Unit - I

Digital systems:

Digital systems have a Prominent role in everyday life that we refer to the Present technological Period as the digital age. digital systems are used in communication business transactions, traffic control, space craft guidance, medical treatement, weather monitoring, the Internet and many other Commercial, industrial and scintific enterprises. we have digital telephones, digital televisions, digital virsatile disks, digital Cameras, handheld devices and digital Computers. These devices have graphical user Interfaces (GUIs) which enable them to execute Commands that apper to the user to be simple but which involves Precise execution of a sequence of Complex Internal instructions. The most stoiking Property of the digital Computer is its generality. It can follow a sequence of instructions, Called Program, that operates on given data. The user can specify and change the program or the data according to the specific need.

Early digital Computers were used for numeric computations In this case the discrete elements were the digits from this application, the term digital Computer emerged. Discrete elements of information are represented in a digital system by Physical quantities Called signals.

Electrical signals such as voltages and currents are most common. Electronic devices called transistors predominate in the circuitry that implement these signals. The signals in present-day electronic digital systems use just two discrete values and are therefore said to be binary. A binary digit called a bit has two values: 0 & 1. Discrete elements of information are represented with group of bits called binary codes. The decimal digits 0 to 9 are represented in a digital system with a code of four bits.

Decimal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A 10	1010
B 11	1011
C 12	1100
D 13	1101
E 14	1110
F 15	1111

Digital systems can be made to operate with extreame reliability by using error correcting codes. A digital system is an interconnection of digital modules. To understand the operation of each module, it is necessary to have a basic knowledge of digital circuits and their logical functions

Binary Numbers:

Binary numbers system also called the base-2 number system, is a method of reprasenting nombers that counts by using combination of only two nombers: 0 & 1

Compater use the binary number system to manipulate and store all of their data including nombers, words, videos, graphics and music.

The nomber system that you are familiar with that you use everyday, is the decimal number system also commonly refered as 10-base 10 system. when you perfoom computation such as $3+2=5$ or $21-7=14$. you are using the decimal nomber system

The base-10 system is a positional system, where the right most digit is the one's position, the next digit to the left is ten position and so on.

eg: 6349

$10^3 10^2 10^1 10^0$

$6\times10^3+3\times10^2+4\times10^1+9\times10^0$

$6\times1000+3\times100+4\times10+9\times1$

Number base conversions

Convert decimal number to Binary number:

The numbers which are present in Decimal format we are going to convert that number into Binary format by using LCM Method

eg: $(25)_{10} = (\ )_2$

2	25
2	12 - 1
2	6 - 0
2	3 - 0
	1 - 1

$(11001)_2$

$(25)_{10} = (11001)_2$

Convert Binary number to decimal number:

The numbers which are present in Binary format we are going to convert that number into Decimal format by using Power of 2 and add all the numbers The we will get Decimal number.

eg: $(11001)_2$

1	1	0	0	1
2^4	2^3	2^2	2^1	2^0

$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1$

$16 + 8 + 0 + 0 + 1 = (25)$

eg: $(7382)_{10} - (\quad)_2$

2	7382
2	3691 - 0
2	1845 - 1
2	922 - 1
2	461 - 0
2	230 - 1
2	115 - 0
2	57 - 1
2	28 - 1
2	14 - 0
2	7 - 0
2	3 - 1
	1 - 1

$(1110011010110)_2$

2048

→ 1 1 1 0 0 1 1 0 1 0 1 1 0

$2^{12}\ 2^{11}\ 2^{10}\ 2^9\ 2^8\ 2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$1\times2^{12} + 1\times2^{11} + 1\times2^{10} + 0\times2^9 + 0\times2^8 + 1\times2^7 + 1\times2^6 + 0\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0$

$4096 + 2048 + 1024 + 0 + 0 + 128 + 64 + 0 + 16 + 0 + 4 + 2 + 0$

$4096 + 2048 + 1024 + 128 + 64 + 16 + 4 + 2$

$= (7382)_{10}$

Octal Number System to Binary number system:

① $(217)_8 = (010001111)_2$

2 1 7

010 001 111

② $(53)_8 = (101011)_8$

5 3

101 011

Binary number system to octal number system:

(1) $(010001111)_2 = (217)_8$

$010 \mid 001 \mid 111 = (217)_8$

2 1 7

Hexa decimal system to Binary system :-

① $(4C4)_{16} = (010011000100)_2$

② $(5A6)_{16} = (010110100110)_2$

Binary number system to Hexadecimal :-

$(0100 \mid 1100 \mid 0100) = (4C4)_{16}$

4 C 4

$(0101 \mid 1010 \mid 0110) = (5A6)_{16}$

5 A 6

Decimal number system to octal number system:

$(78)_{10} = (116)_8$

8 | 78
8 | 9 - 6
1 - 1

$(54)_{10} = (66)_8$

8 | 54
6 - 6

$(240)_{10} = (360)_8$

8 | 240
8 | 30 - 0
3 - 6

octal number system to Decimal number system:

$(360)_8 = 3\ \ 6\ \ 0$

$8^2\ \ 8^1\ \ 8^0$

$3 \times 64 + 6 \times 8 + 0 \times 1$

$192 + 48 + 0$

$(240)_{10}$

Decimal number system to Hexadecimal number system:

$(1220)_{10} = (4C4)_{16}$

16 | 1220
16 | 76 - 4
4 - 12

Hexa decimal number system to Decimal number system:-

$(4C4)_{16} = (\ \ )_{10}$

$4\ \ C\ \ 4$

$16^2\ \ 16^1\ \ 16^0$

$4 \times 16^2 + 12 \times 16 + 4 \times 16^0$

$1024 + 192 + 4 = (1220)_{10}$

Decimal fraction to Binary fraction:

$(21.75)_{10} = (10101.11)_2$

2 | 21	$0.75 \times 2 = 1.50$
2 | 10 - 1	$0.50 \times 2 = 1.00$
2 | 5 - 0	
2 | 2 - 1	
1 - 0	

Binary fraction to decimal fraction:

$(10101.11)_2 = (\quad)_{10}$

1	0	1	0	1	.	1	1
2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}

$1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$

$16 + 4 + 1 = 21$

$(21.75)_{10}$

$1 \times \frac{1}{2} + 1 \times \frac{1}{4}$

$\frac{1}{2} + \frac{1}{4} = \frac{2+1}{4} = \frac{3}{4}$

0.75

$(0.513)_{10} = (\quad)_8$

$0.513 \times 8 = 4.104$

$0.104 \times 8 = 0.832$

$0.832 \times 8 = 6.656$

$0.656 \times 8 = 5.248$

$0.248 \times 8 = 1.984$

$0.984 \times 8 = 7.872$

$(.406517\ldots)_8$

Power of Two :-

n	2^n	n	2^n
0	1	13	8192
1	2	14	16,384
2	4	15	32,768
3	8	16	65,536
4	16	17	131,072
5	32	18	262,144
6	64	19	524,288
7	128	20	1048,576
8	256	21	2,097,152
9	512	22	4,194,304
10	1024	23	8,388,608
11	2048		
12	4096		

Addition of Two Binary Numbers :-

(i) 101101 + 100111

augend = 101101
addend + 100111
1111
sum = 1010100

(ii) 1011 + 1101

1011
+1101
111
10000

Substraction of Two Binary Numbers.

(i) 101101 - 100111

```
  101101
- 100111
--------
  000110
--------
```

Multiplication of Two Binary Numbers :

```
   1011
  x 101
-------
   1011
  0000
 1011
-------
 110111
-------
```

Complements of numbers:

Complements are used in digital computer to simplify the subtraction operation and for logical manipulation simplifying operations leads to similer, less expensive circuits to implement the operations There are two types of complements for each base-r system the radix complement system and diminished radix complement. The first is refered to as the r's complement and the second as the (r-1) Complement when the value of the base r is substituted in the name, the two types are refered to as the 2's complement and 1's complement for binary numbers and the 10's complement & 9's complement for decimal numbers

Diminished Radix Complement:-

Given a number N in base r having n digits, the (r-1)'s complement of N. for decimal numbers

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

r = 10 and r-1 = 9 so the 9's complement of N is $(10^n - 1) - N$ & 10^n represents a number that consists of a single 1 followed by n 0's. $10^n - 1$ is a number represented by n 9's.

eg $(2345)_{10}$

$n = 4$

$r = 10$

$(10^n - 1) - N$

$(10^4 - 1) - 2345$

$(10000 - 1) - 2345$

$9999 - 2345 = 7654)$

The 9's complement of 2345 is = 7654

For binary numbers $r = 2$ and $r - 1 = 1$ so that the 1's complement of N is $(2^n - 1) - N$. 2^n is represented by a binary number that consists of a 1 followed by n 0's.

$2^n - 1$ is a binary number represented by n 1's.

eg: $(1011)_2$

$r = 2$

$n = 4$

$(2^n - 1) - N$

$2^4 - 1 - N$

$2^4 = (10000)_2 = 10000$

$- 1$

$1111)$

```
  1111
- 1011
------
  0100
```

The 1's complement of 1011 is - 0100

Radix complement :-

The r's complement of an n' digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the (r-1)'s complement we note that the r's complement is obtained by adding 1 to the (r-1)'s complement, since $r^n - N = [(r^n - 1) - N] + 1$.

The 10's complement of decimal 2389

10's complement $= [(10^4 - 1) - N] + 1$

$= [(10000 - 1] - N] + 1$

$= [9999 - 2389] + 1$

$= [7610] + 1$

$= [7611]$

Work! $(012398)_{10}$

$(246700)_{10}$

Similarly the 2's complement can be formed by leaving all least significant 0's and first 1 unchanged and replacing 1's with 0's and 0's with 1's in the other higher significant digits

$(1101100)_2$

2's complement $= [(r^n - 1) - N] + 1$

$= [(2^7 - 1) - N] + 1$

$= [(10000000 - 1) - N] + 1$

$= [1111111 - 1101100] + 1$

$= [0010011 + 1]$

$= [0010100]$

→ 2's complement of 1101100 is 0010100

→ $(0110111)_2$ find 2's complement

Subtraction with Complements:

The subtraction of two n-digit unsigned numbers M-N in base r can be done

1. Add the minuend M to the r's complement of the subtrahend N - Mathamatically $M + (r^n - N)$

$$M - N + r^n$$

2. If $M \geq N$ the sum will be produced and end carry r^n, which can be discarded what is left is the result M-N

3. If $M < N$ the sum does not produce an end carry and is equal to $r^n - (N-M)$ which is the r's complement of (N-M). To obtain the answer in fimalier form take the r's complement of the sum and place a negative sign in front.

Using 10's complement, substract 72532 - 3250

minuend - 72532

10's comp subtrahend - 03250

10's complement of 3250 = 03250

$= [(10^5 - 1) - N] + 1$

$= [[100000 - 1] - N] + 1$

$= [99999 - 03250] + 1$

$= [96749] + 1$

10's compl'n = 96750

M = 72532

= 96750

sum = 169282

Discarded Carry 10^5 = 100000

Answer = 69282

M - N = 69282

~~Note~~:- M has 5 digits N has four digits. Both numbers must have the same no of digits, so we write N as 03250.

= 69282 - 2962

Using 10's Complement subtract 3250 - 72532

M = 03250

10's complement of N =

$[(10^5 - 1) - N] + 1$

$[(100000 - 1) - 72532] + 1$

$[99999 - 72532] + 1$

$[27467] + 1$

10's comp − 27468

M = 03250

10's comp N = + 27468

Sum = 30718

−(10's complement of 30718)

~~30718~~ $[(100000 - 1) - N] + 1$

$[99999 - 30718]$

69281 + 1 = 69282

= −69282

There is no end carry. Therefore, the answer is

-(10's complement of 30718) = -69282 Ans

3250 < 72532 The result is negative.

using 2's complement find x-y & y-x by

using the Given No's - x = 1010100 y = 1000011

(a) x-y

x = 1010100

2's comp y = 0111101
+ 1
1001000 1

Discard Carry 2 10000000

x-y = 0010001

$((2^7-1)-N)+1$

$[(10000000-1)-N)+1$

$[1111111-1000011]+1$

0111100+1

2's comp = 0111101

(b) y-x

y = 1000011

2's comp x = 0101100
+
1101111

There is no end carry

The answer is y-x =

-(2's complement of

1101111) =

-0010001

2's comp of x =

$[(2^7-1)-N]+1$

$[(10000000-1]-N)+1$

$[1111111-1010100]+1$

0101011+1

2's comp = 0101100

Signed Binary numbers:

Positive integers (including zero) can be represented as unsigned numbers. A collection of positive and negative integers called as signed numbers.

eg: -2, -1, 0, 1, 2, 3

Computer must represent everything with binary digits. It is customary to represent the sign with a bit placed in the left most position of the number. The convention is to make a sign bit 0 for positive and 1 for negative. both signed and unsigned binary numbers consist of a string of bits when represented in computer

eg: The string of bits 01001 can be considered as 9 (or) +9 because the leftmost bit is '0'. The string of bits 11001 consider as signed number (-9) because the left most bit is 1. when arithamatic operations are implemented in a computer it is more convenient to use a different system. refered to as the signed complement system, for representing -ve numbers. In this system a -ve number is indicated by its complement.

Signed Binary Numbers

Decimal	signed 2's Comp	Signed 1's Comp	Signed magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	—	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	—	—

Mathamatical numbers are generally made up of a sign and a value (magnitude) in which the sign indicates whether the number is positive (+) or negative (-) with the value indicates the size of the number for example 23, +156 or -274 prasenting numberes is this fasion called "sign-magnitude"

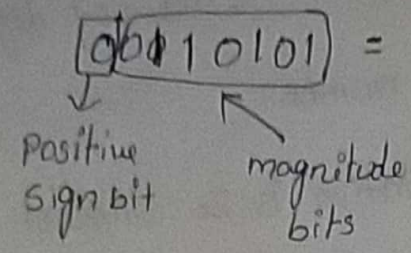
representation since the left most digit can be used to indicate the sign and the remaining digits the magnitude (or) value of the number.

Sign magnitude notation is the simplest and one of the most common methods of representing positive & negative numbers either side of zero. Thus the negative numbers are obtained simply by changing the sign of the corresponding positive number as each positive & unsigned number will have a signed opposite, for example, +2 and -2, +10 and -10 etc.

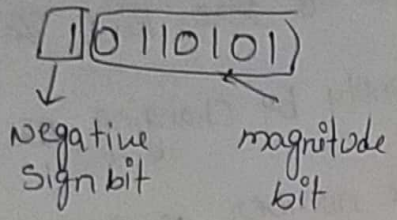
For signed binary numbers the most significant bit (MSB) is used as the sign bit. If the sign bit is '0', this means the number is positive in value. If the sign bit is 1, then the number is negative in value. The remaining bits in the number are used to represent the magnitude of the binary number in the usual unsigned binary number format way.

The sign-and-magnitude notation stores positive and -ve values by dividing the 'n' total bits into two parts: 1 bit for sign and n-1 bits for the value which is a pure binary number.

eg Positive signed binary number



Negative signed binary numbers



Arithamatic Addition:

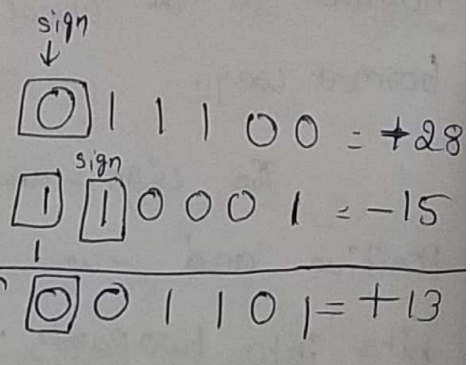
The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. If the signs are same we add the two magnitudes and give the sum the common sign. If the signs are different, we substract the smaller magnitude from the larger and give the difference the size of the larger magnitude.

Eg: Add $(28)_{10}$ and $(-15)_{10}$

$(28)_{10} = 011100$

$+15 = 01111$

$-15 = 10001$



eg:

Add $(-28)_{10}$ and $(-15)_{10}$

$+28 = 011100$ ← $+15 = 01111$

2's comp'of(28) $-28 = 100100$ $\quad -15 = 10001$

$11\ 00100 = -28$

$11\ 10001 = \pm 15$

2's complement ignore carry ①

$$\begin{array}{r} 1 \\ \hline 1\ 0\ 1\ 0\ 1\ 0\ 1 \quad -43 \end{array}$$

eg: Add $(28)_{10}$ and $(15)_{10}$

$(28)_{10} = 011100$

$(15)_{10} = 01111$

$00\ 11100 = +28$

$000\ 1111 = +13$

$$\begin{array}{r} 1\ 1 \\ \hline 0\ 1\ 0\ 1\ 0\ 1\ 1 \quad +43 \end{array}$$

sign bit

2 | 43
2 | 21 - 1
2 | 10 - 1
2 | 5 - 0
2 | 2 - 1
1 - 0

101011
010100
1
010101

eg: Add $(-28)_{10}$ and $(15)_{10}$

$(-28)_{10} = 100100$

$+15 = 001111$

$-13 \quad 110011$

2 | 15
2 | 7 - 1
2 | 3 - 1
1 - 1

2 | 13
2 | 6 - 1
2 | 3 - 0
1 - 1

01101

Arthamatic substraction

Substraction of two signed binary nombers when negative numbers are in 2's Complement form is simple and can be stated

Take the 2's complement of the subtrahend and add it to the minuend. A Carry out of the sign-b position is discarded.

This procedure is adopted because a substract operation can be changed to an addition operation if the sign of the substrahend is changed as is demonstrated by the following relationship

$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

But changing a positive nomber to a -ve nomber is easily done by taking the 2's Complement of the positive nomber. The reverse is also true because the complement of the negative number in complement form produces the equivalent positive nomber

It is worth nothing that Binary number in the signed complement system are added and substracted by the same basic addition and substraction rules as unsigned numbers. Therefore Computers need only one common hardware circuit to handle both type of arthamatic

Binary Codes:

In the coding, when numbers letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code. The digital data is represented, stored and transmitted as group of binary bits. The group is also called as binary code. The binary code represented by the number as well as alphanumaric letters.

Advantages of Binary Code:-

→ Binary Codes are suitable for the computer application

→ Binary Codes are suitable for the digital communication

→ Binary codes make the analysis and design of digital circuits if we use the binary codes.

→ Since only 0 & 1 are being used, implementation becomes easy.

Classification of binary codes:

→ weighted codes

→ Non-weighted codes

→ Binary coded decimal Codes

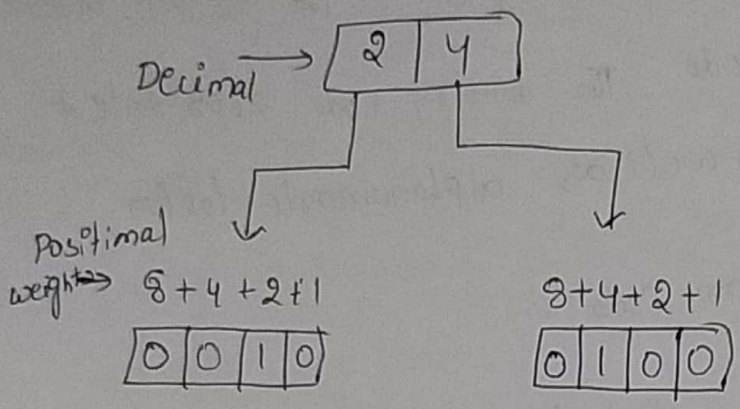
→ Alphanumaric Codes

→ Error Detecting codes

→ error correcting codes.

Weighted Codes:

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight several systems of the codes are used to express the Decimal digits 0 Through 9. In these codes each decimal digit is represented by a group of four bits



Non-weighted Codes:-

In this type of binary codes, the positional weights are not assigned. The example for non-weighted codes are Excess-3 code and Gray code.

Excess-3 code:-

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

The excess-3 codes are obtained as

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

Decimal	BCD 8421	Excess-3 BCD + 0011
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Gray Code: Gray code is also called reflected binary code and developed by frank Gray. It is also called as unweighted code, minimum error code, unit distance code in an ordering of the binary numarical system such that two successive values differ in only one bit.

(XOR) operation

Decimal	BCD	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

Binary coded Decimal (BCD) code :-

In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digit with a binary code. In the BCD with four bits we can represent sixteen numbers (0000 to 1111)

In this decimal digits 0 to 9 are represented by their natural binary equivalent using 4-bits

eg:- $(8 \quad 3)_{10}$

$(1000 \quad 0011)_{BCD}$

BCD addition

Consider the addition of two decimal digits in BCD, together with a possible carry from a previous less significant pair of digits. Since each digit does not exceed 9 the sum cannot be greater than 9+9+1 = 19 with the 1 being a previous carry. Suppose we add the BCD digits as if they were binary numbers. Then the binary sum will produce a result in the range from 0 to 19. In binary, this range will be from 0000 to 1001.

The two BCD digits are added as if they were two binary numbers. If the binary sum is greater than (or) equal to 1010 we add 0110 to obtain the correct BCD sum and carry.

```
  4      0100
+ 5      0101
-----    ------
  9      1001
~~1001~~

  4      0100
+ 8      1000
-----    ------
 12      1100
         0110 +6
      --------
     0001 0010
        1    2
```

8 + 9 = 17

```
      1000  8
      1001  9
    ---------
     10001  17
      0110 +6
    ---------
  0001 0111
```

The addition of two n-digit unsigned BCD numbers follows the same format

184 + 576

184	0001	1000	0100
576	0101	0111	0110
760	0100	1111	1010
	+1	0110	0110
	0111	0110	0000
	7	6	0

Alphanumaric codes:

A Binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters numbers from 0 to 9, punctuation marks and other symbols.

The alphanumaric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represented other characters such as symbol & various instructions necessary for conveying information.

An alphanumaric code should at least represent 10, and 26 letters of alphabet i.e total 36 items.

→ These alphanumaric codes are very commonly used for the data representation.

→ American standard code for information Interchange (ASCII)

→ Extended Binary Coded Decimal InterChange Code (EBCDIC)

→ Five Bit Baudot Code

ASCII code is a 7 bit code whereas EBCDIC is an 8-bit code ASCII code more commonly used worldwide while EBCDIC is used Primarily in large IBM computers.

32 - space
33 - !
34 - "
35 - #
36 - $
37 - %
38 - &
39 - '
40 - (
41 -)
42 - *
43 - +
44 - ,
45 - -
46 - .
47 - /
48-57 - (0-9)
58 - :
59 - ;
60 - <
61 - =
62 - >
63 - ?
64 - @
65-90 - (A-Z)
91 - [
92 - \
93 -]
94 - ^
95 - _
96 - `
97-122 - (a-z)
123 - {
124 - |
125 - }
126 - ~

8 bit

128 64 32 16 8 4 2 1 | 128 64 32 16 8 4 2 1 | 128 64 32 16 8 4 2 1

1 0 0 1 0 0 1 1 | 0 0 0 0 0 1 1 0 | 0 1 0 0 1 1 0 1

73 | 65 | 77

I | A | M

Error: Error is a condition when the output information does not match with the input information. During transmission digital signals suffer from noise that can introduce errors in the binary bits travelling from one system to other. That means a '0' bit may change to 1 (or) a 1 bit may change to '0'.

Error Detecting Codes:

Whenever a message is transmitted it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occured during transmission of the message. a simple example for error detecting code is parity attack.

Error Correcting Codes :

Along with error-detecting code, we can also pass some data to figure out the original message from the corrupt message that we received. This type of code is called an error Correcting Code. Error correcting codes also deploy the same strategy as error detecting codes but additionally, such codes also detect the exact location of the corrupt bit

In error correcting codes, Parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location once the corrupt bit is located, its value is reverted (from 0 to 1, 1 to 0) to get the original message.

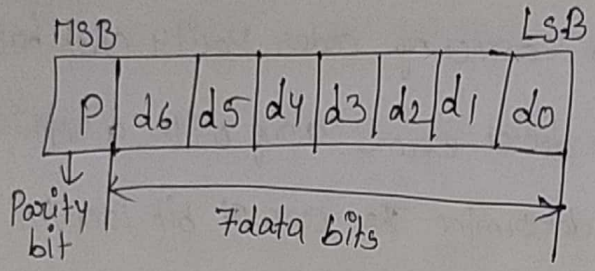
To Detect and Correct the errors, additional bits are added to the data bits at the time of transmission.

→ The additional bits are called Parity bits. They allow detection or correction of the errors.

→ The data bits along with the parity bits form a codeword

Parity Checking (or) Error Detection:

It is a simplest technique for detecting and correction errors. The MSB of an 8-bits word is used as the Parity bit and the remaining 7 bits are used as data bits. The parity of 8 bits transmitted word can be either even parity or odd parity.



Even Parity: Even parity means the number of 1's in the given word including the parity bit should be even (2, 4, 6...)

Odd Parity: Odd parity means the number of 1's in the given word including the parity bit should be odd (1, 3, 5...)

Use of Parity bit: The parity bit can be set to 0 and 1 depending on the type of the parity required

→ For even Parity, This bit is set to 1 or 0 such that the no of '1 bits in the entire word is even.

→ For odd Parity, This bit is set to 1 or 0 such that the no of "1 bits" in the entire word is odd.

Parity Checking at the receiver can detect the presence of an error if the Parity of the receiver signal is different from the expected Parity. That means, if it is known that the Parity of the transmitted signal is always going to be "even" and if the received signal has an odd Parity then the receiver can conclude that the received signal is not Correct. If an error is detected, then the receiver will ignore the received byte and request for re transmission of the same byte to the transmitter.

Binary storage and Registers:

A Binary information in a digital computer must have a Physical existance in some medium for storing individual Bits. A binary Cell is a device that Possesses two stable states and is Capable of storing one bit (0 or 1) of information. The input to the Cell receives excitation signals that set it to one of the two states. The output of the Cell is a Physical quantity that distinguishes between the two states. The information stored in a Cell is 1 when the Cell is in one stable state and 0 when the Cell is in the other stable state.

Registers:

A Register is a group of binary Cells. A Register with n Cells can store any discrete quantity of information that contains n bits.

$$n \text{ Cells} \rightarrow 2^n \text{ Possible states}$$

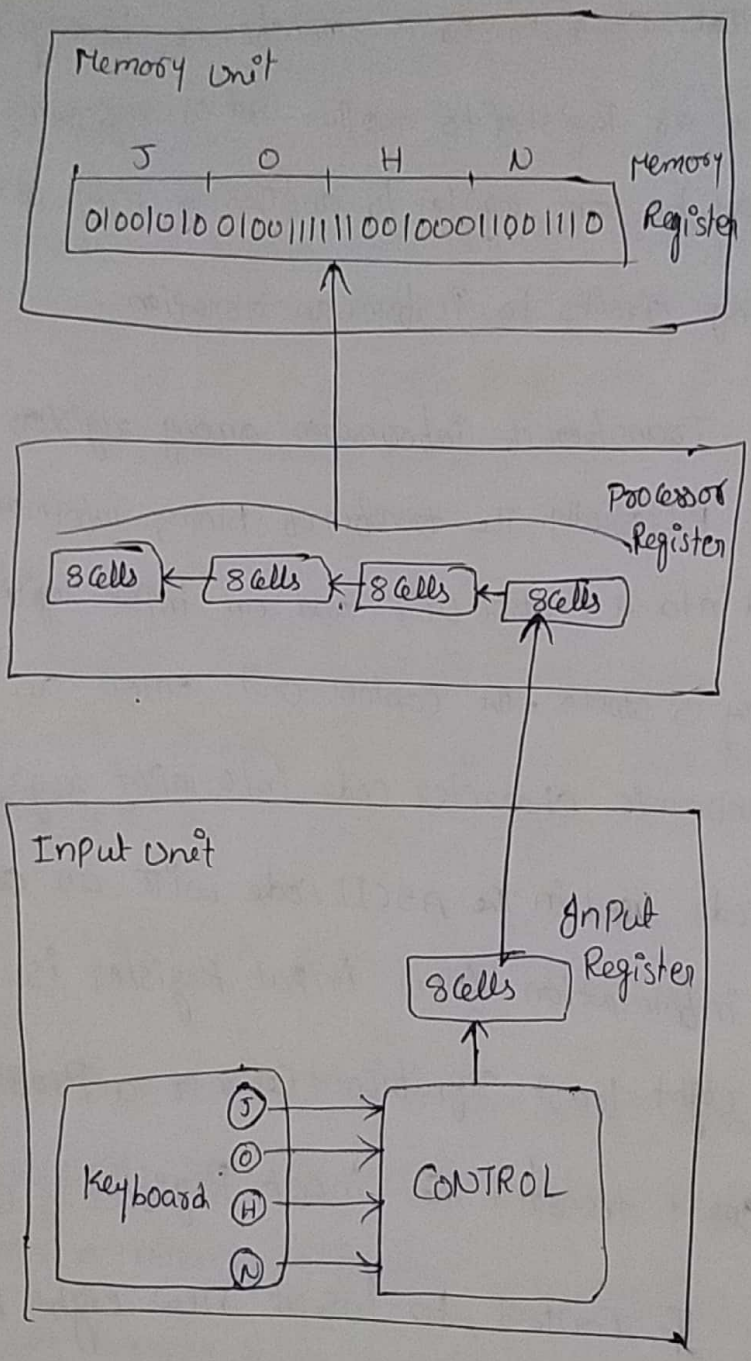
A register with 16 Cells can be in one of 2^{16} Possible states. If one assumes that the content of the register represents a binary integer then the register can store any binary number from 0 to $2^{16}-1$.

Register Transfer

A digital system is characterized by its registers and the components that perform data processing. In digital systems, a register transfer operation is a basic operation that consists of a transfer of binary information from one set of Register to another set of registers. The transfer may be one register to another or may pass through data-processing circuits to perform an operation.

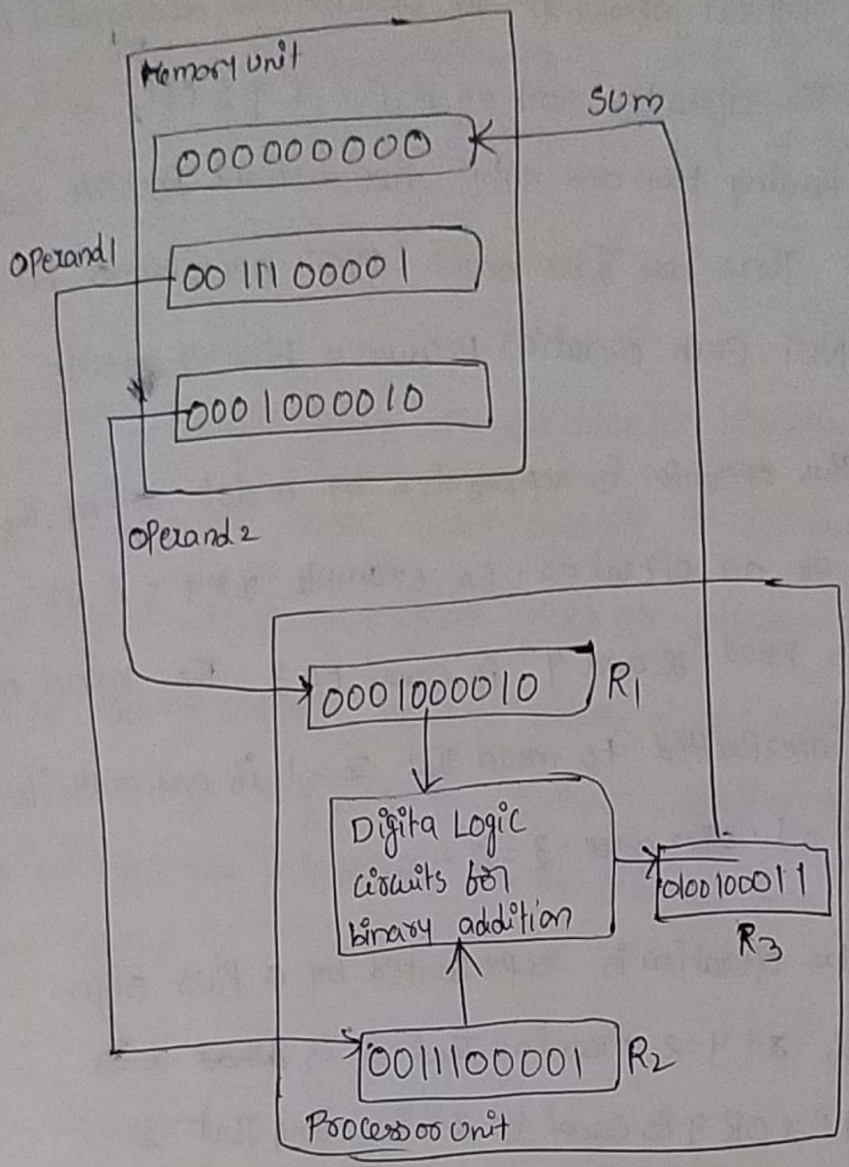
The Transfer of information among registers and demonstrate pictorially the transfer of binary information from a key board into a control unit and an input register. Each time a key is stroke, the control unit enters an equivalent 8 bit alphanumaric character code into input register. That the code used in the ASCII code with an odd parity bit. The information from input Register is transfered into the eight least significant cells of a Processor Register. After every transfer the input Register is cleared to enable the control to insert new eight bit code when key board stroke again. Each eight bit character transferred to the Process Register is preceded by a shift of the previous character to the next eight cells on its left. when a transfer of four characters

is completed, the processor register is full, and its content are transferred into a memory Register. The content stored in the memory register came from the transfer of the characters JOHN after the four appropriate keys were struck



Transfer of information among Registers

The device most commonly used for holding data is a register.



Binary Information Processing.

The process of adding 10-digit binary number. The memory unit, which normally consist of millions of Registers is shown with only of Three Registers. The part of Processor Unit shows Consist of Three Registers R_1, R_2 & R_3

Binary Logic

Binary logic consists of binary variables and a set of logical operation. The variables are designated by letters of the alphabet, such as A, B, C, x, y, z etc. with each variable having two and only two distinct possible values; 1 and 0. There are three basic logical operations: AND, OR and NOT. Each operation produces a binary result.

AND: This operation is represented by a dot or by the absence of an operator. For example $x * y = z$ (or) $xy = z$ is read "x and y" is equal to z. The logical operation AND is interpreted to mean that $z = 1$ if and only if $x = 1$ & $y = 1$. Otherwise $z = 0$.

OR: This operation is represented by a plus sign. for example $x + y = z$ meaning that z is ~~what~~ ~~x is~~ ~~not~~ read "x OR y is equal to z" meaning that $z = 1$ if $x = 1$ ~~or~~ $y = 1$ or if both $x = 1$ and $y = 1$ if $x = 0$ and $y = 0$ Then $z = 0$

NOT: This operation represented by a prime $x' = z$ (or) $\bar{x} = z$ is read "not x is equal to z". if $x = 1$ Then $z = 0$ (or) $x = 0$ Then $z = 1$ The not operation also refered as complement operation

AND

x	y	z = x.y
0	0	0
0	1	0
1	0	0
1	1	1

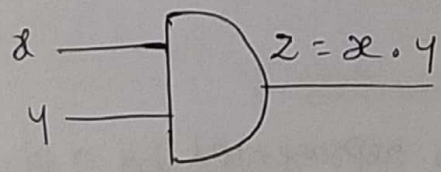
OR

x	y	z = x+y
0	0	0
0	1	1
1	0	1
1	1	1

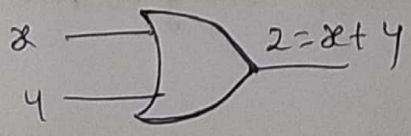
NOT

x	z = x'
0	1
1	0

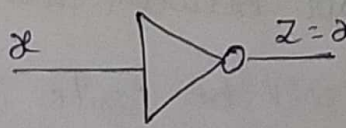
Logic gates: Logic gates are electronic circuits that operate on one (or) more input signals to produce an output signal. Electronic signals such as voltages or current exist as analog signals having values over a given continuous range say 0 to 3V. but in digital systems these voltages are interpreted to be either of two recognizable values 0 (or) 1.



(i) AND Gate



(ii) OR Gate



(iii) NOT Gate

Boolean algebra as logical gates:

Boolean Algebra is used to analyze and simplify the digital circuits. It uses only the binary numbers i.e 0 and 1. It is also called as Binary Algebra or logical Algebra Boolean Algebra was invented by George Boole in 1854

Rules in Boolean Algebra:

→ Variables used can have only two values. Binary 1 for high and Binary 0 for Low

→ Complement of a variable is represented by an overbar (¯). Thus Complement of variable B is represented as $\bar{B}$. Thus if B=0 Then $\bar{B}=1$ and B=1 Then $\bar{B}=0$

→ ORing of the variable is represented by a plus (+) sign between them. for example ORing of A,B,C is represented as A+B+C

→ Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometimes it will be write in ABC form.

Boolean Laws: There are six types of Boolean Laws.

Commutative Law: Any binary operation which satisfies the following expression is refered to as commutative operation.

(i) $A \cdot B = B \cdot A$ (ii) $A + B = B + A$ for all $A, B \in S$

Commutive Law states that changing the sequence of the variable does not have any effect on the output of a logic circuit.

Associative Law: This law states that the order in which the logic operations are performed is irrelavent as their effect is the same.

(i) $(A \cdot B) \cdot C = A \cdot (B \cdot C)$ (ii) $(A + B) + C = A + (B + C)$ for All $A, B, C \in S$

Distributive Law: Distributive Law states the following condition.

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

Inversion Law: This Law use the NOT operation. The inversion law states that double inversion of a variable result in the original variable itself.

$$\bar{\bar{A}} = A.$$

Closure: $N = \{1, 2, 3, 4 \ldots\}$ for any $a, b \in N$ we obtain a unique $c \in N$ by the operation $a+b=c$

The set of natural number is not closed with respect to the binary operator $-$ by the rules of arithmatic substraction, because $2-3=-1$ and $2, 3 \in N$ but $(-1) \notin N$

Identity element: A set S is said to have an identity element with respect to a binary operation

$$x \cdot \blacklozenge = \blacklozenge \cdot x = x$$

$$x + 0 = 0 + x = x$$

The field of real numbers is the basis for arithmatic and ordinary algebra. The operators and postulates have the following meanings.

→ The binary operator + defines addition
→ The additive identity is 0
→ The additive inverse define substraction
→ The binary operator $\cdot$ defines multiplication.
→ The multiplicative identity is 1

for $a \neq 0$, the multiplicative inverse of $a = 1/a$ defines divison (i.e $a \cdot 1/a = 1$).

The only distributive Law applicable is that of $\cdot$ over +

$$a \cdot (b+c) = (a \cdot b) + (a \cdot c)$$

Basic Theorems AND properties of Boolean Algebra:-

In the Boolean algebra we listed six Theorems and four of its Postulates. The notation is simplified by removing (omitting) the binary operators whenever doing so does not lead to confusion. The Theorems and Postulates listed are the most basic relationships in Boolean algebra. The Theorems like the Postulates are listed in pairs: each relation is the dual of the one pair with it. The Postulates are basic axioms of the algebric structure and need no proof. The Theorems must proven from the Postulates. Proofs of the Theorems with one variable are presented next

Postulates and Theorems of Boolean Algebra:-

Postulate 2	(a) $x+0=x$	(b) $x\cdot 1=x$
Postulate 5	(a) $x+x'=1$	(b) $x\cdot x'=0$
Theorem 1	(a) $x+x=x$	(b) $x\cdot x=x$
Theorem 2	(a) $x+1=x$	(b) $x\cdot 0=0$
Theorem 3 involuation	$(x')'=x$	
Postulate 3, Commutative	(a) $x+y=y+x$	(b) $xy=yx$
Theorem 4, associative	(a) $x+(y+z)=(x+y)+z$	(b) $x(yz)=(xy)z$
Postulate 4, distributive	(a) $x(y+z)=xy+xz$	(b) $x+yz=(x+y)(x+z)$
Theorem 5, Demorgan	(a) $(x+y)'=x'y'$	(b) $(xy)'=x'+y'$
Theorem 6, absorption	(a) $x+xy=x$	(b) $x(x+y)=x$

Theorem 1(a): $x + x = x$

Statement	Justification
$x + x = (x + x) \cdot 1$	Postulate 2(b)
$= (x + x)(x + x')$	" 5(a)
$= x + xx'$	" 4(b)
$= x + 0$	" 5(b)
$= x$	

Theorem 1(b): $x \cdot x = x$

Statement	Justification
$x \cdot x = xx + 0$	Postulate 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	

Note that Theorem 1(b) is the dual of Theorem 1(a) and that each step of the proof in part (b) is the dual of its counterpart in part (a). Any dual theorem can be similarly derived from the proof of its corresponding theorem.

Theorem 2(a): $x + 1 = 1$

Statement	Justification
$x+1 = 1 \cdot (x+1)$	Postulate 2(b)
$= (x+x') \cdot (x+1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$x+1 = 1$	5(a)

A=0 or 1=1
A=1 1+1=0

Theorem 2(b): $x \cdot 0 = 0$ by duality

x=0 = 0.0 = 0
x=1 = 1.0 = 0

Theorem 3: $(x')' = x$ from Postulate 5, we have $x + x' = 1$ and $x \cdot x' = 0$ which together defines the complement of x, the complement of x' is x and is also $(x')'$. There fore the complement is unique we have $(x')' = x$

The Theorems involving two or three variables may be proovem algebraically from the Postulate and the Theorems that have already been proovem.

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

Theorem 6(a): $x + xy = x$

Statement	Justification
$x+xy = x \cdot 1 + xy$	Postulate 2(b)
$= x(1+y)$	4(a)
$= x(y+1)$	3(a)
$= x \cdot 1$	Theorem 2(a)
$= x$	2(b)

Theorem 6(b): $x(x+y) = x$ by duality

$x(x+y) = (x+0)(x+y) = x(1+y) = x(1)$

The theorems of Boolean algeboa can be proven by means of touth tables. In touth tables both sides of the relation are checked to see whether they yield identical results for all possible combinations of the variable involved. The following touth tables verifies the The first absorption Theorem:

x	y	xy	$x+xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

The algeboaic Proofs of the associative law and DeMorgan's Theorem are long and will not be shown here Their validity is easily shown with touth tables.

For the first DeMorgan's Theorem $(x+y)' = x'y'$ is as follows

x	y	$x+y$	$(x+y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x'	y'	$x'y'$
1	1	1
1	0	0
0	1	0
0	0	0

Second Demorgan Theorem : $(xy)' = x' + y'$

x	y	xy	$(xy)'$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

x'	y'	$x' + y'$
1	1	1
1	0	1
0	1	1
0	0	0

Duality : one Part may be obtained from the other if the binary operators and the identity elements are interchanged. This important property of boolean Algebra is called the duality principle and states that every algebric expression deducible from the postulate of Boolean algebra remain valid if the operators and identity elements are interchange

AND → OR
OR → AND
1 → 0
0 → 1.

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

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

using the basic Theorems and postulates of Boolean algebra, simplify the following Boolean expression

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

Boolean functions

Boolean Algebra is an algebra that deals with binary variables and logical operations. A Boolean function described by an algebraic expression consists of binary variable the constants 0 and 1, and the logical operation symbols. For a given value of a binary variable the function can be equal to either 1 or 0.

Consider the Boolean function,

$$F_1 = x + y'z$$

The Function F_1 is equal to 1 if x is equal to 1 or if both y' and z are equal to 1 otherwise $F_1 = 0$.

A Boolean function can be represented in tove table. The number of Rows in the tove table is 2^n where n is the no. of variables in the function.

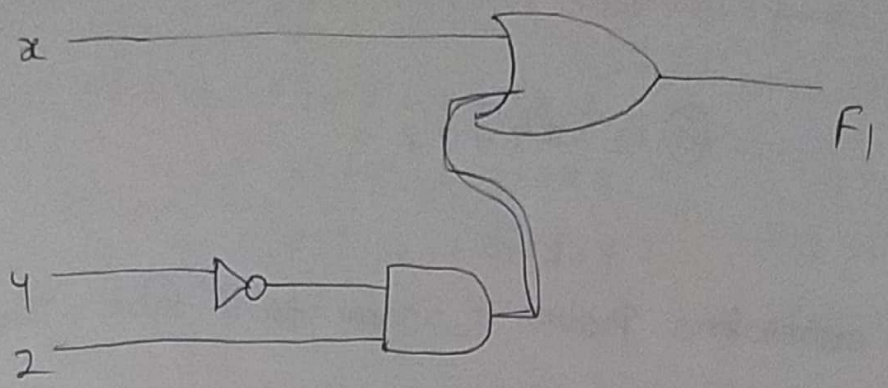
F_1 contains either 0 (or) 1 for each of these combinations. The table shows that the function is equal to 1 when $x = 1$ (or) when $yz = 01$, and otherwise equal to '0'.

A Boolean function can be transfered from an algebraic expression into a circuit diagram composed of logic gates connected in a particular structure.

$F_2 = x'y'z + x'yz + xy'$

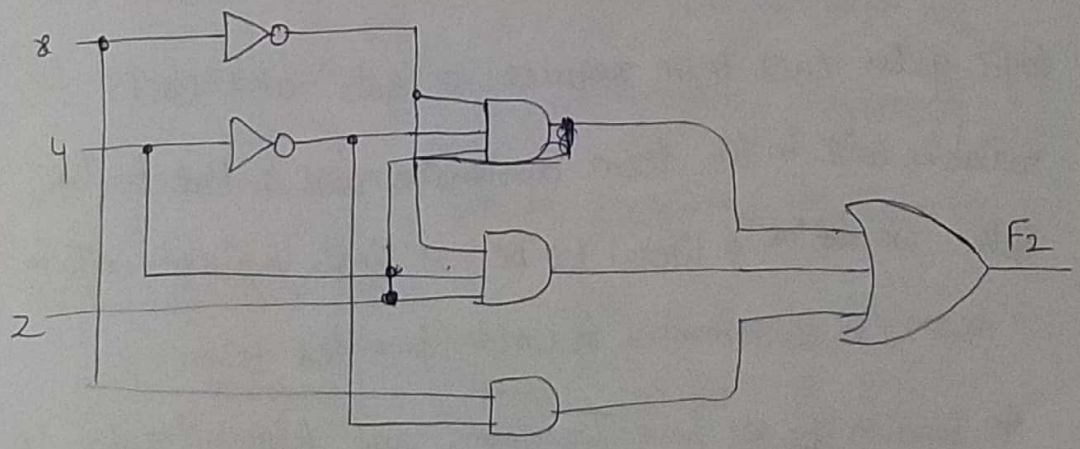
$= x'z(y' + y) + xy'$

$= x'z + xy'$

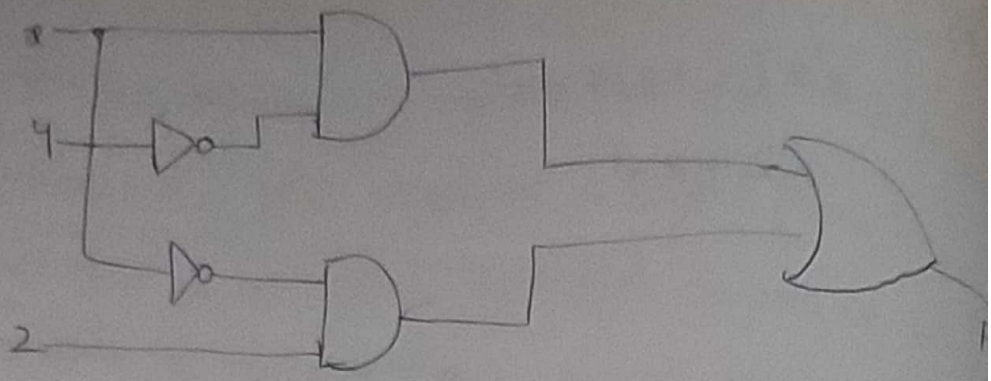


$F_1 = x + y'z$

$F_2 = x'y'z + x'yz + xy'$



(a) $F_2 = x'y'z + x'y'z + xy'$



(b) $F_2 = xy' + x'z$

both expressions produce the same truth table, they are equivalent therefore the two circuits have the same output for all possible binary combinations of inputs of the three variables.

Algebric Manipulation:

When a Boolean expression is implemented with logic gates, each term require a gate and each variable with in the term designates and input to the gate. we define a literal to be a single variable with in a term, in complemented or uncomplemented form.

In function F_2 we have three terms and ~~eight~~ literals. in manipulated function F_2 we have two terms and four literals.

Simplify the following Boolean functions to a minimum no of literals.

1. $x(x'+y) = xx'+xy = 0+xy = xy$

2. $x+x'y = (x+x')(x+y) = 1(x+y) = x+y$

3. $(x+y)(x+y') = x+xy+xy'+yy' = x(1+y+y') = x$

4. $xy+x'z+yz = xy+x'z+yz(x+x')$

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

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

$= xy + x'z$

Complement of a function:-

The Complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F. The complement of a function may be derived algebraically through Demorgan's Theorems

Demorgans Theorems can be extented to three or more variables.

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

$= A'x'$ by Theorem 5(a)(Demorgan)

$= A'(B+C)'$

$= A'(B'C')$ by 5(a) (Demorgan)

$= A'B'C'$

→ Find the complement of the function $F_1 = x'yz' + x'$

$F_2 = x(y'z' + yz)$ By applying Demorgans Theorem

CANONICAL AND STANDARD FORMS:

Minterms and Maxterms:

A binary variable may appear either in its normal form (x) or in its complement form x'

Now consider two binary variables x and y combined with an AND operation. since each variable may appear in either form there are four possible combinations.

$x'y'$, $x'y$, xy', xy. Each of these four AND terms is called a minterm (or) a standard Product.

n variables can be combined to form 2^n minterms

~~Three minterms~~ The binary numbers from 0 to $2^n - 1$ are listed under the n variables. Each minterm is obtained from an AND Term of the n variables

minterm is shown as m_j where the subscript j denotes the decimal equivalent of the binary number of the minterm designated.

n variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations called maxterms (or) standard sums

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.

for example ~~$f_1 = x'y'z + xy'z + xyz$~~

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	$m_0 = 1$	$x+y+z$	M_0
0	0	1	$x'y'z$	$m_1 =$	$x+y+z'$	M_1
0	1	0	$x'yz'$	m_2	$x+y'+z$	M_2
0	1	1	$x'yz$	m_3	$x+y'+z'$	M_3
1	0	0	$xy'z'$	m_4	$x'+y+z$	M_4
1	0	1	$xy'z$	m_5	$x'+y+z'$	M_5
1	1	0	xyz'	m_6	$x'+y'+z$	M_6
1	1	1	xyz	m_7	$x'+y'+z'$	M_7

If the binary variable is '0' then it is represented as complement of variable in minterm as the variable itself in MAXTerm. Similarly, if the binary variable is '1' then it is represented as complement of variable in MAXTerm and as the variable itself in minterm

We can easily notice that the min terms and max terms from the table. They are complement of each other. If there are n Boolean variables then there will be 2^n minterms and 2^n MAX terms

eg:- $f_1 = x'y'z + xy'z + xyz$

$m_1 + m_4 + m_7$

$f_2 = x'yz + xy'z + xyz' + xyz$

$m_3 + m_5 + m_6 + m_7$

X	Y	Z		f_1	f_2
0	0	0	m_0	0	0
0	0	1	m_1	1	0
0	1	0	m_2	0	0
0	1	1	m_3	0	1
1	0	0	m_4	1	0
1	0	1	m_5	0	1
1	1	0	m_6	0	1
1	1	1	m_7	1	1

We can express each output variable in following two ways

→ Canonical SOP form
→ Canonical POS form

Canonical SOP form: Canonical SOP form means Canonical sum of products form. In this form, each product term contains all literals. So these product terms are nothing but the minterms. Canonical SOP form also called as sum of minterms form.

First identify the minterm for which the output variable is one and then do the logical OR of those minterms in order to get the boolean expression corresponding to that output variable. This Boolean function will be in the form of sum of minterms.

Each product term contains all the variables of the function is called standard SOP

Ex: $F(A,B,C) = A'BC + ABC'$

Eg: Express the Boolean function $F = A + B'C$ as a sum of minterms. The function has three variables

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

$B'C(A+A')$

$AB'C + A'B'C$

Combine all terms

$F = A + B'C$

$= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$

But $AB'C$ appears twice and according to theorem 1 ($x+x=x$) it is possible to remove one of those occurances

$F = A'B'C + AB'C' + AB'C + ABC' + ABC$

$m_1 + m_4 + m_5 + m_6 + m_7$

eg: $AB+BC+AC'$ (SOP)

Canonical POS form

Canonical POS form means Canonical Product of Sums form. In this form each sum term contains all literals. So these sum terms are nothing but the Max terms. Hence Canonical POS form is also called as Product of MAX Terms

First identify the MAX Terms for which the output variable is zero and then do the logical AND of those MAX Terms in order to get the Boolean expression corresponding to the output variable thus boolean function will be in the form of Product of MAX Terms

eg. Convert the following Boolean function into standard SOP form

~~$F = P'q'r + Pq'r + Pqr' + Pqr$~~

~~The given boolean function is in Canonical SOP form~~

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

$F = (A+B+CC')(A+BB'+C)(AA'+B+C) \quad x x' = 0$

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

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

→ Express the Boolean function $F = xy + x'z$ as a product of maxterms.

$F = xy + x'z \quad (\because x \cdot y = x)$

$= (xy + x')(xy + z) \quad [\because x' = x \;\; z = z)$

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

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

The function has three variable x, y, z Convert that expression into standard form

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

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

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

$F_1 = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$

$= M_0 M_2 M_4 M_5$

$= \Pi(0, 2, 4, 5)$

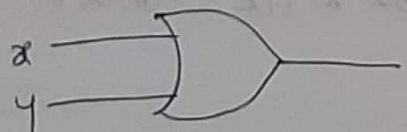
Digital Logic Gates :

Boolean functions are expressed in terms of AND, OR and NOT operations, it is easier to implement a Boolean function with these type of gates, still, the possibility of constructing gates for the other logic operations is of practical interest.

AND gate :-

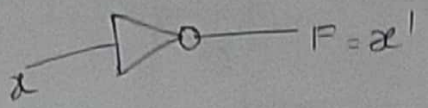
x	y	F=(x.y)
0	0	0
0	1	0
1	0	0
1	1	1

OR gate :-



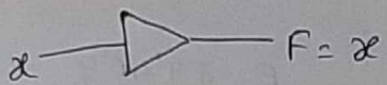
x	y	F=x+y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate :-



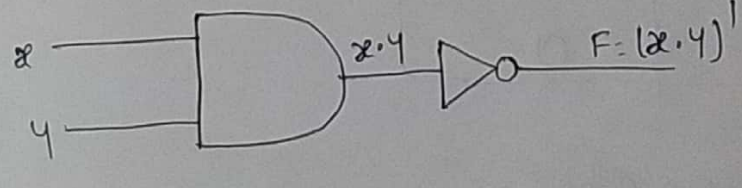
x	F = x'
0	1
1	0

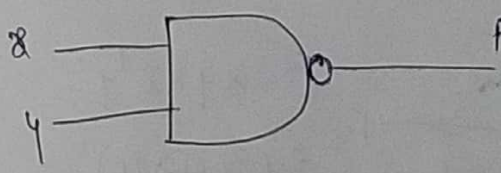
Buffer Gate :-



x	F = x
0	0
1	1

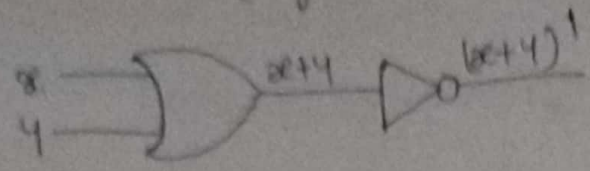
NAND Gate :- (Negated AND)

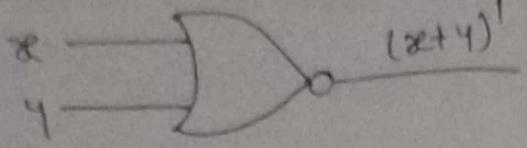




x	y	(x.y)'
0	0	1
0	1	1
1	0	1
1	1	0

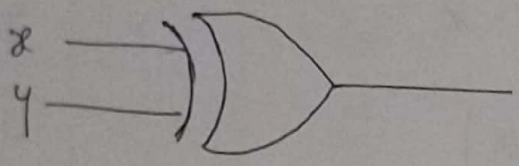
NOR Gate: (Negated OR)





x	y	(x+y)'
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive OR :- (EX-OR)

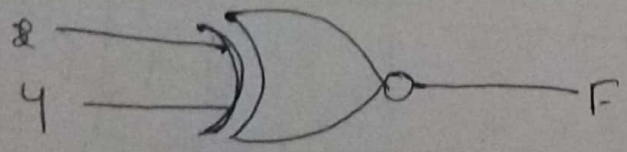


$F = xy' + x'y$

$F = x \oplus y$

x	y	$F = x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR Gate :-



$F = xy + x'y'$

$= (x \oplus y)'$

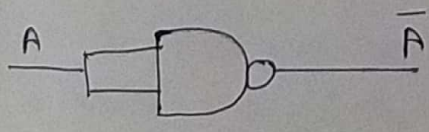
x	y	$F=(x \odot y)'$
0	0	1
0	1	0
1	0	0
1	1	1

Universal Logic gates : NAND and NOR Gates are called as Universal logic gates since They Can be used to implement any digital Circuit without using any other gates. This means That every gate Can be created by NAND or NOR gates only.

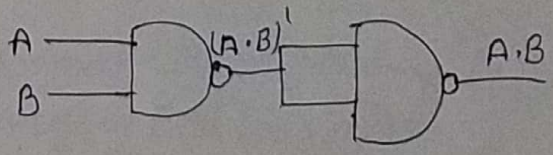
Implementation of Three logic gates using AND and NOR gates.

Using NAND Gate :

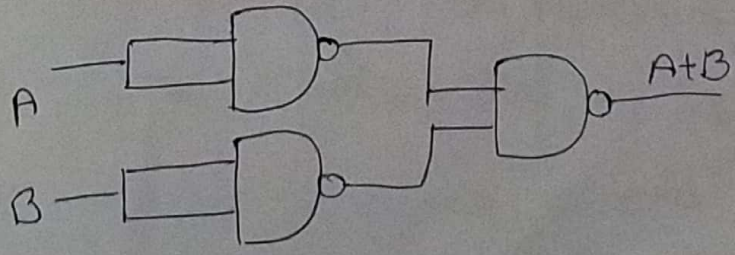
(i) NOT Gate :-



(ii) AND Gate :

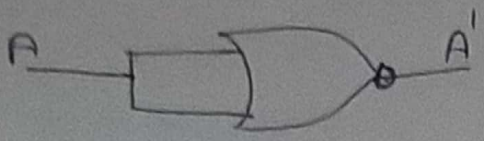


(iii) OR Gate

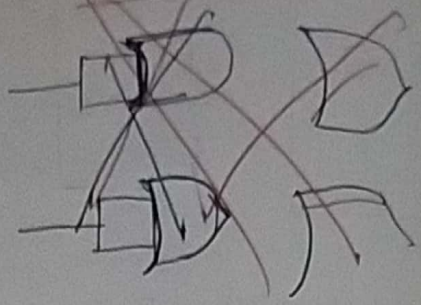


By using NOR Gate :-

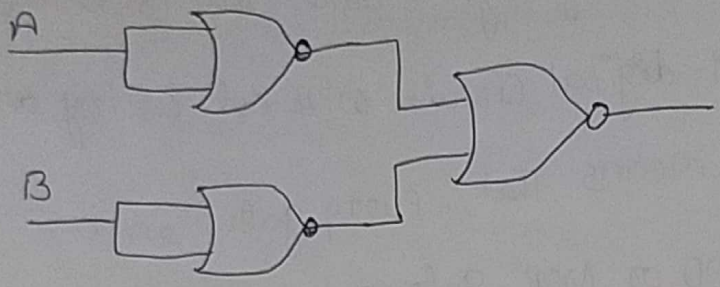
(i) NOT Gate



(ii) AND Gate :-



(ii) AND Gate :-



(iii) OR Gate :-