal conversion :

converts $(11111.10101)_2 \rightarrow ()_{10}$

Integer part :

$$\begin{array}{r} (11111)_2 \\ \begin{array}{l} 1 \times 2^0 = 1 \\ 1 \times 2^1 = 2 \\ 1 \times 2^2 = 4 \\ 1 \times 2^3 = 8 \\ 1 \times 2^4 = 16 \end{array} \\ \hline (31)_{10} \end{array}$$

Fractional part

$$\begin{array}{r} 0.10101 \\ \begin{array}{l} 1 \times 2^{-1} = 1/2 \\ 0 \times 2^{-2} = 0 \\ 1 \times 2^{-3} = 1/8 \\ 0 \times 2^{-4} = 0 \\ 1 \times 2^{-5} = 1/32 \end{array} \\ \hline (0.65625)_{10} \end{array}$$

The digit position in binary number system have weight

$$[2^n \ 2^{n-1} \dots 2^0 \ 2^{-1} \ 2^{-2} \dots 2^{-n}]$$

② Octal to Decimal Conversion & Vice versa :

Convert (a) $(314.734)_8 \rightarrow ()_{10}$

(b) $(3287.687)_{10} \rightarrow ()_8$

① Decimal to octal :

Solⁿ

8	3287	
8	410	→ 7
8	51	→ 2
8	6	→ 3
8	0	→ 6

Top ↑
↓ Bottom

$$(3287)_{10} \leftrightarrow (6327)_8$$

$$\begin{array}{l} 0.685 \times 8 = 5.48 \rightarrow 5 \\ 0.48 \times 8 = 3.84 \rightarrow 3 \\ 0.84 \times 8 = 6.72 \rightarrow 6 \\ 0.72 \times 8 = 5.76 \rightarrow 5 \\ 0.76 \times 8 = 6.08 \rightarrow 6 \end{array}$$

Top ↑
↓ Bottom

$$(0.685)_{10} \leftrightarrow (0.53656\dots)_8$$

③ $(314.734)_8 \rightarrow (\quad)_{10}$

$$\begin{array}{r} 314 \\ \begin{array}{l} 4 \times 8^0 = 4 \\ 1 \times 8^1 = 8 \\ 3 \times 8^2 = 192 \end{array} \\ \hline (204)_{10} \end{array}$$

$$\begin{array}{r} (0.734)_8 \\ \begin{array}{l} 7 \times 8^{-1} \\ 3 \times 8^{-2} \\ 4 \times 8^{-3} \end{array} \\ \hline (0.9296875)_{10} \end{array}$$

$(204.9296875)_{10}$

④ Decimal to hexadecimal and Vice-versa :

(a) $(374.37)_{10} \rightarrow (\quad)_{16}$

(b) $(2B.1FA)_{16} \rightarrow (\quad)_{10}$

⑤ Octal to Binary & Vice-versa :

(a) $(37.12)_8 = (\quad)_2$ (b) $(11011.1011)_2 = (\quad)_8$

Ans (a) $(011\ 111.001010)_2$ (b) $(33.54)_8$

⑥ Hexadecimal to Binary and viceversa :

(a) $(524.36)_{16} \rightarrow (\quad)_2$ (b) $(1010.0111)_2 \rightarrow (\quad)_{16}$

Ans (a) $(010100100100.00110110)_2$ (b) $(A.7)_{16}$

⑦ Convert Hexa to octal & Vice-versa

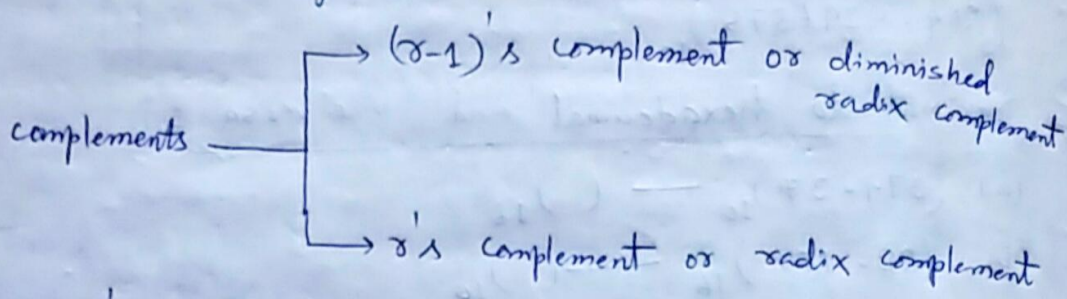
(a) $(1375)_8 = (\quad)_{16}$, (b) $(1375)_{16} = (\quad)_2 = (\quad)_8$

Ans (a) $(2FD)_{16}$ (b) $(1375)_{16}$

(c) $(7FD6)_{16} = (\quad)_8$, $(7FD6)_{16} = (\quad)_2 = (\quad)_8$
 $(7FD6)_{16} = (77726)_8$

Complements :

- * Complements are used in digital systems or computer for simplifying the subtraction operation and for logical manipulation.
- * Basically there are two types of Complements in each number system.



(a) $(r-1)$'s complement :

If 'N' be the number, which complement is to be determined.

r = base of the number system

n = no. of digits required to represent 'N'.

Then $(r-1)$'s complement = $(r^n - 1) - N$

where 'N' is an integer

Notes :

If N is a fractional number having 'm' number of integer digits and 'p' number of fractional bits.

Then $(r-1)$'s complements = $(r^m - r^{-p}) - N$

(b) r's complements :-

$$r's \text{ complements} = (r^n - N)$$

where N = integer number, ~~if~~ which complement is to be determined.

r = base of the number system.

n = number of digits in 'N'.

for binary number system

(a) 1's / $(r-1)$'s (b) 2's / r 's

Binary Number System :

The 1's complement of a binary number is obtained by subtracting each digit from 1.

$$\begin{array}{r} 1111 \\ 1010 \\ \hline 1's \rightarrow 0101 \end{array}$$

let $N = 1010$
 $n = 4, r = 2,$

1's complement of 1010

$$= (r^n - 1) - N$$

$$= (2^4 - 1) - N = [(10000)_2 - 1] - N$$

$$= [1111]_2 - N$$

$$= [1111]_2 - [1010]_2$$

(Q) Find 1's complement

$$\begin{array}{cc} 1100 & 1001 \\ 0011 & 0110 \end{array} \rightarrow 1's$$

* The 1's complement of a binary number is obtained by complementing each bit.

2's complement :

Note: r 's complement is obtained by adding 1

to the $(r-1)$'s complement, i.e. $[(r^n - 1) - N] + 1 = [r^n - N]$.

* 2's complement is defined as the new word obtained by adding 1 to 1's complement.

(Q) Find 2's complement of 0101

$$\begin{array}{r} 0101 \\ 1's \rightarrow 1010 \\ \hline + 1 \\ \hline 2's \rightarrow 1011 \end{array}$$

2's complement of 0101 is 1011

Decimal Number System :

Here 9's & 10's complements are possible.

(r-1)'s complement is 9's complement

Q. Find ~~(8-1)~~ 9's complement of 546700.

$$\begin{aligned} \text{Sol}^n &= (8^n - 1) - N \\ &= (10^n - 1) - N \\ &= (10^6 - 1) - N \\ &= 999999 - 546700 = 453299 \end{aligned}$$

Note: 9's complement of a decimal number is obtained by subtracting each digit from 9.

* The (r-1)'s complement of octal & hexadecimal number is obtained by subtracting each digit from 7 or F (15) respectively.

10's complement :

Find 10's complements of 2389.

$$\begin{array}{r} 9's \text{ complement of } 2389 \rightarrow 7610 \\ \hline + 1 \\ \hline \end{array}$$

$$10's \rightarrow 7611$$

Note :

10's complement of 'N' can be formed also by leaving all least significant 0's unchanged, subtracting the 1st non zero least significant digit from 10 and subtracting all higher significant digits from 9.

Find 10's complement of 246700

$$\begin{array}{r} 1111^{10} \\ 246700 \\ \hline 753300 \end{array} \leftarrow 10's$$

Find 2's complement of 1101100 & 0110111

$$\begin{array}{r} 1111^2 \\ 1101100 \\ \hline 0010100 \end{array} \leftarrow 2's$$

$$\begin{array}{r} 1111^2 \\ 0110111 \\ \hline 1001001 \end{array} \leftarrow 2's$$

Note:

* Complement of Complement is equal to the original number. $x's = (x^n - N)$

$$= x^n - (x^n - N) = N$$

1's complement of (1's complement of x) = x

2's complement of (2's complement of x) = x

Representation of Signed Binary Number :-

Signed-magnitude form (used in ordinary arithmetics)

Signed-complements form

1's Complement form

2's complement form

(used in computers)

- In ordinary arithmetics negative number is indicated by minus sign and a positive number is indicated by +ve sign. $(-9, +9)$

→ Because of hardware limitation, computer must represents every things with binary digits.

① Signed Magnitude form :

- ★ a bit placed in the leftmost position of the number that represents sign.
- ★ the sign bit '0' for positive & 1 for negative.
- ★ if the binary number is signed, then the leftmost bit represents sign and rest of the bits represents the number.
- ★ if the binary number is unsigned then the leftmost bit represents the MSB of the number.

e.g. $\left[\begin{array}{l} \text{Unsigned binary} \rightarrow 01001 \rightarrow 9 \\ \text{signed binary} \rightarrow \boxed{0}1001 \rightarrow +9 \end{array} \right.$

↑
Sign bit

[Unsigned binary $\rightarrow$ 1 1 0 0 1 $\rightarrow$ 25
Signed binary $\rightarrow$ 1 1 0 0 1 $\rightarrow$ -9
 $\nearrow$
sign bit

Signed magnitude $\rightarrow$ ^{sign bit} symbol (+ or -) as a bit (0 or 1)
This is the representation of signed numbers used in ordinary arithmetics.

* When arithmetic operation are implemented in computer we used different system called signed complements system for representing negative numbers.

⑧ Signed complements form :

③ Signed Complements form :

* In this system negative number is indicated by its complements and positive number is same as Binary Number.

as binary
* whereas the signed magnitude system negates a number by changing its sign, the signed complement system negates a number by taking its complements, i.e. 1's or 2's.

e.g. Number '9' represented in binary with 3 bit

signed Magnitude form : $+9 \rightarrow 00001001$
↑
signed bit

signed bit

$$-9 \rightarrow 10001001$$

signed bit

1's complement form:

$+9 \rightarrow 0000\ 1001$

$\downarrow 1's$

$-9 \rightarrow 11110110$

2's Complement form:

$+9 \rightarrow 0000\ 1001$

2's

$$-9 \rightarrow 11110111$$

2's complement form:

Q.1 6 bit representation of $+26$ & -26 in all forms.

$$\underline{+26}$$

Unsigned $\rightarrow$ binary of 26 $\rightarrow 011010$

signed $\rightarrow$ $\left\{ \begin{array}{l} \text{sign mag of } +26 \rightarrow \underline{0}11010 \\ 1\text{'s rep. of } +26 \rightarrow 011010 \\ 2\text{'s rep. of } +26 \rightarrow 011010 \end{array} \right.$

representation of (-26) in all form

Signed magnitude $\rightarrow 1 \overset{\text{Sign bit}}{11010} \rightarrow [-26]$

1's rep. of $(-26) \rightarrow 100101$

2's rep. of $(-26) \rightarrow 100110$

Note:

* The positive numbers in all 3 representation are identical and have zero in the leftmost position.

* If leftmost bit '1' that indicates it's a negative number, it may be in any form.

(Q). Each of the following number is a signed binary number. Determine the decimal value in each case if they are in (1) sign magnitude form (2) 1's complement form (3) 2's complement form.

	<u>SM Representation</u>	<u>1's Comp</u>	<u>2's Comp</u>
(1)	01101 $\rightarrow +13$	+13	+13
(2)	010111 $\rightarrow +23$	+23	+23
(3)	10111 $\rightarrow -7$	-8	-9
(4)	1101010 $\rightarrow -42$	-21	-22

(3) $\frac{1}{\text{Sign}} \frac{0111}{\text{magnitude}} = -7$

1's comp
01000
 $\downarrow$
-8

2's comp
01001
 $\downarrow$
-9

(Q) Express -45 in 8 bit 2's complement form.

Solⁿ +45 in 8 bit

$$+45 \rightarrow 00101101$$

↓ 2's

$$11010011$$

11010011 is the 2's complement form of -45.

(Q) Express -73.25 in 12 bit 2's complement form.

$$+73.25 = 01001001.1100$$

↓ 1's

$$10110110.0011$$

$$+ 1$$

$$10110110.0100 \leftarrow$$

is the 2's Complement form of negative number -73.25.

Subtraction : (using complements)

Subtract 14 from 46 using the 8 bit 2's complement arithmetic.

$$+14 \rightarrow 00001110$$

↓ 2's

$$-14 \rightarrow 11110010$$

* In the 2's complement

subtraction add the 2's

complement of the subtrahend

to the minuend. If

there is a carry out,

ignore it.

$$+46 \rightarrow 00101110$$

$$-14 \rightarrow 11110010$$

$$\textcircled{1} 00100000$$

↑ ignore the carry

* Look at the sign bit i.e. MSB of the sum term.

If MSB is a 0 the result is +ve & is in true

binary form.

If the MSB is a 1 (Whether there is carry or no carry at all) the result is negative and is in its 2's complement form. Take its 2's complement to find its magnitude in binary.

In the above example, there is a carry, so ignore it. The MSB is 0 so, the result is positive and is in normal binary form. Therefore result is

$$+00100000 = +32$$

(Q) Add -75 to +26 using 8bit 2's complement arithmetics.

$$+26 \rightarrow 00011010$$

$$-75 \rightarrow 10110101$$

$$\begin{array}{r} 00011010 \\ + 10110101 \\ \hline 11001111 \end{array}$$

$$+75 \rightarrow 01001011$$

$$-75 \rightarrow 10110101 \text{ (in 2's complement form)}$$

(MSB is 1 & No carry)

The result is negative and is in 2's complement form.

The magnitude is 2's complement of (11001111) is

(00110001) $\rightarrow$ 49. The result is -49.

Subtraction using 1's complement:

(Q) Subtract 14 from 25 using 8bit 1's complement arithmetic.

$$25 \rightarrow 00011001$$

$$-14 \rightarrow 11110001 \text{ (1's comp form)}$$

(Add end around carry)

$$\begin{array}{r} 00001010 \\ + 1 \\ \hline 00001011 \end{array}$$

The result is (+11)₁₀.

(Q) Add -25 to $+14$ using 8 bit 1's complement arithmetic.

$$+14 \rightarrow 0000 \ 1110$$

$$-25 \rightarrow 1110 \ 0110 \text{ (1's comp. form)}$$

(No carry)

$$1111 \ 0100$$

There is no carry and MSB is a 1, so result is negative and in its 1's complement form.

So 1's complement of $1111 \ 0100$ is $0000 \ 1011$.

The result is $(-11)_{10}$.

* In 1's complement subtraction add the 1's complement of subtrahend to the minuend, if there is a carry out bring the carry around and add it to the LSB. This is called the end around carry.

* Look at sign bit (MSB) if this is 0, the result is positive and is in true binary form.

* If the MSB is a 1 (whether there is carry or no carry at all) the result is negative and is in its 2's complement form. Take its 2's complement to find its magnitude in binary.

Books :

Fundamentals of digital circuits by

A. Anand Kumar, PHI.

BINARY CODE :

Code is a symbolic representation of discrete information, which may be present in the form of numbers, letters or physical quantities, special character.

Various codes :

- ① Weighted code ② Non-weighted code
- ③ Error detecting codes ④ Error correcting code
- ⑤ Alphanumeric code.

① Weighted code :

weighted binary codes have definite weights assigned for each bit position. The bits are multiplied by the weights indicated and sum of these weight give equivalent decimal numbers.

positive weighted code $\rightarrow$ 8421, 2421, 5211, 3321, 4311

negative weighted code $\rightarrow$ 642-3, 631-1, 84-2-2, 74-2-1

e.g. 8421 code / BCD code : Binary Coded Decimal (BCD) #

* # is a combination of 4 binary digits that represents decimal numbers.

* 4 bit code for 1 decimal digits.

* A number with K decimal digits will require 4K bits in BCD.

* BCD code means 8421 code. where 8, 4, 2, 1 are weights of 4 bits respectively.

* For the decimal numbers greater than 9, each digits in decimal is represented as 4 bit in BCD.

e.g. Decimal - 396 in BCD with 12 bits

0011 1001 0110

* A decimal number ~~in~~ in BCD is the same as its equivalent binary number only when the number is between 0 & 9.

* BCD numbers are decimal numbers but only difference is decimal numbers written with symbol (0, 1, ..., 8, 9) & BCD numbers use binary code (0000, 0001, ..., 1001)

e.g. 3 6 5 → decimal

/ | \

0011 0110 0101 → BCD

9 5 — decimal

/ \

1001 0101 — BCD

Four different Binary codes for decimal digits:

Digits	BCD/8421	2421	Excess-3	84-2-1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
10	1010	0101	0000	0001
11	1011	0110	0001	0010
12	1100	0111	0010	0011
13	1101	1000	1101	1100
14	1110	1001	1110	1101
15	1111	1010	1111	1110

e.g. (a) 0110 → interpreted as decimal 2 in 8, 4, -2, -1 code

$$= 0 \times 8 + 1 \times 4 + 1 \times -2 + 0 \times -1 = 2$$

(b) 1011 → in 2421 code

$$1 \times 2 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 5$$

Non-weighted code :-

Gray Codes :

1 bit	2 bit	3 bit	4 bit
0	0 0	0 0 0	0 0 0 0 — 0
1	0 1	0 0 1	0 0 0 1 — 1
	1 1	0 1 1	0 0 1 1 — 2
	1 0	0 1 0	0 0 1 0 — 3
		1 1 0	0 1 1 0 — 4
		1 1 1	0 1 1 1 — 5
		1 0 1	0 1 0 1 — 6
		1 0 0	0 1 0 0 — 7
			1 1 0 0 — 8
			1 1 0 1 — 9
			1 1 1 1 — 10
			1 1 1 0 — 11
			1 0 1 0 — 12
			1 0 1 1 — 13
			1 0 0 1 — 14
			1 0 0 0 — 15

Binary to Gray Conversion :

n bit binary number $\rightarrow B_n B_{n-1} \dots B_1$

its Gray code equivalent $\rightarrow G_n G_{n-1} \dots G_1$

B_n, G_n are MSB bits

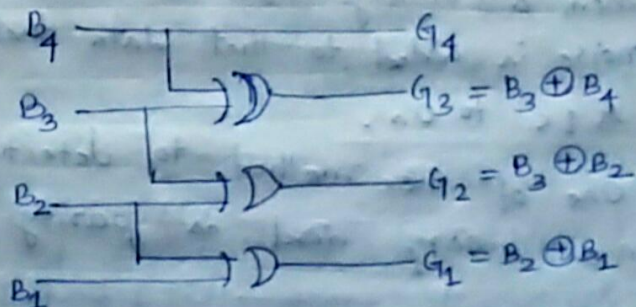
Gray code bits are obtained from binary code

$$G_n = B_n$$

$$G_{n-1} = B_n \oplus B_{n-1}$$

$$G_{n-2} = B_{n-1} \oplus B_{n-2}$$

$$G_1 = B_2 \oplus B_1$$



(for $n=4$)

Gray to Binary Conversion:

(for $n=4$)

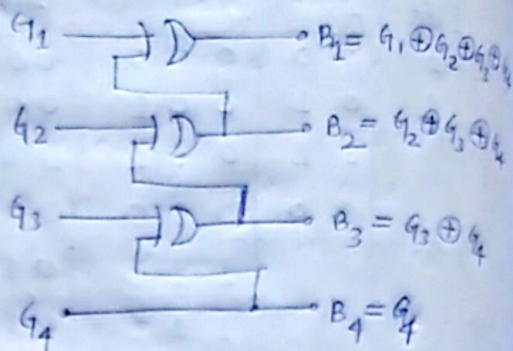
$$B_n = G_n$$

$$B_{n-1} = G_{n-1} \oplus G_n$$

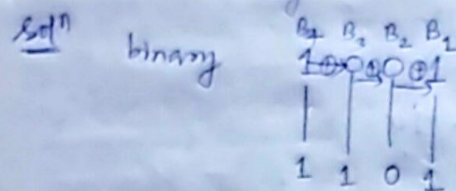
$$B_{n-2} = G_{n-2} \oplus G_{n-1} \oplus G_n$$

...

$$B_1 = G_1 \oplus G_2 \oplus \dots \oplus G_n$$



(p) Convert binary to Gray 1001.



(q) Convert Gray to Binary code



Code for serial data transmission and storage:

(1) Error detecting Code:

When a binary data is transmitted and processed, noise is added to that data that can alter 0s to 1s & 1s to 0s.

One of the methods to detecting single bit error in a binary word is parity check.

Parity checker:

The simplest technique for detecting error is that of adding an extra bit known as the parity bit to each word being transmitted.

Two types of parity check $\left\{ \begin{array}{l} \rightarrow \text{odd parity method} \\ \rightarrow \text{even parity method} \end{array} \right.$

odd parity:

For odd parity the parity bit is set to a 0 or 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an odd number.

even parity:

For even parity the parity bit is set to a 0 or 1 at the transmitter such that the total number of 1 bits in word including parity bit is an even number.

Odd & Even parity:

	odd parity	even parity
0 0 1 0	0	1
0 1 0 1	1	0
0 1 1 1	0	1

When digital data received a parity checking circuit generates an error signal if the total number of 1's is even in an odd parity system or odd in an even parity system.

This parity check can detect a single bit error but can't detect two or more errors within the same word.

Error correcting code:

If the correct code word can always be deduced from an erroneous word. e.g. Hamming code - single bit error detecting and correcting code)

Alphanumeric Code

The code that represents numbers, alphabetic letters and special symbols are called alphanumeric code.

A complete and adequate set of necessary characters includes (i) 26 lower case letters (ii) 26 upper case letters (iii) 10 numeric digits (iv) 25 special characters as + . / # ... etc. These total up to 87 symbols.

* Each character is represented by 7 bit code.

(8th bit is the parity).

ASCII Code : (7 bit)

(American standard code for Information Interchange)

A standard code that has been accepted by the industry, the ASCII used in most μ -computers by its manufacturers.

The ASCII code represents characters with 7 bits which can be stored as one byte with 1 bit unused.

e.g.	character	7 bit ASCII
	A	100 0001
	B	100 0010
	:	
	Blank	010 0000
	+	010 1011
	3	011 0011
	9	011 1001

EBCDIC code :

- * (Extended binary coded Decimal Information Code)
- alphanumeric code used in IBM equipment.
- * It uses 8 bit for each character and 9th bit for parity.

Boolean Algebra :

- * It was introduced by George Boole and is the algebra of logic, applied in the design and analysis of digital systems.
- * It deals with binary variables and logic operations and a number of laws and theorems.
- * Laws are the basic assumptions and need no proof but the theorems must be proven by using laws and other theorems. That is the difference between laws and theorems.

Common Boolean Algebra Laws :

(1) Commutative law :

commutative law of addition : $x + y = y + x$

" " multiplication : $x \cdot y = y \cdot x$

(2) Associative laws :

Addition - $x + (y + z) = (x + y) + z$

Multiplication - $x(y \cdot z) = (x \cdot y)z$

(3) Distributive Laws :

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

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

AND LAW

- Law 1 : $A \cdot 0 = 0$ (Null law)
- Law 2 : $A \cdot 1 = A$ (Identity law)
- Law 3 : $A \cdot A = A$ (Idempotence law)

OR LAW

- Law 1 : $A + 0 = A$
- Law 2 : $A + 1 = 1$
- Law 3 : $A + A = A$

Comparison of Boolean Algebra with Ordinary Algebra:-

- ① The distributive law i.e. $x + yz = (x+y)(x+z)$ is valid for Boolean Algebra only.
- ② Boolean Algebra doesn't have additive or multiplicative inverse, so there is no subtraction or division operation.
Additive Inverse = $a + (-a) = 0$, Multiplicative Inverse = $a \times \frac{1}{a} = 1$
- ③ Complements is not valid in ordinary algebra.
- ④ Two valued Boolean Algebra is defined as a set with only two elements 0 & 1.

Basic laws, Theorems and properties of Boolean Algebra:

Laws:

- 1(a) $x + 0 = x$, (b) $x \cdot 1 = x$
- 2(a) $x + y = y + x$, (b) $xy = yx$
- 3(a) $x(y+z) = xy + xz$ (b) $x + yz = (x+y)(x+z)$
- 4(a) $x + x' = 1$ (b) $x \cdot x' = 0$

Theorems:

- 1.(a) $x + x = x$ (b) $x \cdot x = x$
- 2.(a) $x + 1 = 1$ (b) $x \cdot 0 = 0$
- 3.(a) $(x')' = x$

Demorgan's Theorem:

- 4(a) $(x+y)' = x'y'$ + (b) $(xy)' = x' + y'$

Absorption Theorem:

- 5(a) $x + xy = x$ 5(b) $x(x+y) = x$

Operator Precedence:

- (1) parenthesis
- (2) NOT
- (3) AND
- (4) OR

Generalized De Morgan's Theorem:

$$(a+b+\dots+z)' = (a'b'\dots z')$$

$$(a \cdot b \cdot \dots \cdot z)' = (a' + b' + \dots + z')$$

Duality Theorem:

on dual of an algebraic expression we interchange OR & AND operators and replace 1's by 0's and 0's by 1's.

e.g. relations $\longleftrightarrow$ Dual relations

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

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

$$A + \bar{A} = 1 \longleftrightarrow A + \bar{A} = 0$$

$$A + A = A \longleftrightarrow A \cdot A = A$$

Boolean function:

* A Boolean function is an expression formed with binary variables, the two binary operators OR and AND, the unary operator NOT, parenthesis and equal sign.

* For a given value of the variables the function can be either '0' or '1'.

e.g. $f_1 = x + y'z$

(Q.) find the truth table of given function $f_1 = x + y'z$ and also draw the logic circuit.

Truth Table			y'	$y'z$	$x + y'z$
x	y	z			
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

So we can write the function
$$f_1 = (x'y'z + xy'z' + xy'z + xyz' + xyz)$$
$$= (x + y'z)$$

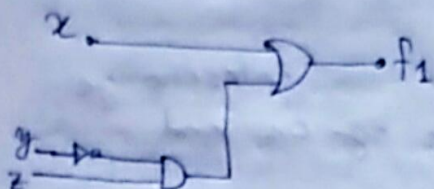
(minimized form)

Truth Table: a unique representation of a Boolean function.

Literal:- (complemented & uncomplemented variables)

literal is the variable in complemented or uncomplemented form.

Logic circuit:



Algebraic Reduction:

(Q) Simplify the following Boolean expression to a minimum number of literals.

(1) $ABC + A'B + ABC' + AC$ (2) $x'yz + xz + x'z$

(3) $(x+y)'(x'+y')$ (4) $(bc' + a'd)(ab' + cd')$

(5) $(x+y'+z')(x'+z')$ (6) $A'd + ABC + AC' + AB'$

Q1 $ABC + A'B + ABC' + AC$

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

$$= AB + A'B + AC$$

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

Q2 $x'yz + xz + x'z$

$$= x'z(y+1) + xz = x'z + xz = z(x'+x) = z \cdot 1 = z$$

Q3 $(x+y)'(x'+y')$

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

$$= x'y'z' + x'y'y'$$

$$= x'y' + x'y'$$

$$= x'y'$$

→ by demorgan's theorem
 $(x+y)' = x'y'$

$$(4) (BC' + A'D)(AB' + CD')$$

$$= BC'AB' + BC'CD' + A'DAB' + A'DCD'$$

$$= BB' \cdot C'A + C'CB'D' + AA'DB' + A'C \cdot DD'$$

$$= 0 \cdot C'A + 0 \cdot BD' + 0 \cdot DB' + A'C \cdot 0$$

$$= 0$$

$$(5) (x+y'+z')(x'+z')$$

$$= \underline{xx'} + \underline{xz'} + \underline{y'x'} + \underline{y'z'} + \underline{z'x'} + \underline{z'z'}$$

$$= z' [(x+y'+x'+1)] + y'x'$$

$$= z' \cdot [1] + y'x'$$

$$= z' + x'y'$$

$$(6) \underline{A'C'} + ABC + \underline{A'C'} + AB'$$

$$= C'(A'+A) + ABC + AB'$$

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

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

$$= C' + AB + AB'$$

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

Canonical form :

Minterms and Maxterms :

Minterms : (standard product)

- * Two or more variables is complemented and uncomplemented form combined with AND operations is called minterm or standard product and is designated as m_j .
- * n variables can be combined to form 2^n minterms.
- * When the variable being primed / complemented the corresponding bit of the binary number is 0 and unprimed or uncomplemented means the bit is 1. $x' = 0$, $x = 1$

MAXTERMS : (standard sum)

- * n variable forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations called maxterms or standard sums. It is designated as M_j .
- * Each maxterm is the complements of its corresponding minterm and viceversa.
- * $x' = 1$, $x = 0$, when variable being primed then the corresponding bit of binary number is 1 and unprimed means the bit is 0.

			<u>Minterms</u>		<u>Maxterm</u>	
<u>x</u>	<u>y</u>	<u>z</u>	<u>Term</u>	<u>Designation</u>	<u>Term</u>	<u>Designation</u>
0	0	0	$x'y'z'$	m_0	$x+y+z$	M_0
1	0	0	$x'y'z$	m_1	$x+y+z$	M_1
2	0	1	$x'yz'$	m_2	$x+y+z$	M_2
3	0	1	$x'yz$	m_3	$x+y+z$	M_3
4	1	0	$xy'z'$	m_4	$x+y+z$	M_4
5	1	0	$xy'z$	m_5	$x+y+z$	M_5
6	1	1	xyz'	m_6	$x+y+z$	M_6
7	1	1	xyz	m_7	$x+y+z$	M_7

Note

0 means the variable is primed | $x'y'z'$ is designated as m_0 .
1 means the variable is unprimed

A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms. / Maxterm for each combination of the variable that produces a 0 in the function and then taking the AND of all those terms.

Canonical form:

Boolean function expressed as a sum of minterms or product of maxterms are said to be in canonical form.

① Sum of minterms ② product of maxterms

① sum of minterms : (Canonical sum of product (SOP))

The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table. Since the function can be either 0 or 1 for each minterms, the number of possible function with n -variable is 2^{2^n} .

e.g. $f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$

$f_1 = \sum(1, 4, 7)$

x	y	z	f_1	f_2	f_3
0	0	0	0	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	0	1	1
1	1	0	0	1	0
1	1	1	1	1	0

$f_2 = m_3 + m_4 + m_5 + m_6 + m_7$
 $= \sum m(3, 4, 5, 6, 7)$

$f_3 = m_0 + m_1 + m_4 + m_5$
 $= \sum(0, 1, 4, 5)$

It is defined as the logical sum of all the minterms derived from the rows of a truth table, for which the value of the f^n is 1.

procedure for obtaining the canonical sum of product or sum of minterms form of a logic function:

- (1) Examine the each term in the given logic fⁿ. Retain it, if it is a minterm, continue to examine the next term in the same manner.
- (2) Check for variable that are missing in each product which is not a minterm. Multiply the product by $(x + \bar{x})$, if x is a missing variable.
- (3) Multiply all the products and omit the redundant terms.
- (4) Obtain the canonical / standard sum of product (SOP) form of the function $Y(A, B) = A + B$.

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

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

$$= AB + A\bar{B} + \bar{A}B = \sum(1, 2, 3) = m_1 + m_2 + m_3$$

Product of Maxterms : (Canonical product of sum (POS)) :

This is defined as the logical product of all the maxterms derived from the rows of truth-table, for which the values of function is 0.

e.g. $Y = (A + B + C)(A + \bar{B} + C)(\bar{A} + B + C)(\bar{A} + \bar{B} + \bar{C})$
 $= \Pi(0, 2, 4, 7) = M_0 \cdot M_2 \cdot M_4 \cdot M_7$

Procedure for obtaining the canonical product of sum form of a logic function.

- (1) Examine each term in the given logic function. Retain it if it is a maxterm, continue to examine the next term in the same manner.

(2) check for variable that are missing in each sum, which is not a maxterm. Add $x\bar{x}$ to the sum term, for each variable 'x' that is missing.

(3) Expand the expression using the distributive property and eliminate the redundant terms.

e.g. Express the function in canonical POS form.

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

$$\begin{aligned} \text{sol}^n \quad y &= A + \bar{B}C = (A + \bar{B})(A + C) = (A + \bar{B} + C\bar{C})(A + C + B\bar{B}) \\ &= (A + \bar{B} + C)(A + \bar{B} + \bar{C})(A + C + B)(A + C + \bar{B}) \\ &= \underbrace{(A + \bar{B} + C)}_{M_2} \underbrace{(A + \bar{B} + \bar{C})}_{M_3} \underbrace{(A + C + B)}_{M_0} \end{aligned}$$

$$y = M_0 M_2 M_3 = \Pi(0, 2, 3)$$

Relation Between Minterms and Maxterms:

$$\bar{m}_j = M_j \quad \text{or} \quad \bar{M}_j = m_j$$

where m_j = minterms, M_j = maxterms

$$j = 0, 1, 2, \dots$$

so, complement of minterms = Maxterm.

standard form:

Logical functions are generally expressed in terms of logical variables. An arbitrary logic function can be expressed in (1) sum of products (SOP) (2) Product of sum (POS).

Sum of Product (SOP):

The logical sum of two or more logical product term is called a sum of products expressions.

e.g. $Y = AB + BC + AC$
 $Y_1^* = AB + \bar{A}C + BC$

Product of sums (POS):

A product of sums expression is a logical product of two or more logical sum terms.

e.g. $y = (A+B)(B+C)(C+\bar{A})$
 $y = (A+B+C)(A+\bar{C})$

(Q) Express the following function in sum of minterms and product of maxterms $f(A, B, C, D) = B'D + A'D + BD$

Solⁿ

$$f(A, B, C, D) = B'D + A'D + BD$$

$$= (A+A')B'(c+c')D + A'(B+B')(c+c')D + (A+A')B(c+c')D$$

$$= AB'CD + AB'C'D + A'B'CD + A'B'C'D + A'BCD + A'BC'D + A'B'CD + A'B'C'D + ABCD + ABC'D + A'BCD + A'BC'D$$

$$= m_{11} + m_9 + m_3 + m_1 + m_7 + m_5 + m_3 + m_1 + m_{15} + m_{13} + m_7 + m_5$$

$$= m_1 + m_3 + m_5 + m_7 + m_9 + m_{11} + m_{13} + m_{15}$$

$$f = \sum m(1, 3, 5, 7, 9, 11, 13, 15)$$

$$= \prod M(0, 2, 4, 6, 8, 10, 12, 14)$$

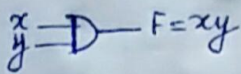
Logic Gates :

These are the logic circuits that form the building blocks of complex digital systems which can do arithmetic and logic functions.

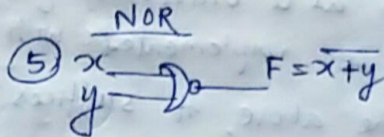
They can accept one or more input voltages but with one output voltage. They work on basis of Boolean algebra.

Most common logic gates are NOT, AND, OR, NAND, NOR, X-OR, XNOR.

① AND

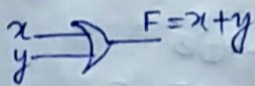


x	y	F
0	0	0
0	1	0
1	0	0
1	1	1



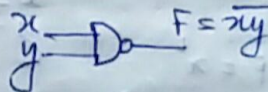
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

② OR



x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

③ NAND



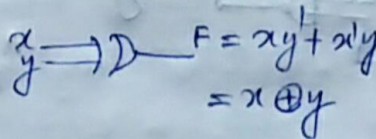
x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

④ NOT



x	F
0	1
1	0

⑤ X-OR (Exclusive-OR)



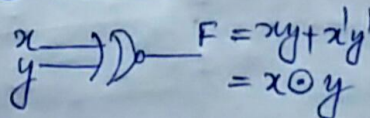
x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

⑥ Buffer



x	F
0	0
1	1

⑦ X-NOR



x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Universal Logic Gates:

NAND and NOR gates are called Universal gates.

All fundamental gates (NOT, AND, OR) can be realized by using either only NAND or only NOR gate.

A universal gate provides flexibility and offers enormous advantages to logic designers.

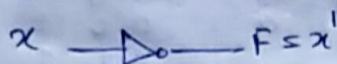
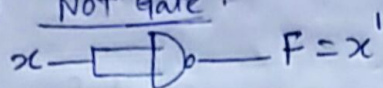
* The logic gates, with the help of which, all other logic gates can be implemented known as Universal gates.

NAND as a Universal Gate:

NAND known as a universal gate because any digital circuit can be implemented with NAND gates alone.

To prove the above, it suffices to show that AND, OR & NOT can be implemented using NAND gates only.

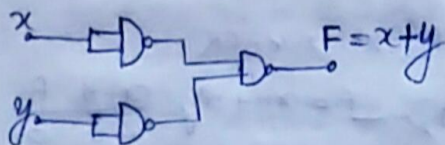
① NOT Gate:



② AND Gate:

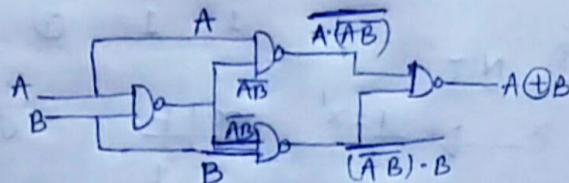


③ OR Gate:



④ $A \oplus B$

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



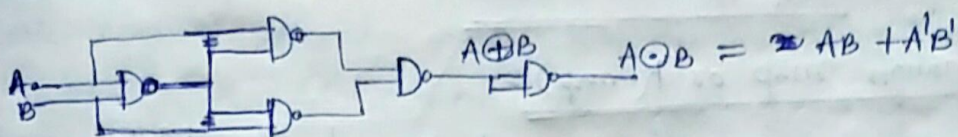
$$F = A \oplus B = A\bar{B} + \bar{A}B$$

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

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

* Digital circuits are constructed with NAND or NOR gates rather than with AND or OR gates. NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.

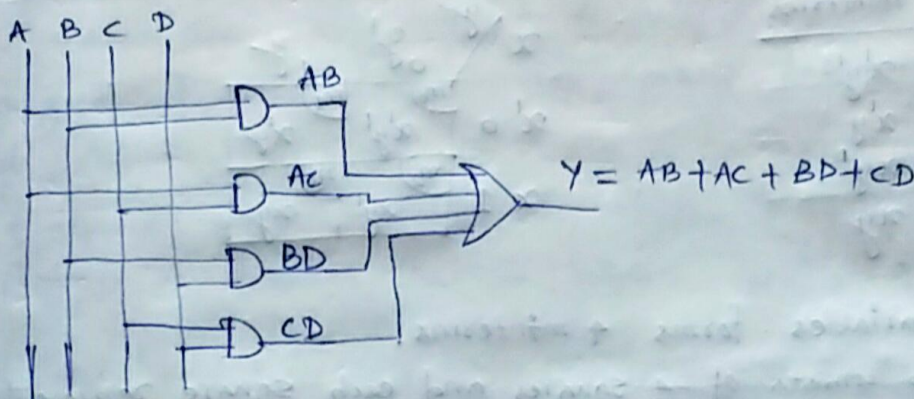
⑤ XNOR : using NAND :



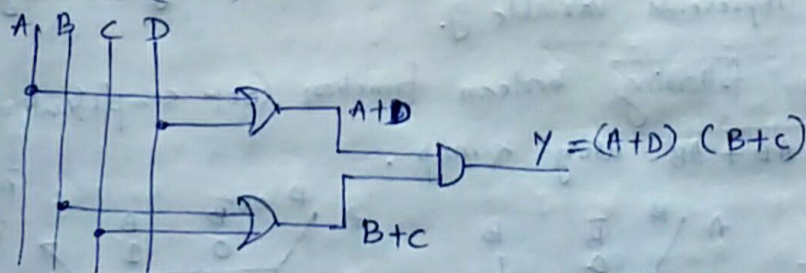
Q(a) Draw a logic circuit for the Boolean equation
 $Y = AB + AC + BD + CD$.

(b) Simplify the expression and draw logic circuit for the simplified expression.

Ans (a) It requires four AND gate & 1 OR gate (4 input)



⑥ $Y = AB + AC + BD + CD$
 $= A(B + C) + D(B + C)$
 $= (A + D)(B + C)$



The Map Method for Minimization:

Karnaugh map or (K-map):

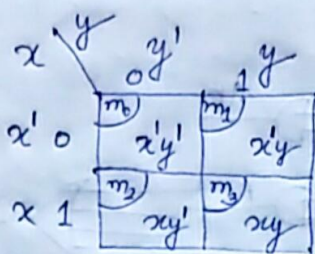
Karnaugh map allow us to reduction of the Boolean Expression using visual approach.

A k-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized.

Two variable K-map:

x	y	minterms
0	0	$x'y'$
0	1	$x'y$
1	0	xy'
1	1	xy

General structure of 2-variable k-map:



→ Two variables . forms 4 minterms.

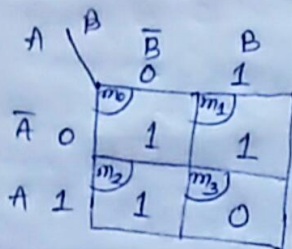
- The map consists of 4 squares and each square represents a minterm. 0 & 1 marked in each row & column designates the value of variables. row 0 represents x' (x primed) & row 1 represents variable x . column 0 presents y' primed y' and column 1 represents variable y .

(Q) Reduced the following boolean function (SOP simplification)

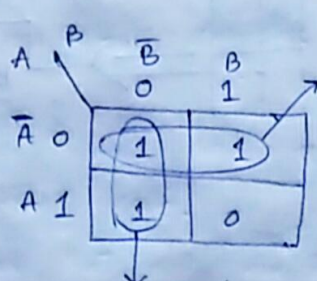
$$f = A'B' + A'B + AB' \text{ using K-map.}$$

soln

A	B	f
0	0	1
0	1	1
1	0	1
1	1	0



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



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

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

(Variable $\bar{B}$ is common to minterm m_0 & m_2 but A is not common. So we remove A) (differs by A)

$\bar{A}\bar{B} + A\bar{B}$
 $\bar{A}(\bar{B} + B)$
 $= \bar{A}$
 (two adjacent squares differ by the variable B , which can be removed)

- fill the K-map with 1's & 0's as defined by the function table. Combined adjacent square, and the combination must be power of 2. i.e. (4, 2, 1).
- Here the adjacent square are combined to produce term with fewer variable.

The rules to simplify a 2-variable K-map:

- ① The number of adjacent squares that may be combined must always represent a number that is a power of two, such as $2^0, 2^1, 2^2$. (1, 2, 4).
 - ② A single square represents a 2 variable term.
 - ③ A group of 2 adjacent squares combine to give a single variable term.
 - ④ 4 adjacent squares forms the entire map and produces a function that is always equal to 1.
- (Q) $f(A, B) = (\bar{A}B + A\bar{B} + AB) = \sum m(1, 2, 3)$

	B	
	$\bar{B}$	B
A		
$\bar{A}$ 0	m_0 0	m_1 1
A 1	m_2 1	m_3 1

$f = A + B$

(Q) $f(A, B) = \bar{A}\bar{B}$

	B	
	0	1
A		
0	1	0
1	0	0

$f = \bar{A}\bar{B}$

here a single square represents a 2 variable.

(Q) $f(A, B) = \sum m(0, 3)$

(Q) $f = \sum (0, 1)$

$f(A, B) =$

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

$f = \overline{A}\overline{B} + AB$

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

$\rightarrow \overline{A}$

$f = \overline{A}$

(Q) $f(A, B) = \sum m(0, 1, 2, 3)$

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

$f = 1$

3-variable K-map :

General structure of 3-variable K-map :

	$\overline{y}z$	$\overline{y}z'$	$y'z$	yz	yz'
	00	01	11	10	
x	m_0	m_1	m_3	m_2	
x' 0	$\overline{x}\overline{y}z$	$\overline{x}\overline{y}z'$	$\overline{x}yz$	$\overline{x}yz'$	
x 1	$xy'z$	$xy'z'$	xyz	xyz'	

- There are 8 minterms for 3 binary variables, so the map consists of 8 squares.
- The minterms are arranged, not in a binary sequence but in a sequence similar to the Gray code.
- The characteristics of this sequence is that only one bit changes in value from one adjacent column to the next.

- $m_5 = xy'z$ means row masked x and the column belonging to $y'z$ (Column 01).
- Basic property possessed by adjacent square: Any two adjacent squares in the map differ by only one variable, which is primed in one square and unprimed in the other.
e.g. m_5 & m_7 are two adjacent square (differ by variable y).
- Any two minterms in adjacent square (vertically or horizontally but not diagonally adjacent)
- removes the dissimilar variables.

Rules to simplify a 3-variable K-map:

- ① The number of adjacent squares that may be combined must always represent a number that is a power of two such as $2^0, 2^1, 2^2, 2^3$ (1, 2, 4, 8).
- As more adjacent squares are combined we obtain a term with fewer variables.
- ② One square represents one minterm, with 3-variable.
- ③ Two adjacent squares represents a term with two variables.
- ④ Four adjacent squares represents a term with one variable.
- ⑤ Eight adjacent squares produces a function that is always equal to 1.

Q) Reduce the following Boolean f

$$f(x, y, z) = \sum (1, 3, 4, 5, 7) \text{ using K-map.}$$

	yz	$y'z$	$y'z$	yz	yz'
x	00	01	11	10	
x'	0	m_1	m_3	m_2	
	0	0	1	0	
x	1	m_4	m_5	m_7	m_6
	1	1	0	1	1
	xz'		yz		

$$f = yz + xz'$$

(Two square are adjacent if they are differ by 1 variable. They are not touching each other but they are adjacent. here m_4 & m_6 differ by one variable.)

Note

Special cases :

- The two square in the map are considered to be adjacent even though they don't touch each other.
- Here m_0 is adjacent to m_2 & m_4 is adjacent to m_6 , because their minterms differ by one variable.

$$m_0 + m_2 = x'y'z' + x'yz' = x'z'(y' + y) = x'z'$$

$$m_4 + m_6 = xy'z' + xyz' = xz'(y' + y) = xz'$$

In the above example

There are 4 squares marked with 1's, one for each minterm of the function. Two adjacent square are combined in the 3rd column to give a two variable term yz .

The remaining two squares with 1's are also adjacent by the new special case. These two squares combined to give term xz' .

(②) Reduced the following Boolean function.

$$f(x, y, z) = \sum (2, 3, 4, 5) \text{ using K-map.}$$

		yz			
		$y'z'$	$y'z$	yz	yz'
x	0	m_0 0	m_1 0	m_3 1	m_2 1
	1	m_4 1	m_5 1	m_6 0	m_7 0

Annotations: An arrow points from the circled 1s in the yz column to the label $x'z'$. Another arrow points from the circled 1s in the $x=1$ row to the label xy' .

- first minterms m_2, m_3, m_4, m_5 marked with 1's.
- The next step is to find possible adjacent square. The upper combination represent $x'z'$ and lower combination represents xy' . The sum of 4 minterms can be replaced by sum of only two product term $f = x'z' + xy'$.

(Q) Simplify the Boolean function

$$f(x, y, z) = \sum (0, 2, 4, 6)$$

	yz	$y'z'$	$y'z$	yz	yz'
	00	01	11	10	
x'	0	1	0	0	1
x	1	1	0	0	1

$$f = z'$$

→ The logical sum of the four adjacent minterms 0, 2, 4, 6 reduces to the single variable term z' .

(Q) Simplify the Boolean function $f = \sum (1, 3, 5, 7)$.

	yz	$y'z'$	$y'z$	yz	yz'
	00	01	11	10	
x'	0		1	1	
x	1		1	1	

$$f = z$$

The function 'f' having 4 minterms with each minterms has 3-variable. reduces to single variable z .

(Q) Simplify the Boolean function $f(x, y, z) = \sum (0, 2, 4, 5, 6)$ using K-map.

	yz	$y'z'$	$y'z$	yz	yz'
	00	01	11	10	
x'	0	m_0	m_1	m_3	m_2
		1			1
x	1	m_4	m_5	m_7	m_6
		1	1		1

$$f = (xy' + z')$$

• Combine the 4 adjacent squares in the 1st & last column to give the single variable term z' .

• The remaining single square representing minterm-5 is combined with an adjacent square that has already been used once.

• The single square represents the 3-variable minterm $xy'z$ but two adjacent squares give the two variable term xy' .

(Q) Simplify the Boolean function $f(x, y, z) = \sum(1, 2, 4, 7)$

		$y'z$	$y'z'$	yz	yz'
x	0		1		1
	1	1		1	

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

• There is no adjacency existing among the elements, grouping is done individually, resulting in an output function $f = \sum(1, 2, 4, 7)$.

(Q) Simplify the function $\Rightarrow f = \sum(1, 2, 3, 5, 6, 7)$

		$y'z$	$y'z'$	yz	yz'
x	0		1	1	1
	1	1	1	1	1

$$f = (z + yz')$$

		$y'z$	$y'z'$	yz	yz'
x	0		1	1	1
	1	1	1	1	1

$$f = y + y'z$$

		$y'z$	$y'z'$	yz	yz'
x	0		1	1	1
	1	1	1	1	1

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

		$y'z$	$y'z'$	yz	yz'
x	0		1	1	1
	1	1	1	1	1

$$f = y + x'z + xz$$

		$y'z$	$y'z'$	yz	yz'
x	0		1	1	1
	1	1	1	1	1

$$f = (y + z)$$

- (More accurate result) than other combination.
- Combine more number of square.

Four Variable K-map :

		yz			
		$y'z'$	$y'z$	yz	yz'
wx	$w'x'$ 00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
	$w'x$ 01	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
	wx 11	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
	wx 10	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

(General structure of 4-variable K-map)

$\{ (m_0, m_8), (m_1, m_9), (m_3, m_{11}), (m_2, m_{10}) \}$ are also adjacent
 $\{ (m_0, m_2), (m_4, m_6), (m_{12}, m_{14}), (m_8, m_{10}) \}$ to each other.

- 4 variable means $2^4 = 16$ minterms
- 16 minterms are placed in 16 squares.
- The rows & columns are numbered in Gray code sequence.

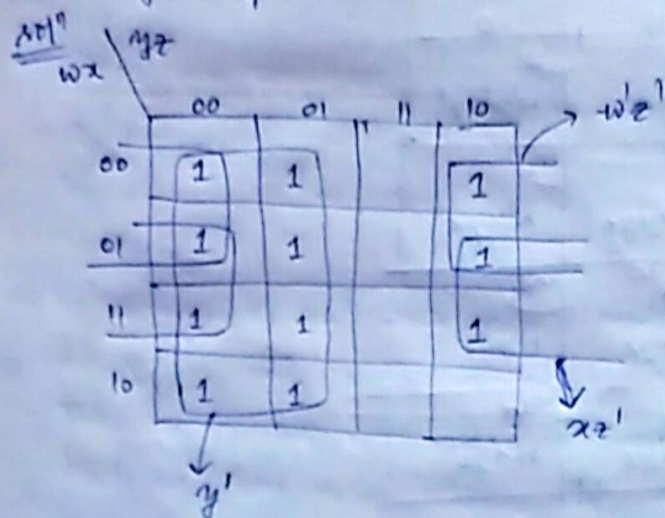
Rules to simplify a 4-variable K-map :

- ① 1 square represents one minterms with 4 variable.
- ② 2 adjacent squares represent a term with 3 variables.
- ③ 4 adjacent squares represent a term with 2 variable.
- ④ 8 adjacent squares represent a term with 1 variable.
- ⑤ 16 adjacent squares produce a function that is equal to 1.

(Q) Simplify the Boolean function

$$f(w, x, y, z) = \sum (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

using K-map.

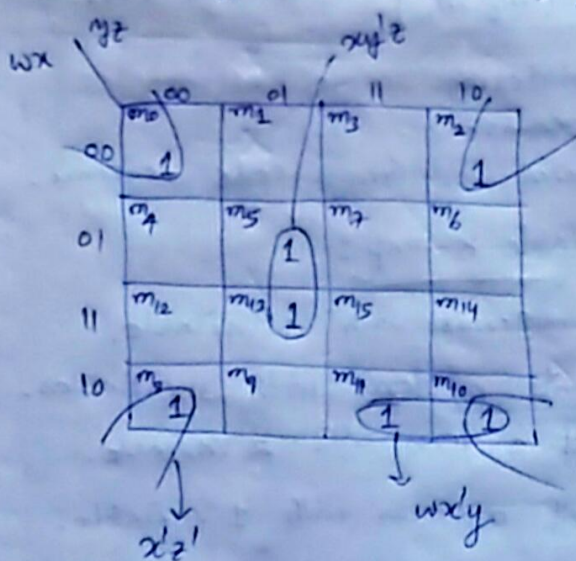


$$f_1 = (y' + w'z' + xz')$$

(minimized form)

(Q) Simplify the Boolean function

$$f(w, x, y, z) = \sum (0, 2, 5, 8, 10, 11, 13)$$



$$f = x'z' + w'x'y + x'yz$$

(a) simplify the function $f(w, x, y, z) = \sum(0, 1, 2, 3, 8, 9, 10, 11, 15)$

		yz			
wx		00	01	11	10
00	m ₀	1	1	1	1
01	m ₄				
11	m ₁₂			1	
10	m ₈	1	1	1	1

Annotations: x' points to the first column (m₀, m₄, m₁₂, m₈). wyz points to the cell m₁₁ (row 11, column 11).

$$f = x' + w y z$$

(b) simplify the function $f = \sum(0, 2, 3, 7, 8, 9, 10, 11, 13, 15)$

		yz			
		00	01	11	10
00	1			1	1
01				1	
11			1	1	
10	1	1	1	1	

Annotations: $x'z'$ points to the first column (m₀, m₄, m₁₂, m₈). wz points to the cell m₁₁ (row 11, column 11).

$$f = (wz + yz + x'z')$$

Don't Care Conditions:

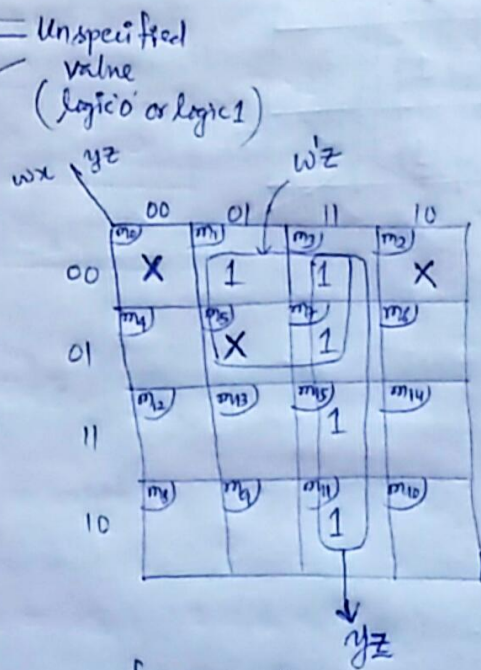
- (a) Simplify the Boolean function $f(w, x, y, z) = \sum(1, 3, 7, 11, 15)$ which has the don't care condition $d(w, x, y, z) = \sum(0, 2, 5)$

Solⁿ

Don't care Conditions:

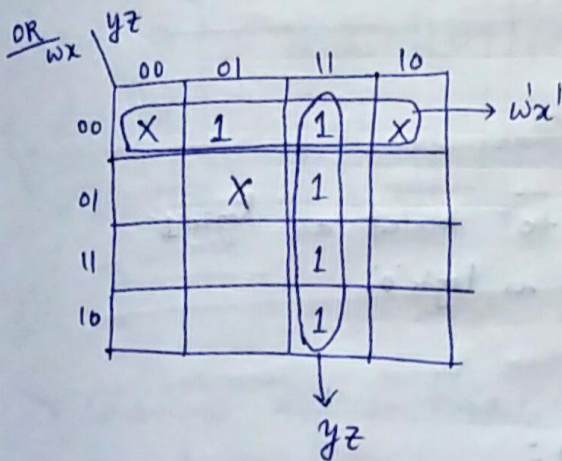
- A don't care minterm is a combination of variable whose logical value is not specified. It can be either 0 or 1. (logic 0 or logic 1).
- To distinguish the don't care condition from 1's and 0's an 'X' is used. It is denoted by symbol 'X'.
- Thus an symbol 'X' inside a square in the map indicates that we don't care whether the value of 0 or 1 is assigned to 'f' for the particular minterms.
- We can choose each don't care minterm with 1's depending on which combination gives the simplest expression.

	w	x	y	z	f
0	0	0	0	0	X
1	0	0	0	1	1
2	0	0	1	0	X
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	X
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1



• The minterms of d are don't care minterms that may be either 0 or 1.

• All don't care minterms may or may not be grouped but all 1's must be grouped.

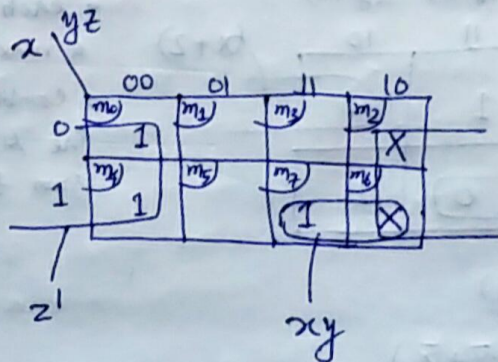


$$f = w'x' + yz$$

(Q) Simplify the $f(x, y, z) = \sum(0, 4, 7)$ which has the don't care conditions $d(x, y, z) = \sum(2, 6)$.

$$f(x, y, z) = \sum(0, 4, 7) + d\sum(2, 6)$$

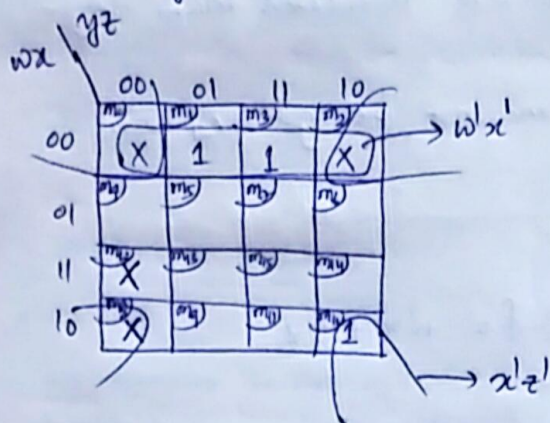
x	y	z	f
0	0	0	1
1	0	0	0
2	0	1	X
3	0	1	0
4	1	0	1
5	1	0	0
6	1	1	X
7	1	1	1



$$f = z' + xy$$

• If don't care minterms m_2, m_6 are logic 1 then we combine 4 adjacent square m_0, m_4, m_2, m_6 .

(Q) Simplify the Boolean function $f(w, x, y, z) = \sum(1, 3, 10) + d \sum(0, 2, 8, 12)$



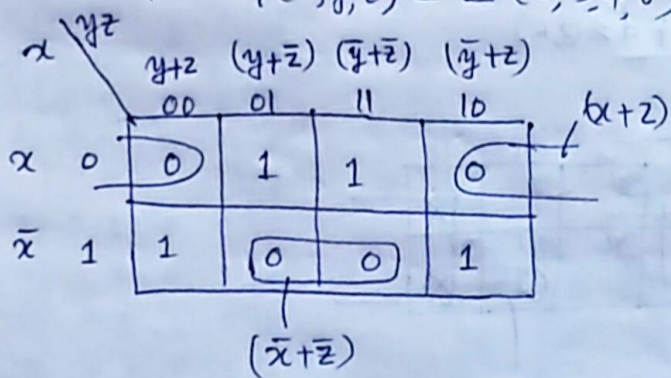
$$f = w'x' + x'z'$$

Here don't care terms 0, 2, 8 may be treated as logic 1 while don't care term 12 is treated as logic 0.

POS form:

(Product of Sums Simplifications) :

(Q) Simplify the following Boolean function into product of sum form $f(x, y, z) = \sum(1, 3, 4, 6)$



- each square represents a maxterm
- Combined 0's to obtain the simplified product of sum terms.

$$f = (x+z)(\bar{x}+\bar{z})$$

(Q) Simplify the following Boolean function into product of sum form.

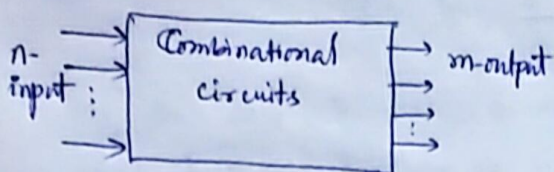
$$f(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$$

Combinational Digital System :

Logic circuits can be classified into two types

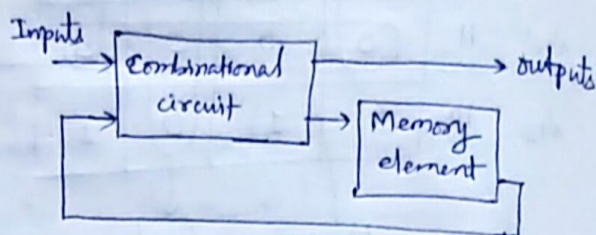
(a) combinational logic (b) Sequential logic circuits

Combinational logic circuit



Block diagram of combinational circuit

Sequential Logic Circuit
(Sequential Digital System)



Block diagram of sequential circuit

(1) The output of combinational circuit is determined by the present values of inputs only.

(2) It doesn't have a memory.

(3) It doesn't have a feedback path from output to input.

(4) It doesn't have a clock signal.

(5) It's circuit is simpler than that of sequential logic circuit.

(6) It is designed using basic gates i.e. NOT, NAND, NOR, AND, OR, XOR gates.

(1) It's output is determined by the present values of inputs as well as past values of outputs.

(2) It has a memory.

(3) It has a feedback path from output to input.

(4) It may or maynot have a clock signal. Most sequential circuits have a clock signal.

(5) It's circuit is more complex than that of combinational logic circuits.

(6) It is designed using basic gates and combinational logic circuits.

- (6) examples of combinational logic circuits are adders, subtractors, multiplexers, demultiplexers, decoders, encoders, comparators etc.
- (7) examples of sequential circuits are flip flops, counters, shift registers etc.

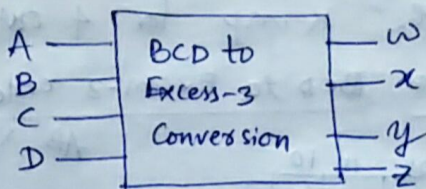
Analysis and Design Procedure of Combinational Circuits :

- ① From the problem, identify the inputs and outputs and draw a block diagram.
- ② Draw a truth table that completely describes the operation of the circuit for different combinations of inputs.
- ③ Write down the Boolean expression for the output.
- ④ Simplify the output expression using either algebraic or K-map method.
- ⑤ Implement the simplified output expression using logic gates.
- ⑥ Design Logic diagram for BCD to Excess-3 Code converter for the decimal digits.

Soln

Since each BCD code uses 4 bits to represent a decimal digit, there must be 4 input variables and 4 output variables.

Excess-3 code is obtained from the corresponding binary value plus 3.



(Block diagram)

Truth Table for Code Conversion :

Decimal Digit	Input BCD				Output Excess-3 code			
	A	B	C	D	w	x	y	z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10	1	0	1	0	X	X	X	X
11	1	0	1	1	X	X	X	X
12	1	1	0	0	X	X	X	X
13	1	1	0	1	X	X	X	X
14	1	1	1	0	X	X	X	X
15	1	1	1	1	X	X	X	X

→ From 10 to 15 values have no meaning in BCD.
So we assume that they are donot care combinations.

→ Draw the K-map for 4 outputs (w, x, y, z).

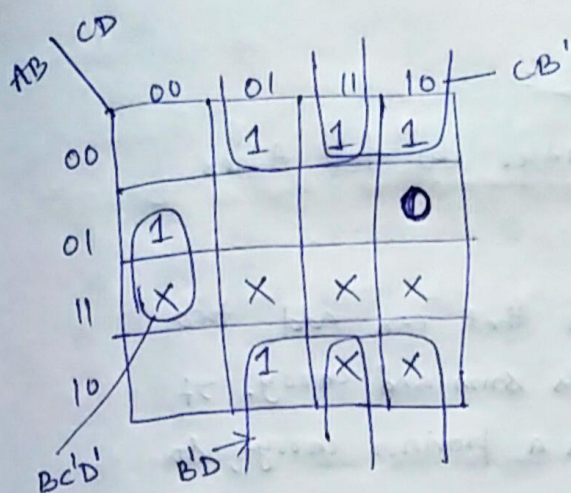
K-Maps for BCD to Excess-3 code Converter :

AB	CD			
	00	01	11	10
00	1			1
01	1			1
11	X	X	X	X
10	1		X	X

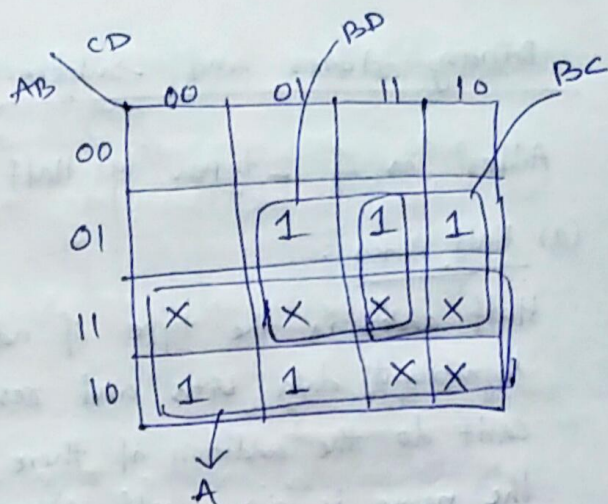
$z = D'$

AB	CD			
	00	01	11	10
00	1		1	
01	1		1	
11	X	X	X	X
10	1		X	X

$y = CD + CD'$



$$X = B'C + B'D + B'C'D'$$



$$W = A + BC + BD$$

outputs

$$Z = D', \quad Y = CD + C'D = CD + (C+D)'$$

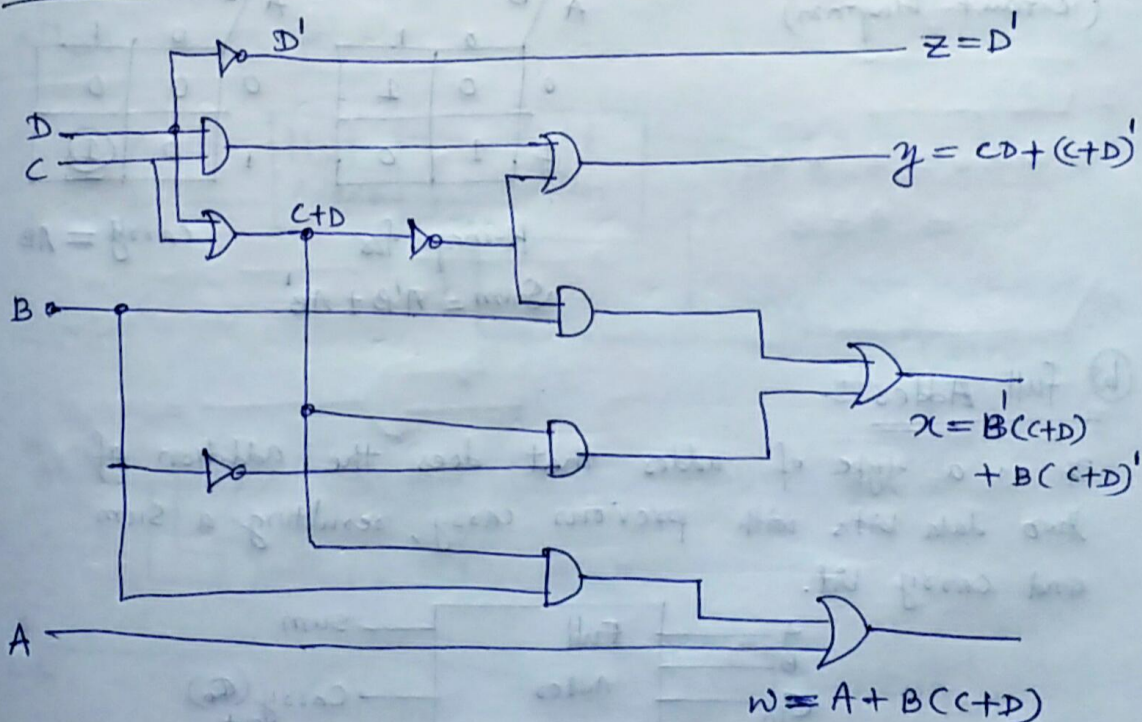
$$X = B'C + B'D + B'C'D'$$

$$W = A + BC + BD$$

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

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

Logic diagram for BCD to Ex-3 code converter:

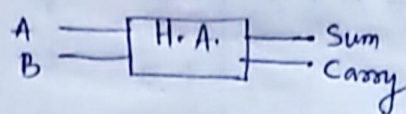


Binary Adder and Subtractors :

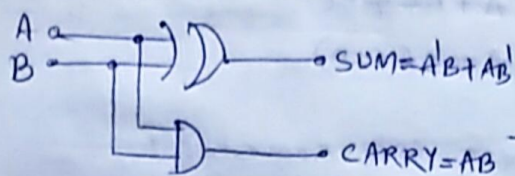
Adder is of 2 types (a) Half Adder (b) full Adder

(a) Half Adder :

Half adder is the type of adder that can add two significant data bits and results sum and carry. It can't do the addition if there is a previous carry, so the name is given half Adder.



(Block diagram)



(Circuit Diagram)

Truth Table

A	B	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A \ B	0	1
0	0	1
1	1	0

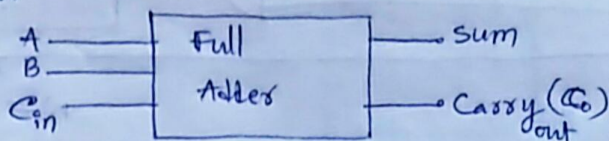
A \ B	0	1
0	0	0
1	0	1

K-map for
 $\text{Sum} = A'B + AB'$

$\text{Carry} = AB$

(b) full Adder :

It is a type of adder that does the addition of two data bits with previous carry, resulting a sum and carry bit.



Truth Table

Inputs			Outputs	
A	B	C _{in}	Sum	Carry _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K-map for full Adder :

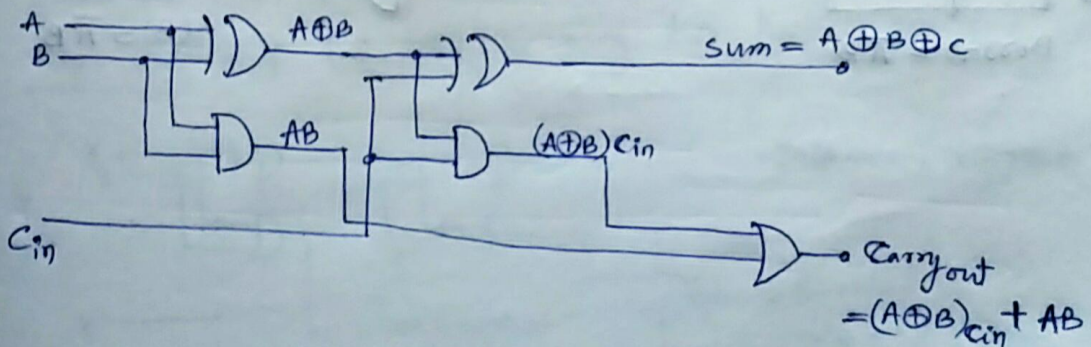
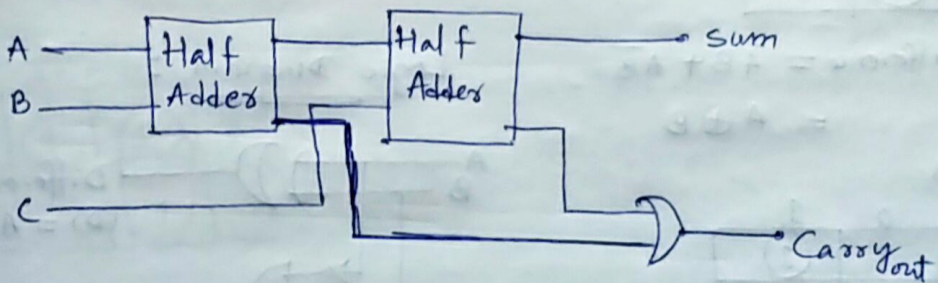
A \ BC _{in}	00	01	11	10
0		1		1
1	1		1	

$$\begin{aligned}
 \text{Sum} &= \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} \\
 &\quad + A\bar{B}\bar{C}_{in} + AB C_{in} \\
 &= A \oplus B \oplus C_{in}
 \end{aligned}$$

A \ BC _{in}	00	01	11	10
0			1	
1		1	1	1

$$\text{Carry}_{out} = AB + BC_{in} + C_{in}A$$

Implementation of full adder with two half adders and OR gate :-



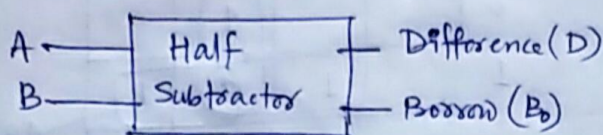
Subtractor :

It is a combinational circuit that can do the subtraction of binary digits resulting a difference bit and a borrow bit.

It is 2 types (a) Half Subtractor (b) full Subtractor

Half Subtractor :

Half Subtractor is the type of subtractor that subtracts two binary data bits resulting a difference and borrow bit. It cannot do the subtraction if there is a previous borrow.



Truth Table

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-map

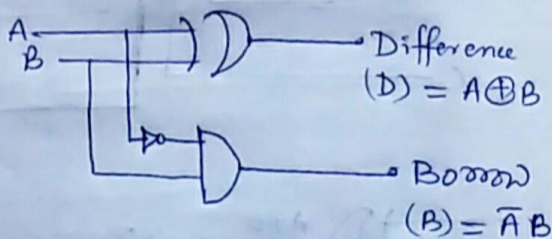
A \ B	0	1
0		1
1	1	

$$\begin{aligned}\text{Difference} &= \bar{A}B + A\bar{B} \\ &= A \oplus B\end{aligned}$$

A \ B	0	1
0		1
1		

$$\text{Borrow} = \bar{A}B$$

Logic Diagram :



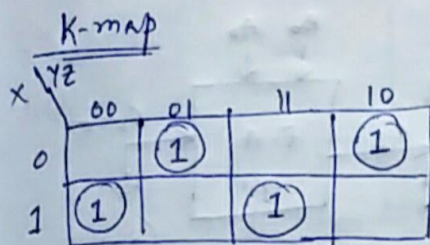
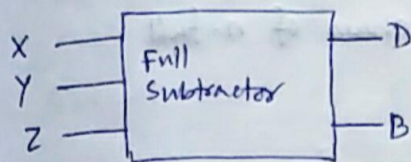
Full Subtractor :

A full subtractor has 3 inputs and two outputs.

x, y, z are the inputs to be subtracted in which z represents borrow from the next stage. D & B are outputs.

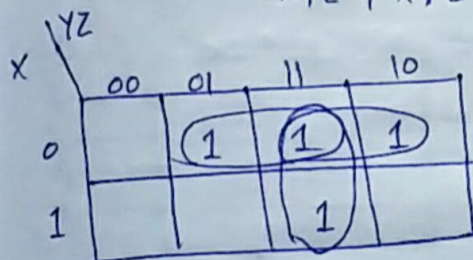
$z = \text{input Borrow} = \text{borrow from the next stage}$

$B = \text{output borrow}, D = \text{Difference}$



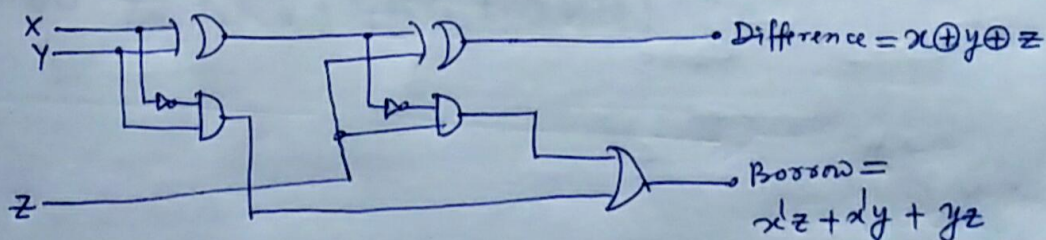
$$\text{Difference} = \bar{x}\bar{y}z + \bar{x}y\bar{z}$$

$$+ x\bar{y}z + xy\bar{z}$$



$$B = \bar{x}z + \bar{x}y + yz$$

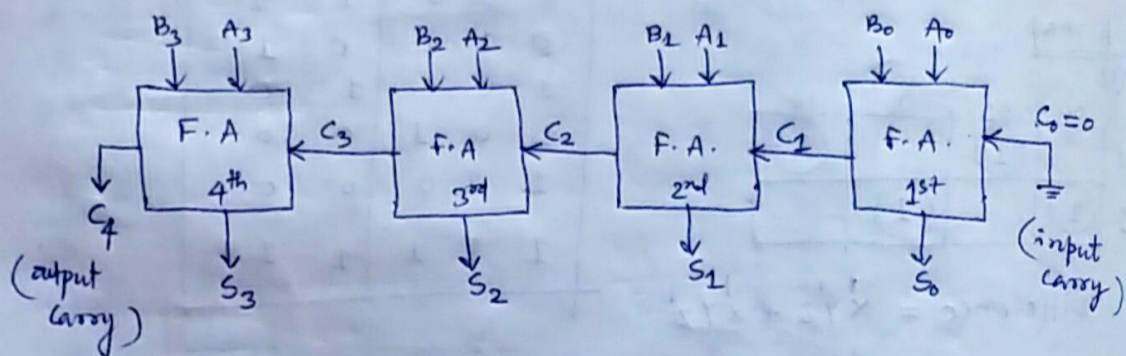
Logic Diagram of full Subtractor using two half Subtractor :



Parallel Binary Adder / Binary Adder :

- A parallel binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n -bit numbers requires a chain of n -full adders.

Four bit Parallel Binary Adder :



where

F. A. — full Adder

A_i = Augend bits (i^{th} bit numbers)

B_i = Addend bits (i^{th} bit numbers)

C_i = carry bits

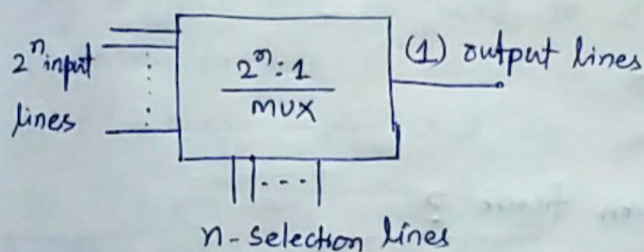
S_i = sum bits

$i = 0, 1, 2, 3$

Multiplexer (MUX) :

- A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line, and it is also called a data selector.
- The selection of a particular input line is controlled by a set of selection lines.
- The size $2^n:1$ MUX means, there are 2^n input lines and single output line.

General size of a Multiplexer:



$2^n:1$ MUX means

2^n : input lines

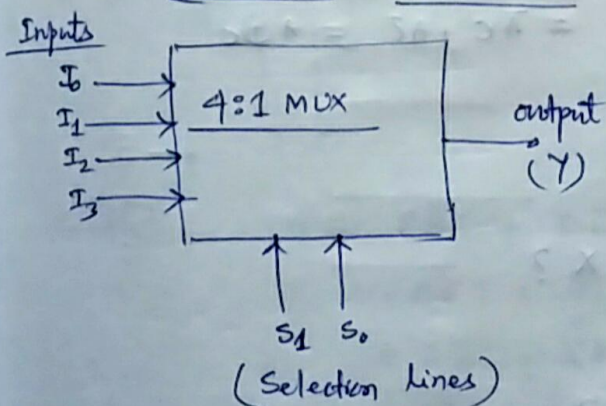
1: output line

n : Selection line

e.g.

$n=2$

$2^2:1$ MUX — $4:1$ MUX

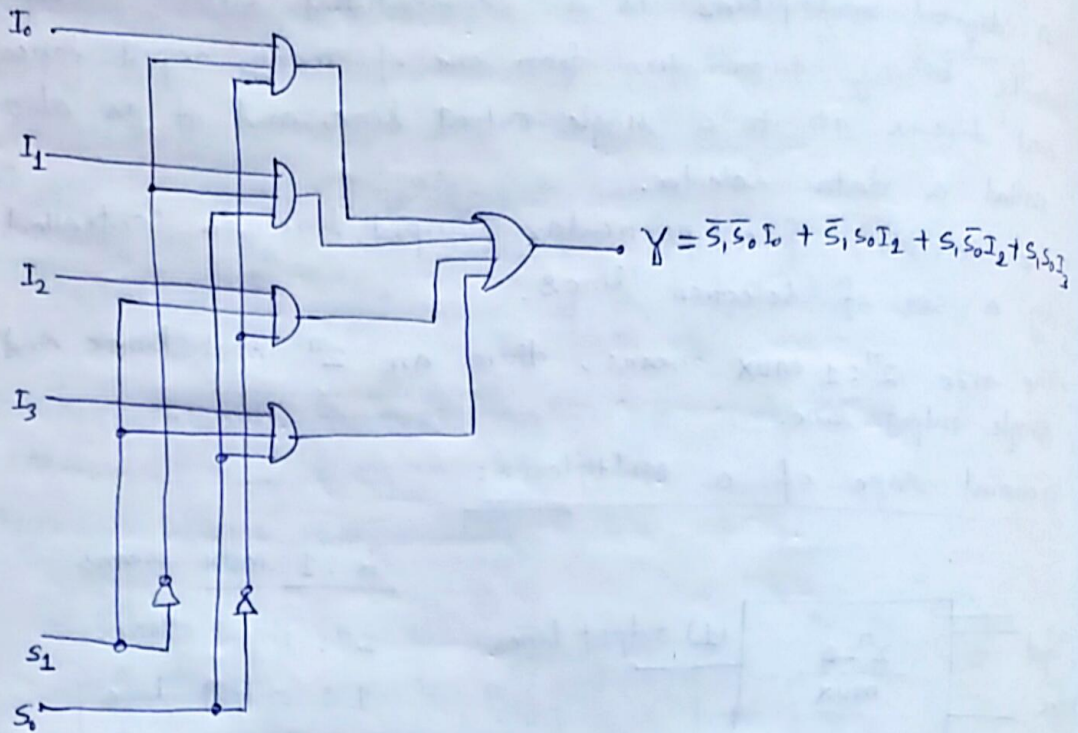


function Table

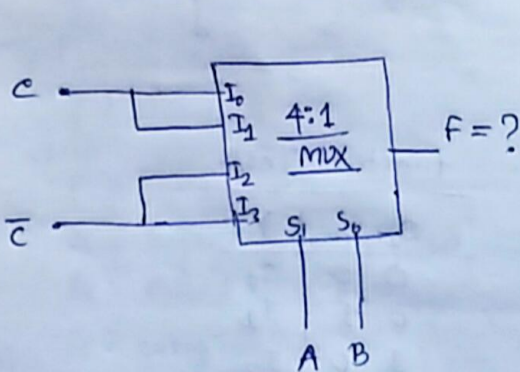
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Output $Y = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_0 I_3$

Logic diagram of 4:1 MUX :



(Q) Find the output of the given figure ?



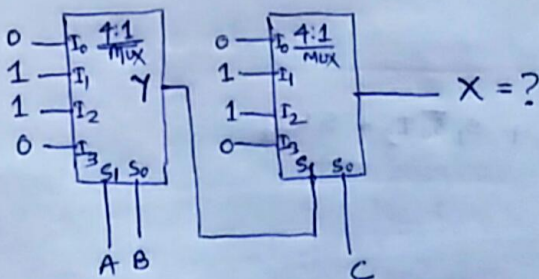
Solⁿ

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

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

$$= \bar{A}C + A\bar{C} = A \oplus C$$

(Q) In the below circuit, Find the X ?



Ans - $Y = \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + A\overline{B}C + ABC$

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

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

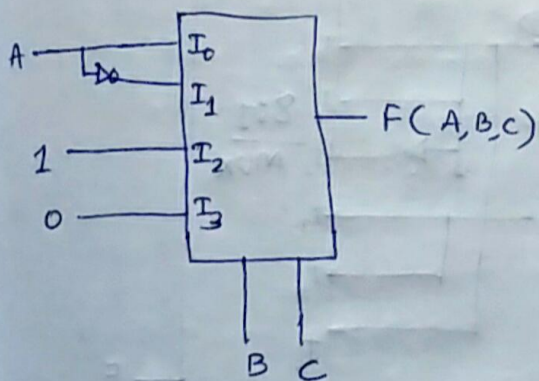
$$= (\overline{A}B + A\overline{B}) = A \oplus B$$

$$X = \overline{Y}C + Y\overline{C}$$

$$\begin{aligned} &= (\overline{A \oplus B}) \cdot C + (A \oplus B) \cdot \overline{C} = (\overline{A\overline{B} + AB}) \cdot C + (\overline{A\overline{B} + AB}) \cdot \overline{C} \\ &= A\overline{B}\overline{C} + \overline{A}B\overline{C} + \overline{A}\overline{B}C + ABC \\ &= \Sigma (1, 2, 4, 7) \end{aligned}$$

$$\boxed{X = \Sigma (1, 2, 4, 7)}$$

(Q) A 4:1 mux is used to implement a 3-input Boolean function as shown below. The Boolean function $f(A, B, C)$ implemented is



(a) $f(A, B, C) = \Sigma (1, 2, 4, 6)$

(b) $f(A, B, C) = \Sigma (1, 2, 6)$

(c) $f(A, B, C) = \Sigma (2, 4, 5, 6)$

(d) $f(A, B, C) = \Sigma (1, 5, 6)$

Solution

$$f = A\overline{B}\overline{C} + \overline{A}\overline{B}C + 1 \cdot B\overline{C} + 0 \cdot B \cdot C$$

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

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

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

$$= \Sigma (1, 2, 4, 6)$$

(Q) Implement the Boolean function $f(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$ using Multiplexer.

Solⁿ

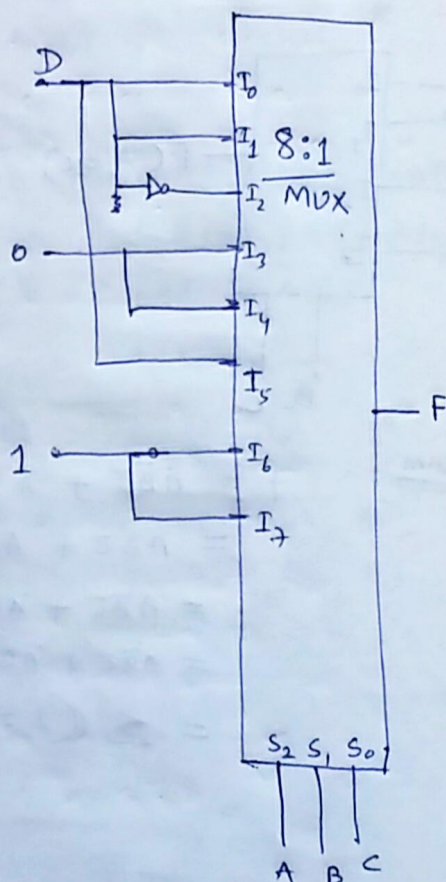
* Here the number of variable is 4 means, 3 variable assign to selection line and 1 variable connects to data input. So $2^3:1 = 8:1$ MUX used. ($2^4:1 = 2^3:1$).

* 3 variable A B C are connected to the selection inputs of the 8:1 MUX. in the order of A connected to S_2 , B connected to S_1 & C connected to S_0 .

* The variable 'D' will connect to data inputs. The value for the data inputs lines are determined from the truth table of the function.

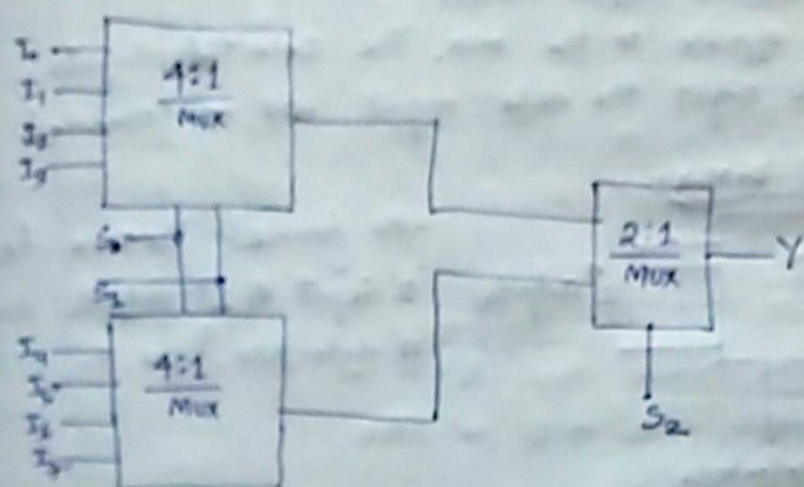
Truth Table

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



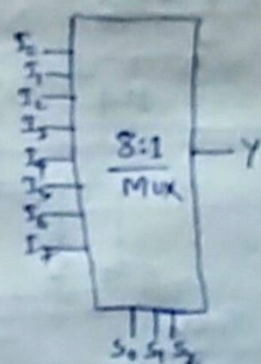
(Q) construct a 8:1 MUX using 4:1 MUX & 2:1 MUX.

Ans. It is possible to implement higher order mux using lower order mux.



Truth Table

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



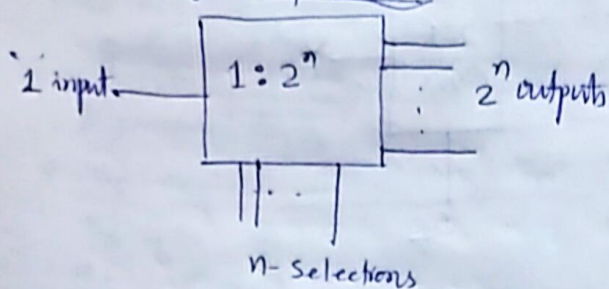
→ S_1, S_0 are the selection lines for two 4:1 MUX,
and S_2 is the selection line for 2:1 MUX.

→ 1st 4 inputs (I_0, I_1, I_2, I_3) represents 1st 4:1 MUX
2nd 4 inputs (I_4, I_5, I_6, I_7) represents 2nd 4:1 MUX

Demultiplexer (DEMUX) : (Data Distributors) :

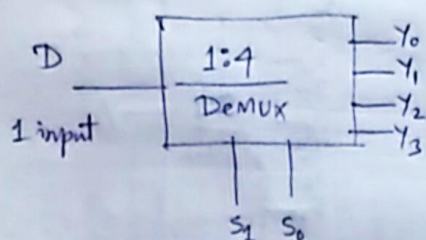
- A demux is a logic circuit that receives information on a single input and ~~transmits~~ transmits the same information over one of 2^n possible output lines.
- It's operation is opposite to the mux. The selection inputs determine to which output the data input will be connected.

Block diagram of DEMUX:



- In general Demux has 1 input & 2^n outputs and n selection lines.

e.g. 1:4 Demux



There are 4 outputs, out of which one output will be selected according to the selection line. That selected output receives the input signal.

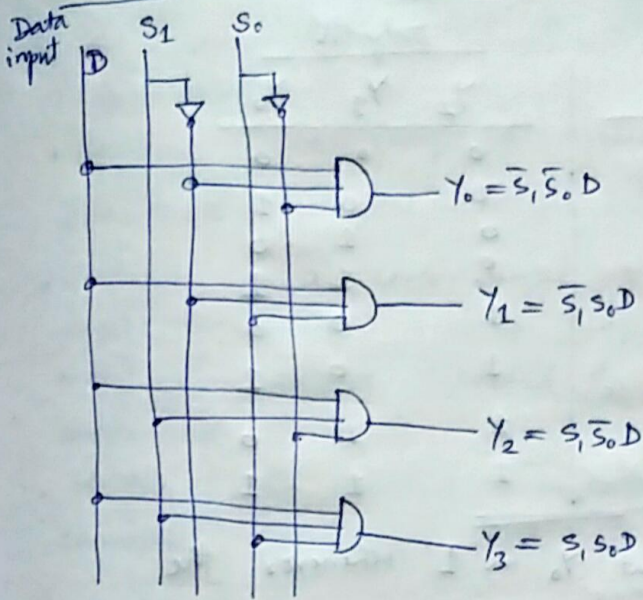
Truth Table

Data input	Select input		Outputs			
D	S ₁	S ₀	Y ₃	Y ₂	Y ₁	Y ₀
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

- 1:4 Demux has a single input (D), 4 outputs (Y₀ to Y₃) and two select input (S₁ & S₀).
- From the truth table, it is clear that the data input is connected to output Y₀ when S₁=0 & S₀=0 and so on.

- From truth table, the expression for output can be written as $Y_0 = \bar{S}_1 \bar{S}_0 D$, $Y_1 = \bar{S}_1 S_0 D$, $Y_2 = S_1 \bar{S}_0 D$, $Y_3 = S_1 S_0 D$

Logic Diagram of output :



Encoders :

- * An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or less) input lines and n -output lines.
- The output lines generate the binary code for the 2^n input variables.

Octal to Binary Encoder :

- An octal to binary encoder performs the opposite function of a 3-to-8 decoder or binary to octal decoder.
- It accepts 8 inputs and produces a 3-bit output code corresponding to the activated input.

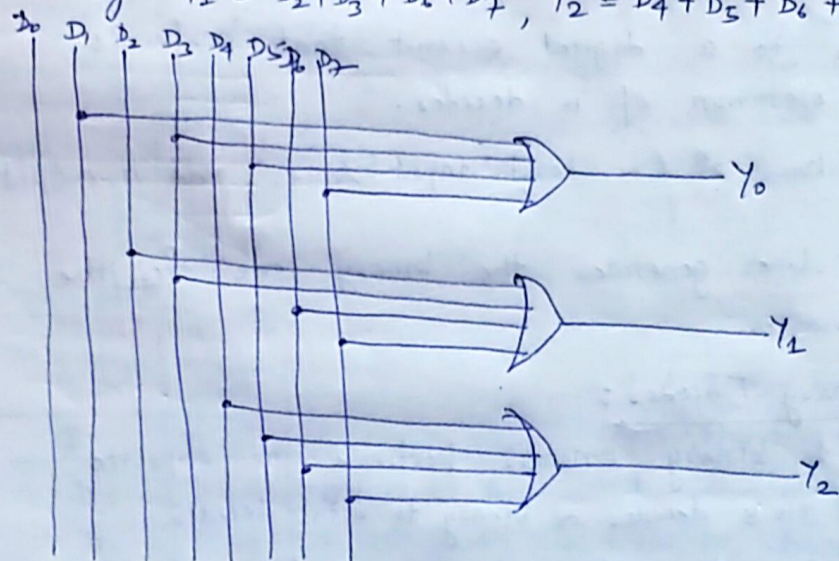
Truth Table of Octal to Binary ENCODER:

<u>Inputs</u>								<u>Outputs</u>		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	Y_2	Y_1	Y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

The truth table shows that Y_0 is 1 whenever the input D_1 or D_3 or D_5 or D_7 is 1 or HIGH.

$$\text{So } Y_0 = D_1 + D_3 + D_5 + D_7$$

Similarly $Y_1 = D_2 + D_3 + D_6 + D_7$, $Y_2 = D_4 + D_5 + D_6 + D_7$

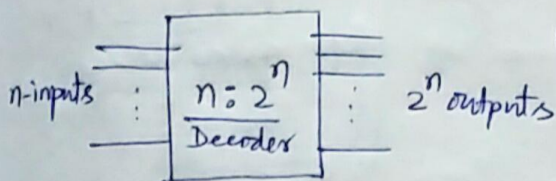


(Logic Diagram for Octal to Binary Encoder)

- Using the above expressions the octal to binary encoder can be implemented using three 4-input OR gates.
- The circuit is designed in such a way that when D_0 is HIGH, the binary code 000 is generated, when D_1 is HIGH, the binary code 001 is generated and so on. Here we are taken 8 input combinations from total 28 possible input combinations.

DECODER :

- A decoder is similar to Demux but without any data input.
- Decoding is necessary in applications such as data demultiplexing, digital display, D/A converter and memory addressing.
- A decoder is a combinational circuit that converts an n -bit binary input into 2^n output lines, such that each output line will be activated for only one of the possible combinations of inputs.
- If the n -bit decoded information has unused or don't care combinations, the decoder output will have less than 2^n outputs.



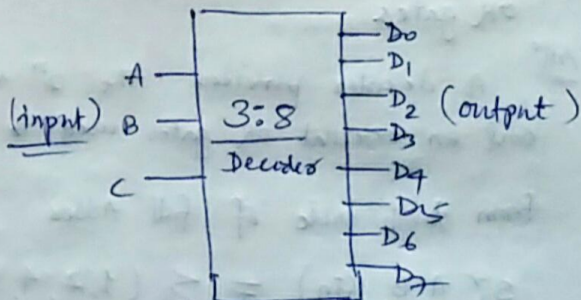
(• from 2^n outputs one output will be active according to input binary combination.)

3 to 8 Decoder :

- A 3 to 8 decoder has 3 inputs (A, B, C) and 8 outputs D_0 to D_7 .
- based on the 3 inputs, one of the output is selected.

Truth Table of 3:8 decoder :

<u>Inputs</u>			<u>Outputs</u>							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



from the truth table, the logic expressions for the outputs can be written as follows

A	BC			
	00	01	11	10
0	(1)			
1				

$$D_0 = A'B'C$$

A	BC			
	00	01	11	10
0		(1)		
1				

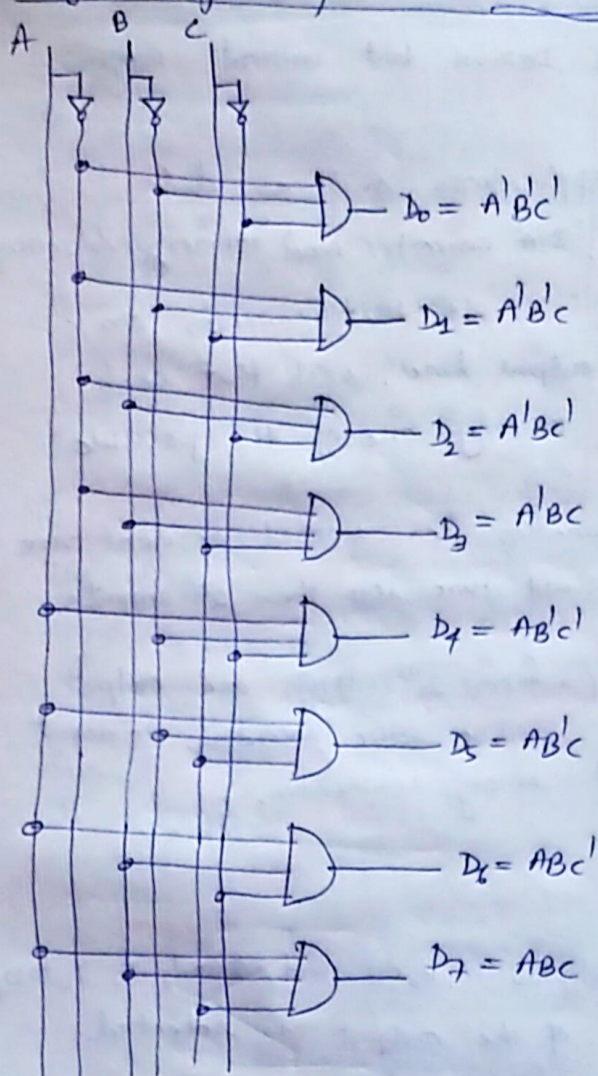
$$D_1 = A'B'C'$$

Similarly

$$D_2 = AB'C', D_3 = \bar{A}BC, D_4 = AB'C$$

$$D_5 = AB'C, D_6 = ABC, D_7 = ABC$$

Logic Diagram of 3 to 8 decoder:



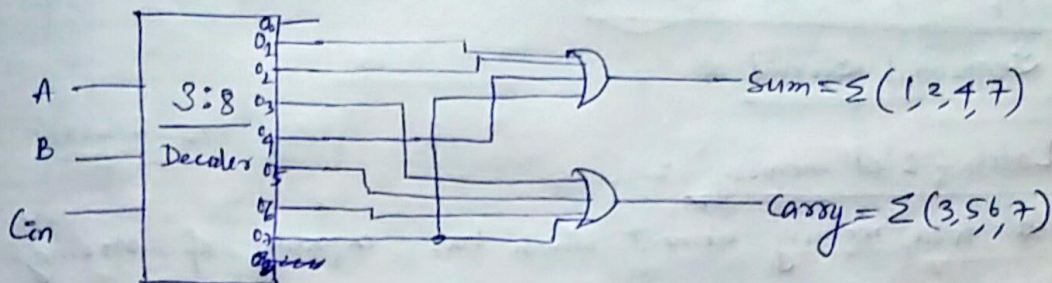
(Q) Implement a full adder circuit with a decoder and two OR gates.

Solⁿ A decoder provides the 2^n minterms of n input variable and an external OR gate is used to form the sum.

from truth table of full adder

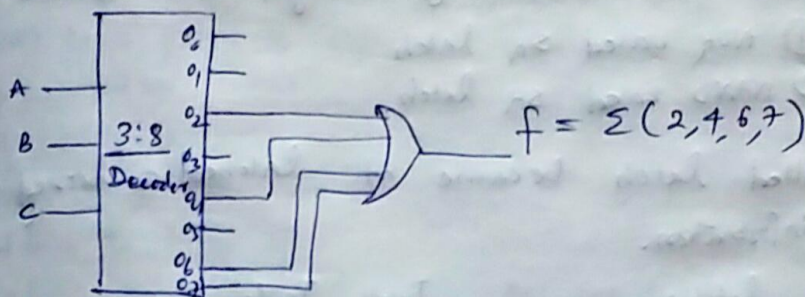
$$S(A, B, C_{in}) = \sum (1, 2, 4, 7)$$

$$C(A, B, C_{in}) = \sum (3, 5, 6, 7)$$



(Q) Implement the Boolean function using Decoder & OR gates.

$$f(A, B, C) = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + AB\bar{C} + ABC = \Sigma(2, 4, 6, 7)$$



Sequential digital System :-

Sequential Circuit

Latch (1 bit memory) :

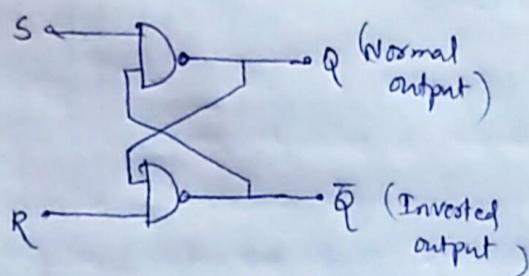
It is a memory element which has capable of storing 1 bit of information i.e. logic 1 or 0. This sequential circuit is called a latch, since one bit of information can be locked or latched.

We can construct the latches by using NAND gates & NOR gates.

- ① NOR based SR latch
- ② NAND based SR latch.

- its name called latch because it latched or locked one bit of information.

NAND based SR latch :



Logic diagram

Inputs		Outputs		Cond ⁿ
S	R	Q _{n+1}	Q-bar _{n+1}	
0	0	1	1	Invalid
0	1	1	0	Set
1	0	0	1	Reset
1	1	Q _n	Q-bar _n	Nochange

Q_n - present state of F/F, Q_{n+1} - next state of F/F

NAND property

A	B	AB
0	0	1
0	1	1
1	0	1
1	1	0

{ NAND gate property
any '1' input = 0
means output = 1 }

• It has two input one is Set (S) & another is Reset (R).

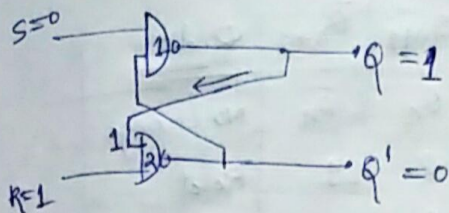
• It has two output i.e. normal (Q) & complemented (Q').

• When output is 1 it is called set condⁿ of F/F. In set condⁿ it store '1'.

Set condⁿ → input S=0, R=1

• When output is 0 it is called Reset condⁿ. In reset condⁿ it store '0'.

for input $S=0, R=1$

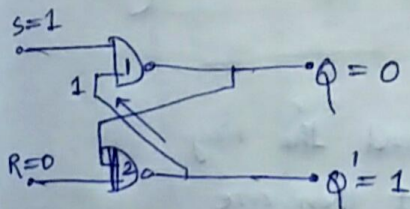


- $S=0$ means $Q=1$
- from NAND gate property Gate-1 input is $S=0$ so its output is $Q=1$.
- NAND gate property any 1 input $=0$ means output 1.
- output $Q=1$ from Gate-1 feed back to input of Gate-2.

→ Gate-2 both input are 1 so output $Q'=0$.

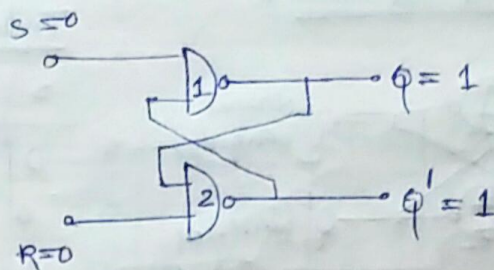
→ for $S=0, R=1$
 $\Rightarrow Q=1$ & $Q'=0$

For input $S=1$ & $R=0$



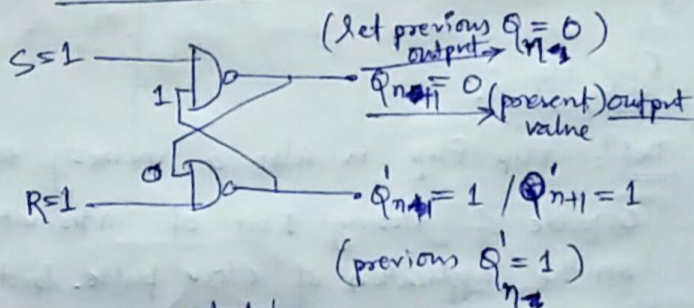
- Gate-2 has one input 0 so its output $Q'=1$ from NAND gate properties.
- $Q'=1$ feedback to Gate-1 so Gate-1 input has both input 1 so its output $Q=0$.

for input $S=0, R=0$



- Both Gate-1 & Gate-2 one of the input is 0 so both output Q & $Q'=1$.
- $Q=1, Q'=1$ is a Invalid case because Q & Q' are complement to each other.
- So $S=0, R=0$ is a Invalid case.

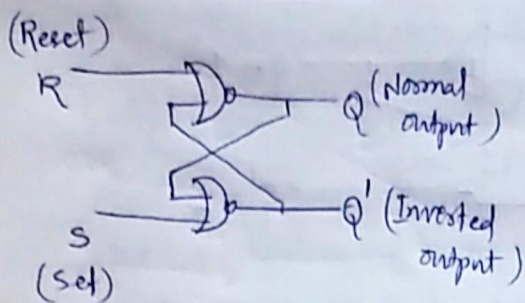
for input $S=1, R=1$



If previous output has value $Q=0$, $Q'_{n+1}=1$
 we will get present value also $Q_{n+1}=0$, $Q'_{n+1}=1$

So there is no change of Q value.
 output value when both $S=1$ & $R=1$.

NOR based SR latch :



NOR

A	B	A+B
0	0	1
0	1	0
1	0	0
1	1	0

→ NOR property
any one input 1
means output 0.

Truth Table

S	R	Q_{n+1}	$\bar{Q}_{n+1}$	Condition
0	0	Q_n	$\bar{Q}_n$	No change
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Invalid

$\{Q_n = \text{present state of FIF}\}$
 $\{Q_{n+1} = \text{next state of FIF}\}$

Flip Flop :

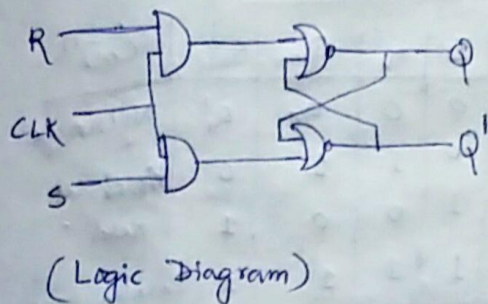
Defⁿ: Flip Flop is also a memory element, which has capable of storing 1 bit of information, either logic 1 or 0, on the application of clock pulse. But latch has no clock pulse.

Types of Flip Flop :

⊗ Depending on the numbers of inputs and the way the output changes with respect to input they are divided into different types.

- (1) ~~SR~~ SR Flip flop
- (2) D-Flipflop
- (3) JK Flip Flop
- (4) T-Flip Flop

1) SR FIF (using NOR)



Truth Table

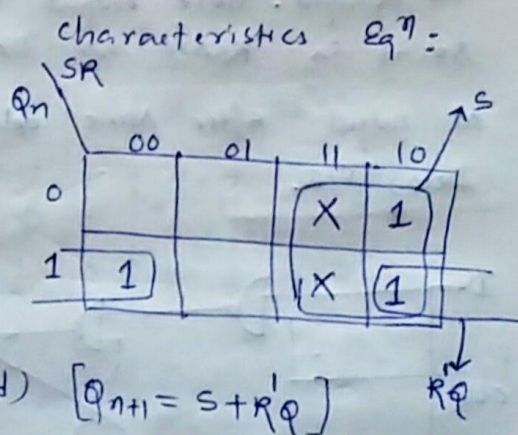
S	R	Q_{n+1}	$\bar{Q}_{n+1}$	Cond ⁿ
0	0	Q_n	$\bar{Q}_n$	Nochange
0	1	0	1	Reset
1	0	1	0	Set
1	1	1	1	Invalid

Characteristics ~~Flip flop~~ Table :

characteristics table define the Next state of flip flop of input and present state of flip flop known to us.

Characteristics Table :

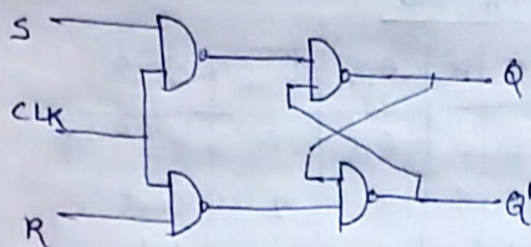
present state of flip flop (Q_n)	Input of flip flop		Next state of flip flop
Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X (Invalid)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X (Invalid)



characteristics Equation

$$Q_{n+1} = (S + R'Q_n)$$

SR flip flop (using NAND)

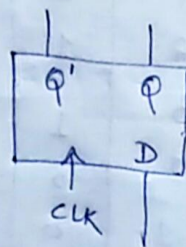
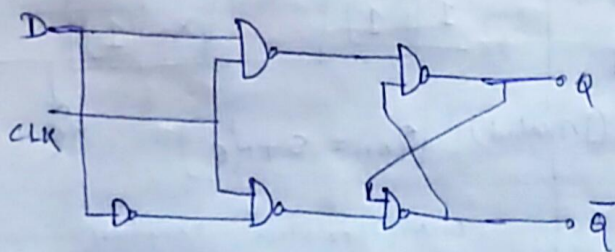


Truth Table

S	R	Q_{n+1}	Q'_{n+1}	Cond ⁿ
0	0	Q_n	Q'_n	No change
0	1	1	0	Reset set
1	0	0	1	Reset
1	1	1	1	(Invalid case)

D-flip flop : (using NAND)

- The D-F/F is a modification of the clocked RS flip flop.
- It is basically an RS flip flop with an inverter in the R input. Here output is same as the input to D so it is known as Data Flip flop. The data is appearing at output after some delay. So it is also known as Delay F/F.



Truth Table

D	Q_{n+1}	$\overline{Q}_{n+1}$	Cond ⁿ
0	0	1	Reset
1	1	0	Set

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

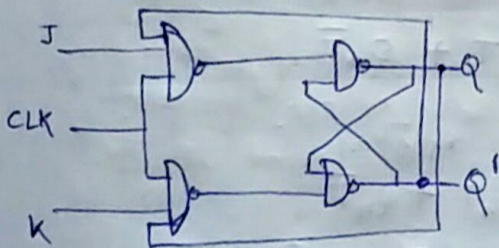
Characteristics Table :

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

$Q_{n+1} = D$ → Characteristic Equations

J-K Flip flop :

- A JK flip flop is a refinement of the RS flip flop in that the indeterminate state of the RS flip flop is defined.
- The invalid condition of RS flip flop is overcome with the help of extra feedback to first stage.
- When both inputs are HIGH then present output is complement of previous output and the output continuously changes between 1 & 0, on the application of clock pulse. so the output Toggle between 1 & 0. It is known as Toggling condition. The output continuously changes between 1 & 0 and output is uncertain. This problem ^{also} called race around condition. Race around condition can be removed in JK-MasterSlave Flip flop.



(JK flip flop using NAND gates)

Truth Table

J	K	Q_{n+1}	Cond ⁿ
0	0	Q_n	No change
0	1	0	Reset
1	0	1	Set
1	1	Q_n'	Toggle / Race around

Characteristics Table :

present state	Input of f/f		next state of f/f
Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

characteristics Q_{n+1} is obtained from k-map:

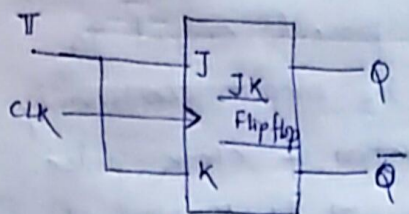
Q/JK				
	00	01	11	10
0			1	1
1	1			1

Characteristics Q_{n+1}

$$Q_{n+1} = JQ' + K'Q$$

T-flip flop : (T-stands for Toggle)

- The T-flip flop is a single-input version of the JK flip flop. It is obtained from a JK type F/F if both inputs are tied together. It is known as toggle F/F as its output is continuously changing between 1 & 0.



Truth Table

T	Q_{n+1}	condition
0	Q_n	No change
1	Q_n'	Toggle

Characteristics Table :

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

$Q_n \backslash T$	0	1
0		1
1	1	

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

Characteristics Equation

$$\begin{aligned} Q_{n+1} &= \bar{Q}_n T + Q_n \bar{T} \\ &= \underline{Q_n \oplus T} \end{aligned}$$

Registers : REGISTERS :

Register :

A Register is a collection of flip flops. For storing a large number of bits, the storage capacity is increased by grouping more than one flip flops.

An n -bit register is a cascade of n -flip flops and can store an n -bit binary data.

Shift Registers :

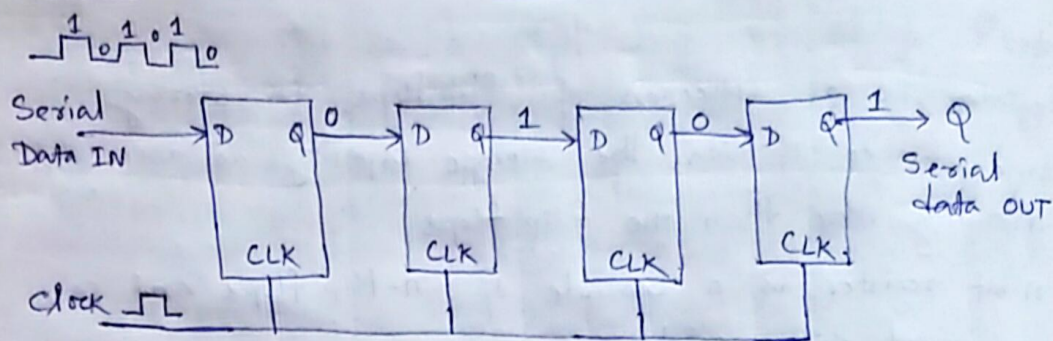
A register is capable of shifting its binary information in one or both directions is known as shift register. The logical configuration of shift register consist of a D-Flip flop cascaded with output of one flip flop connected to input of next flip flop. All flip flops receive common clock pulses which causes the shift in the output of the flip flop. The output of a given flip flop is connected to the input of next flip flop of the register. Each clock pulse shift the contents of register one bit position to right.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as

- (1) Serial in Serial out (SISO) (2) Serial in Parallel out (SIPO)
- (3) ~~Serial~~ Parallel in Serial out (PISO) (4) Parallel in Parallel out (PIPO)

• Shift registers are a type of logic circuits closely related to counters. They are basically for the storage and transfer of digital data.

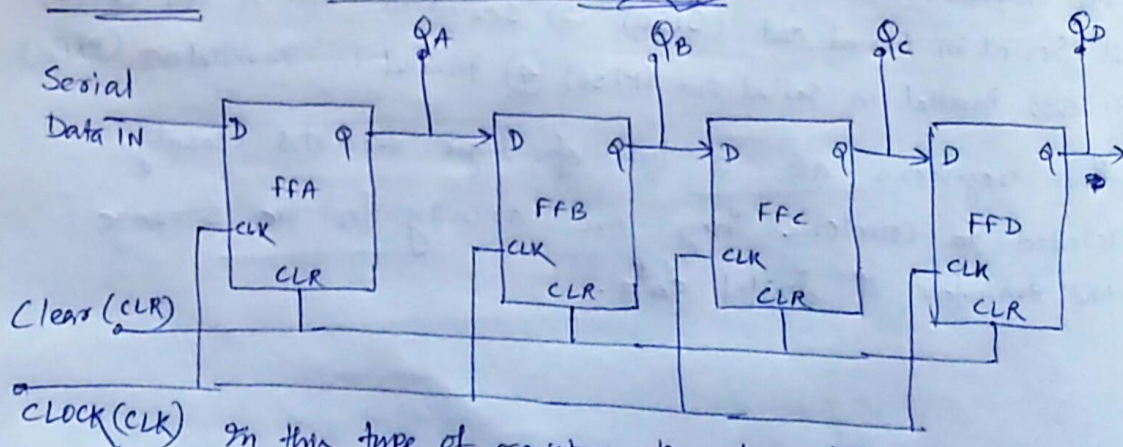
Serial IN, Serial OUT Shift register :



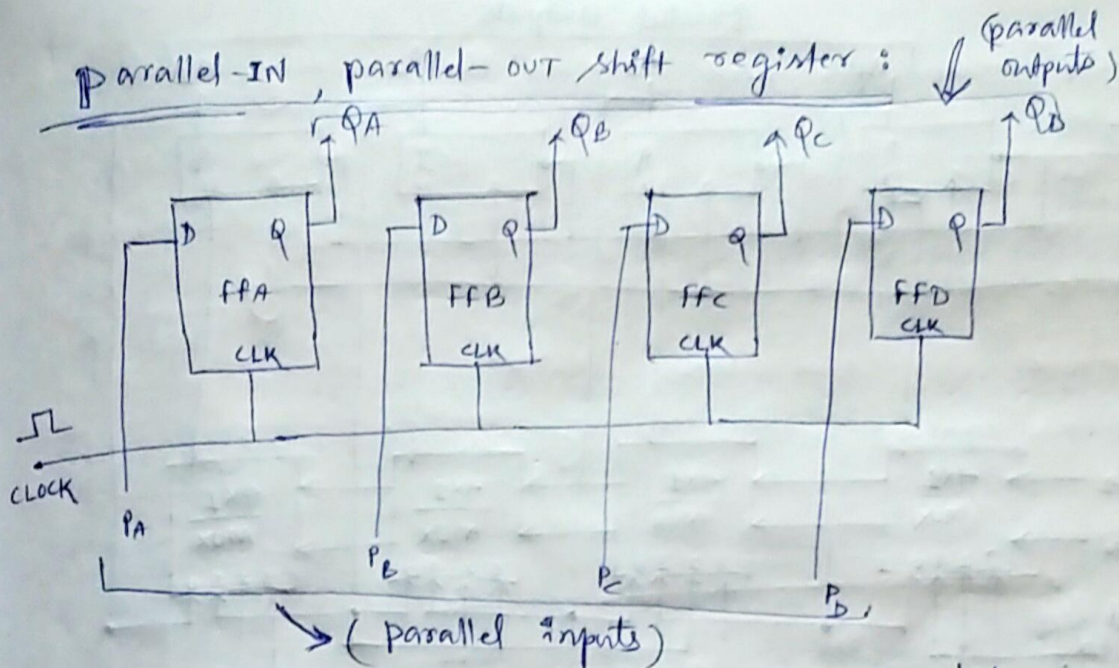
The logic diagram of 4-bit serial out, right shift registers with 4 stages. The registers can store 4 bits of data. Serial data is applied at the input D of the first F/F. The Q output of the 1st F/F is connected to the D input of another F/F. The data is outputted from the Q terminal of the last F/F.

When serial data is transferred into a register, each new bit is clocked into the 1st F/F at the positive going edge of each clock pulse. The bit that was previously stored by the 1st F/F is transferred to the 2nd F/F. The bit that was stored by the 2nd F/F is transferred to the 3rd F/F.

Serial IN, parallel OUT shift register :



In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.



In a parallel-in, parallel-out shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form.

Universal Shift Register :-

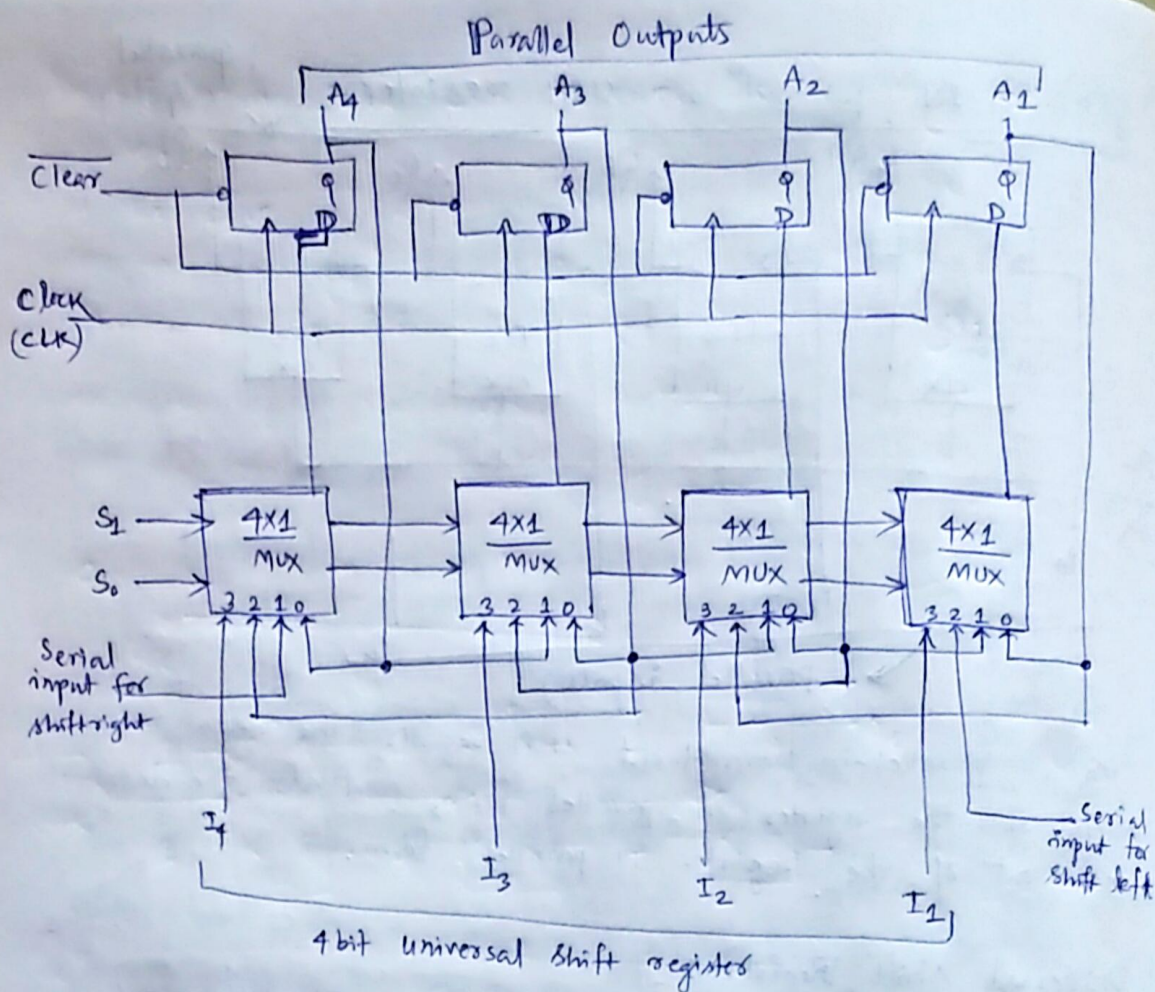
A register is capable of shifting in one direction only is a unidirectional shift register. One that can shift both directions is a bidirectional shift register.

→ If the register has both direction shift and serial, and parallel load capabilities, it is referred to as a universal shift registers.

→ Universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be in serial form or parallel form.

Function table for Universal Shift register:

		MODE CONTROL
S_0	S_1	Register operation:
0	0	No change
0	1	Shift Right
1	0	Shift Left
1	1	Parallel load

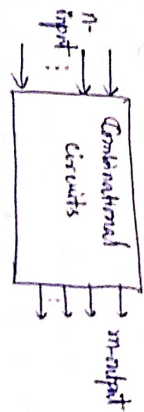


(figure: Logic diagram 4 bit Universal Shift register.)

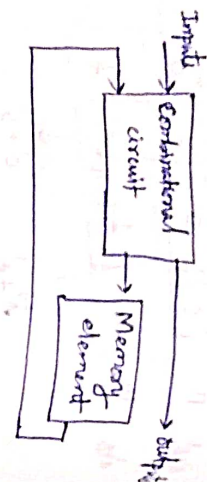
Combinational Digital System :

Logic circuits can be classified into two types

- combinational logic
- Sequential logic circuit
(Sequential Digital System)



Block diagram of combinational circuit



Block diagram of sequential circuit

(1) The output of combinational circuit determined by the present values of inputs only.

(1) It's output is determined by the present values of inputs as well as past values of outputs.

(2) It doesn't have a memory

(2) It has a memory.

(3) It doesn't have a feedback path from output to input.

(3) It has a feedback path from output to input.

(4) It doesn't have a clock signal.

(4) It may or maynot have a clock signal. Most sequential circuits have a clock signal.

(5) It's circuit is simpler than that of sequential logic circuit.

(5) It's circuit is more complex than that of combinational logic circuits.

(6) It is designed using basic gates i.e. NOT, NAND, NOR, AND, OR, XOR gates.

(6) It is designed using basic gates and combinational logic circuits.

(7) examples of combinational logic circuits are address, multipliers, demultiplexers, decoders, encoders, comparators etc.

(7) examples of sequential circuits are flip flops, counters, shift registers etc.

Analysis and Design Procedure of Combinational Circuits :

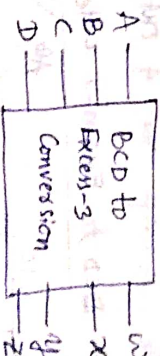
- From the problem, identify the inputs and outputs and draw a block diagram.
- Draw a truth table that completely describes the operation of the circuit for different combinations of inputs.
- Write down the Boolean expression for the output.
- Simplify the output expression using either algebraic or K-map method.
- Implement the simplified output expression using logic gates.

(8) Design Logic diagram for BCD to Excess-3 code converter for the decimal digits.

Soln

Since each BCD code uses 4 bits to represent a decimal digit, there must be 4 input variables and 4 output variables.

Excess-3 code is obtained from the corresponding binary value plus 3.



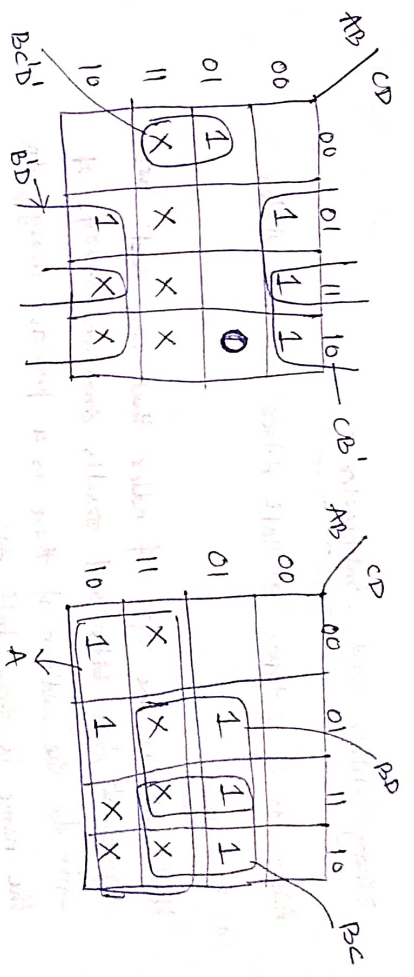
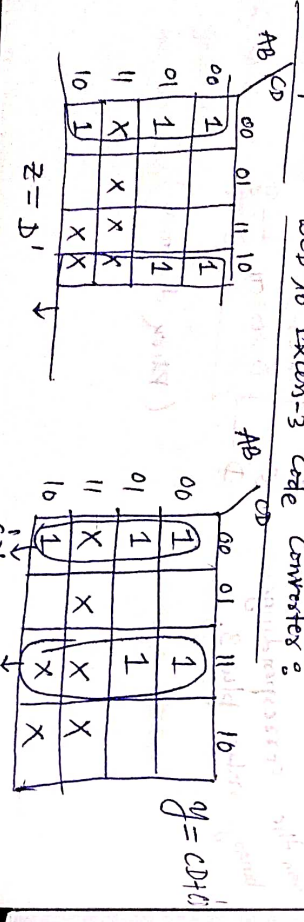
(Block diagram)

Truth Table for code conversion:

Decimal Input	BCD Input	Output Excess-3 code
	A B C D	w x y z
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 1 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 1
4	0 1 0 0	1 0 1 1
5	0 1 0 1	1 0 1 0
6	0 1 1 0	1 1 0 1
7	0 1 1 1	1 1 0 0
8	1 0 0 0	1 1 1 1
9	1 0 0 1	1 1 1 0
10	1 0 1 0	X X X X
11	1 0 1 1	X X X X
12	1 1 0 0	X X X X
13	1 1 0 1	X X X X
14	1 1 1 0	X X X X
15	1 1 1 1	X X X X

→ From 10 to 15 values have no meaning in BCD.
 So we assume that they are don't care combinations.

→ Draw the K-map for 4 outputs (w, x, y, z).

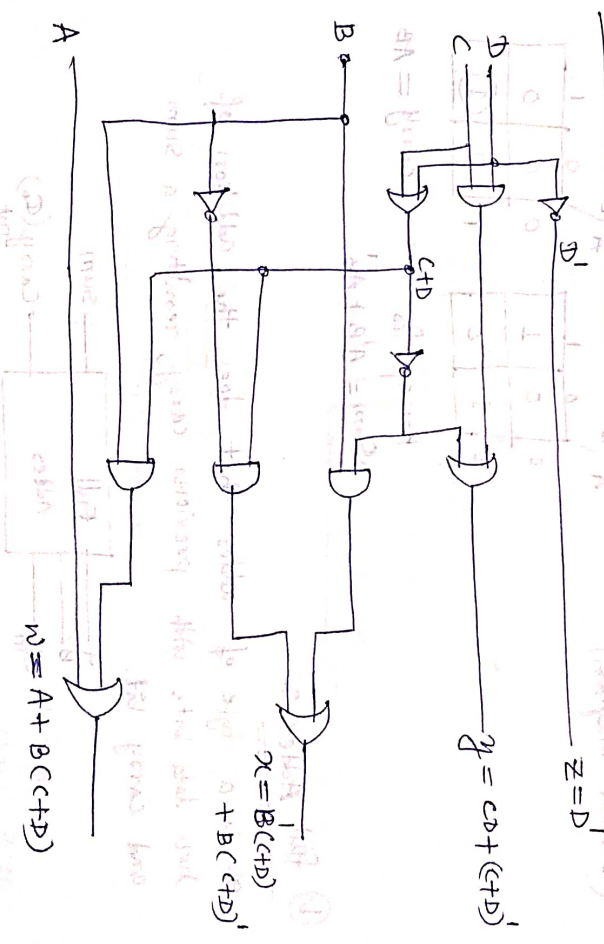


$$w = B'C + BD + B'CD$$

$$w = A + B + C + D$$

Outputs
 $z = D'$, $y = CD + C'D = CD + (C+D)'$
 $x = B'C + BD + B'CD$
 $= B'(C+D) + B(C+D)$
 $w = A + B + C + D$
 $= A + B(C+D)$

Logic diagram for BCD to Ex-3 code converter:



Binary Adder and Subtractors :-

Adder is of 2 types

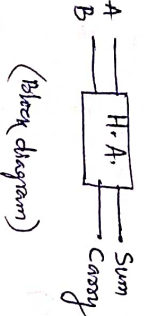
- (a) Half Adder (b) Full Adder

(a) Half Adder :-

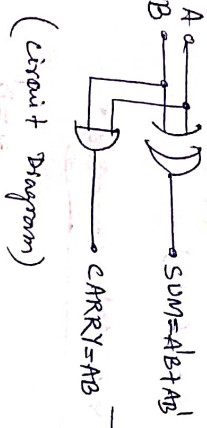
Half adder is the type of adder that can add two significant data bits and results sum and carry. It can't do the addition if there is a previous carry, so the name is given half adder.

Truth Table

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



(Block diagram)



(Circuit Diagram)

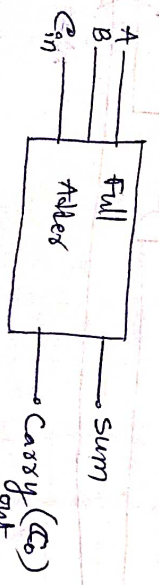
0	0	1
1	0	1

0	0	0
0	0	1

K-map for Sum = $A \oplus B$
Carry = AB

(b) Full Adder :-

It is a type of adder that does the addition of two data bits with previous carry, resulting a sum and carry bit.



Truth Table

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

K-map for Full Adder :-

A \ B Cin	00	01	11	10
0	0	1	1	1
1	1	1	0	0

$$\text{Sum} = AB'C_{in} + A'B C_{in} + AB C_{in}' + A'B'C_{in}$$

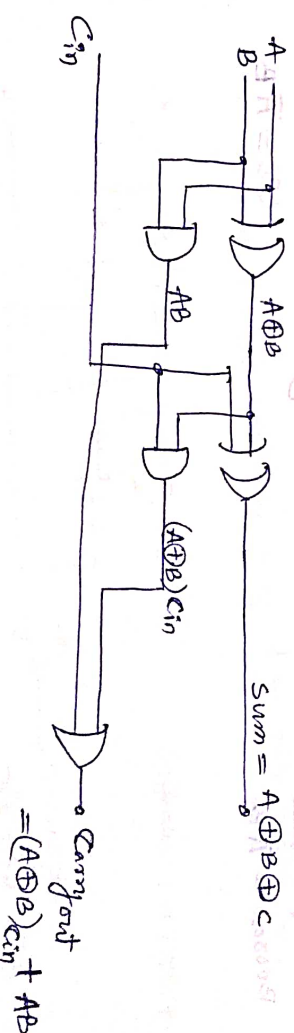
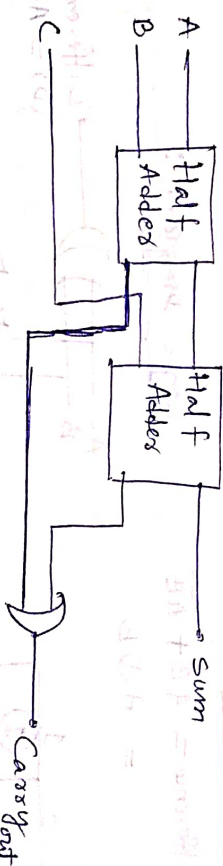
$$= A \oplus B \oplus C_{in}$$

K-map for Carry out

A \ B Cin	00	01	11	10
0	0	0	0	0
1	0	1	1	1

$$\text{Carry out} = AB + B C_{in} + C_{in} A$$

Implementation of full adder with two half adders and OR gate :-



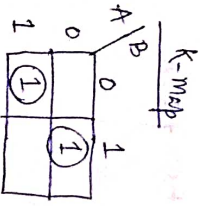
Subtractor :

It is a combinational circuit that can do the subtraction of binary digits resulting a difference bit and a borrow bit.

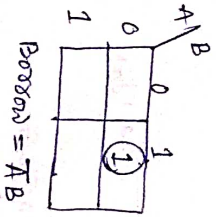
It is 2 types (a) Half Subtractor (b) Full Subtractor

Half Subtractor :

Half Subtractor is the type of subtractor that subtracts two binary data bits resulting a difference and borrow bit. It cannot do the subtraction if there is a previous borrow.



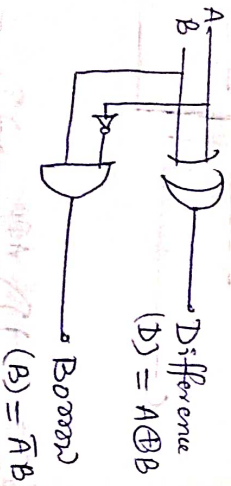
$$\text{Difference} = \bar{A}B + A\bar{B} = A \oplus B$$



$$\text{Borrow} = \bar{A}B$$

Inputs		Outputs	
A	B	Difference (D)	Borrow (B0)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

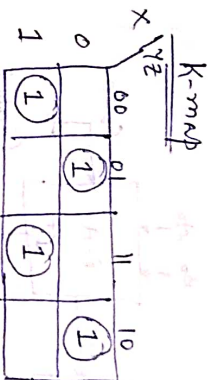
Logic Diagram :



Full Subtractor :

A full subtractor has 3 inputs and two outputs. X, Y, Z are the inputs to be subtracted in which Z represents borrow from the next stage. D & B are outputs.

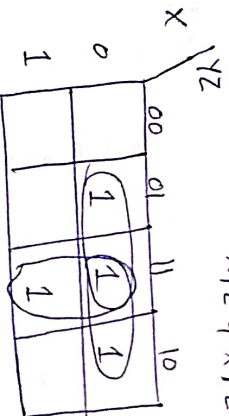
Z = input borrow = borrow from the next stage
B = output borrow, D = Difference



$$\text{Difference} = X'Y'Z + X'YZ + XY'Z' + XYZ'$$

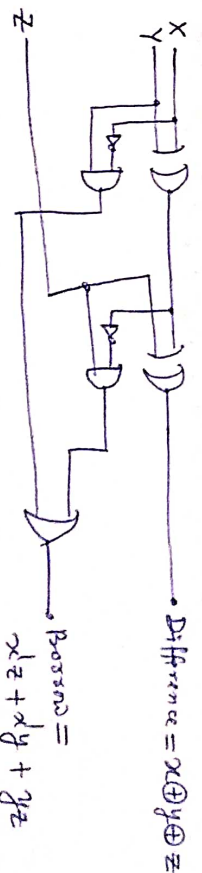
$$XY'Z' + XYZ'$$

Inputs			Outputs	
X	Y	Z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



$$B = X'Z + X'Y + YZ$$

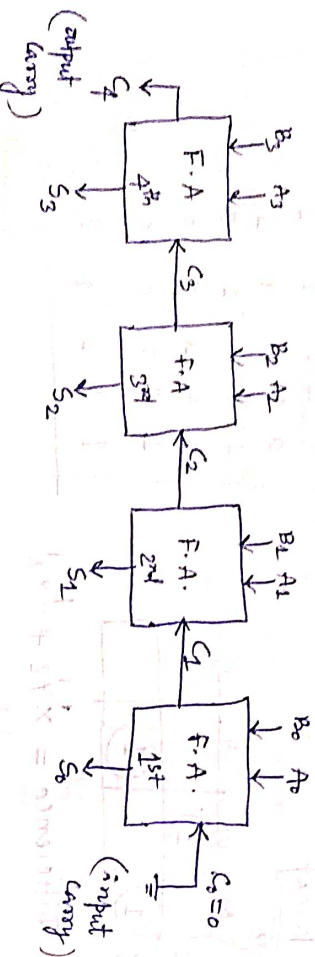
Logic Diagram of full Subtractor using two half Subtractor :



Parallel Binary Adder / Binary Adder :

- A parallel binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n bit numbers requires a chain of n -full adders.

Four bit Parallel Binary Adder :



where
F.A. — full adder

A_i = Augend bits (1^{st} + bit numbers)

B_i = Addend bits (2^{nd} 4 bit numbers)

C_i = carry bits

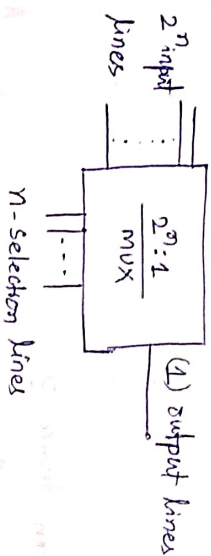
S_i = sum bits

$i = 0, 1, 2, 3$

Multiplexer (MUX) :

- A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line, and it is also called a data selector.
- The selection of a particular input line is controlled by a set of selection lines.
- The size $2^n:1$ MUX means, there are 2^n input lines and single output line.

General size of a Multiplexer :



$2^n:1$ MUX means

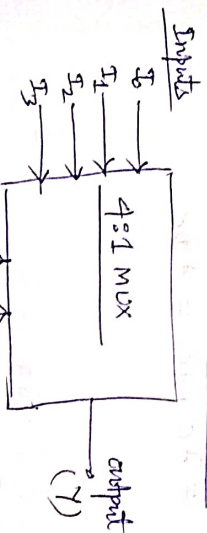
2^n : input lines

1 : output line

n : selection line

e.g. $n = 2$

$2^2:1$ MUX — $4:1$ MUX

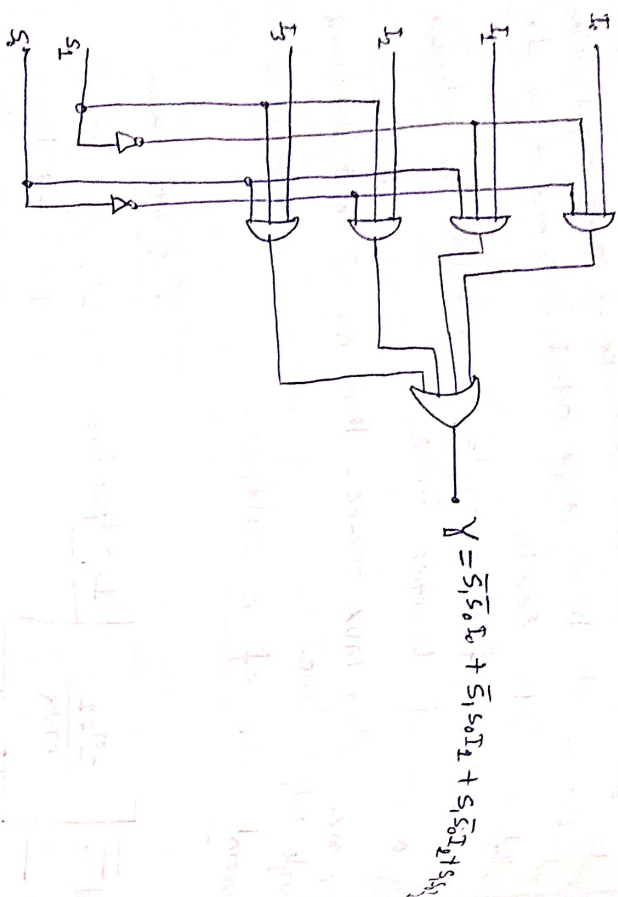


function Table

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

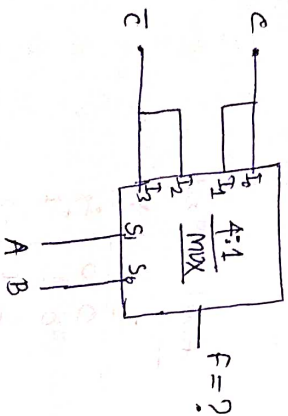
$$\text{Output } Y = \overline{S_1} \overline{S_0} I_0 + \overline{S_1} S_0 I_1 + S_1 \overline{S_0} I_2 + S_1 S_0 I_3$$

Logic diagram of 4:1 MUX



$$Y = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_0 I_3$$

(b) Find the output of the given figure ?



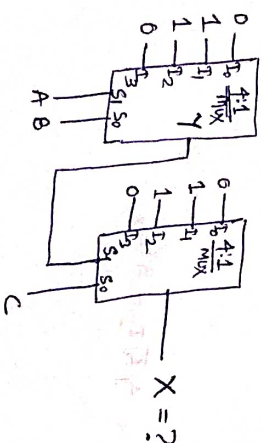
Solⁿ

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

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

$$= \bar{A} C + A \bar{C} = A \oplus C$$

(b) In the below circuit, find the X ?



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

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

$$= \bar{A} B \cdot 1 + A \bar{B} \cdot 1$$

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

$$X = \bar{Y} C + Y \bar{C}$$

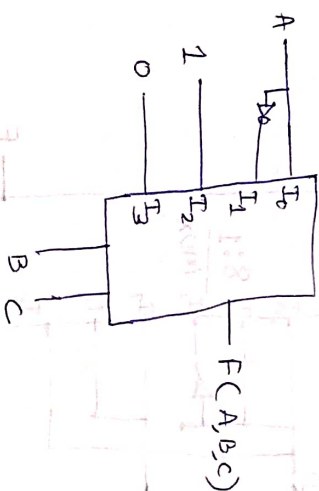
$$= (\bar{A} \oplus B) \cdot C + (A \oplus B) \cdot \bar{C} = (\bar{A} \bar{B} + A B) \cdot C + (\bar{A} B + A \bar{B}) \cdot \bar{C}$$

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

$$= \sum (4, 2, 1, 7)$$

$$X = \sum (1, 2, 4, 7)$$

(Q) A 4:1 MUX is used to implement a 3-input Boolean function as shown below. The Boolean function $f(A, B, C)$ implemented is



(a) $f(A, B, C) = \sum (1, 2, 4, 6)$

(b) $f(A, B, C) = \sum (1, 2, 6)$

(c) $f(A, B, C) = \sum (2, 4, 5, 6)$

(d) $f(A, B, C) = \sum (1, 5, 6)$

Solution

$$f = A \bar{B} \bar{C} + \bar{A} \bar{B} \bar{C} + 1 \cdot B \bar{C} + 0 \cdot B \cdot C$$

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

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

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

$$= \sum (1, 2, 4, 6)$$

(8) Implement the problem function $f(A, B, C, D) = \sum(1, 3, 4, 11, 12, 14, 15)$ using Multiplexers.

Solⁿ

• Here the number of variable is 4 means, 3 variable line and 1 variable connects to data input.

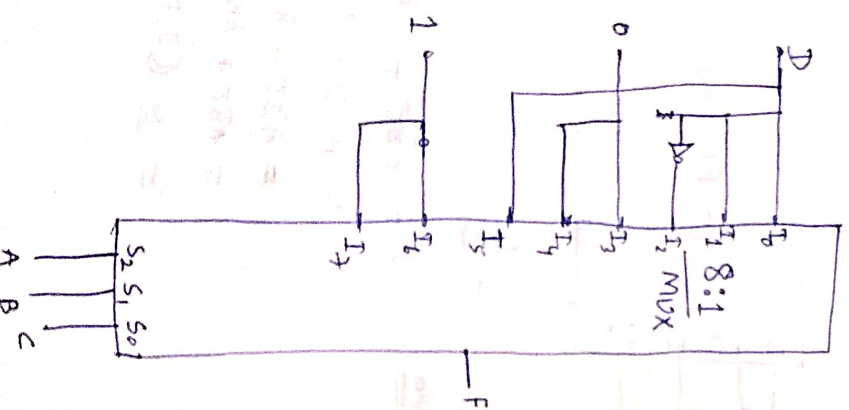
So $2^3:1 = 8:1$ MUX used.

• 3 variable A, B, C are connected to the selection input, if the $5:1$ MUX, in the order of A connected to S_2 , B connected to S_1 & C connected to S_0 .

• The variable D will connect to data inputs. The value for the data inputs lines are determined from the truth table of the function.

Truth Table

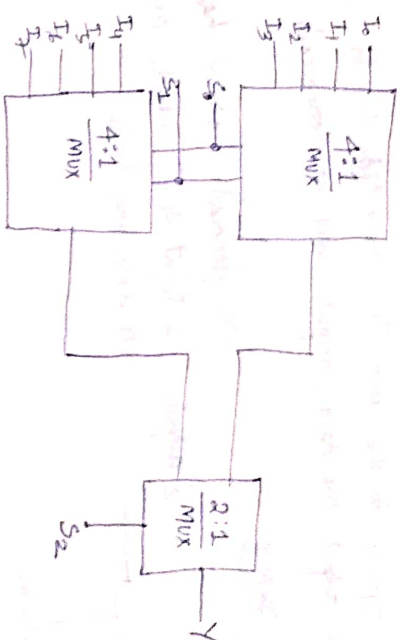
A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



(9) Construct a $5:1$ MUX using $4:1$ MUX & $2:1$ MUX.

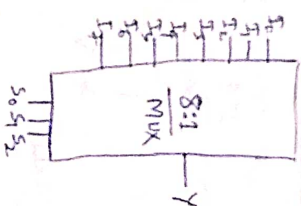
Solⁿ

• It is possible to implement higher order MUX using lower order MUX.



Truth Table

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



→ S_1, S_0 are the selection lines for two $4:1$ MUX.

and S_2 is the selection line for $2:1$ MUX.

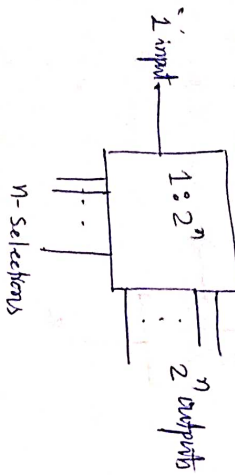
→ 1st 4 inputs (I_0, I_1, I_2, I_3) represents 1st $4:1$ MUX

2nd 4 inputs (I_4, I_5, I_6, I_7) represents 2nd $4:1$ MUX.

Demultiplexers (DEMUXES) : (Data Distributors)

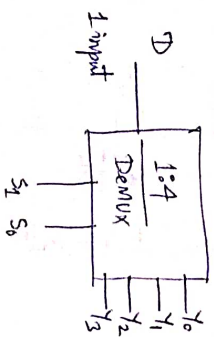
- A demux is a logic circuit that receives information on a single input and ~~transmits~~ transmits the same information over one of 2^n possible output lines.
- If a operation is opposite to the mux. The selection inputs determine to which output the data input will be connected.

Block diagram of Demux:



- In general Demux has i input & 2^n outputs and n selection lines.

e.g. 1:4 Demux



There are 4 outputs, out of which one output will be selected according to the selection line. That selected output receives the input signal.

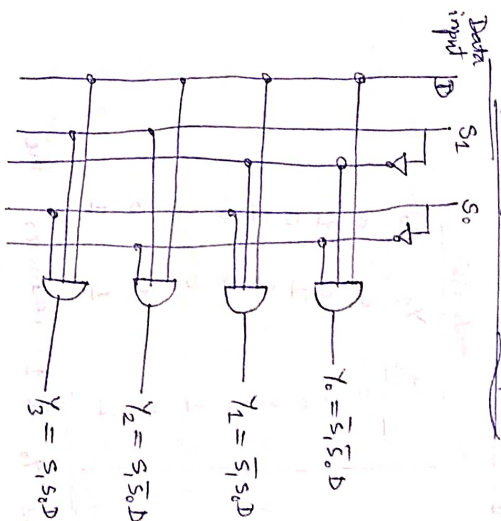
Truth Table

Data input	Select input	outputs				
D	S_1	S_0	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

- 1:4 Demux has a single input (D), 4 outputs (Y_0 to Y_3) and two select input (S_1 & S_0).
- From the truth table, it is clear that the data input is connected to output Y_0 when $S_1 = 0$ and $S_0 = 0$.

From truth table, the expression for output can be written as $Y_0 = \bar{S}_1 \bar{S}_0 D$, $Y_1 = \bar{S}_1 S_0 D$, $Y_2 = S_1 \bar{S}_0 D$, $Y_3 = S_1 S_0 D$.

Logic Diagram of output :



Encoders :

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or less) input lines and n -output lines.
- The output lines generates the binary code for the 2^n input variables.

Octal to Binary Encoder :

- An octal to binary encoder performs the opposite function of 3-to-8 decoder or binary to octal decoder.
- It accepts 8 inputs and produces a 3-bit output code corresponding to the activated input.

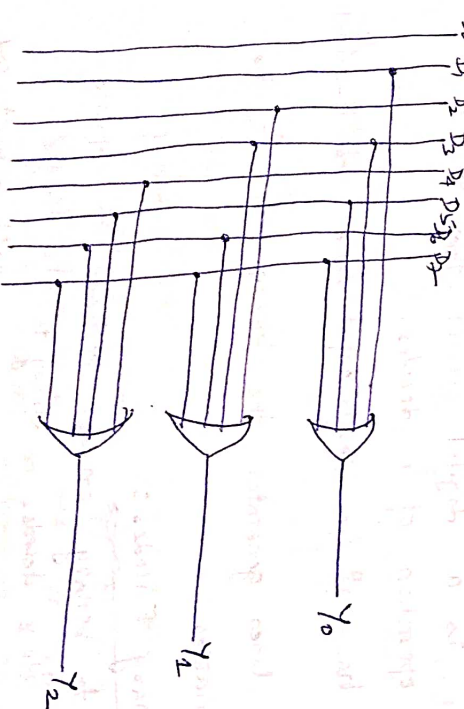
Truth Table of Octal to Binary ENCODER:

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	Y_2	Y_1	Y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	1	0	1
0	0	0	0	0	1	0	0	1	1	0
0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	1	1	1	1

The truth table shows that Y_0 is 1 whenever the input D_1 or D_3 or D_5 or D_7 is 1 or HIGH.

$$\text{So } Y_0 = D_1 + D_3 + D_5 + D_7$$

Similarly $Y_1 = D_2 + D_3 + D_6 + D_7$, $Y_2 = D_4 + D_5 + D_6 + D_7$



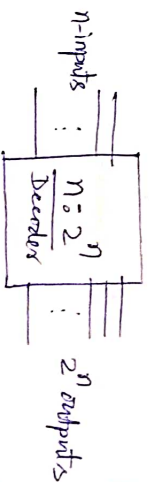
(Logic Diagram for Octal to Binary Encoder)

- Using the above expressions the octal to binary encoder can be implemented using three 4-input OR gates.

The circuit is designed in such a way that when D_0 is HIGH, the binary code 000 is generated, when D_1 is HIGH, the binary code 001 is generated and so on. Here we are taken 8 input combinations from total 28 inputs.

DECODER :

- A decoder is similar to Demux but without any data input.
- Decoding is necessary in applications such as data demultiplexing, digital display, DMA converter and memory addressing.
- A decoder is a combinational circuit that converts an n -bit binary input into 2^n output lines, such that each output line will be activated for only one of the possible combinations of inputs.
- If the n -bit decoded information has unused or don't care combinations, the decoder output will have less than 2^n outputs.



(from 2^n outputs one output will be active according to input binary combination.)

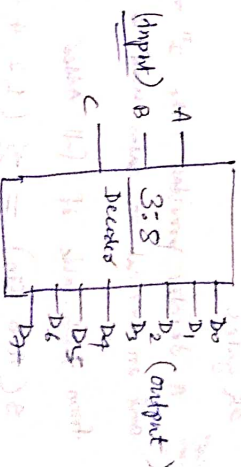
3 to 8 Decoder :

- A 3 to 8 decoder has 3 inputs (A, B, C) and 8 outputs D_0 to D_7 .
- Based on the 3 inputs, one of the output is selected.

Truth Table of 3:8 decoders :

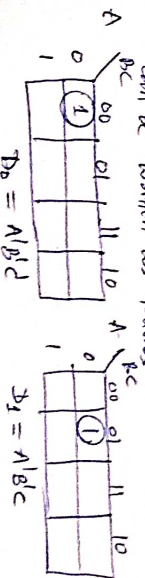
Inputs			Outputs							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

From the truth table, the logic expressions for the outputs can be written as follows:



Similarly

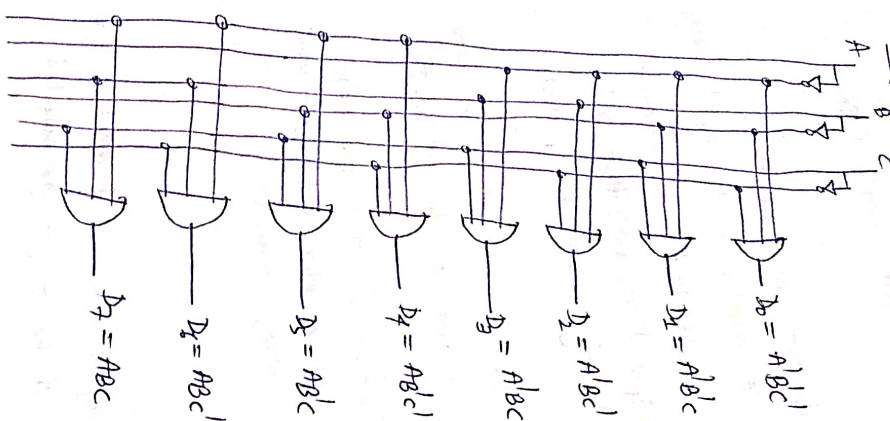
$$D_2 = \bar{A}B\bar{C}, D_3 = \bar{A}BC, D_4 = A\bar{B}\bar{C}, D_5 = A\bar{B}C, D_6 = AB\bar{C}, D_7 = ABC.$$



$$D_0 = \bar{A}\bar{B}\bar{C}$$

$$D_1 = \bar{A}\bar{B}C$$

Logic Diagram of 3 to 8



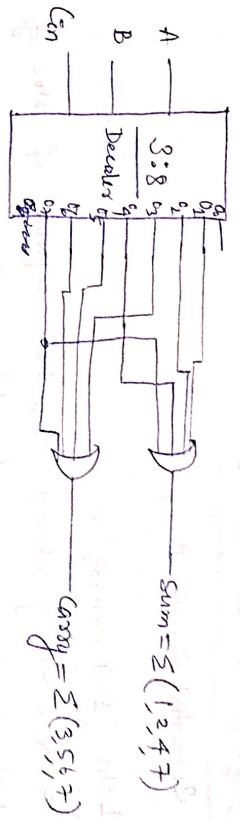
(Q) Implement a full adder circuit with a decoder and two OR gates

A decoder provides the 2^n minterms of n input variable and an external OR gate used to form the sum.

from truth table of full adder

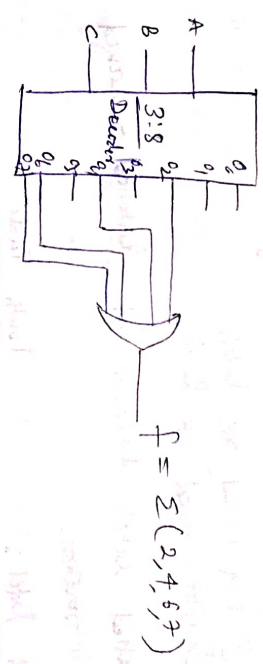
$$S(A, B, C_{in}) = \sum (1, 2, 4, 7)$$

$$C(A, B, C_{in}) = \sum (3, 5, 6, 7)$$



(Q) Implement the boolean function using Decoder & OR gates.

$$f(A, B, C) = \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} + ABC = \sum (2, 4, 6, 7)$$



Sequential digital system :-

Sequential circuit

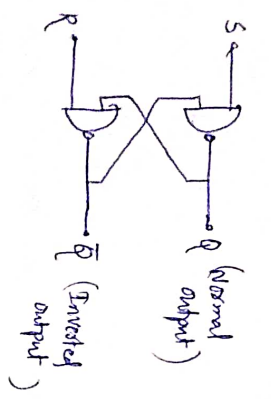
Latch (1 bit memory) :-

It is a memory element which has capable of storing 1 bit of information i.e. logic 1 or 0. This sequential circuit is called a latch, since one bit of information can be latched or latched.

We can construct the latches by using NAND gates & NOR gates. ① NOR based SR Latch ② NAND based SR Latch.

• its name called latch because it latched or latched one bit of information.

NAND based SR Latch :-



Logic diagram

NAND property

A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

{ NAND gate property }
Any '1' input = 0 means output = 1

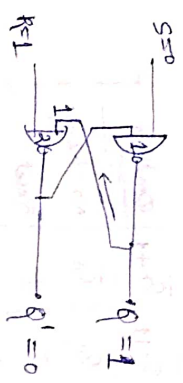
- It has two input i.e. normal (S) & another is Reset (R).
- It has two output i.e. normal (Q) & complemented ($\overline{Q}$).
- When output is 1 it is called set condition of flip. On set condition it stores '1'.
- Set condition $\rightarrow$ input $S=0, R=1$
- When output is 0 it is called Reset condition. When Reset condition it stores '0'.

Truth Table

Inputs		Outputs		
S	R	Q_{n+1}	$\overline{Q}_{n+1}$	Condition
0	0	1	1	Invalid
0	1	1	0	Set
1	0	0	1	Reset
1	1	Q_n	$\overline{Q}_n$	No change

Q_n - present state, Q_{n+1} - next state of flip

For input $S=0, R=1$



$\rightarrow S=0$ means $Q=1$

$\rightarrow$ from NAND gate property gate-1 input is $S=0$ so

at output is $Q=1$.

$\rightarrow$ NAND gate property any 1 input = 0 means

output 1.

$\rightarrow$ output $Q=1$ from gate-1

feed back to input of gate-2.

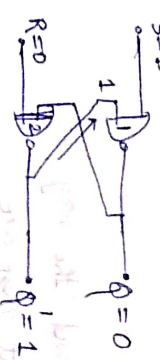
$\rightarrow$ Gate-2 both input are 1

so output $Q'=0$.

$\rightarrow$ For $S=0, R=1$

$\Rightarrow Q=1 \text{ \& } Q'=0$

For input $S=1 \text{ \& } R=0$



$\rightarrow$ Gate-2 has one input '0'

so its output $Q'=1$

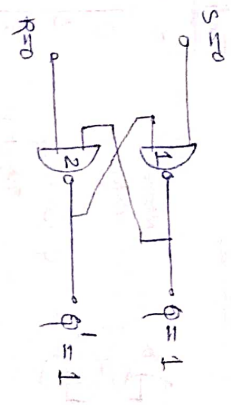
from NAND gate properties.

$\rightarrow Q'=1$ feedback to gate-1

so gate-1 input has both input 1 so its

output $Q=0$.

For input $S=0, R=0$



$\rightarrow$ Both gate-1 & gate-2 one

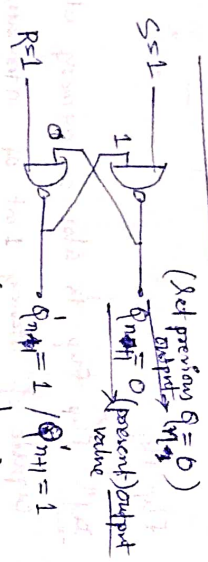
of the input is 0 so

both output $Q \text{ \& } Q'=1$.

$Q=1, Q'=1$ is a Invalid case.

because $Q \text{ \& } Q'$ are complement to each other.

$\rightarrow$ So $S=0, R=0$ is a Invalid case.

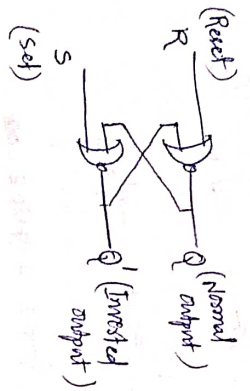


For input $S=1, R=1$

If previous value $Q=0, Q'=1$ we will get present value also $Q_{n+1}=0, Q'_{n+1}=1$

So there is no change of Q value / output value when both $S=1 \text{ \& } R=1$.

NOR based SR latch:



Truth Table

S	R	Q_{n+1}	$\bar{Q}_{n+1}$	Condition
0	0	Q_n	$\bar{Q}_n$	No change
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Invalid

$\left\{ \begin{array}{l} Q_n = \text{present state of F/F} \\ Q_{n+1} = \text{next state of F/F} \end{array} \right\}$

→ NOR properly
any one input 1
means output 0.

NOR

A	B	A+B
0	0	1
0	1	0
1	0	0
1	1	0

Flip Flop:

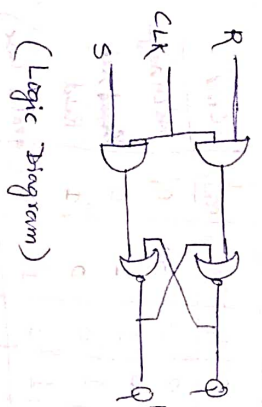
Defⁿ: Flip Flop is also a memory element, which has capable of storing 1 bit of information, either logic 1 or 0, on the application of clock pulse. But latch has no clock pulse.

Types of Flip Flop:

⊗ Depending on the number of inputs and the way the output changes with respect to input they are divided into different types.

- (1) SR Flip Flop
- (2) D-Flip Flop
- (3) JK Flip Flop
- (4) T-Flip Flop

① SR F/F (using NOR)



Truth Table

S	R	Q_{n+1}	$\bar{Q}_{n+1}$	Cond ⁿ
0	0	Q_n	$\bar{Q}_n$	No change
0	1	0	1	Reset
1	0	1	0	Set
1	1	1	1	Invalid

Characteristics Table:

Characteristics table define the Next state of Flip Flop if input and present state of Flip Flop known to us.

Characteristics Table:

present state of flip flop (Q_n)	input of flip flop	Next state of flip flop
Q_n	S	Q_{n+1}
0	0	0
0	1	1
0	0	0
0	1	1
1	0	0
1	1	1
1	0	0
1	1	1

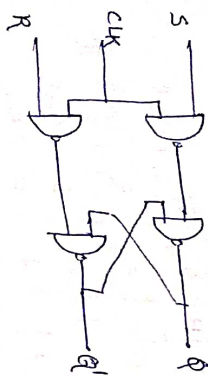
Characteristics Q_n :

Q_n	00	01	11	10
0	0	0	X	1
1	1	X	1	1

Characteristics Equation
 $[Q_{n+1} = (S + R'Q)]$

$[Q_{n+1} = S + R'Q]$

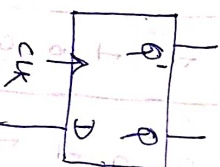
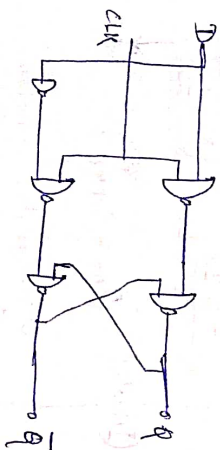
SR Flip Flop (using NAND)



Truth Table					
S	R	Q_{n+1}	Q_n'	Q_n	Cond ⁿ
0	0	Q_n	Q_n'	0	No change
0	1	1	0	0	Reset set
1	0	0	1	1	Reset
1	1	1	1	1	(Invalid case)

D-Flip Flop : (using NAND)

The D-FF is a modification of the clocked RS flip flop. It is basically an RS flip flop with an inverter in the R input, here output is same as the input to D so it is known as Data Flip flop. The data is appearing at output after some delay. So it is also known as Delay FF.

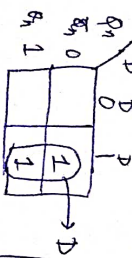


Truth Table

D	Q_{n+1}	Q_n	Cond ⁿ
0	0	0	Reset
1	1	0	Set

Characteristics Table

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

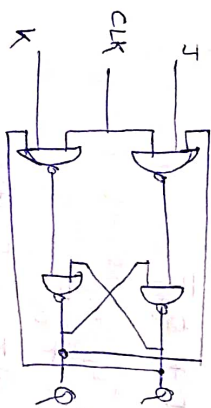


$$Q_{n+1} = D$$

Characteristics Equation

J-K Flip Flop :

A JK flip flop is a refinement of the RS flip flop in that the indeterminate state of the RS flip flop is defined. The invalid condition of RS flip flop is overcome with the help of extra feedback to first stage. When both inputs are HIGH then present output is complement of previous output and the output continuously changes between 1 & 0, on the application of clock pulse. So the output Toggle between 1 & 0. It is known as Toggling condition. The output continuously changes between 1 & 0 and output is uncertain. This problem is called race around condition. Race around condition can be removed in JK-Master Slave Flip flop.



(JK flip flop using NAND gates)

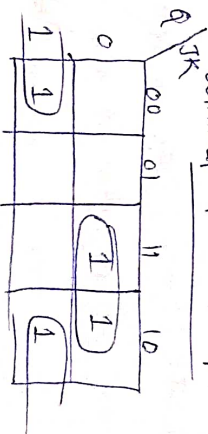
J	K	Q_{n+1}	Cond ⁿ
0	0	Q_n	No change
0	1	0	Reset
1	0	1	Set
1	1	Q_n'	Toggle / Race around

Truth Table

Characteristics Table :

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

Characteristics Eqⁿ is obtained from K-map:

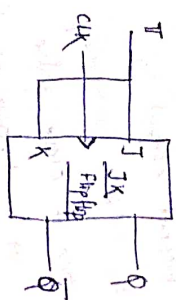


Characteristics Eqⁿ

$$Q_{n+1} = JQ' + K'Q$$

T-Flip Flop : (T stands for Toggle)

The T-Flip Flop is a single-input version of the JK flip flop. It is obtained from a JK type F/F if both inputs are tied together. It is known as toggle F/F as its output is continuously changing between 1 & 0.



Truth Table

T	Q _{n+1}	condition
0	Q _n	No change
1	Q _n '	Toggle

Characteristics Table :

Q _n	T	Q _{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Q _n	T	Q _{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

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

Characteristics Equation

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

$$= Q_n \oplus T = Q_n \oplus T$$

Registers : REGISTERS : #

Registers :

A Register is a collection of flip flops. For storing a large number of bits, the storage capacity is increased by grouping more than one flip flops.

An n-bit register is a cascade of n flip flops and can store an n-bit binary data.

Shift Registers :

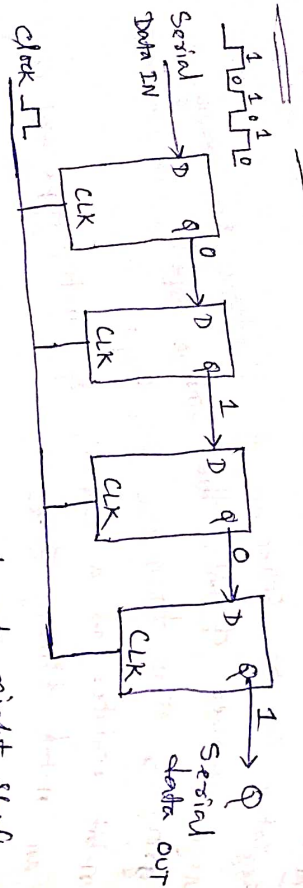
A register is capable of shifting its binary information in one or both directions is known as shift registers. The logical configuration of shift registers consist of a D-type flip flop cascaded with output of one flip flop connected to input of next flip flop. All flip flops receive common clock pulses which causes the shift in the output of the flip flop. The output of a given flip flop is connected to the input of next flip flop of the registers. Each clock pulse shift the contents of registers one bit position to right.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as

- (1) Serial in Serial out (SISO) (2) Serial in Parallel out (SIPO)
- (3) Parallel in Serial out (PISO) (4) Parallel in Parallel out (PIPO)

Shift registers are a type of logic circuits closely related to counters. They are basically for the storage and transfer of digital data.

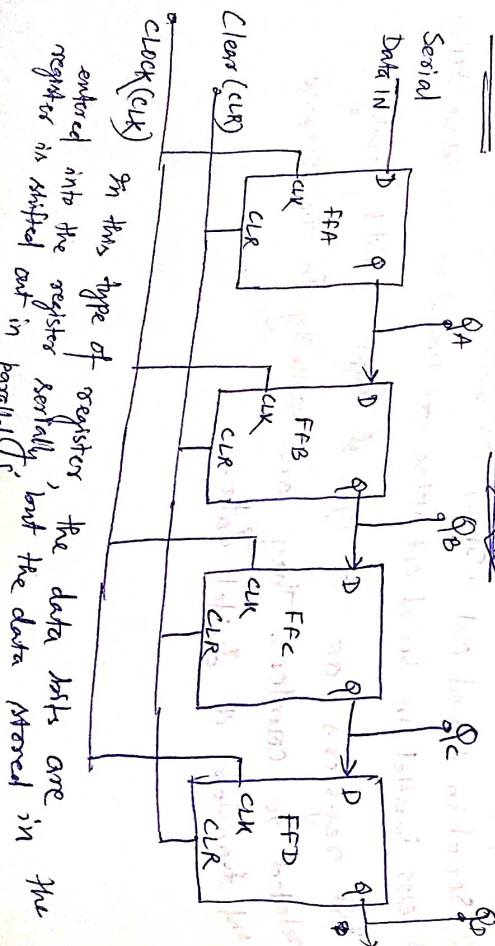
Serial IN, Serial OUT shift register:



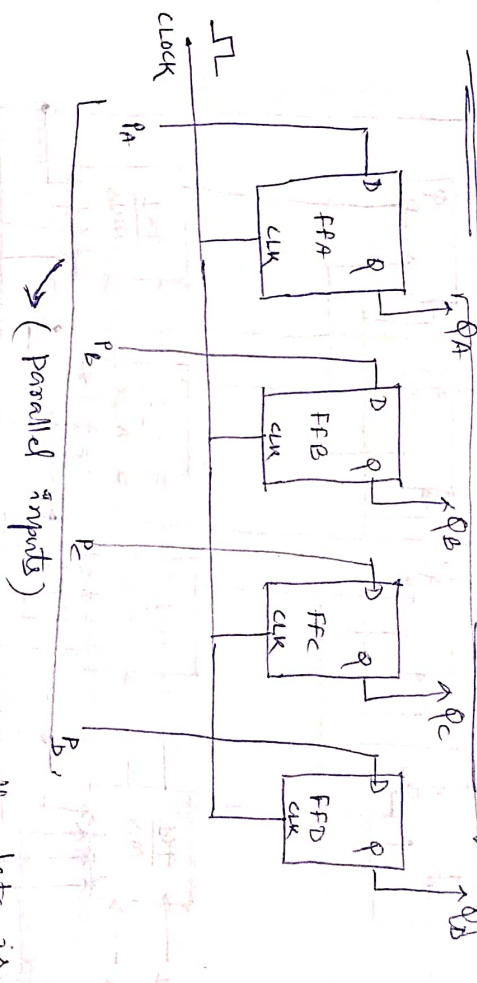
The logic diagram of 4-bit serial out, right shift registers with 4 stages. The registers can store 4 bits of data. Serial data is applied at the input D of the first F/F. The Q output of the 1st F/F is connected to the D input of another F/F. The data is outputted from the Q terminal of the last F/F.

When serial data is transferred into a register, each new bit is clocked into the 1st F/F at the positive going edge of each clock pulse. The bit that was previously stored by the 1st F/F is transferred to the 2nd F/F. The bit that was stored by the 2nd F/F is transferred to the 3rd F/F.

Serial IN, parallel OUT shift register:



Parallel-IN, parallel-OUT shift register:



On a parallel-in, parallel-out shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form.

Universal Shift Register:

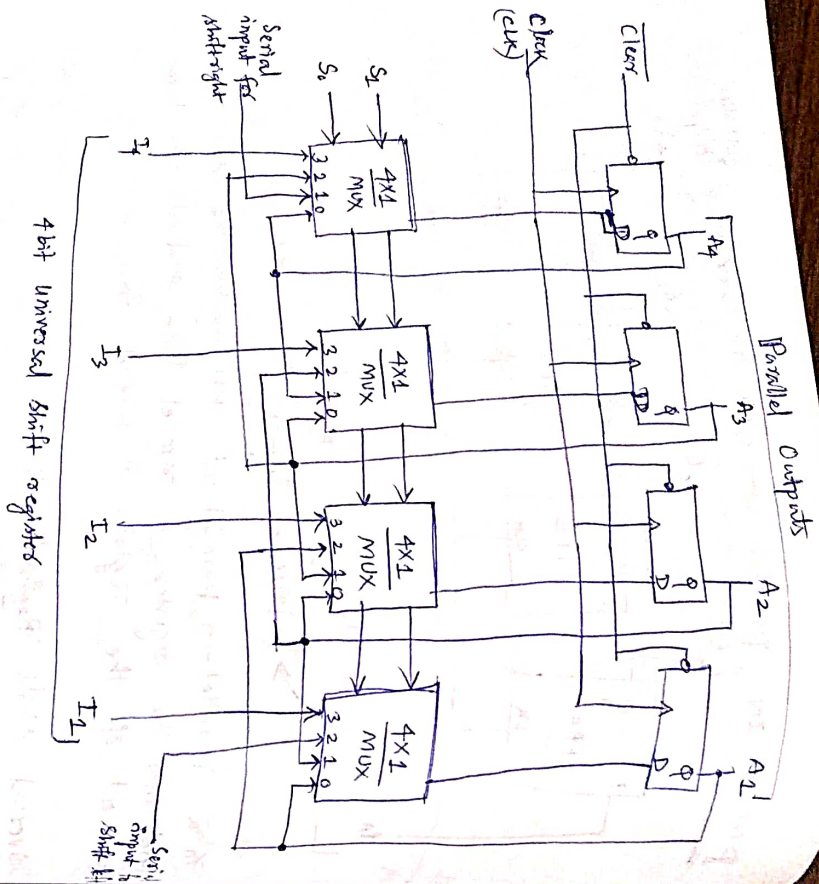
A register is capable of shifting in one direction only is a unidirectional shift register. One that can shift both directions is a bidirectional shift register.

→ If the register has both direction shift and serial, and parallel load capabilities, it is referred to as a universal shift register.

→ Universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be in serial form or parallel form.

Function table for Universal shift register:

MODE CONTROL		register operation:
S ₀	S ₁	
0	0	No change
0	1	Shift Right
1	0	Shift Left
1	1	Parallel Load



(figure: Logic diagram 4-bit universal shift registers.)

COUNTERS :-

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulses arrived. A specified sequence of states appears as counter output.

In electronic counters can be implemented quite easily using registers-type circuits such as the flip flops and a wide variety of classification exist :-

(1) Asynchronous (ripple) counters
(b) Synchronous counter $\rightarrow$ (Common clock is given to all flip flop).

(2) (a) UP counters } classified based on counting
(b) down counters }

(3) Ring counter
Johnson counter

(4) Modulus Counter

Asynchronous counters / (Ripple counters) :-

In asynchronous counter first flip flop is clocked by external pulse and then each successive flip flop is clocked by Q or Q' output of previous stage. As soon the clock of 2nd stage is triggered by output of 1st stage.

Because of inherent propagation delay time all flip flops are not activated at same time which results in

asynchronous operation.

$\rightarrow$ A binary ripple counter consists of a series connection of complementing flip flops (T or JK type) with the output of each F/F connected to the clock pulse input of the next higher order F/F.

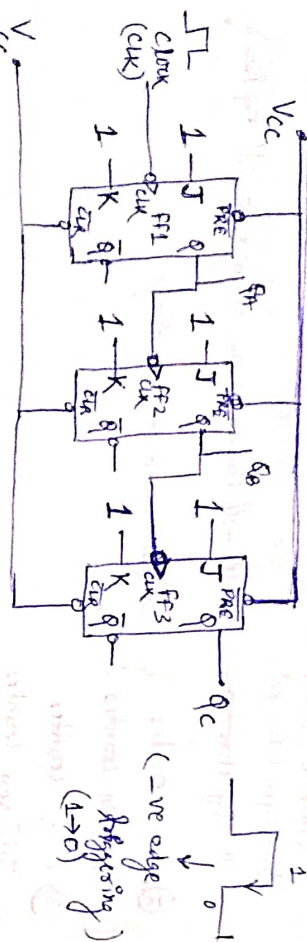
the FF holding the LSB receives the incoming count pulses.

On the case of an asynchronous counter, all the FFs are not clocked simultaneously, whereas in a synchronous counter all the FFs are clocked simultaneously.

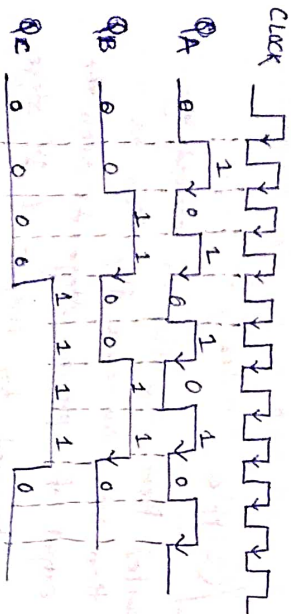
examples of synchronous counters

- (a) Ring counter
- (b) twisted-ring counter

Circuit Diagram: 3bit Asynchronous Up Counter :



Output waveform of Counter :



Counting sequence of 3bit binary Counter:

Counter State	QA	QB	QC
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

From the waveform

(i) QA - makes a transition whenever clock pulse changes from 1 to 0 (-ve edge)

(ii) QB makes a transition (changes from 0 to 1 or 1 to 0) whenever QA changes from 1 to 0 (-ve edge of pulse)

(iii) QC makes a transition whenever QB changes from 1 to 0 (-ve edge of pulse)

NSF

If $n = \text{No. of FF used in the counter}$
 $N = \text{No. of binary states that counter can count.}$

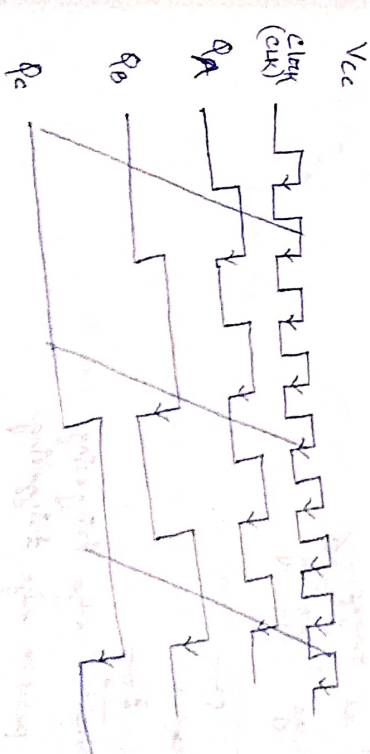
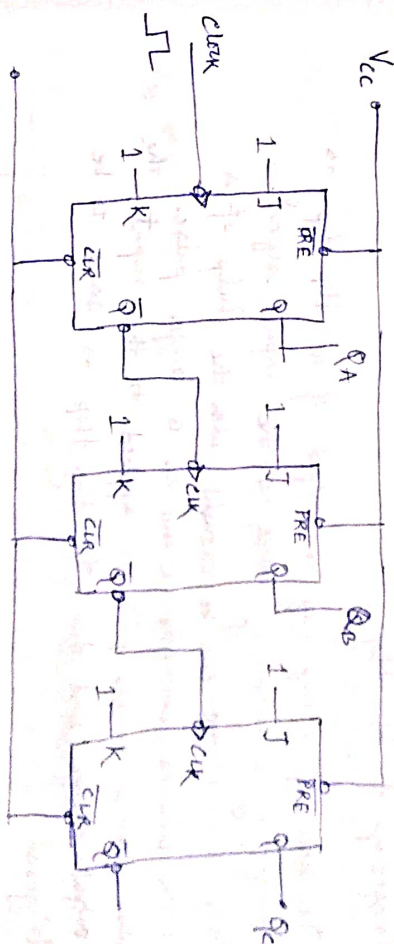
$$N \leq 2^n$$

N must be less than or equal to 2^n .

n bit asynchronous counter can count 2^n states from 0 to $2^n - 1$.

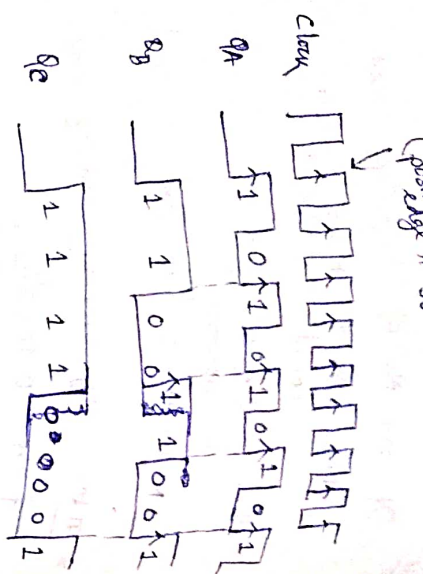
eg 3bit asynchronous counter can count $2^3 = 8$ states from 0 to $2^3 - 1$ (0 to 7).

Circuit Diagram :-
 3bit Asynchronous Down Counter :-



3bit asynchronous down counter

clock	Qc	Qb	Qa
0	1	1	1
1	1	1	0
2	1	0	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1
7	0	0	0
8	1	1	1
9	1	1	0



Triggering of flip flop :

The state of a flip flop or output of a flip flop is changed by a momentary change in the input signal. This small change can be brought with the help of a clock pulse or commonly known as a trigger pulse. when such a trigger pulse is applied to the input, the output changes and thus the flip flop is said to be triggered.

Types : (1) Level triggering
(2) Edge triggering

Trigger on high clock level



Negative edge triggering

positive edge triggering

Modulus of Counter (MOD-N counter) :

If the counter counts N states from 0 to N-1, then it is called MOD-N counter.

e.g. MOD-5 counter, that counts 5 states from 0 to 4.

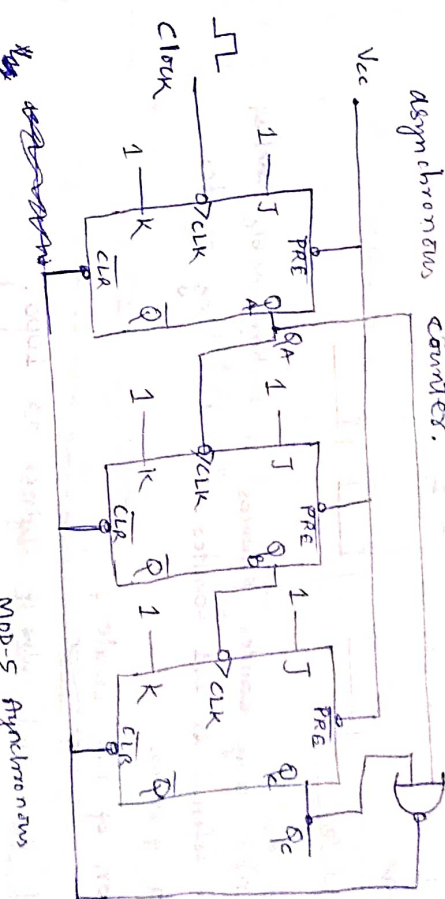
Designing

MOD-5 asynchronous counter :

$$N=5$$

$$5 \leq 2^n \Rightarrow n=3$$

So, 3 no. of f/f's are used to design MOD-5 asynchronous counter.



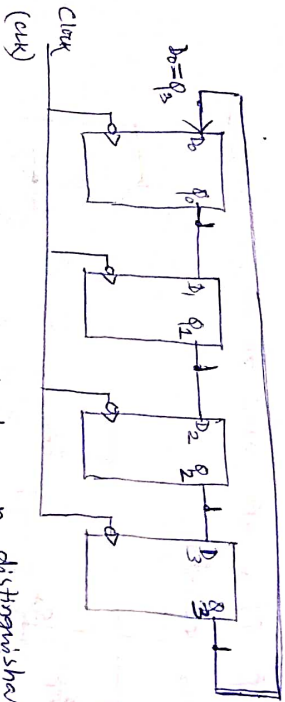
MOD-5 asynchronous counter

Qa	Qb	Qc
1	1	1
0	1	1
1	0	1
0	0	1
1	1	0
0	1	0
1	0	0
0	0	0

clock	Qc	Qb	Qa
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	0	0	0

Ring Counter : where all the Flip flops

- It is a synchronous counter. all the flip flops are clocked simultaneously.
- In the shift register if the serial output Q of the last flip flop is connected back to the serial input of 1st flip flop then an injected pulse will keep circulating. This circuit is referred to as a ring counter.



- A n -bit ring counter produces n distinguishable states. It is referred to as $n:1$ counter. e.g. 4-bit ring counter counts 4 states.

Operation of ring counter :

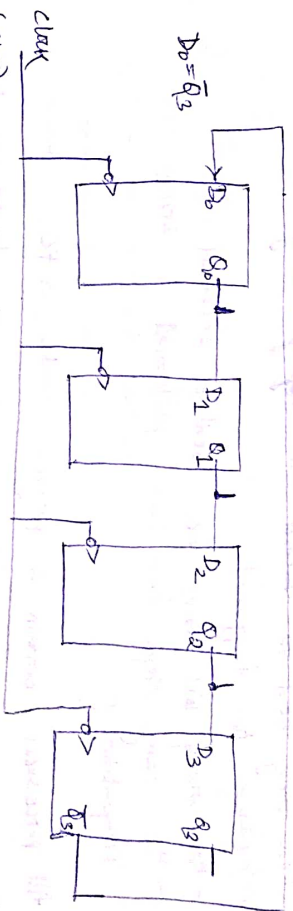
Let the initial value of register is 1000.

clock	input $D_0=Q_3$	output Q_0, Q_1, Q_2, Q_3
1	0	1 0 0 0
2	0	0 1 0 0
3	0	0 0 1 0
4	1	0 0 0 1

- The single bit is shifted right with every clock pulse and circulates around the circuit.
- 4-bit ring counter counts 4 distinguishable states.
- These sequential nonoverlapping pulses are used in stepper motors to rotate it in steps.

Johnson Counter (Twisted Ring) :

- It is a synchronous counter.



- In the shift register if $\bar{Q}_0$ of last flip flop is connected to the serial input of first flip flop the resulting circuit is called twisted ring or Johnson counter.

- A n -bit Johnson counter produces $2n$ -distinguishable states.
- It produces the same input sequences as output sequence after $2n$ clock pulses. It is referred to as $2n:1$ counter.

Operation :

clock	input $D_0=\bar{Q}_3$	output Q_0, Q_1, Q_2, Q_3
1	1	1 0 0 0
2	1	1 1 0 0
3	1	1 1 1 0
4	1	1 1 1 1
5	0	0 1 1 1
6	0	0 0 1 1
7	0	0 0 0 1
8	0	0 0 0 0

initial content of counter

Counted Application: to keep track of simple clocks

- Counter can act as simple clocks to keep track of time.
- You may need to record how many times something has happened.
- How many bits have been sent or received?
- How many bits have been performed in some computation?
- How many steps have been performed in some computation?

→ All processors contain a program counter, or PC. The PC keeps track of the instruction currently being executed. The PC increments once on each clock cycle, and the next program instruction is then executed.

→ They are used as frequency dividers, as divide by 'N' counters.

→ Application of Synchronous counters - Alarm clock, Set AC Timer, Set time in camera to take the picture, Flashing light indicators in automobiles, Car parking control etc.

→ Commonly used in home appliances like washing machine, microwave, Time schedule led indicators, Key board controllers.

Logic family :

The entire range of digital ICs is fabricated using either bipolar devices or MOS devices or combination of the two.

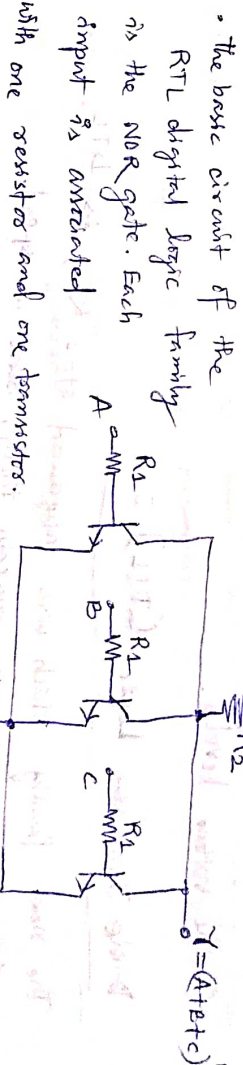
Bipolar family include:

- (1) Resistor transistor Logic (RTL)
- (2) Diode transistor Logic (DTL)
- (3) Transistor transistor Logic (TTL)
- (4) Emitter Coupled Logic (ECL)
- (5) Integrated injection Logic (IIL)

MOS families

- (1) PMOS family
- (2) NMOS family
- (3) CMOS family

Resistor Transistor Logic (RTL) :



The collectors of the transistors (Fig: Basic RTL NOR gate) are tied together at the output.

• If any input of the RTL gate is high, the corresponding transistor is driven into saturation and the output goes low, regardless of the states of the other transistors. If all inputs are low, all transistors are cut off because $V_{BE} < 0.6\text{V}$ and the output of the circuit goes high, approaching the value of the supply voltage V_{CC} .

<u>NOR gate (2 input)</u>		High (V_{cc})
Low (0.2V)	Low (0.2V)	Low
Low (0.2V)	High (3.5V)	Low
High (3.5V)	Low (0.2V)	Low
High	High	Low

Parameters of the RTL logic circuit are

Parameters of the RTL logic circuit are
 Fan-out: 5, Noise Margin: 0.2V, Propagation Delay: 25ns
 Power dissipation: 12mW/gate, Power supply voltage: 3-5V
 * RTL has only advantage that it has low power dissipation.
 * It has poor noise immunity and slow in speed. (disadv.)

On digital electronics, the fan-out is the number of gate inputs driven by the output of another single logic gate.
 Noise Margin: Ability of the gate to tolerate fluctuations of the voltage levels.

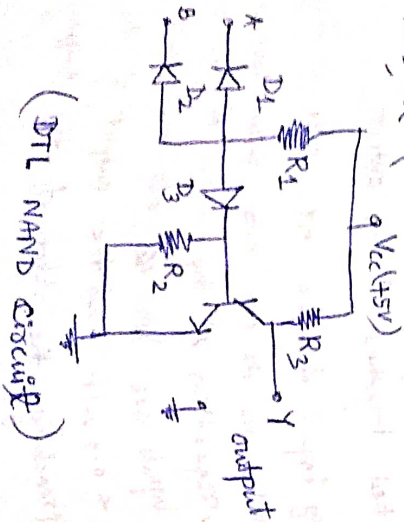
Diode Transistor Logic (DTL):

The next family which was introduced after RTL, is DTL, which have better noise margin though slow in speed.

The basic DTL gates are AND, OR & NAND.

DTL NAND gate:

DTL NAND circuit is shown in figure.



If both the inputs are high (logic 1), the diodes D_1 & D_2 are reverse biased and passes no current and point X is clamped to +5V; the diode D_3 is forward biased and current flows through R_1 & D_3 to the base of Q_1 thus saturating the transistor, this causes the output point Y to go low (logic 0).

If either input change to its low logic (logic 0) the corresponding input diode conducts and drives current from base of Q_1 , thus the transistor is off and the output voltage rise to logic 1.

Parameters: fan-out: 5, Noise margin: 0.7V, propagation delay: 30ns, Power Dissipation: 8-12mW/gate, Power supply Voltage: +5V.

Advantages: (1) It has better noise immunity than RTL circuit. (2) It is much more economical because of the use of diode in place of resistors and capacitors (3) It has low power dissipation.

Disadvantages: (1) It has poor noise margin (2) It has high propagation delay

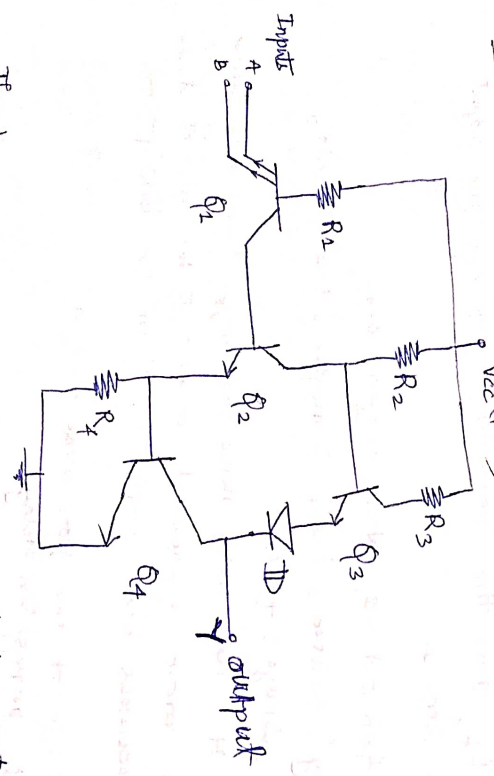
Transistor-Transistor Logic (TTL):

TTL logic is modified form of DTL logic, the input diodes in the case of DTL logic are replaced by transistors with more than one emitters with more than one emitters as input.

TTL gates available in 3 different types of output configuration : —

- ① Open-collector output
- ② Totem-pole output
- ③ Three-state output

TTL NAND gate with Totem-pole output :



The basic TTL logic element is a NAND gate and is shown in figure. The circuit consists of 3 stages one input stage associated with multiple emitters transistors Q_1 , a switching stage consisting of transistors Q_2 and output stage having transistors Q_3 & Q_4 .

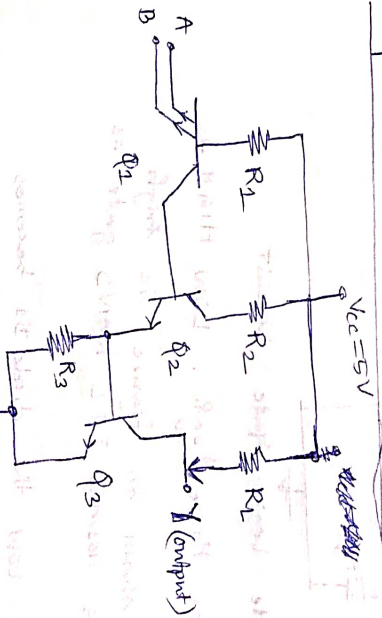
of either or both the inputs A & B are at logic 0, Q_1 conducts and no current flows into the base of Q_2 . Therefore Q_2 is OFF, the voltage at the collector of Q_2 rises to V_{cc} .

If both input A & B are at logic 1 level, no current flows out of the emitter of Q_1 but it does flow through the base of Q_2 through R_1 thus turning on Q_2 .

As Q_2 is ON, it supplies base current to Q_4 causing it to turn ON, when Q_4 is ON its collector voltage and thereby output Y is at zero level. Note that because Q_2 is ON, it collector voltage drops and keeps Q_3 in the OFF state.

The function of diode in the circuit is to prevent both Q_3 & Q_4 from being turned ON simultaneously because if both transistors becomes ON they will offer low impedance to the supply which will draw excessive current and produce large noise spike at the output.

Open-collector TTL NAND gate :



- In open collector TTL logic in the output stage, the emitter followers transistors Q_4 & diode D are absent as compared to the totem-pole logic.
- The collector of Q_3 is not connected anywhere internally in the circuit and is open which is connected to common external collector resistor R_L .

Each IC's are used for operations with an enabling using resistive loads and for operations with an enabling loads.

→ With any input LOW, Q_2 is ON thus making Q_3 & Q_4 OFF drawing output HIGH resulting in Q_1 OFF and Q_2 & Q_3 & Q_4 ON which makes the output LOW.

TTL Tristate Logic :

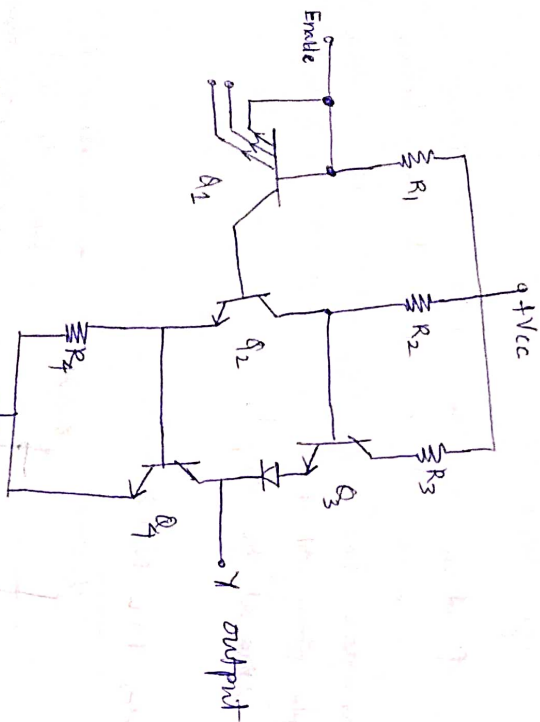


Fig:- TTL Tristate Logic Gate Circuit

• A Tristate Logic has three logic. LOW, HIGH & IMPEDANCE state, when enable lines as shown in figure is high the circuit works as normal TTL NAND gate as explained above.

• When enable line is LOW the diode D1 becomes forward biased which keeps Q_2 ON, Q_2 , Q_3 & Q_4 OFF, with both output transistors Q_3 & Q_4 OFF, the output impedance is VERY HIGH, this is the HIGH impedance state.

The major parameters, advantages and disadvantages of TTL IC's are listed below:

Parameters :

fanout : 10, propagation delay : 7-10 ns,
Power dissipation : 10mW/gate, Noise Margin : 0.4V
Power Supply Voltage : +5V

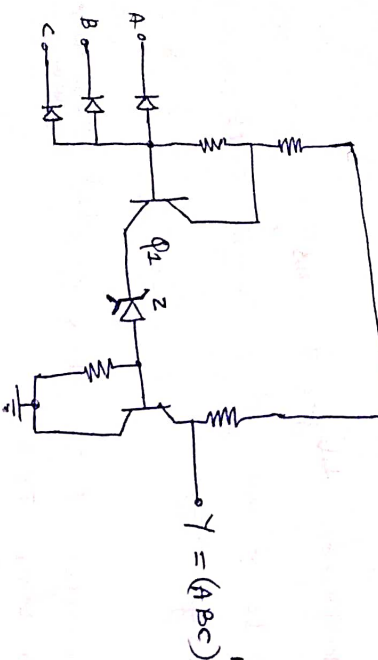
Advantages

- The main advantages of TTL IC's is that they are compatible with other IC's.
- It has high speed of operation.
- It's smaller size yields more function on an IC.
- It has low output impedance which improves the fan out capability.
- It has good noise immunity, in worst case it is 0.4V & typically it touches 1V.
- It is less expensive.

Disadvantages :

It generates switching transients, which can be eliminated by the use of bypass capacitors.

HTL (High Threshold)



→ High Threshold Logic (HTL) is a variant of Diode-transistor logic which is used in such environments where noise is very high.

→ The threshold values at the input to a logic gate determine whether a particular input is interpreted as a logic '0' or a logic '1'. (For example anything less than 1 volt is a logic '0' and anything above 3 volt is a logic '1'. In this example the threshold values are 1V & 3V.

→ HTL incorporates Zener diodes to create a large offset between logic 1 & logic 0 voltage levels.

These devices usually run off a 15VOLT power supply and were found in industrial control where the effect of high differential was intended to minimize the effect of noise.

Advantages

- * increased noise margin
- * spike control
- * High Noise Threshold value

Disadv:

- slow speed
- High power drain.

DETL (Direct Coupled Transistor Logic)

→ DETL is similar to resistor-transistor logic (RTL), but the input transistor bases are connected directly to the collector outputs without any base resistors.

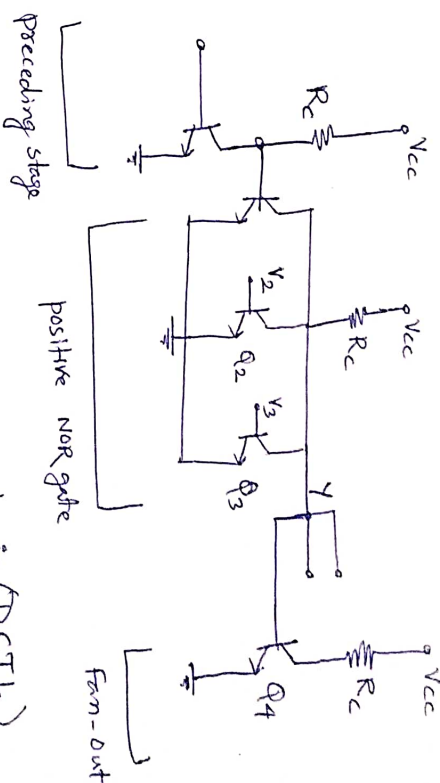


Figure: Direct Coupled Transistor Logic (DETL) positive NOR Gate.

→ DETL gates have fewer components, are more economical and are simpler to fabricate onto ICs than RTL gates.

→ Unfortunately, DETL has much smaller signal levels, has more susceptibility to ground noise and required matched transistor characteristics.

As