DIGITAL DESIGN AND COMPUTER ORGANIZATION.

UNIT-I.

Digital systems, Binary Numbers, Number Base Conversions, Octal, Hexadecimal and other base numbers, Complements, Signed binary numbers, Floating Point Number Representation, Binary Codes, Error detection and Correction, Binary storage and Registers, Binary logic, Boolean algebra and logic gates, Basic theorems and Properties of Boolean Algebra, Boolean functions, Canonical and Standard forms, Digital logic Gates.

UNIT-II

Gate level Minimization, The K-Map method, Three-Variable Map, four Variable Map, five Variable Map, Sum of Products, Product of Sums Simplification, Don't Care conditions, NAND and NOR implementation and other two level implementations, Exclusive-OR function.

Combinational Circuits: Adders, Subtractors, Comparators, Multiplexers, Demultiplexers, Decoders, Encoders and Code Converters.

UNIT-III

PART A :- Sequential Circuits: Introduction. Basic Architectural Distinctions between Combinational and Sequential Circuits, The Binary Cell, Latches, Flip flops: SR, Jk, Race Around-Condition in JK, JK Master Slave flipflop, D and T type flipflops, Excitation table of all flip flops, Design of a clocked flip flop, Timing and triggering Consideration, Clock Skew, Conversion from one type of flip flop to another.

PART B: Registers and Counters: Shift Registers, Data transmission in shift Register, Operation of Shift Register, Shift Register-Configuration, Bidirectional shift Register, Applications of Shift-Register, Design & operation of Ring & twisted Ring counter, Operation of Asynchronous and Synchronous Counters.

UNIT IV:

Register Transfer and Microoperations - Register Transfer language, Register transfer, Bus and memory transfers, Arithmetic microoperations, logic Microoperations, Shift Microoperations, Arithmetic logic Shift unit.

UNIT V:

Processor organization: Introduction to CPU, Register Transfers, Execution of Instructions, Multiple Bus organization, Hardwired-Control, Microprogrammed Control memory organization: Concept of memory, RAM, ROM memories, memory hierarchy, Cache memories, Virtual memory, Secondary storage, memory management-requirements.

Text Books:-

→ Digital logic Design, A.P. Godse, D.A Godse.

→ Digital Design, M. Morris Mano, M. D. Ciletti, 5th Edition, Pearson.

→ Computer system Architecture, M. Morris Mano, 3rd Edition, Pearson.

* DIGITAL SYSTEMS *

Every day digital Concepts are being applied to problems that could only be solved by analog methods several years ago.

→ Fast and reliable solutions using switching techniques provide the tremendous Power and usefulness of the digital systems.

→ Now-a-days digital systems are used in wide variety of Industrial and Consumer products such as automated industrial machinery, Pocket Calculators, microprocessors, digital Computers, digital watches, TV games, Signal Processing. and many.

→ The areas of applications of digital Systems have been increasing Everyday.

→ The digital Computer can be programmed to perform different tasks.

→ The users are allowed to specify and change the program or the data according to the specific need.

→ Because of this flexibility, general Purpose digital Computers Can perform a variety of information processing tasks that range over a wide spectrum of applications.

* Characteristics of Digital Systems :

→ Digital systems manipulate discrete elements of information.

→ Discrete elements are nothing but the digits such as 10 decimal digits, 26 letters of alphabets . . .

→ Digital systems use physical quantities called Signals to represent discrete Elements.

→ In digital systems, the signals have two discrete values and are therefore said to be binary.

→ A signal in digital system represents one binary digit, called a bit. The bit has a value either 0 or 1.

→ Discrete element of information are represented with group of bits called binary codes.
For example; the decimal digits 0 through 9 are represented in a digital system with 4-bit code called BCD (Binary coded Decimal) code.

→ The digital systems like a digital computer is programmable device which can be programmed to perform variety of tasks.

→ The digital systems like digital computer is an interconnection of digital modules. The major units/modules of digital computer are a central processing unit, memory unit and input-output unit.

→ The memory unit stores programs and data.

→ CPU performs Arithmetic and logical operations on data stored in memory specified by the program.

→ The input device such as keyboard is used to store program and data into the memory and display device such as pointer, monitor is used to display the result of operation.

* Definition:-

A digital system is an interconnection of the digital modules and it is a system that manipulates discrete elements of information that is represented internally in the binary form.

Number Systems

There are four Basic types of Number Systems which are listed below.

(1) Binary Number system
(2) Octal Number system
(3) Decimal Number system
(4) Hexa Decimal Number system

* Binary Number System:

→ Computers Cannot operate on decimal number, they do their operation on binary.
→ So, It is important to study the binary number system
→ for Example, we enter decimal number for booking the bus ticket ; but system will Convert to binary.
→ We have to Consider Base (or) Radix.
→ **Base or Radix :-**

The Value representing Particular number system.

(or)

→ It Defines the no. of digits or Symbols or letters used in a number system.

→ **How Base or Radix is defined ?** [How many numbers we have to use?]

To find how many digits or numbers used in a number system we have the formula;

→ If 'r' is the base then, 0 to r-1 are the digits present in that number system.

→ So, for Binary the base value is '2'

ie; the Base or Radix for Binary number system is 2.

→ Then the number of digits in binary number system are;

For Binary, $r = 2$

0 to $r-1$ digits

0 to $(2-1)$

0 to 1

i.e., 0 and 1

→ So, Binary digits (0 and 1) are called as "Bits"

→ Similarly for octal Number system, the Base value is 8,

then; $r = 8$

For Octal; $r = 8$

0 to $r-1$ digits

0 to $(8-1)$

0 to 7.

i.e; 0, 1, 2, 3, 4, 5, 6, 7 digits are present in octal.

→ For Decimal Number system; the Base Value is 10;

then, $r = 10$.

For Decimal; $r = 10$

0 to $(r-1)$

0 to $(10-1)$

0 to 9

i.e; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 digits are present in Decimal.

→ For Hexadecimal number system; the Base Value is 16;

then, $r = 16$.

For Hexadecimal; $r = 16$

0 to $(r-1)$

0 to $(16-1)$

0 to 15.

i.e; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 digits are present in Hexadecimal.

→ In Hexadecimal we are having decoding from 10 to 15.

i.e; 10 – A ; 11–B ; 12–C ; 13–D ; 14–E ; 15–F.

→ Why decoding is Required?

If we want to send Hexadecimal data to someone.

and if the data is 2and 15 ; we will send it as 2f5.

→ So, the person on otherside will understand it as 215 ie two-hundred and fifteen ; for that reason if we decode 15 with f

→ Then we send the data as 2f.

→ The Maximum Value of digits in

→ Binary number system → 1
→ Octal number system → 7
→ Decimal number system → 9
→ Hexadecimal no. System → 15(or) F.

→ In Binary number system ; weight is Expressed as Power of 2.
→ For Decimal number system ; weight is expressed as power of 10
→ For Hexadecimal number system ; weight is expressed as Power of 16.
→ For Octal number system ; weight is Expressed as Power of 8.

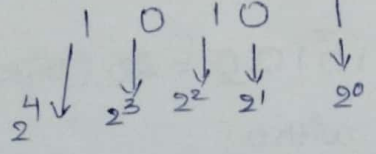
→ For Example :

Consider Binary number ; 10101

So, the weight is associated at Each and Every Position.

→ Here ; Base (or) Radix is 2.

So;



i.e; $\Rightarrow 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

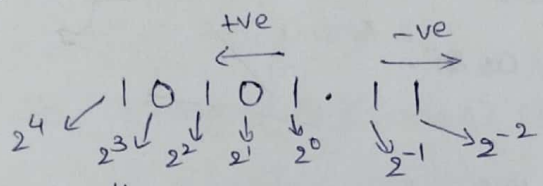
$\Rightarrow 16 + 0 + 4 + 0 + 1 \Rightarrow 21$

Ex2:- 10101.11

In this Example we are having decimal value so,

→ The values before the decimal point we consider from right left to right direction with Positive Sign.

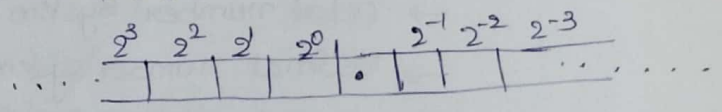
→ The values after the decimal point we consider from right to left to right direction with negative sign.



$\Rightarrow 1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}$

$\Rightarrow 16+0+4+0+1+0.5+0.25$

$\Rightarrow 21.75$



→ Bit is the smallest unit of data.

1 Nibble	= 4 Bits (used for BCD and Hexa)
1 Byte	= 8 Bits
1 Word	= 16 Bits = 2 Bytes
1 Doubleword	= 32 Bits = 4 Bytes

→ MSB and LSB :-

Ex :- 1 0 1 0 1 = $(21)_{10}$

b_4 b_3 b_2 b_1 b_0
1 0 1 0 1
MSB ↙ ↘ LSB

So, Replace the LSB bit with 0;

then it becomes 1010<u>0</u> = 20 (Difference is 1)

Replace the MSB bit with 0;

10101 → 00101 = (5) (Difference is 16)

more difference.

* Significance of left most bit is important.

For Decimal number System:

$$\overset{10^3\ 10^2\ 10^1\ 10^0}{(7\ 3\ 9\ 2)_{10}} = 7\times10^3 + 3\times10^2 + 9\times10^1 + 2\times10^0$$

For Octal number System:

$$\overset{8^2\ 8^1\ 8^0\ 8^{-1}\ 8^{-2}}{(1\ 3\ 6.\ 4\ 1)_8} = 8^2\times1 + 8\times3 + 8^0\times6 + 8^{-1}\times4 + 8^{-2}\times1$$

For Hexadecimal number System:

$$\overset{16^3\ 16^2\ 16^1\ 16^0\ 16^{-1}\ 16^{-2}}{(1\ 7\ 9\ 5.\ A\ B)_{16}} = 16^3\times1 + 16^2\times7 + 16^1\times9 + 16^0\times5 + 16^{-1}\times10 + 16^{-2}\times11$$

Similarly;

$$\overset{r^2\ r^1\ r^0\ r^{-1}}{(1\ 2\ 1.\ 3)_r} = r^0\times1 + r^1\times2 + r^2\times1 + r^{-1}\times3.$$

→ Decimal numbers are represented by binary 8421 Code.

Decimal Numbers	Binary 8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

* NUMBER BASE CONVERSIONS *

→ Computer uses binary number systems where as we use decimal numbers.

→ So, It is necessary to Convert decimal number into its equivalent binary while feeding number into the computer and to Convert binary number into its decimal equivalent while displaying result of operation to the human beings.

→ However, dealing with a large quantity of binary number of many bits is inconvenient for humans.

→ Therefore, octal and hexadecimal numbers are used as a short hand means of Expressing large binary numbers.

→ Digital circuits strictly work in Binary.

* Why Conversions?

We have decimal values, but system understand binary So, we Cannot give binary Everytime.

(1.) Decimal to Any other (i.e; Binary, octal, Hexadecimal)

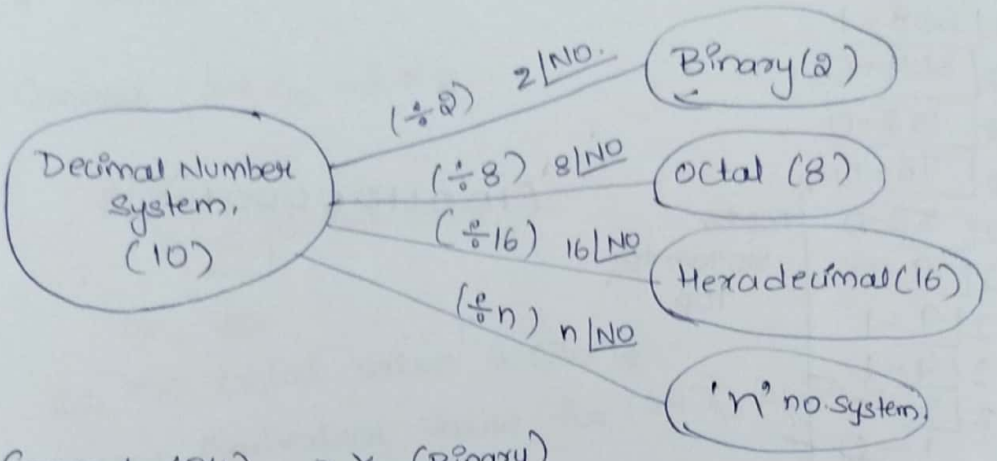
(2.) Any other (i.e; Binary, octal, Hexadecimal) to Decimal.

(3.) Octal to Binary and Binary to octal.

(4.) Hexadecimal to Binary and Binary to Hexadecimal.

(5.) Octal to Hexadecimal and Hexadecimal to Octal.

* Conversion of Decimal number system to Any Base (or) Radix.



Ex:- Convert $(24)_{10} \rightarrow X_2$ (Binary)

(i) Decimal To Binary.

For Binary Base is 2.

Divide 24 by 2

```
2 | 24   Remainder
2 | 12 — 0
2 | 6  — 0
2 | 3  — 0
    1  — 1
```

Bottom to top

* Consider the remainder values from bottom to top

i.e; 11000 is the binary equivalent for $(24)_{10}$

i.e., $(24)_{10} = (11000)_2$.

Ex 2:- Convert $(24.25)_{10} \rightarrow X_2$ (Binary)

Same procedure as above for decimal before value.

→ For After decimal digits we have to multiply by "2" ie Base Value.

For 24 we know binary equivalent is 11000.

(24.25)
↓ ↳ multiply by base value.
11000

$0.25 \times 2 = 0.5 = 0$
$0.5 \times 2 = 1.0 = 1$
Top to Bottom

Consider top to bottom
for (0.25) = 01

So; $(24.25)_{10} = (11000.01)_2$.

Ex:- Convert $(1217)_{10} = X_2$.

2 | 1217
2 | 608 – 1
2 | 304 – 0
2 | 152 – 0
2 | 76 – 0
2 | 38 – 0
2 | 19 – 0
2 | 9 – 1
2 | 4 – 1
2 | 2 – 0
1 – 0

Bottom to TOP

$(10011000001)_2$

The binary Equivalent for $(1217)_{10} = (10011000001)_2$.

Ex:- Convert $(41.6875)_{10} = X_2$.

2 | 41
2 | 20 – 1
2 | 10 – 0
2 | 5 – 0
2 | 2 – 1
1 – 0

Bottom to TOP

$0.6875 \times 2 = 1.375 = 1$
$0.375 \times 2 = 0.75 = 0$
$0.75 \times 2 = 1.50 = 1$
$0.50 \times 2 = 1.0 = 1$

Top to Bottom

The binary Equivalent for $(41.6875)_{10} = (101001.1011)_2$

Ex:- Convert $(24.625)_{10} = X_2$

2 | 24
2 | 12 – 0
2 | 6 – 0
2 | 3 – 0
1 – 1

Bottom to TOP

$0.625 \times 2 = 1.25 = 1$
$0.25 \times 2 = 0.5 = 0$
$0.5 \times 2 = 1.0 = 1$

Top to bottom

The binary Equivalent for $(24.625)_{10} = (11000.101)_2$

(ii) Conversion of Decimal to octal.

* Divide the given decimal number with Base value (8).

Ex: Convert $(24)_{10} \rightarrow X8$.

8 | 24
3 - 0 ↑ Bottom to top

i.e; 30

so, the octal value is $(30)_8$

The octal equivalent value for $(24)_{10} = (30)_8$.

Ex: Convert $(153)_{10} \Rightarrow X8$

8 | 153
8 | 19 - 1 ↑ Bottom to top
2 - 3

ie., $(231)_8$.

The octal equivalent value for $(153)_{10} = (231)_8$.

Ex: Convert $(0.513)_{10} \rightarrow X8$.

$0.513 \times 8 = 4.104 = 4$
$0.104 \times 8 = 0.832 = 0$
$0.832 \times 8 = 6.656 = 6$
$0.656 \times 8 = 5.248 = 5$
$0.248 \times 8 = 1.984 = 1$
$0.984 \times 8 = 7.872 = 7$
⋮

Top to bottom

The octal equivalent value for $(0.513)_{10} = (.406517\ldots)_8$

Ex; Convert $(204.25)_{10} \rightarrow X8$

8 | 204
8 | 25 - 4 ↑
3 - 1

$0.25 \times 8 = 2.00$

ie; The octal equivalent for $(204.25)_{10} = (314.2)_8$

(iii) Conversion of Decimal to Hexadecimal.

* Divide the given decimal number with the base value (ie, 16)

Ex:- Convert $(24)_{10} \rightarrow X_{16}$

16 | 24
1 – 8 ↑ Bottom to top

$(24)_{10} \rightarrow (18)_{16}$ or H

Ex:- Convert $(4769)_{10} \rightarrow X_{16}$

* Divide with base 16

16 | 4769
16 | 298 – 1 ↑
16 | 18 – A(10) Bottom to top
1 – 2

The Hexadecimal Equivalent value for $(4769)_{10} = (12A1)_{16}$

Ex :- Convert $(422.675)_{10} \rightarrow X_{16}$

16 | 422
16 | 26 – 6 ↑ Bottom to top
1 – 10 ↓ A

$0.675 \times 16 = 10.8$ → A (Top to bottom)
$0.8 \times 16 = 12.8$ → C
$0.8 \times 16 = 12.8$ → C

The Hexadecimal Equivalent value for $(422.675)_{10} = (1A6.ACC)_{16}$

Ex:- Convert Decimal $(24)_{10} \rightarrow X_4$ (Any Base)

Base = 4

4 | 24
4 | 6 – 0 ↑
1 – 2
Base 4

The Equivalent Value of $(24)_{10}$ is $(120)_4$.

Ex: Convert $(164.25)_{10} \rightarrow X_6$

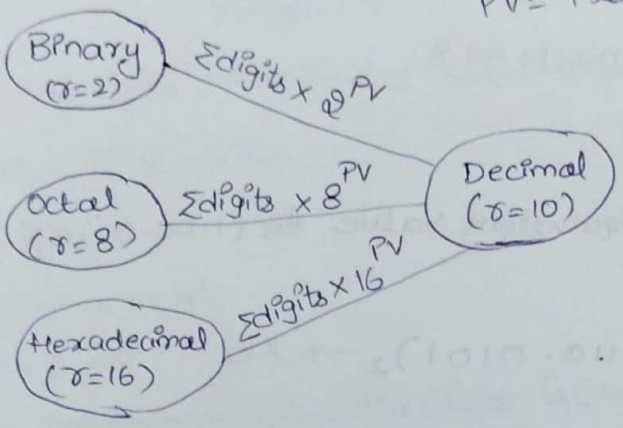
6 | 164
6 | 27 – 2 ↑
4 – 3

$0.25 \times 6 = 1.5 = 1$ ↓
$0.5 \times 6 = 3.0 = 3$

The Base 6 Equivalent Value for $(164.25)_{10} = (432.13)_6$

* Conversion of Anyother (Binary, Hexadecimal, octal) into Decimal Number System:

[Σ = Sum of digits
PV = Place value]



(i) Binary to Decimal.

* NOTE:- Let us take an example abc.de

So, Here for after decimal ie the numbers present after the decimal point are assigned by negative sign.

i.e; abc.de

In this d and e are assigned by negative values.

So, If we want to Convert binary to decimal for abc.de

$$\begin{matrix} 2 & 1 & 0 & -1 & -2 \\ a & b & c & . d & e \end{matrix}$$

$$a\times 2^2 + b\times 2^1 + c\times 2^0 + d\times 2^{-1} + e\times 2^{-2}$$

Ex:- Convert $(1100)_2 \rightarrow x_{10}$

$$\begin{matrix} 1 & 1 & 0 & 0 \\ 2^3 & 2^2 & 2^1 & 2^0 \end{matrix}$$

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

$$= 8+4+0+0 = 12$$

The decimal equivalent value for $(1100)_2 = (12)_{10}$.

Ex:- Convert $(1100.01)_2 \rightarrow X_{10}$

$$\overset{3}{1}\overset{2}{1}\overset{1}{0}\overset{0}{0}.\overset{-1}{0}\overset{-2}{1}$$

$\Rightarrow 1\times2^{-2}+0\times2^{-1}+0\times2^{0}+0\times2^{1}+1\times2^{2}+1\times2^{3}$

$\Rightarrow 0.25+0+0+0+4+8$

$\Rightarrow (12.25)_{10}$

The decimal Equivalent value for $(1100.01)_2 = (12.25)_{10}$

Ex:- Convert $(10110.0101)_2 \rightarrow X_{10}$

$$\overset{4}{1}\overset{3}{0}\overset{2}{1}\overset{1}{1}\overset{0}{0}.\overset{-1}{0}\overset{-2}{1}\overset{-3}{0}\overset{-4}{1}$$

$\Rightarrow 1\times2^{4}+0\times2^{3}+1\times2^{2}+1\times2^{1}+0\times2^{0}+0\times2^{-1}+1\times2^{-2}+0\times2^{-3}+1\times2^{-4}$

$\Rightarrow 16+4+2+(1/4)+(1/16)$

$\Rightarrow 22.3125$

The decimal Equivalent value for $(10110.0101)_2 = (22.3125)_{10}$

Ex:- Convert $(11011.01)_2 \rightarrow X_{10}$

$$\overset{4}{1}\overset{3}{1}\overset{2}{0}\overset{1}{1}\overset{0}{1}.\overset{-1}{0}\overset{-2}{1}$$

$\Rightarrow 1\times2^{4}+1\times2^{3}+0\times2^{2}+1\times2^{1}+1\times2^{0}+0\times2^{-1}+1\times2^{-2}$

$\Rightarrow 16+8+2+1+(1/4)$

$\Rightarrow 27.25.$

The decimal equivalent value for $(11011.01)_2 = (27.25)_{10}$

Ex:- Convert $(101101)_2 \rightarrow X_{10}$

$$\overset{5}{1}\overset{4}{0}\overset{3}{1}\overset{2}{1}\overset{1}{0}\overset{0}{1}$$

$\Rightarrow 2^{5}\times1+2^{4}\times0+2^{3}\times1+2^{2}\times1+2^{1}\times0+2^{0}\times1$

$\Rightarrow 32+8+4+1 \Rightarrow 45.$

The decimal equivalent value for $(101101)_2 = (45)_{10}$

(ii) Octal to Decimal Conversion:

radix = 8 → radix 10

$\Sigma digit \times 8^{PV}$

i.e; Sum(digits) multiplied with 8 power Place Value.

Ex:- Convert $(12)_8 \rightarrow \times 10$.

$(1^{1}\ 2^{0})_8$

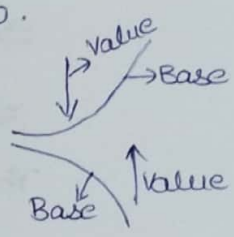
$\Rightarrow 2 \times 8^0 + 1 \times 8^1$

$\Rightarrow 2 + 8 = 10$

The decimal Equivalent Value for $(12)_8$ is $(10)_{10}$

* Lower base to Higher base; we get low value.
  i.e., In the above 8→10 (Base value); Value decreased from 12→10.

* Higher base to lower base; we get higher Value.



Ex:- Convert $(12.10)_8 \rightarrow \times 10$

$(1^{1}\ 2^{0}.1^{-1}\ 0^{-2})$

$\Rightarrow 1 \times 8^1 + 2 \times 8^0 + 1 \times 8^{-1} + 0 \times 8^{-2}$

$\Rightarrow 8 + 2 + 0.125 + 0$

$\Rightarrow (10.125)_{10}$

The decimal equivalent Value for $(12.10)_8$ is $(10.125)_{10}$.

Ex:- Convert $(126.5)_8 \rightarrow \times 10$

$(1^{8^2}\ 2^{8^1}\ 6^{8^0}.\ 5^{8^{-1}})$

$\Rightarrow 1 \times 8^2 + 2 \times 8^1 + 6 \times 8^0 + 5 \times 8^{-1}$

$\Rightarrow 64 + 16 + 6 + (5/8)$

$\Rightarrow 86.625$

(iii) Convert Hexadecimal to Decimal.

PV → Place Value.

Base = 16 → Base = 10

$\Sigma digit \times 16^{PV}$

Ex:- Convert $(129)_{16} \rightarrow X_{10}$

$16^2\ 16^1\ 16^0$
$1\ 2\ 9$

$\Rightarrow 16^2 \times 1 + 16^1 \times 2 + 16^0 \times 9$

$\Rightarrow 256 + 32 + 9$

$\Rightarrow 297.$

The decimal Equivalent value for $(129)_{16}$ is $(297)_{10}$

Ex:- Convert $(1F)_{16} \rightarrow X_{10}$

$\Rightarrow 15 \times 16^0 + 1 \times 16^1$

$\Rightarrow 15 + 16 = 31$

The decimal Equivalent value for $(1F)_{16}$ is $(31)_{10}$

Ex:- Convert $(AB.24)_{16} \rightarrow X_{10}$

$16^1\ 16^0\ 16^{-1}\ 16^{-2}$
$A\ B . 2\ 4$
$10\ 11$

$\Rightarrow 16^1 \times 10 + 16^0 \times 11 + 2 \times 16^{-1} + 4 \times 16^{-2}$

$\Rightarrow 160 + 11 + 0.125 + 0.015625$

$\Rightarrow 171.140625$

The decimal Equivalent value for $(AB.24)_{16}$ is $(171.140625)_{10}$.

Ex: Convert $(BABA)_{16} \rightarrow X_{10}$

$16^3\ 16^2\ 16^1\ 16^0$
$B\ A\ B\ A$

$\Rightarrow 11 \times 16^3 + 10 \times 16^2 + 11 \times 16^1 + 10 \times 16^0$

$\Rightarrow 45056 + 2560 + 176 + 10$

$\Rightarrow 47802.$

The decimal Equivalent value for $(BABA)_{16}$ is $(47802)_{10}$.

* Base 4 / Base3 TO Decimal Conversion :-

Base 3 — 0, 1, 2

Base 4 — 0, 1, 2, 3

Ex: Convert $(21)_3 \rightarrow X_{10}$

3^1 ← 2 ; 1 → 3^0

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

$\Rightarrow 6 + 1 = 7$

The ~~Base 3~~ decimal equivalent for Base 3 $(21)_3$ is $(7)_{10}$

Ex: Convert $(112.22)_3 \rightarrow X_{10}$; Convert $(3122.33)_4 \rightarrow X_{10}$

(a) $(112.22)_3$

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

$1\ 1\ 2 . 2\ 2$

$\Rightarrow 1 \times 3^2 + 1 \times 3^1 + 2 \times 3^0 + (2/3) + (2/9)$

$\Rightarrow 9 + 3 + 2 + 0.66 + 0.22$

$\Rightarrow 14.88$

$(112.22)_3 = (14.88)_{10}$

(b) $(3122.33)_4$

$4^3\ 4^2\ 4^1\ 4^0\ 4^{-1}\ 4^{-2}$

$3\ 1\ 2\ 2 . 3\ 3$

$\Rightarrow 4^3 \times 3 + 4^2 \times 1 + 4^1 \times 2 + 4^0 \times 2 + 4^{-1} \times 3 + 4^{-2} \times 3$

$\Rightarrow 192 + 16 + 8 + 2 + (3/4) + (3/16)$

$\Rightarrow 192 + 16 + 10 + 0.75 + 0.1875$

$\Rightarrow 218.9375$

$(3122.33)_4 = (218.9375)_{10}$.

Ex: Convert $(321)_4 \Rightarrow X_{10}$

$3\ 2\ 1$

$4^2\ 4^1\ 4^0$

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

$\Rightarrow 57.$

$(321)_4 = (57)_{10}$.

* Octal to Binary and Binary to Octal Conversion

(i) Octal to Binary Conversion:

The relation between octal and Binary number system is

Octal(8) ; Binary(2)

i.e; $\text{Binary}_{(2)}^{3} = 8$

Octal	Binary
	2^2 2^1 2^0
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

* To convert octal to Binary, each digit of the octal number is individually converted to its binary equivalent to get octal to binary conversion of the number.

Ex:- Convert $(64)_8 \rightarrow X_2$

So; 64

Binary value for 6 is 110

Binary value for 4 is 100

$(110100)_2$

ie the Binary value for $(64)_8 = (110100)_2$.

Ex:- Convert $(64.25)_8 \rightarrow X_2$

$(110100.010101)_2$.

(for two value is 010
For 5 value is 101)

(ii) Conversion of Binary to Octal :-

We know that base for octal number is 8 and the base for binary number is 2.

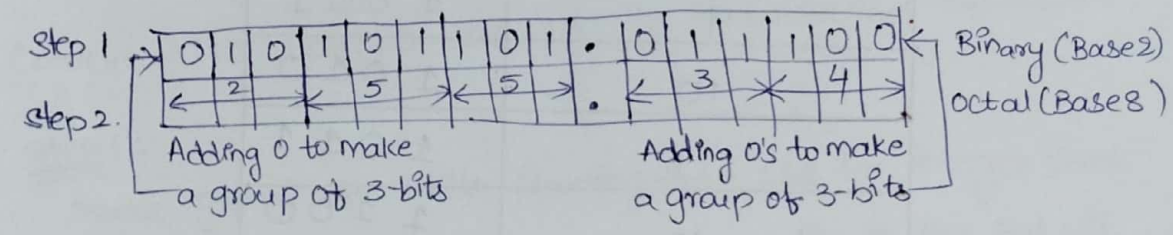
→ The base for octal number is the third power of the base for binary numbers.

→ Therefore; by grouping 3 digits of binary numbers and then Converting Each group digit to its octal Equivalent we Can Convert binary number to its octal Equivalent.

Ex :- Convert $(10101101.0111)_2$ to octal Equivalent.

Sol :- Step 1 : Make group of 3-bits starting from LSB for integer Part and MSB for fractional part, by adding 0's at the End; if required.

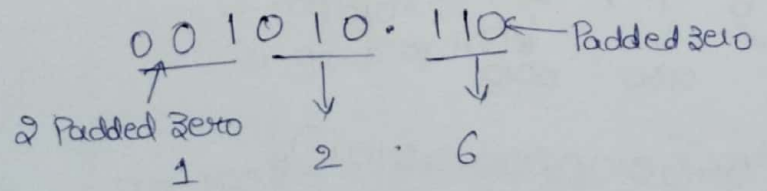
Step 2 : Write Equivalent Octal number for Each group of 3 bits



$\therefore (10101101.0111)_2 = (255.34)_8$.

Ex :- Convert $(1010.11)_2 \rightarrow X_8$.

Sol : Need 3 Binary bits ; So group 3 bits left grouping for decimal before and right grouping for after decimal.



$(12.6)_8$.

$(1010.11)_2 \rightarrow (12.6)_8$.

✱ Conversion of Hexadecimal to Binary and Binary to Hexadecimal:-

(1) Hexadecimal to Binary:

→ Conversion depends on relation of Hexa radix and Binary radix.

Hexadecimal = 16 = 2^4	Binary = 2
	2^3 2^2 2^1 2^0
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

Ex:- Convert $(64.25)_H \rightarrow X_2$

6 4 . 2 5 → 0101

0110 0100 0010

$(01100100.00100101)_2$.

Ex: Convert $(BABA)_{16} \rightarrow X_2$.

B A B A → 1010

1011 1010 1011

$(1011101010111010)_2$

Ex: Convert $(16.5)_{16} \rightarrow X_2$.

16 . 5 → 0101

0001 0110

$(16.5)_{16} \rightarrow (00010110.0101)_2$.

(ii) Binary to Hexadecimal:

→ We know base for hexadecimal numbers is 16 and the base for binary numbers is 2.

→ The base for hexadecimal number is the fourth power of the base for binary numbers.

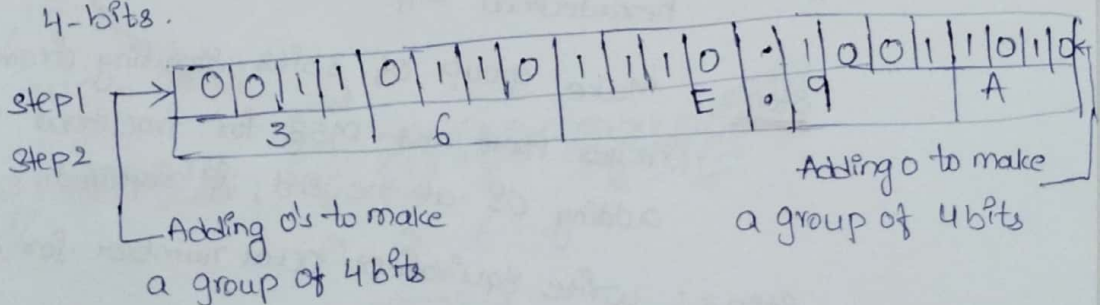
→ Therefore, by grouping 4 digits of binary numbers and then Converting each group digit to its hexadecimal Equivalent we Can Convert binary number to its hexadecimal equivalent.

Ex: Convert 1101101110.1001101 to hexadecimal Equivalent.

Sol: Step1:

Make group of 4-bits Starting from LSB for Integer Part and MSB for fractional Part; by adding 0's at the End; if required.

Step2: Write Equivalent hexadecimal number for Each group of 4-bits.



$\therefore (1101101110.1001101)_2 = (36E.9A)_{16}$

Ex: Convert $(1101101100)_2 \rightarrow X_{16}$

00 | 1101 | 1100
Padding | 3 | 6 | C

$\therefore (1101101100)_2 \rightarrow (36C)_{16}$.

Ex: Convert $(101011.01101)_2 \rightarrow X_{16}$

Pad ← 0010 | 1011 . 0110 | 1000 → Pad
2 | B | 6 | 8

i.e., $(101011.01101)_2 \rightarrow (2B68)_{16}$

* Conversion of Hexadecimal to octal and octal to Hexadecimal :-

There is no direct Conversion.

→ So, we need a mediator i.e., Binary (or) Decimal.

→ Binary mediator is simple.

* Hexa decimal to octal Conversion :-

Ex: Convert $(BC66.AF)_{16} \rightarrow X_8$.

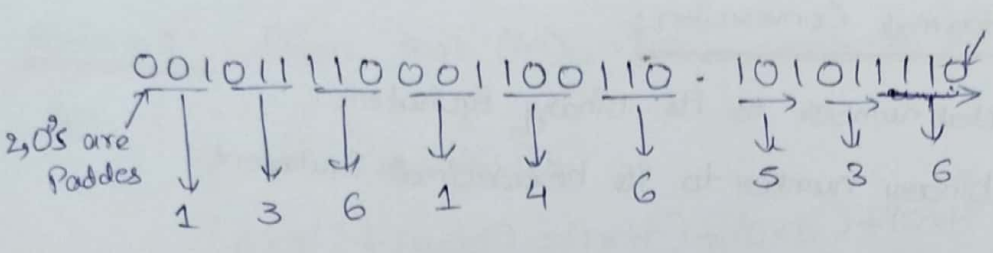
Sol: Step 1: write equivalent 4-bit binary number for each hexadecimal digit.

Step 2: Make group of 3bits starting from LSB for integer part and MSB for fractional part by adding 0's at the end; if required.

Step 3: write equivalent octal number for each group of 3bits.

B C 6 6 . A F → 1111
1011 | 1100 | 0110 | 0110 | 1010

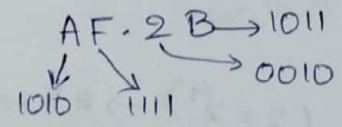
10111100011001101010111 → Binary Mediator.



$(BC66.AF)_{16} \longrightarrow (136146.536)_8$.

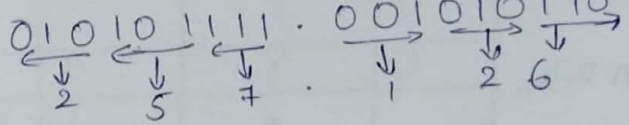
Ex: Convert $(AF.2B)_H \longrightarrow X8$.

Convert first Hexadecimal to Binary then Binary to Octal.



Binary:- 10101111 . 00101011
(Mediator)

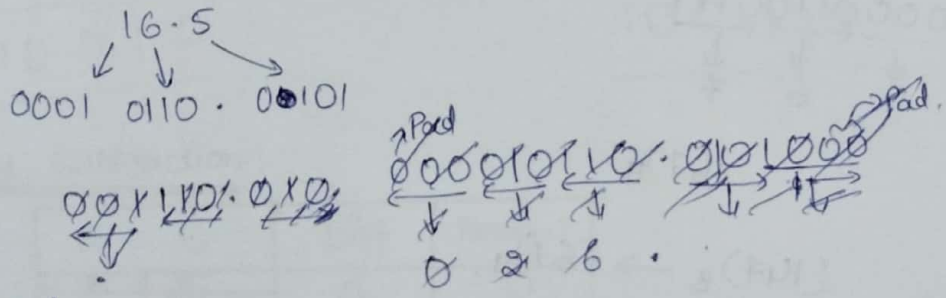
So, Binary to octal, we need to group 3bits.

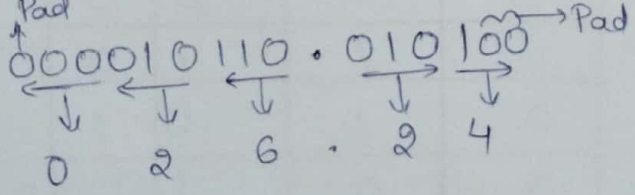


octal : $(257.126)_8$.

$(AF.2B)_H \rightarrow (257.126)_8$

Ex:- Convert $(16.5)_{16} \rightarrow X8$.





$(26.24)_8$.

$(16.5)_{16} \rightarrow (26.24)_8$.

*Octal to Hexadecimal Conversion:-

→ Convert octal number to its binary Equivalent.

→ Convert binary number to its hexadecimal Equivalent.

Ex:- Convert $(72.1)_8 \rightarrow X_H$

Mediator is binary ; $8 - 2^3 \rightarrow$ 3 bits in binary data.

7 2 . 1 → 001

111 010

$(111010.001)_2$

→ Binary to Hexadecimal. $(2 = 2^4) \rightarrow$ Four Bits.

Pad 0's

0011 1010 . 0010 ← Pad '0'

3 A 2

$\Rightarrow (3A.2)_H$

$(72.1)_8 \rightarrow (3A.2)_H$.

Ex:- Convert $(147)_8 \rightarrow X_H$

1 4 7

001 100 111

0000 0110 0111

0 6 7

$(67)_H$

$(147)_8 \rightarrow (67)_H$.

Problem 1: Given that $(64)_{10} = (100)_b$. determine the value of b.

Sol:

$$(\overset{1}{6}\overset{0}{4})_{10} = (\overset{2}{1}\overset{1}{0}\overset{0}{0})_b$$

$$(6 \times 10^1) + (4 \times 10^0) = (1 \times b^2) + (0 \times b^1) + (0 \times b^0)$$

$$60 + 4 = b^2$$

$$64 = b^2$$

$$(8)^2 = b^2$$

$$\therefore b = 8$$

$\therefore$ The Base is 8.

* **Binary Addition :-**

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

Ex: Add 1101 and 0101

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

Ex: Add 110110 and 011001

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

* **Binary Subtraction :**

A − B		Diff	Borrow
A	B		
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Ex: Subtract 11001 and 00110

```
  11001
(-)00110
-------
  10011
-------
```

Ex: Subtract 0110 and 0011

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

Ex: Subtract 1100 and 1011

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

Diff Borrow
0-1 = 1 1

1100 is 12 and 1011 is 11 and the difference of 12 and 11 is 1.

After subtracting we got the result 0001 ie 1.

Ex: Subtract $(0011)_2$ from $(1100)_2$

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

COMPLEMENTS:

Complements are used in digital computers to simplify the operation of Subtraction for logical manipulation.

* There are two types of Complements for Each base-r system:

* The radix Complement (or) r's Complement.
* The diminished radix Complement (or) $(r-1)$'s Complement.

→ For Binary number system – 2's Complement and
1's Complement.

→ For Decimal number System – 10's Complement and
9s Complement.

1's Complement:

To find the one's Complement of a given number of bits we have; $(r^n - 1) - N$.
↳ Given no. of bits

Here; $r \rightarrow$ Base or Radix.
$n \rightarrow$ Number of bits

Ex: Find the 1's Complement for 1101.

$1101 = 4 \text{ bits} = n$

As it is a binary number; Base $= 2 = r$

$\Rightarrow (r^n - 1) - N$

$\Rightarrow (2^4 - 1) - 1101$

$\Rightarrow (16 - 1) - 1101$

$\Rightarrow 15 - 1101$

Binary Value for 15 is 1111

$$\begin{array}{r} 1111 \\ (-)\,1101 \\ \hline 0010 \end{array}$$

1's Complement for 1101 is 0010.

Similarly,

1's Complement for;

11011001 $\xrightarrow{\text{1's Complement}}$ 00100110

1111110101 $\xrightarrow{\text{1's Complement}}$ 0000001010.

* **9's Complement :-** $(r^n - 1) - N$.

Ex: Find The 9's Complement for 4.

$(r^n - 1) - N$ ↳ given no.(4)

$r = 10$; $n = 1$.

$\Rightarrow (10^1 - 1) - 4$

$\Rightarrow 9 - 4 = 5$.

Ex: 24 $\xrightarrow{\text{9's Complement}}$ $\begin{array}{r} 99 \\ (-)\ 24 \\ \hline 75 \end{array}$

ie; $r = 10$; $n = 2$

$(r^n - 1) - N \Rightarrow (10^2 - 1) - 24 \Rightarrow 99 - 24 = 75$.

Ex 789 $\xrightarrow{\text{9's Complement}}$ $999 - 789 = 210$

Ex: 5476000 $\xrightarrow{\text{9's Complement}}$ $9999999 - 5476000$

$= 4523999$.

Ex: 4716 $\xrightarrow{\text{9's Complement}}$ $9999 - 4716 = 5283$.

* 2's Complement :- $(r^n - N)$

Ex: Find the 2's complement for $(1010)_2$.

$(r^n - N)$

Given no. is binary ; Base, $r = 2$

no. of digits, $n = 4$

Given number, $N = 1010$.

$(2^4 - 1010) = (16 - 1010)$

Binary Equivalent for 16 is 10000

$$\begin{array}{r} 10000 \\ (-)\ 1010 \\ \hline 00110 \end{array}$$

(or)

2's Complement = 1's complement + 1.

i.e; Given $(1010)_2$.

1010 $\xrightarrow{\text{1's Complement}}$ 0101

Add '1' to 0101 ;

$$\begin{array}{r} 0101 \\ +\ \ 1 \\ \hline 0110 \end{array}$$

Ex: find 2's complement for $(11011010)_2$.

$(11011010)_2 \xrightarrow{\text{2's Complement}} 00100101$

$$\begin{array}{r} 00100101 \\ +\ \ \ \ \ \ 1 \\ \hline 00100110 \end{array}$$

2's Complement for $(11011010)_2$ is $(00100110)_2$.

* 10's Complement :- $(r^n - N)$

Ex: Find the 10's Complement for $(24)_{10}$.

$(24)_{10}$

$(r^n - N)$

$r = 10$; $n = 2$; Given number, $N = 24$.

$(10^2 - 24) \Rightarrow 100 - 24 = 76$.

(OR)

10's complement = 9's Complement + 1.

i.e; 24 $\xrightarrow{\text{10's Complement}}$

```
  99
(-) 24
  75
+  1   → Add '1'
  76
```

Ex: Find 10's Complement for $(102398)_{10}$

$(102398)_{10}$ $\xrightarrow{\text{10's complement}}$

```
  999999
(-) 102398
  897601
+      1
  897602
```

Ex: Find the 10's Complement for $(246700)_{10}$

$(246700)_{10}$ $\xrightarrow{\text{10's Complement}}$

```
  999999
(-) 246700
  753299
+      1
  753300
```

The 10's Complement for $(246700)_{10}$ is 753300.

* Subtraction with Complements:

The direct method of subtraction uses the borrow concept. In this method, we borrow a 1 from a higher significant position when the minuend digit is smaller than the subtrahend digit.

→ The subtraction of two n-digit unsigned number M-N is base 'r' can be done as follows:

(1) Add the minuend M to the r's complement of the subtrahend N.

(2) If $M \geq N$; the sum will produce an end carry r^n which can be discarded.

(3) If $M < N$; the sum does not produce an end carry.

* 1's Complement Subtraction:

Case (i): Subtraction of smaller number from larger number.

Ex: 1101 − 0110

(i) Take 1's complement of subtrahend 0110.

i.e., 0110 $\xrightarrow{\text{1's Complement}}$ 1001

(ii) Add the 1's complement of subtrahend to minuend.

i.e.,
```
  1 0 0 1
+ 1 1 0 1
-------------
1 0 1 1 0
-------------
```
Carry → 1

(iii) Remove the carry and add it to the final result.

i.e;
```
  0 1 1 0
+       1
-------------
  0 1 1 1  ⇒ 7
-------------
```

Verification:-
```
  1 1 0 1
(-) 0 1 1 0
-------------
  0 1 1 1
-------------
```

Decimal:
```
  13
- 6
----
  7
----
```

Ex 2:- 1010100 − 1000011

(i) 1000011 $\xrightarrow{\text{1's Complement}}$ 0111100

(ii)
$$\begin{array}{r} 0111100 \\ +\ 1000000 \\ \hline \text{Carry} \leftarrow (1)\ 0010000 \end{array}$$

(iii)
$$\begin{array}{r} 0010000 \\ +\qquad 1 \\ \hline 0010001 \Rightarrow 17 \end{array}$$

Verification:-

$$\begin{array}{rll} & 1010100 & \longrightarrow 84 \\ (-) & 1000011 & \longrightarrow (-)67 \\ \hline & 0010001 & 17 \text{ Verified.} \end{array}$$

Case (ii): Subtraction of larger number from smaller number.

Ex: 0101 − 1010

(i) Find 1's Complement of the subtrahend.

1010 $\xrightarrow{\text{1's Complement}}$ 0101

(ii) Add 1's Complement of subtrahend to minuend.

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

(iii) The result will be negative; take 1's Complement and assign −ve sign.

1010 $\xrightarrow{\text{1's Complement}}$ −0101

Ex2: 1001 − 101000

(i) 101000 —1's Complement→ 010111

(ii)
```
  010111
+ 001001
--------
  100000
```

(iii) 100000 —1's Complement→ 011111

Verification :- 1001 − 101000 ⇒ 9 − 40 ⇒ −31

* 2's Complement Subtraction :-

Case (i) : Subtraction of smaller number from larger number.

Ex: (i) Subtract $(01010)_2$ from $(11001)_2$.

Minuend → $(11001)_2 - (01010)_2$ ← Subtrahend

(i) Take 2's complement of the subtrahend.

$(01010)_2$ —2's Complement→ 1's complement + 1

```
  10101
+     1
-------
  10110
```

(ii) Add 2's complement of subtrahend to minuend.

```
  10110
+ 11001
---------
 (1)01111
```

Carry → 1

(iii) Discard the carry.

i.e., 01111

Verification :-

```
  11001
- 01010
-------
  01111
```

ie., 25 − 10 = 15

(ii) 10011 − 10001

(i) 10001 $\xrightarrow{\text{2's Complement}}$ 01110+1 = 01111

(ii)
```
  10011
+ 01111
-------
 100010
```
Carry → 1

(iii) Discard Carry 00010

Verification:

10011	19
(−) 10001	(−) 17
00010	2

Case(ii) :- Subtraction of larger number from smaller number.

Ex(i):- 100 − 10101

(i) Take 2's Complement of Subtrahend.

10101 $\xrightarrow{\text{2's Complement}}$
```
 01010
+    1
-----
 01011
```

(ii) Add 2's Complement of Subtrahend to minuend.

```
 01011
+  100
-----
 01111
```

(iii) Take 2's complement and assign negative sign.

i.e; 01111 $\xrightarrow{\text{2's Complement}}$
```
 10000
+    1
-----
-10001
```

Verification:-

4 − 21

⇒ −17.

(Smaller number from larger number).

Ex(ii) :- 1000011 − 0101100

(i) 0101100 — 2's Complement → 1010011 + 1 = 1010100

(ii)
```
  1000011
+ 1010100
----------
 (1)0010111
```
Carry → 1

(iii) Discard Carry i.e; 0010111 //—

* 9's Complement :-

Case (i) :- Subtraction of Smaller number from larger number.

Minuend → 52532 − 3250 ← Subtrahend

Ex(i) :-

(i) Take 9's complement of Subtrahend.

```
  99999
(-> 03250
----------
  96749
```

(ii) Add 9's Complement of Subtrahend to minuend.

```
  52532
(+)96749
----------
(1)49281
```
Carry → 1

(iii) Remove the Carry and add to final result.

```
  49281
+     1
----------
  49282
```

Verification :-

```
  52532
(-> 3250
----------
  49282
```

Case (ii) : Subtraction of larger number from Smaller number :

Ex :- 3250 - 52532

→ 99999
(-) 52532
47467

→ 47467
+ 3250
50717

→ Take 9's Complement of the result.

99999
(-) 50717
49282

* 10's Complement :-

Case (i) :- Subtraction of Smaller number from larger number.

Ex1: 6428 - 3409

(i) Take 10's Complement of subtrahend.

9999
(-) 3409
6590
+ 1
6591

(ii) Add 10's Complement of subtrahend to minuend.

6428
+ 6591
Carry → (1) 3019

(iii) Discard the Carry.

3019.

Verification :-

6428
(-) 3409
3019

Case (ii): Subtraction of larger number from smaller number.

Ex:- 125 - 1800

(i) Take the 10's Complement of Subtrahend.

```
  9999
(-) 1800
------
  8199
(+)    1
------
  8200
------
```

(ii) Add 10's Complement of Subtrahend to minuend.

```
   0125
(+)8200
------
  8325
------
```

(iii) Take 10's Complement (and assign -ve Sign).

8325 ——10's Complement——→

```
  9999
(-)8325
------
  1674
+    1
------
 -1675 //-
------
```

i.e; -1675//-

* SIGNED BINARY NUMBERS * :

→ Positive Integers (including zero) can be represented as unsigned numbers.

→ To represent negative numbers, we need a notation for negative values.

→ In ordinary arithmetic, a negative number is indicated by minus sign and positive number is by plus sign.

→ Computers should represent everything with binary digits. The sign should be represented in the left most position of the number.

→ 0 → Positive and 1 → Negative.

→ It is important to realize that both signed and unsigned binary numbers consists of a string of bits when represented in a computer.

If the Binary number is signed; then the left most (MSB Bit) bit represents the sign and the rest of bits represent number.

→ If the binary number is assumed to be unsigned; then the left most bit is the most significant bit of the number.

For Example: $+9 = \underset{\text{Sign}}{0}\ \underset{\text{Magnitude}}{1001}$

$-9 = 11001 \rightarrow$ Sign magnitude

$\Rightarrow 10110 \rightarrow$ Sign 1's Complement

$\Rightarrow 10111 \rightarrow$ Signed 2's Complement.

Example:- -6 is represented in 3 formats.

Consider the number 6 represented in binary with 8 bits.

Signed magnitude representation : 10000110

Signed-1's Complement representation : 11111001

Signed-2's Complement representation : 11111010.

→ In Signed-magnitude; −6 is obtained from +6 by changing the Sign bit in the leftmost (MSB) position from 0 to 1.

→ In Signed 1's complement; −6 is obtained by complementing all the bits of +6; including Sign bit.

→ The Signed-2's complement representation of −6 is obtained by taking 2's complement of Positive number; including the Sign bit.

Decimal	Signed-2's Complement	Signed-1's Complement	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	[illegible]	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	[illegible] 1101	1100	1011
−4	[illegible] 1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111

Arithmetic 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 subtract the smaller magnitude from the larger and give the difference sign of the larger magnitude.

→ The rule for adding numbers in the signed complement system does not require a comparison or subtraction, but only addition.

The addition of two signed binary numbers with negative numbers represented in signed 2's complement form is obtained from the addition of the two numbers, including their sign bits.

→ A carry out of the sign bit position is discarded.

Ex: Add +6 and +13.

0 → Positive
1 → Negative

+13 ⇒ 00001101
+6 ⇒ 00000110

```
  00000110
+ 00001101
----------
  00010011
----------
```

Ex: Add −6 and +13.

−6 ⇒ 00000110 → Sign Magnitude.
⇒ 11111001 → Signed 1's complement
⇒ 11111010 → Signed 2's complement

```
−6
+13
---
+7
---
  11111010
+ 00001101
----------
 (1)00000111
----------
```

Discard the carry.

Ex: +6 and -13.

$-13 \Rightarrow 10001101 \rightarrow$ Sign Magnitude

$\Rightarrow 11110010 \rightarrow$ Sign 1's Complement

$\Rightarrow 11110011 \rightarrow$ Sign 2's Complement

$+6 \Rightarrow 00000110$

Ex:- +6, -13, -7

$$\begin{array}{r} 00000110 \\ +\ 11110011 \\ \hline 11111001 \end{array} \rightarrow \text{Take 2's Complement.}$$

$\Downarrow$ 1's complement + 1

$$\begin{array}{r} 10000110 \\ +\ \quad 1 \\ \hline 10000111 \end{array} \rightarrow (-7)$$

Ex:- -6, -13, -19

$-6 \Rightarrow 11111010$

$-13 \Rightarrow 11110011$

$$\begin{array}{r} 11111010 \\ +\ 11110011 \\ \hline (1)\ 11101101 \end{array} \rightarrow \text{2's Complement}$$

Discard

$\Downarrow$

$$\begin{array}{r} 10010010 \\ +\ \quad 1 \\ \hline 10010011 \end{array} \Rightarrow (-19)$$

* Binary Codes *

When numbers, alphabets or words are represented by a specific group of symbols, we can say that they are encoded.

→ The group of symbols used to encode them are called codes.

→ The digital data is represented, stored and transmitted as groups of binary digits (bits).

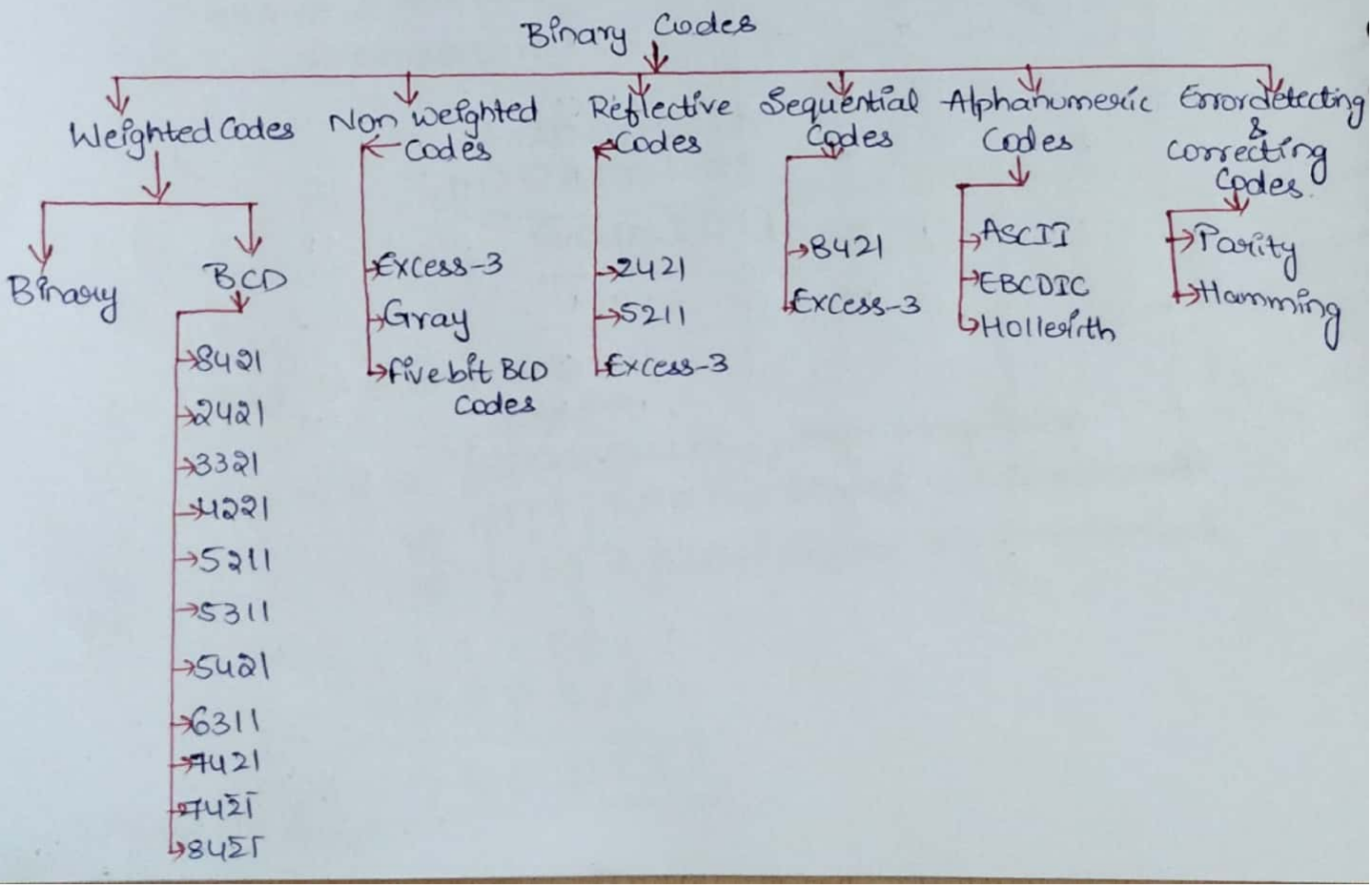
→ The group of bits, also known as Binary code, represent both numbers and letters of the alphabets as well as many special characters and control functions.

→ They are classified as numeric or alphanumeric.

- Numeric codes are used to represent numbers.
- Alphanumeric codes are used to represent characters. i.e Alphabetic letters & numerals.
  - In these codes, a numeral is treated simply as another symbol rather than as a number or numeric-value.

* Classification of Binary Codes :

These codes are classified as ;



*Weighted Codes :-

In Weighted Codes; Each digit position of the number represents a specific weight.

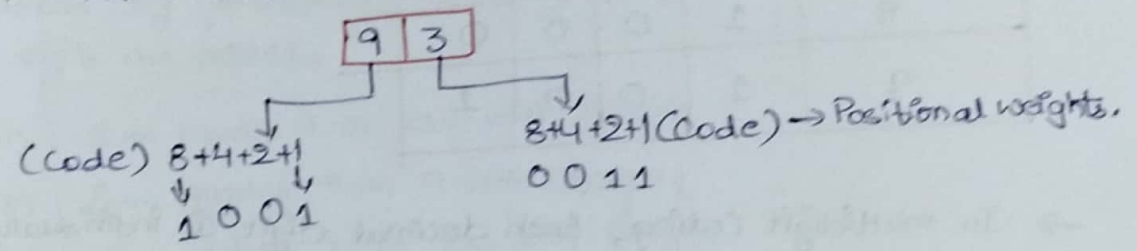
Ex: For a decimal number 567; the weight of 5 is 100; 6 is 10 and 7 is 1.

→ In Weighted Binary Code each bit has a weight 8,4,2,1 and

→ Each decimal digit is represented by a group of four bits.

Ex:- Decimal number 93.

For decimal number 93



The Binary code is (10010011)

*BCD (Binary Coded Decimal) Codes :-

→ BCD is a numeric code in which each digit of a decimal-number is represented by a separate group of bits.

→ The most common BCD code is 8421 BCD; in which each decimal digit is represented by a 4-bit binary number.

→ It is called 8-4-2-1 BCD because the weights associated with 4 bits are 8-4-2-1 from left to right.

→ This means that, bit-3 has weight 8, bit-2 has weight 4, bit-1 has weight 2 and bit-0 has weight 1.

Decimal Digit	BCD Code 8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

→ In multidigit coding, each decimal digit is individually coded with 8-4-2-1 BCD code.

Ex:- 58 in decimal can be encoded in 8-4-2-1 BCD as,

5	8	→ Decimal
0101	1000	→ BCD(8-4-2-1)

Here, each decimal digit requires 4-bits.

∴ Total 8 bits are required to encode $(58)_{10}$ in 8-4-2-1 BCD.

→ If we represent in Binary: $(111010)_2$ i.e only 6 bits are required.

*→ This means that representing numbers in 8-4-2-1 BCD is less efficient than pure binary number system.

*→ The Advantage of a BCD code is that it is easy to convert between BCD and decimal.

*→ The Arithmetic operations of BCD(8-4-2-1) are more complex than they are in pure Binary.

→ The combinations 1010 to 1111 i.e; 6 combinations are not used.

Ex:- $(4)_{10} \xrightarrow{BCD} 0100$

$(27)_{10} \xrightarrow{BCD} \underbrace{0010}_{2}\ \underbrace{0111}_{7}$

$(3805)_{10} \xrightarrow{BCD} 0011100000000101$

*BCD ADDITION :-

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

Case(1) : Sum Equals 9 or less with carry 0

Case(2) : Sum greater than 9 with Carry 0

Case(3) : Sum Equals 9 or less with Carry 1.

*Case(1) : Sum Equals 9 or less with Carry 0.

let us consider addition of 3 and 6 in BCD.

```
 6  →  0110 → BCD for 6
+3  →  0011 → BCD for 3
---------------------
 9     01001 → BCD for 9
       ↑          └→ Valid BCD Number
     Carry
```

let us consider addition of 5 and 2 in BCD

```
 5  → 0101 → BCD for 5
+2  → 0010 → BCD for 2
---------------------
 7    00111 → BCD for 7
      ↑          └→ Valid BCD Number
    Carry
```

*Case 2 :- Sum greater than 9 with Carry 0.

let us Consider addition of 6 and 8 in BCD.

```
  6 BCD→   0110 → BCD for 6
+ 8 BCD→   1000 → BCD for 3
---------------------------
  14       01110 → Invalid BCD number (1110) > 9
           ↑
         Carry
```

The Sum 1110 is an invalid BCD number.

→ This has occured because the sum of the two digits Exceeds 9.

→ Whenever this occurs the Sum has to be Corrected by the addition of six (0110) in the invalid BCD number;

```
  6        1110 → Invalid number.
+ 8      + 0110 → Add 6 for Correction
-----    -------
  14      10100
          ↓
        Carry.
          ↓
  0 0010100  → BCD for 14 Decimal
    1   4
```

→ After addition of 6 Carry is produced into the Second-decimal Position.

*Case 3 :- Sum Equals 9 or less with Carry 1.

let us Consider addition of 8 and 9 in BCD.

```
  8        1000 → BCD for 8
+ 9      + 1001 → BCD for 9
-----    -------
  17      10001 ──┐
          ↑       │
        Carry     ├─ Incorrect BCD result
          ↓       │
  0001 0001 ──────┘
    1    1
```

→ In this case, result (00010001) is valid BCD number; but it is incorrect.

→ To get correct BCD result correction factor of 6 has to be added to the least significant digit sum.

```
  8          1000  → BCD for 8
 +9        + 1001  → BCD for 9
 17.      ---------
          0001 0001  → Incorrect BCD result.
        + 0000 0110  → Add 6 for Correction
        ---------
          0001 0111  → BCD for 17.
          1    7
```

* Procedure for BCD Addition :-

By considering above three cases of BCD addition, we can summarize BCD addition as follows:-

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

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

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

(4.) To correct the invalid sum; add 6 $(0110)_2$ to the four-bit sum. If a carry results from this addition; add it to the next higher-order BCD digit.

Ex :- Perform the code conversion :

$(137)_{10} = ?$ NBCD.

Sol :- NBCD = 8421 BCD.

$(137)_{10} = (0001\ 0011\ 0111)$ NBCD

Ex: Perform Each of the following decimal additions in 8-4-2-1 BCD.

(a) 24 + 18

(b) 48 + 58.

Sol:

(a) 24 + 18 = 42.

0 0 1 0 0 1 0 0
+ 0 0 0 1 1 0 0 0
0 0 1 1 1 1 0 0 → Invalid BCD number 1100 > 9
+ 0 1 1 0 → Add 6 for correction
1 1 1 ← (Carry added to next higher digit)
0 1 0 0 0 0 1 0 → BCD for 42.
4 2.

(b) 48 + 58 = 106

0 1 0 0 1 0 0 0
+ 0 1 0 1 1 0 0 0
1 0 1 0 0 0 0 0 → Invalid sum (>9)
→ Add 6 for Correction
0 1 1 0 0 1 1 0 1010 > 9; Add 6 for correction.
1 0 0 0 0 0 1 1 0
Carry. 1 0 6

Ex: Perform Addition of 6428 and 3205.

Sol:

6428 → 0 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0
+ 3205 → 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1
9633.
1 0 0 1 0 1 1 0 0 0 1 1 1 1 0 1 (1101 > 9 Add 6)
(+) 0 1 1 0
1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 1 → BCD for 9633.
9 6 3 3

*Other 4 Bit BCD Codes:

(i) 2-4-2-1 Codes:

2-4-2-1 BCD is another self complementing code or Reflective Code. Its 4-bit code groups are weighted.

→ In this case; the weights are 2-4-2-1, meaning that bit 1 and bit 3 have the same weight (2).

→ It is sometimes reffered to as 2*-4-2-1 code; where the asterisk simply distinguishes one position with weight 2 from the other.

→ Since two positions have same weight, there are two Possibilities for bit patterns that could be used to represent the same decimal digit.

→ But we have to assign only one of the code.

Decimal digit	2-4-2-1 Code
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3.	0 0 1 1
4.	0 1 0 0
5.	1 0 1 1
6.	1 1 0 0
7	1 1 0 1
8	1 1 1 0
9	1 1 1 1

Table for 2-4-2-1 Code.

* Reflective Property for 2421 Code is:

It has reflective or Self Complementing Property.

Decimal	Complement relation	Decimal
0 →	0000 / 1111	← 9
1 →	0001 / 1110	← 8
2 →	0010 / 1101	← 7
3 →	0011 / 1100	← 6
4 →	0100 / 1011	← 5

Reflective Property of 2421 Code.

* Other weighted BCD Codes:-

Decimal	3321	4221	5211	5311	5421	6311	7421
0	0000	0000	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0001	0001	0001
2	0010	0010	0011	0011	0010	0011	0010
3	0011	0011	0110	0100	0011	0100	0011
4	0101	1000	0111	0101	0100	0101	0100
5	1010	0111	1000	1000	1000	0111	0101
6	1100	1100	1001	1001	1001	1000	0110
7	1101	1101	1100	1011	1010	1001	1000
8	1110	1110	1110	1100	1011	1011	1001
9	1111	1111	1111	1101	1100	1100	1010

Ex: Represent $(7)_{10}$ using all the weighted 4-bit BCD codes listed in the above table.

Sol: (i) 3321 Code.

$(7)_{10} = 3+3+2+1$
$\quad\quad\quad\quad\quad\downarrow\ \downarrow\ \downarrow\ \downarrow$
$\quad\quad\quad\quad\quad 1\ 1\ 0\ 1 = (1101)_{3321\ BCD}$

(ii) 4221 Code

$(7)_{10} = 4+2+2+1$
$\quad\quad\quad\downarrow\ \downarrow\ \downarrow\ \downarrow$
$\quad\quad\quad 1\ 0\ 1\ 1 = (1011)_{4221\ BCD}$

(iii) 5211 Code

$(7)_{10} = 5+2+1+1$
$\quad\quad\quad\downarrow\ \downarrow\ \downarrow\ \downarrow$
$\quad\quad\quad 1\ 1\ 0\ 0 = (1100)_{5211\ BCD}.$

(iv) 5311 Code

$(7)_{10} = 5+3+1+1$
$\quad\quad\quad\downarrow\ \downarrow\ \downarrow\ \downarrow$
$\quad\quad\quad 1\ 0\ 1\ 1 = (1011)_{5311\ BCD}.$

(v) 5421 Code

$(7)_{10} = 5+4+0+1$
$\quad\quad\quad\downarrow\ \downarrow\ \downarrow\ \downarrow$
$\quad\quad\quad 1\ 0\ 1\ 0 = (1010)_{5421\ BCD}$

(vi) 6311 Code

$(7)_{10} = 6+3+1+1$
$\quad\quad\quad\downarrow\ \downarrow\ \downarrow\ \downarrow$
$\quad\quad\quad 1\ 0\ 0\ 1 = (1001)_{6311\ BCD}$

(vii) 7421 Code

$(7)_{10} = 7+4+2+1$
$\quad\quad\quad\downarrow\ \downarrow\ \downarrow\ \downarrow$
$\quad\quad\quad 1\ 0\ 0\ 0 = (1000)_{7421\ BCD}.$

* **Non Weighted Codes:**

These codes are not assigned with any weight to each digit position. Excess-3 and Gray codes are non-weighted codes.

* Excess-3 Code :

Excess-3 code is a modified form of BCD number.

→ The Excess-3 code can be derived from the natural BCD code by adding 3 to each coded number.

Ex :- Decimal number 12 can be represented in BCD as 0001 0010.

Now adding 3 to each digit we get Excess-3 code as, 01000101. (12 in decimal).

```
   0001 0010
(+) 0011 0011 → Excess 3
    ---------
    0100 0101
```

→ It is a non weighted code.

→ The table below shows Excess-3 codes to represent single decimal digit.

→ It is sequential code because we get any code word by adding binary 1 to its previous code word in the table below. (Adding 3 ie (0011) to Binary code.

Decimal digit	Excess-3 code
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

The Excess-3 codes has 6 invalid states. ie., 0000, 0001, 0010, 1101, 1110, 1111.

Excess-3 Code.

→ In BCD subtraction we have to compute 9's (or 10's) complement of number before subtraction.

→ In Excess-3 Code we get 9's Complement of a number by just Complementing Each bit.

→ Due to this Excess-3 Code is Called Self Complementing Code or Reflective Code.

Ex:- Find the Excess-3 Code and its 9's Complement for following decimal numbers.

(a) 592 (b) 403.

Sol:- By above table.

(a) $(592)_{10}$ = 1000 (5) 11000101 (9, 2)

Complement of Each bit we get 9's Complement.

$(592)_{10}$ – 9's Complement → 0111001110 10

(b) $(403)_{10}$ = 0111 (4) 00110110 (0, 3)

Complement of Each bit we get 9's Complement.

$(403)_{10}$ – 9's Complement = 1000 11001001.

* Excess-3 Addition:

To Perform Excess-3 addition we have to,

→ Add two Excess-3 numbers using binary addition.

→ If Carry

⇒ 1 → Add 3 $(0011)_2$ to the sum of two digits.

⇒ 0 → Subtract 3 $(0011)_2$ from the Sum.

Ex:- Perform Excess-3 addition of (a) 8, 6. (b) 1, 2.

Sol:-

8 → 1011 → Excess-3 for 8

+6 → (+) 1001 → Excess-3 for 6

14

10100 Carry = 1; So, Add Excess-3 to sum of two digits

Carry (+) 0011 → Excess-3 Addition

00010111

0011 0000

0100 0111 → 0100 (1) 0111 (4) ⇒ 14.

(b) 1, 2.

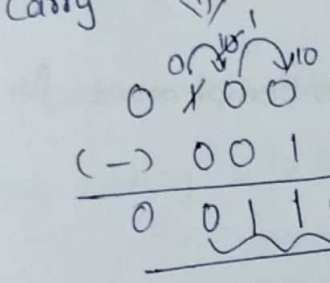
$$\begin{array}{r} 1 \\ +2 \\ \hline 3 \end{array}$$

→ 0100 → Excess-3 for 1

(+) 0101 → Excess-3 for 2

01001 → Carry = 0, so Subtract (0011) from sum.

Carry ↑



(−) 0011 → Excess-3 Subtraction

0 0 1 1 0 ⇒ 3 in decimal result.

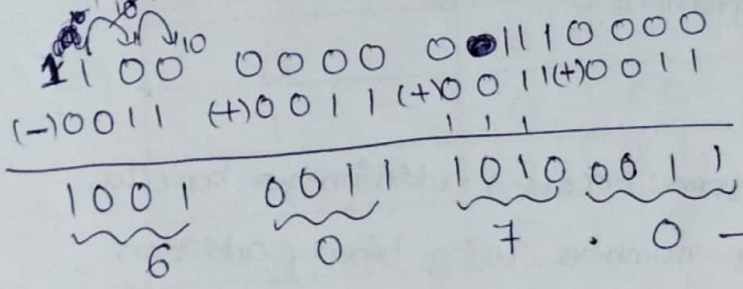
Ex: Perform Excess-3 Addition for 247.6 + 359.4.

Sol:

247.6	0 1 0 1 0 1 1 1 1 0 1 0 . 1 0 0 1
359.4	(+) 0 1 1 0 1 0 0 0 1 1 0 0 . 0 1 1 1
607.0	1 1 0 0 0 0 0 0 0 . 1 1 1 . 0 0 0 0

If Carry is regenerated add 3 or Subtract 3 if carry is 0.

1100.00000111 0000 → Result



→ Excess-3 value

Excess-3 Solution is 607.0//-

* Gray Codes:

Gray Codes is non weighted code and is a special case of "unit-distance Code".

→ In unit distance Code, bit patterns for two Consecutive numbers "differ in only one bit Position."

→ These codes are also called "Cyclic Codes."

→ The bit pattern assigned for gray code from decimal 0 to 15 is shown below.

Decimal Code	Gray code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Gray Code.

→ In the table; for gray code any two adjacent code groups differ only in one bit Position.

→ It has "Reflective - Property".

→ The gray code is also called "reflected code"

NOTE:
Two least significant LSB bits for 4_{10} through 7_{10} are the mirror images of those for 0_{10} through 3_{10}.

→ Similarly, 3 LSB bits for 8_{10} through 15_{10} are mirror images of those for 0_{10} through 7_{10}.

→ In general, the n least significant bits for 2^n through $2^{n+1}-1$ are the mirror images of those for 0 through 2^n-1.

*Relation with Decimal Number :-

→ Another Property of gray code is that gray Coded number Corresponding to the decimal number $2^n - 1$; for any n; differs from gray Coded 0 (0000) in one bit position only.

→ The n-1 indicate the bit position that differs from gray-code (0000).

→ The table below shows the change in single bit position for different values of n.

→ This property places the gray code for the largest n-bit binary number at unit distance from 0.

n	Decimal number $(2^n - 1)$	Gray code 3 2 1 0	Bit Position
1	$2^1 - 1 = (1)_{10}$	0 0 0 1	n-1 = 1-1 = 0 only bit in bit 0 Position differs
2	$2^2 - 1 = (3)_{10}$	0 0 1 0	n-1 = 2-1 = 1 only bit in bit 1 Position differs
3	$2^3 - 1 = (7)_{10}$	0 1 0 0	n-1 = 3-1 = 2 only bit in bit 2 Position differs.
4	$2^4 - 1 = (15)_{10}$	1 0 0 0	only bit in bit 3 Position differs n-1 = 4-1 = 3

Relation of gray code with decimal number.

*Gray to Binary Conversion :-

steps for Gray to Binary Conversion.

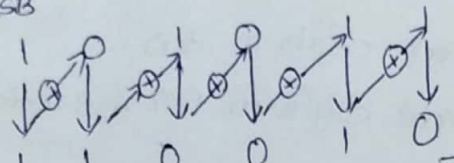
(1.) → The MSB bit of binary number is the same as the MSB of gray code number. So, write down MSB as it is.

(2.) → To obtain the next binary digit, perform an Exclusive-OR-Operation between the bit just written down and the next gray code bit. write down the result.

(3.) → Repeat step 2 until all gray code bits have been Exclusive-ORed with binary bits.

Ex:- Convert gray code 101011 into its Binary Equivalent.

Sol:- Gray Code :- 1 0 1 0 1 1 (MSB ... LSB)



Binary Code : 1 1 0 0 1 0

$\therefore (101011)_{Gray} = (110010)_2$.

Truth Table for XOR.

A	B	A ⊕ B (Ex-OR)
0	0	0
0	1	1
1	0	1
1	1	0

*Binary to Gray Code Conversion :-

Step for Binary to Gray Code Conversion.

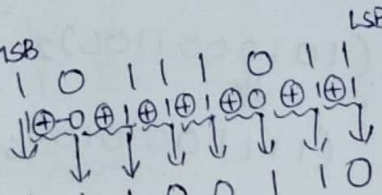
(1.) → MSB of gray code is same as MSB of Binary. So, write down MSB as it is.

(2) → To obtain the next gray digit, perform an Exclusive-OR operation between previous and current binary bit. Write down the result.

(3) → Repeat step 2 until all binary bits have been Exclusive-ORed with their previous one's.

Ex: Convert 10111011 in binary into its Equivalent gray code.

Sol: Binary : 1 0 1 1 1 0 1 1 (MSB ... LSB)

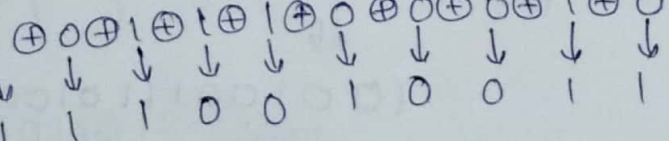


Gray Code : 1 1 1 0 0 1 1 0

$\therefore (10111011)_2 = (11100110)_{gray}$.

Ex:- Convert binary 1011100010 to its gray code.

Sol: Binary : 1 0 1 1 1 0 0 0 1 0



Gray Code : 1 1 1 0 0 1 0 0 1 1

$\therefore (1011100010)_2 = (1110010011)_{gray}$.

Ex: The state of a 12 bit register is 100010010111. What is its contents if it represents.

(a) 3 decimal digits in BCD.

(b) 3 decimal digits in Excess-3 code.

Sol:- (a) 1000 | 1001 | 0111
8 9 7

$(897)_{10}$

(b) 1000 | 1001 | 0111 → Excess-3 code.
5 6 5

$(565)_{10}$

Ex: Convert the following into Gray number

(i) $(527)_8$ (ii) $(652)_{10}$ (iii) $(3A7)_{16}$.

Sol:- (i) $(527)_8$

$\Rightarrow (101010111)_2$ → Binary.

$(111111100)_{Gray}$ → Gray code.

(ii) $(652)_{10}$.

2 | 652
2 | 326 - 0
2 | 163 - 0
2 | 81 - 1
2 | 40 - 1
2 | 20 - 0
2 | 10 - 0
2 | 5 - 0
2 | 2 - 1
1 - 0

$\Rightarrow (1010001100)_2$

$(1111001010)_{Gray}$

(iii) $(3A7)_{16} \Rightarrow (001110100111)_2$

$(001001110100)_{Gray}$.

Ex: Convert gray number 10110010 into
(i) Hexa (ii) octal (iii) Decimal

Sol:- $(10110010)_{Gray}$.

(i) $(11011100)_2 \Rightarrow (DC)_{16}$.

(ii) $(11011100)_2 = (334)_8$

(iii) $\overset{2^7}{1}\overset{2^6}{1}\overset{2^5}{0}\overset{2^4}{1}\overset{2^3}{1}\overset{2^2}{1}\overset{2^1}{0}\overset{2^0}{0}$ $= (220)_{10}$

$2^7\times1 + 2^6\times1 + 2^5\times0 + 2^4\times1 + 2^3\times1 + 2^2\times1 + 2^1\times0 + 2^0\times0 = 220$.

Ex: Convert the following Binary numbers to Gray code.
(a) 1011 (b) 100001 (c) 110110010

Sol:- (a) ~~Binary~~

(a) 1011 → Binary

1 ⊕ 0 ⊕ 1 ⊕ 1
↓ ↓ ↓ ↓
1 1 1 0

(1110)Gray.

(b) 100001 → Binary

1 ⊕ 0 ⊕ 0 ⊕ 0 ⊕ 0 ⊕ 1
↓ ↓ ↓ ↓ ↓ ↓
1 1 0 0 0 1

(110001)Gray.

(c) 110110010 → Binary

1 ⊕ 1 ⊕ 0 ⊕ 1 ⊕ 1 ⊕ 0 ⊕ 0 ⊕ 1 ⊕ 0
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
1 0 1 1 0 1 0 1 1

(101101011)Gray.

* Alphanumeric Codes :

→ In order to Communicate, we need not only numbers, but also letters and other symbols commonly known as non-numeric data.

→ Computer manipulates both numbers and symbols.

→ Most of the programs written by Computer users are in the form of characters. ie set of symbols consists of letters, digits, and various special characters such as +, −, >, <, & . . . .

→ The codes which consists of both numbers and alphabetic characters are called Alphanumeric Codes.

→ The most commonly used Alphanumeric codes are :

- ASCII (American standard Code for Information Interchange)
- EBCDIC (Extended Binary Coded Decimal Interchange Code).

→ These two codes consists of symbols to represent :

- → 26 alphabets with Capital and Small letters
- → Numbers from 0 to 9
- → Punctuation marks and other symbols.

* ASCII :

One standardized alphanumeric code, called ASCII, is the most widely used type.

→ It is a seven-bit code in which the decimal digits are represented by the BCD code preceded by 011.

→ Since it is a 7-bit code; it represent $2^7 = 128$ symbols.

→ The letter of the alphabet and other symbols and instructions are represented by other coded combinations as shown in the Table.

→ for Example: ASCII Code for $N = (4E)_H = (1001110)_2$.

1.6.6.1 ASCII

One standardized alphanumeric code, called the American Standard Code for Information Interchange, is perhaps the most widely used type. It is a seven-bit code in which the decimal digits are represented by the BCD code preceded by 011. Since it is a 7-bit code, it represent $2^7 = 128$ symbols. The letter of the alphabet and other symbols and instructions are represented by other code combination as shown in Table 1.14. For example, ASCII code for $N = (4E)_H = (1001110)_2$.

LSBs	MSBs							
	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000 (0)	NUL	DLE	SP	0	@	P	`	p
0001 (1)	SOH	DC_1	!	1	A	Q	a	q
0010 (2)	STX	DC_2	"	2	B	R	b	r
0011 (3)	ETX	DC_3	#	3	C	S	c	s
0100 (4)	EOT	DC_4	$	4	D	T	d	t
0101 (5)	ENQ	NAK	%	5	E	U	e	u
0110 (6)	ACK	SYN	&	6	F	V	f	v
0111 (7)	BEL	ETB	'	7	G	W	g	w
1000 (8)	BS	CAN	(	8	H	X	h	x
1001 (9)	HT	EM	)	9	I	Y	i	y
1010 (A)	LF	SUB	*	:	J	Z	j	~z
1011 (B)	VI	ESC	+	;	K	[	k	{
1100 (C)	FF	FS	,	<	L	\	l	|
1101 (D)	CR	GS	-	=	M	J	m	}
1110 (E)	SO	RS	.	>	N	↑	n	~
1111 (F)	SI	US	/	?	O	←	o	DEL

Table 1.14 American standard code for information interchange

Definition of control abbreviations :

ACK	Acknowledge
BEL	Bell
BS	Backspace
CAN	Cancel
CR	Carriage return
DC_1-DC_4	Direct control
DEL	Delete idle
DLE	Data link escape
FS	Form separator
GS	Group separator
HT	Horizontal tab
LF	Line feed
NAK	Negative acknowledge
NUL	Null
RS	Record separator
SI	Shift in

EM	End of medium	SO	Shift out
ENQ	Enquiry	SOH	Start of heading
EOT	End of transmission	STX	Start text
ESC	Escape	SUB	Substitute
ETB	End of transmission block	SYN	Synchronous idle
ETX	End text	US	Unit separator
FF	Form feed	VT	Vertical tab

Note : The hexadecimal digit representing each bit pattern is shown in parentheses.

1.6.6.2 EBCDIC

Another alphanumeric code also frequently encountered is called the **Extended Binary Coded Decimal Interchange Code (EBCDIC)**. It is an 8-bit code in which both uppercase and lowercase letters are represented in addition to numerous other symbols and commands as shown in Table 1.15.

LSB	Most Significant Byte (MSB)															
	0000 (0)	0001 (1)	0010 (2)	0011 (3)	0100 (4)	0101 (5)	0110 (6)	0111 (7)	1000 (8)	1001 (9)	1010 (A)	1011 (B)	1100 (C)	1101 (D)	1110 (E)	1111 (F)
0000 (0)	NUL		DS		SP	&	-									0
0001 (1)			SOS				/		a	j			A	J		1
0010 (2)			FS						b	k	s		B	K	S	2
0011 (3)		TM							c	l	t		C	L	T	3
0100 (4)	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101 (5)	HT	NL	LF	RS					e	n	v		E	N	V	5
0110 (6)	LC	BS	EOB	UC					f	o	w		F	O	W	6
0111 (7)	DL	IL	PRE	EOT					g	p	x		G	P	X	7
1000 (8)									h	q	y		H	Q	Y	8
1001 (9)									i	r	z		I	R	Z	9
1010 (A)		CC	SM		¢	!		:								
1011 (B)						$	.	#								
1100 (C)					<	*	%	@								
1101 (D)					(	)	_	'								
1110 (E)					+	;	>	=								
1111 (F)	CU1	CU2	CU3		|		?	"								

Table 1.15 Partial EBCDIC table

Definitions of control abbreviations and symbols

BS	: Backspace	LC	: Lowercase		
BYP	: Bypass	LF	: Line feed	|	Vertical bar : logical OR
CC	: Cursor control	NL	: New line	¬	Logical NOT
CU1	: Customer use	PF	: Punch off	-	Hypen or minus sign
CU2	: Customer use	PN	: Punch on	–	Underscore (01101101)
CU3	: Customer use	PRE	: Prefix		
DL	: Delete	RES	: Restore		
DS	: Digit select	RS	: Reader stop		
EOB	: End of block	SM	: Set Mode		
EOT	: End of transmission	SP	: Space		
FS	: Field separator	TM	: Tape mark		
HT	: Horizontal tab	UC	: Uppercase		
IL	: Idle				

Example 1.39 : *Obtain the EBCDIC code for letter 'a' using Table 1.15.*

Solution :

	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	
a =	1	0	0	0	0	0	0	1	= 81H

Example 1.40 : *Write the ASCII code for 'ELECTRONICS'.*

Solution :

45, 4C, 45, 43, 54, 52, 4F, 4E, 49, 43, 53 i.e.,

1000101 1001100 1000101 1000011 1010100 1010010

1001111 1001110 1001001 1000011 1010011

Example 1.41 : *Encode the word 'BINARY' in ASCII form.*

Solution :

42, 49, 4E, 41, 52, 59 i.e.,

1000010 1001001 1001110 1000001 1010010 1011001

*Error Detection using Parity Bit and Error Correction:

When binary data is transmitted and processed, it is susceptible to noise that can alter or distort its contents. The 1's may get changed to 0's and 0's to 1's.

*Error Detecting Codes:

A parity bit is used for the purpose of detecting errors during transmission of binary information.

→ A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even.

→ The message, including the parity bit is transmitted and then checked at the receiving end for errors.

→ An error is detected if the checked parity doesnot correspond with the one transmitted.

→ The circuit that generates parity bit in the transmitter is called a parity generator and

→ The circuit that checks the parity in the receiver is called a parity checker.

There are two types of parity → Even, odd

→ for odd parity, the parity bit is set to 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.

→ for Even parity, the parity bit is set to 0 or 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an Even number.

→ Odd and Even Parity in 8421 BCD Code.

Decimal	8 4 2 1	Odd Parity	Even Parity
0	0 0 0 0	1	0
1	0 0 0 1	0	1
2	0 0 1 0	0	1
3	0 0 1 1	1	0
4	0 1 0 0	0	1
5	0 1 0 1	1	0
6	0 1 1 0	1	0
7	0 1 1 1	0	1
8	1 0 0 0	0	1
9	1 0 0 1	1	0

Ex: Write a ASCII Code for alphabet 'A' with an Odd Parity. place Parity bit in the most significant Position.

Sol: The 7-bit ASCII Code for the Alphabet 'A' is 1000001. This requires the addition of a 1 in the MSB place to give Odd Parity,

Added Parity Bit → 1 1000001.

Ex: In an Even Parity Scheme; which of the following words contain an Error.

(a) 10101010 (b) 11110110 (c) 10111001

Sol:- (a) 10101010
No. of 1's are Even
∴ No Error

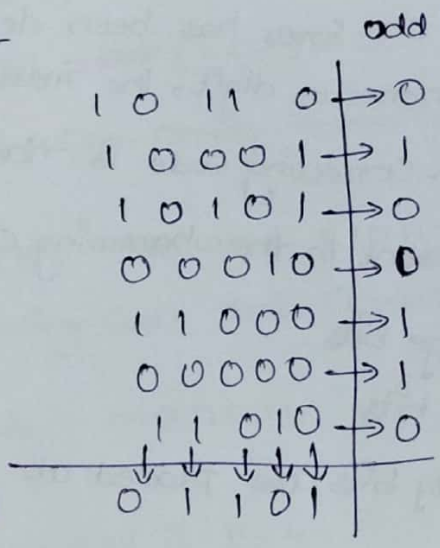
(b) 11110110
No. of 1's are Even
∴ No Error

(c) 10111001
No. of 1's are odd.
∴ Error Exists.

Ex :- The block of data shown in table is to stored on a magnetic tape. Create row and column parity bits for the data using odd Parity.

1	0 1 1 0
1	0 0 0 1
1	0 1 0 1
0	0 0 1 0
1	1 0 0 0
0	0 0 0 0
1	1 0 1 0 .

Sol:- Given :-



Ex: In the odd parity scheme; which of the following words contail Error.

(a) 10110111 (b) 10011010 (c) 11101010

Sol:- (a) 10110111

Even no. of 1's

∴ Error is present

(b) 10011010

Even number of 1's

∴ Error is present

(c) 11101010

Odd number of 1's

∴ No Error.

* Error Correcting Codes:

A code is said to be an Error correcting code, if the correct code word can always be deduced from an erroneous word.

→ A code with minimum distance of three cannot only correct single-bit errors, but also detect two-bit errors.

→ the key to Error correction is that it must be possible to detect and locate erroneous digits.

→ If the location of an error has been determined, then by complementing the erroneous digit, the message can be corrected.

→ one type of Error-Correcting code is Hamming code.

→ The total bits present in the hamming code are $(m+p)$ bits

where, $m \rightarrow$ message bits

$p \rightarrow$ Parity bits

→ Always the parity bits are placed at locations 2^{k-1} ; $k=1,2\cdots$

Ex: Encode data bits 1101 into the 7 bit Even parity Hamming code.

Sol: Given data = 1101

$\therefore m=4$

To find Parity bits; $2^p \geq m+p+1$

If $p=1$;

$2^1 \geq 4+1+1$

$2 \geq 6$, not satisfied.

If $p=2$;

$2^2 \geq 4+2+1 \Rightarrow 4 \geq 7$; not satisfied.

If $p=3$

$2^3 \geq 4+3+1 \Rightarrow 8 \geq 8$; Satisfied.

$\therefore p=3$

-Always Parity bits are located at 2^{k-1} ; $k=1,2\ldots$

Total bits in Hamming Code = $m+P = 4+3 = 7$ bits

1	2	3	4	5	6	7
001	010	011	100	101	110	111
P_1	P_2	1	P_4	1	0	1

P_1 = [redacted] $(3,5,7)$ = [redacted] $(1,1,1) = 1$ (First bit 1's in the Code consider.)

P_2 = [redacted] $(3,6,7)$ = [redacted] $(1,0,1) = 0$ (2^{nd} bit 1's)

P_4 = [redacted] $(5,6,7)$ = [redacted] $(1,0,1) = 0$ (for Even Parity)

∴ The 7 bit Even parity Hamming Code is 1010101.

Ex: Given a 8 bit data word 01011011. Generate a 12 bit Composite Hamming Code that Corrects and detects single Errors.

Sol: Given; Data = 01011011

$m = 8$

Parity bits required is $P = 4$.

Total bits in Hamming Code = $8+4 = 12$.

$2^P \geq m+P+1$

If $P=1$

$2 \geq 8+1+1$, not satisfied

If $P=3$

$8 \geq 3+8+1$; not satisfied

If $P=4$

$16 \geq 8+4+1$; satisfied

∴ $P = 4$.

1	2	3	4	5	6	7	8	9	10	11	12
0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100
P_1	P_2	0	P_4	1	0	1	P_8	1	0	[redacted]	1

$P_1 = (3,5,7,9,11) = (0,1,1,1,1) = 0$

$P_2 = (3,6,7,10,11) = (0,0,1,0,1) = 0$

$P_3 = (5,6,7,12) = (1,0,1,1) = 1$

$P_8 = (9,10,11,12) = (1,0,1,1) = 1$

∴ The 12-bit Hamming Code is 000110111011.

Ex: A 12bit Hamming Code word containing 8 bits of data and 4 Parity bits is read from Memory. What is the original 8 bit word if the 12 bit read out as follows?

(i) 100011101010 (ii) 101110000110 (iii) 101111110000.

Sol: (i) 1 2 3 4 5 6 7 8 9 10 11 12
1 0 0 0 1 1 1 0 1 0 1 0
2^0 2^1 2^2 2^3
P_1 P_2 P_3 P_4 Powers of 2 & Parity.

8 bit word : 01111010

(ii) 1 2 3 4 5 6 7 8 9 10 11 12
1 0 1 1 1 0 0 0 0 1 1 0
2^0 2^1 2^2 2^3

8 bit word : 11000110

(iii) 1 2 3 4 5 6 7 8 9 10 11 12
1 0 1 1 1 1 1 1 0 0 0 0
2^0 2^1 2^2 2^3

8bit word :- 11110000

Ex: Detect and Correct Errors; if any in the Even parity Hamming Code words and write the correct Code.

(i) 1100110 (ii) 0111110

Sol:- (i) 1100110

1	2	3	4	5	6	7
001	010	011	100	101	110	111
2^0	1	0	0	1	1	0
	2^1		2^2			

$P_1 = (1,3,5,7) = (1,0,1,0) = 0$

$P_2 = (2,3,6,7) = (1,0,1,0) = 0$

$P_3 = (4,5,6,7) = (0,1,1,0) = 0$

∴ NO Error Present

(ii) 0111110

1	2	3	4	5	6	7
001	010	011	100	101	110	111
2^0 0	2^1 1	1	2^2 1	1	1	0

$P_1 = (1,3,6,7) = (0,1,1,0) = 0$; $P_2 = (2,3,6,7) = (1,1,1,0) = 1$

$P_3 = (4,5,6,7) = (1,1,1,0) = 1$ ↑ Bottom to Top

(010) → Error Position . ie Error is in 3rd Position; Replace 1 with 0

∴ The Corrected Hamming Code is 0101110.

* Binary storage and Registers:

In digital computers, the binary information is stored using Binary Cells. A binary cell is capable of storing one bit of information.

→ The cell can have two possible states; Either logic 1 or logic 0.
→ When cell is in logic 1 state, the information stored in it is 1.
→ If it is 0; then the information stored in it is 0.

* Registers :-

A register is a group of binary cells.

→ A register with 'n' binary cells can store n-bit information.
→ We know; Each bit in register can have Either 0 or 1 value.
∴ 16 bit register can store one of 2^{16} possible numbers.
→ The information stored in the register may be interpreted differently for different types of data.

Example : If register is used to store the number represented in Excess-3 code; the number has different interpretation when same number is treated as binary number.

* Register Transfer :

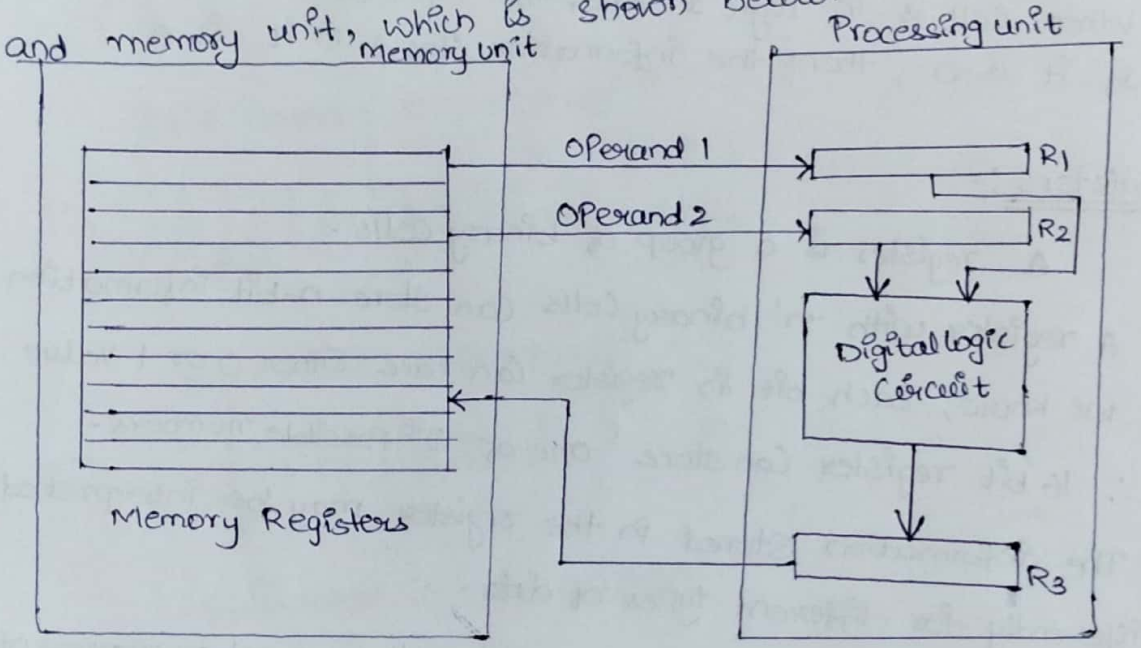
The Registers are Part of digital system.

→ The Processing unit in the digital system gets the information to be processed from registers.
→ The processed information is again stored in the register.
Sometimes it is necessary to transfer information from one register to another register. Such operation is known as register transfer operation.
→ Usually, the information is stored in the memory.

→ At the time of processing it is brought into the registers so that processing unit gets access to it.

→ The processed information available in the registers is stored into the memory.

→ The information is transferred between processing unit, registers and memory unit, which is shown below.



Binary Information processing Blocks.

* Binary Logic :

Binary logic operates on variables that take two discrete values and operations that assume logic meanings.

→ The binary variables are designated by alphabets such as A, B, C, D, P, Q, x, y, z ...

→ The operations that have logic meaning are called logical-operators.

* Logical operators :

We know that to represent and solve arithmetic Expressions we use arithmetic operators such as +, −, × and ÷.

→ Similarly, we can use logical operators to represent and solve logical expressions. There are 3 basic logical operators :→ NOT/INVERT.
↳ AND
↳ OR.

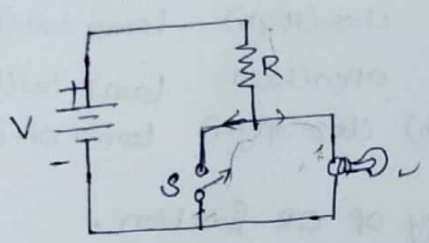
* Logical Operator NOT / INVERT :

The inversion (or complementing or Negation) operator is written as a bar over its argument.

→ Sometimes it is written "NOT"

→ Thus, the inverse of A is $\overline{A}$ or NOT A.

→ The logic operator NOT can be better understood if the switching circuit realization of NOT function as shown below is considered.



Input	output
Switch open (low)	lamp on (High)
Switch close (High)	lamp OFF (low)

→ When the switch is open; lamp is 'ON' and when the switch is close lamp is "off".

* Logical Operator AND.

→ It is denoted by 'x' or '.'.

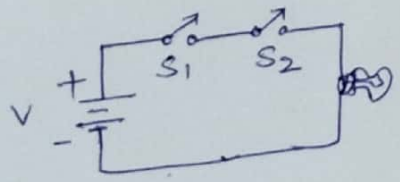
→ These signs may be omitted, similar to ordinary algebra.

Ex: A · B; A×B (or) AB has same meaning.

→ A·B may be read as 'A' and 'B' or 'A' times 'B'.

→ A·B is high if A and B are both high and otherwise low.

→ The logic operator AND can be better understood if the switching-circuit realization of AND function as shown below is considered. AND gate denotes lowest value of input variables.



Input S_1	Input S_2	output
open (low)	open (low)	lamp off (low)
open (low)	close (High)	lamp off (low)
close (High)	open (low)	lamp Off (Low)
close (High)	close (High)	lamp ON (High)

Switching circuit Analogy of AND function.

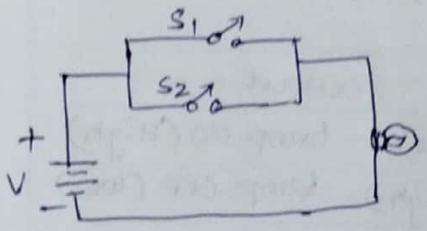
*Logical operator OR :-

→ It is written '+'. 'A' + 'B' is read as A or B.

→ A+B is high if either A is high or B is high or both are high.

→ The switching circuit realization of OR function is shown below.

→ OR gate denotes highest value of input variable.



Input		Output
S_1	S_2	
open (low)	open (low)	lamp off (low)
open (low)	close (High)	lamp ON (High)
close (High)	open (low)	lamp ON (High)
close (High)	close (High)	lamp ON (High)

Switching circuit Analogy of OR function.

→ If either or both of two seperate switches S_1 OR S_2 are closed; the lamp will be ON.

→ only when both S_1 and S_2 are off, the lamp will be off.

* BOOLEAN ALGEBRA AND LOGIC GATES *

In 1854, George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system now called Boolean Algebra.

→ The Boolean Algebra is defined with a set of Elements, set of operators and number of rules, laws, theorems & Postulates.

→ Boolean algebra is formulated by a defined set of Elements, together with two binary operators, + and .

→ Associative law:

law 1: $A+(B+C)=(A+B)+C$.

law 2: $(AB)C = A(BC)$

law

→ Commutative law:

law 1: $A+B=B+A$

law 2: $AB=BA$

→ Identity Element:

$$a+0=0+a=a$$

→ Distributive law:

$$A\Delta(B*C)=(A\Delta B)*(A\Delta C)$$

law: $A(B+C)=AB+AC$.

* Axiomatic Definition of Boolean Algebra :-

→ Closure (a): Closure with respect to the operator '+' when two binary Elements are operated by operator '+' the result is a unique binary Element.

→ closure (b): Closure with respect to the operator '.' (dot) when the two binary Elements are operated by operator '.' result is a unique binary Element.

→ An identity element with respect to + ; designated by 0 :

$$A+0 = 0+A = A.$$

→ An identity element with respect to · , designated by 1 :

$$A \cdot 1 = 1 \cdot A = A.$$

→ Commutative → with respect to + : $A+B = B+A$
→ with respect to · : $A \cdot B = B \cdot A$

→ Distributive property of · over + :

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

→ Distributive property of + over · :

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

→ For every binary element, there exists complement element.

Ex: If A is an element; we have $\bar{A}$ is a complement of A.

ie:, If $A=0$; $\bar{A}=1$

If $A=1$; $\bar{A}=0$.

Idempotency: $A+A=A$
$A \cdot A = A$.

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

*Basic Theorems and Properties of Boolean Algebra:

*Duality:

. The principle of duality theorem says that, starting with a Boolean relation; you can derive another Boolean relation by,

(1.) Changing each OR sign to an AND sign

(2.) Changing each AND sign to an OR sign and

(3). Complementing any 0 or 1 appearing in the expression.

Ex:- Dual Relation $A+\bar{A}=1$ is $A \cdot \bar{A}=0$

Duality is a very important property of Boolean Algebra.

* Basic Theorems :-

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

$0 + 0 = 0$
$1 + 1 = 1$ $\Rightarrow A + A = A$ $\left[\because A \cdot 1 = A,\ A + \bar{A} = 1,\ A \cdot \bar{A} = 0 \right]$

Proof :- $A + A = (A + A) \cdot 1$

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

$= A + A\bar{A}$

$= A + 0$

$= A$

Theorem (1(b)) :- $A \cdot A = A$

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

Proof : $A \cdot A = A \cdot A + 0$

$= AA + A\bar{A}$

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

$= A \cdot 1$

$= A$

$(\because A + \bar{A} = 1,\ A \cdot \bar{A} = 0,\ A \cdot 1 = A)$

Theorem 2(a) :- $A + 1 = 1$

$1 + 0 = 1$
$1 + 1 = 1$ $\Rightarrow 1 + A = 1$ or $A + 1 = A$

Proof :- $A + 1 = 1 \cdot (A + 1)$

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

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

$= A + \bar{A}$

$= 1$

$\left[\because A \cdot 1 = A,\ A + \bar{A} = 1,\ A + BC = (A + B)(A + C) \right]$

* **Theorem 2(b):** $A \cdot 0 = 0$

$0 \cdot 0 = 0 \Rightarrow 0 \cdot A = 0$ or $A \cdot 0 = 0$

$0 \cdot 1 = 0$

Proof: $A \cdot 0 = 0$ (Duality)

* **Theorem 3:** $\bar{\bar{A}} = A$

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

$\bar{\bar{1}} = 1$

Proof :- Complement of $\bar{A}$ is A and also $\bar{\bar{A}}$

* **Theorem 4(a):** $A + AB = A$

Proof: $A + AB = A \cdot 1 + AB$

$= A(1 + B)$ (Distributive. ($\because A + BC = (A+B)(A+C)$)

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

$= A \cdot 1$

$= A$.

* **Theorem 4(b):** $A(A+B) = A$

Proof: $A(A+B) = A \cdot A + AB$

$= A + AB$

$= A$

* **Theorem 5(a):** $A + \bar{A}B = A + B$

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

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

$= A + B \cdot 1$

$= A + B$

* **Theorem 5(b):** $A \cdot (\bar{A} + B) = AB$

Proof: $A(\bar{A} + B) = (A + AB) \cdot (\bar{A} + B)$

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

$= AB + ABB$

$= AB + AB$

$= AB$.

* Demorgan's Theorems :

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

Truth Table :

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

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

Truth Table :

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

Ex : $(A+B+C)' = A'.B'.C'$

let $A+B = x$

$(x+C)' = x'.C'$

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

$\Rightarrow A'.B'.C'$

* Consensus Theorem :

In simplification of Boolean expression; an expression of the form $AB + \overline{A}C + BC$ the term BC is redundant and can be eliminated to form the Equivalent Expression $AB + \overline{A}C$.

→ The theorem used for this simplification is known as Consensus theorem. And it is stated as,

$$AB + \overline{A}C + BC = AB + \overline{A}C.$$

→ The key to organize Consensus terms is to 1st find a Pair of terms, one of which Contains a Variable and the other contains its Compliment.

→ 3rd term which should Contain the remaining Variables from pair of terms Eliminating Seleded Variable and its Complement.

Proof :- $AB + \overline{A}C + BC = AB + \overline{A}C + (A+\overline{A})BC = AB + \overline{A}C + ABC + \overline{A}BC$

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

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

Ex: Solve the given Expression using Consensus theorem.

$\bar{A}\bar{B} + AC + B\bar{C} + \bar{B}C + AB$.

Sol: Given $\bar{A}\bar{B} + AC + B\bar{C} + \bar{B}C + AB$.

$\bar{A}\bar{B} + AC + \bar{B}\bar{C} + B\bar{C} + AB$

[$\because$ $AB + \bar{A}C + BC = AB + \bar{A}C$ (Theorem)]

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

$\bar{A}\bar{B} + AC + B\bar{C} + AB$

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

$\therefore \bar{A}\bar{B} + AC + B\bar{C}$.

* Dual of Consensus Theorem:

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

Proof: $(A\bar{A} + AC + \bar{A}B + BC)(B+C) = A\bar{A} + AC + \bar{A}B + BC$

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

$ABC + \bar{A}BB + BBC + ACC + \bar{A}BC + BCC = AC + \bar{A}B + BC$

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

$\bar{A}B + BC + AC = AC + \bar{A}B + BC$.

Ex: Solve the following Boolean Expression using dual of Consensus theorem $(A+B)(\bar{A}+C)(B+C)(\bar{A}+D)(B+D)$

Sol:- $(A+B)(\bar{A}+C)(B+C)(\bar{A}+D)(B+D)$

$\Downarrow$

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

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

*Boolean function:

Bodean Expressions are constructed by Connecting the Boolean-Constants and Variables with the Boolean operations.

→ These Bodean Expressions are also known as Boolean formulas.

→ We use Bodean Expressions to describe Boolean functions.

Ex: Boolean Expression $(A+\bar{B})C$ is used to describe function f, then Boolean function is written as,

$$f(A,B,C) = (A+\bar{B})C \quad (or)$$

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

→ Let us Consider four variable Boolean function.

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

(Product terms. / literals.)

→ Each occurrence of a variable in either a complemented or an uncomplemented form is called a literal.

→ Above function Consists of 6 literals.; They appear in product terms.

→ A product term is defined as either a literal or a product of literals.

Ex: $f(A,B,C,D) = (B+\bar{D}) \cdot (A+\bar{B}+C) \cdot (\bar{A}+C)$

(Sum terms. / literals.)

→ The above function consists of 7 literals.

→ They appear in sum terms.

→ Sum term is defined as either a literal or sum of literals.

→ These literals and terms are arranged in one of the two forms:

- Sum of product form (SOP) and
- Product of Sum form (POS).

*Sum Of Product form:

→ The words sum and product are derived from the symbolic representation of the OR and AND functions by + and ·

→ A product term is any group of literals that are ANDed together.

Ex: ABC, XY, UVWX ...

→ A Sum term is any group of literals that are ORed together.

Ex: A+B+C; X+Y; U+V+W.

→ A sum of product (SOP) is a group of product terms ORed together.

Ex: (i) $f(A,B,C) = \underset{\text{Product terms.}}{ABC} + \underset{\text{Product terms.}}{A\bar{B}\bar{C}}$ (Sum)

→ The sum of product form is also known as disjunctive - normal form or formula.

*Product of sum form:

A product of sum is any groups of sum terms ANDed together.

Ex:- (i) $f(P,Q,R,S) = \underset{\text{Sum terms.}}{(P+Q)\cdot(R+\bar{S})\cdot(P+S)}$ (Product)

→ The product of sum form is also known as conjunctive - normal form or formula.

* Canonical and Standard forms:-

The Canonical forms are the special cases of SOP and POS forms.

→ These are also known as Standard SOP and POS forms.

* Standard SOP Form (Or) Minterm Canonical form :

We can realize that in SOP form; all individual terms donot involve all literals.

Ex : $AB + AB\bar{C}$; The first product term donot contain literal C.

→ If each term in SOP form consists all the literals then SOP form is known as standard or canonical SOP form.

→ Each individual term in the standard SOP form is called Minterm.

Ex :- $f(A,B,C) = A\bar{B}C + ABC + \bar{A}B\bar{C}$

↳ Each product term consists of all literals in either complemented form or uncomplemented form.

- Standard SOP form.

* Standard POS form (or) Maxterm Canonical form :

→ If each term in POS form contains all the literals then the POS form is known as standard or canonical POS form.

→ Each individual term in standard POS form is called Maxterm.

Ex: $f(A,B,C) = (A+B+C) \bullet (A+\bar{B}+C)$

↳ Each sum term consists of all literals in either complemented / uncomplemented form.

* Converting Expressions in standard SOP or POS forms :-

Sum of product form can be converted to standard sum-of products by ANDing the terms in the expression with terms formed by ORing the literal and its complement which are not present in that term.

Ex :- For a three literal expression with literals A, B and C ; if there is a term AB; where C is missing; the we form term $(C+\bar{C})$ and AND it with AB.

$\therefore AB(C+\bar{C}) = ABC + AB\bar{C}$.

* Steps to Convert SOP to Standard SOP form :-

(1) Find the missing literal in Each product term if any.

(2) AND Each Product term having missing literals with terms form by ORing the literal and its Complement.

(3) Expand the terms by applying distributive law and Reorder the literals in the product term

(4). Reduce the expression by omitting repeated product terms if any.
Because $A + A = A$.

Ex:- Convert the given Expression in standard SOP form.

$f(A, B, C) = AC + AB + BC$.

Sol:- Step1:- $f(A, B, C) = AC + A\bar{B} + BC$ → literal 'C' missing, literal 'A' missing, literal 'B' missing

Step2:- AND product term with (missing literal + its complement)

$f(A, B, C) = A.C \cdot (B + \bar{B}) + AB(C + \bar{C}) + BC(A + \bar{A})$

Step3:-

$f(A, B, C) = ABC + A\bar{B}C + A\bar{B}C + AB\bar{C} + ABC + \bar{A}BC$. (omit)

Step4: Omit repeated product terms

$f(A, B, C) = ABC + A\bar{B}C + AB\bar{C} + \bar{A}BC$.

Ex:- Convert given Expression in standard SOP form.

$f(A, B, C) = A + ABC$

Sol:- $f(A, B, C) = A + ABC$
↳ BC missing.

$f(A, B, C) = A(B + \bar{B}).(C + \bar{C}) + ABC$.

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

$= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + ABC$ → Omit (Repeated)

$f(A, B, C) = ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C}$.

* Steps to Convert Pos to Standard Pos form :-

Step1: find the missing literals in each sum term if any.

Step2: OR each sum term having missing literals with term/s form by ANDing the literal and its complement.

Step3: Expand the terms by applying distributive law and reorder the literals in the sum terms.

Step4: Reduce the expression by omitting repeated sum terms if any. Because $A \cdot A = A$.

Ex: Convert the given expression in standard Pos form.

$$f(A,B,C) = (A+B)(B+C)(A+C)$$

Sol:- (i) $f(A,B,C) = (A+B)(B+C)(A+C)$

'C' ↓ , ↓ 'A' , ↓ 'B' → missing literals

(ii) $f(A,B,C) = [(A+B)+(C\cdot\bar{C})] \cdot [(B+C)+(A\cdot\bar{A})] \cdot [(A+C)+(B\cdot\bar{B})]$

$\because A+BC = (A+B)(A+C)$; Distributive law.

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

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

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

Ex: Convert the given expression in standard Pos form.

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

Sol:

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

↑ B & C missing

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

(A; B, C)

$\because A+BC = (A+B)(A+C)$

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

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

(A; B, C) (A; B, C)

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

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

* M-NOTATIONS : Minterms and Maxterms :-

→ Each individual term in standard SOP form is called minterm.

→ Each individual term in standard POS form is called maxterm.

→ The number of minterms as well as maxterms are $2^3 = 8$. For table below.

→ In General, for an n-variable logical function there are 2^n minterms and an equal number of maxterms.

Variables			Minterms	Maxterms
A	B	C	m_i	M_i i → Decimal no Equivalent of natural Binary No.
0	0	0	$\bar{A}\bar{B}\bar{C} = m_0$	$A+B+C = M_0$
0	0	1	$\bar{A}\bar{B}C = m_1$	$A+B+\bar{C} = M_1$
0	1	0	$\bar{A}B\bar{C} = m_2$	$A+\bar{B}+C = M_2$
0	1	1	$\bar{A}BC = m_3$	$A+\bar{B}+\bar{C} = M_3$
1	0	0	$A\bar{B}\bar{C} = m_4$	$\bar{A}+B+C = M_4$
1	0	1	$A\bar{B}C = m_5$	$\bar{A}+B+\bar{C} = M_5$
1	1	0	$AB\bar{C} = m_6$	$\bar{A}+\bar{B}+C = M_6$
1	1	1	$ABC = m_7$	$\bar{A}+\bar{B}+\bar{C} = M_7$

$\overline{SOP} = POS$

If, $f(A,B,C,D) = \sum m(0,2,4,6,8,10,12,14)$

Then

$f(A,B,C,D) = \pi M(1,3,5,7,9,11,13,15)$

Minterms and Maxterms for three Variables.

→ With these shorthand notations logical function can be represented as follows :-

(i) $f(A,B,C) = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + AB\bar{C}$

$= m_0 + m_1 + m_3 + m_6$

$= \sum m(0,1,3,6)$

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

$= M_1 + M_3 + M_6$

$= \pi M(1,3,6)$

Where; Σ denotes Sum of Products

π denotes Product of Sum.

Simplification of Boolean functions:-

The Boolean functions can be realized by logic gates.

→ The simplification of Boolean function is very important as it saves the hardware required and hence the cost for design of specific Boolean function.

Ex1:- $A \cdot \bar{A}C = 0 \cdot C$

$= 0$ $\quad [\because A\cdot\bar{A}=0,\ A\cdot 0=0]$

Ex2:- $ABCD + ABD$

$\Rightarrow ABD(C+1)$

$\Rightarrow ABD(1)$ $\quad [\because A+1=1,\ A\cdot 1=A]$

$\Rightarrow ABD$

Ex3:- $ABCD + A\bar{B}CD$

$\Rightarrow ACD(B+\bar{B})$

$\Rightarrow ACD \cdot 1$ $\quad [\because B+\bar{B}=1,\ A\cdot 1=A]$

$\Rightarrow ACD$

Ex4: $A(A+B)$

$\Rightarrow AA + AB$ $\quad [\because A\cdot A=A]$

$\Rightarrow A + AB$

$\Rightarrow A(1+B)$ $\quad [\because 1+B=1]$

$\Rightarrow A$

Ex5:- $AB + ABC + AB(D+E)$

$\Rightarrow AB(1 + \underbrace{C+D+E}_{A})$ $\quad [\because A+1=A]$

$\Rightarrow AB$

Ex6 :- $XY + XYZ + XY\bar{Z} + \bar{X}YZ$

$\Rightarrow XY(1+Z) + XY\bar{Z} + \bar{X}YZ$ $\quad [\because A+1=1]$

$\Rightarrow XY + XY\bar{Z} + \bar{X}YZ$

$\Rightarrow XY(1+\bar{Z}) + \bar{X}YZ$

$\Rightarrow XY + \bar{X}YZ$ $\quad (\because A+\bar{A}B = A+B)$

$\Rightarrow Y(X+\bar{X}Z)$

$\Rightarrow Y(X+\bar{Z})$

Ex7 :- $\bar{A}B\bar{C} + \bar{A}B\bar{C} + \bar{A}BC$

$\Rightarrow \bar{A}\bar{C}(\bar{B}+B) + \bar{A}BC$ $\quad (\because A+\bar{A}=1)$

$\Rightarrow \bar{A}\bar{C} + \bar{A}BC$

$\Rightarrow \bar{A}(\bar{C}+BC)$ $\quad (\because A+\bar{A}B = A+B)$

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

Ex8 :- Prove $ABC + A\bar{B}C + AB\bar{C} = A(C+B)$

LHS :- $\Rightarrow AC(B+\bar{B}) + AB\bar{C}$ $\quad (A\cdot 1 = A)$ $(A+\bar{A}=1)$

$\Rightarrow AC + AB\bar{C}$ $\quad (\because A+\bar{A}B = A+B)$

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

$\Rightarrow A(C+B)$.

Ex9 :- $A + \bar{A}B + A\bar{B} = A+B$

LHS : $\Rightarrow A + \bar{A}B + A\bar{B}$ $\quad (\because A+\bar{A}B = A+B)$

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

$\Rightarrow A+A+B$ $\quad [\because A+A=A]$

$\Rightarrow A+B$

Ex10 :- $\overline{\bar{A}B + \bar{A} + AB}$

$\Rightarrow \overline{\bar{A}+\bar{B}+\bar{A}+AB}$ $\quad [\because \bar{A}\bar{B} = \bar{A}+\bar{B}]$

$\Rightarrow \overline{\bar{A}+\bar{B}+\bar{A}+B}$ $\quad [\because A+\bar{A}B = A+B]$

$\Rightarrow \overline{\bar{A}+1}$ $\quad (\because A+\bar{A}=1)$

$\Rightarrow \bar{1}$ $\quad [\because A+1=1]$

$\Rightarrow 0$.

Ex: Simplify the following three Variable Expression using Boolean Algebra $Y = \Sigma m(1,3,5,7)$

Sol:- Sum of products.

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

$$\Rightarrow \bar{A}C(\bar{B}+B) + AC(\bar{B}+B)$$

$$\Rightarrow \bar{A}C + AC \qquad (\because \bar{A}+A=1)$$

$$\Rightarrow C(\bar{A}+A)$$

$$= C$$

Ex: Reduce the following boolean Expression to three literals.

$$\overline{[(\bar{C}\bar{D})+A)} + A + CD + AB.$$

Sol:- $F = \overline{((\bar{C}\bar{D})+A)} + A + CD + AB$

$$\Rightarrow (\overline{\bar{C}\bar{D}} \cdot \bar{A}) + A + CD + AB$$

$$\Rightarrow \bar{A}CD + A + CD + AB$$

$$\Rightarrow CD(\bar{A}+1) + A(1+B) \qquad [\because 1+\bar{A}=1] \quad [\because 1+B=1]$$

$$\Rightarrow CD + A$$

Ex: find the complement of the function $F_1 = x'y'z' + x'y'z$.

$$F_1 = x'y'z' + x'y'z.$$

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

$$F_1' = (x'y'z')' \cdot (x'y'z)'$$

$$\Rightarrow [(x')' + y' + (z')'] [(x')' + (y')' + z']$$

$$F_1' \Rightarrow (x + y' + z)(x + y + z')$$

Ex: find the Complement of (a) $xy' + x'y$

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

Sol: (a) $xy' + x'y$

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

$\Rightarrow (x'+y)(x+y')$

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

$\Rightarrow [(x'+y+z')(x+y')(x+z)]'$

$\Rightarrow (x'+y+z')' + (x+y')' + (x+z)'$

$\Rightarrow (x')' y' (z')' + x'(y')' + x' \cdot z'$

$\Rightarrow xy'z + x'y + x'z'$.

Ex:- Simplify the following Boolean Expressions to a minimum number of literals.

(a) $x'yz + xz + x'z$

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

(c) $(x+y)'(x'+y')$

Sol:- (a) $x'yz + xz + x'z$

$\Rightarrow x'z(1+y) + xz$

$\Rightarrow x'z + xz$

$\Rightarrow z(x+x')$ [$\because A + \bar{A} = 1$]

$\Rightarrow z$

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

$\Rightarrow \underbrace{xx'}_{0} + \underline{xz'} + x'y' + y'z' + \underline{x'z'} + z'z'$ [$\because A \cdot \bar{A} = 0$]

$\Rightarrow z'(x+x') + x'y' + z'(y'+1)$ [$\because x + \bar{x} = 1$]

$\Rightarrow z' + x'y' + z'$ [$1 + \bar{y} = 1$]

$\Rightarrow x'y' + z'$.

Sol:- (c) $(x+y)'(x'+y')$

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

$\Rightarrow x'y'x' + x'y'y'$

$\Rightarrow x'y' + x'y'$

$\Rightarrow x'y'$ =.

Ex: Prove $(A+\bar{A})(AB+AB\bar{C}) = AB$

Sol:- $(AB+AB\bar{C})$ $[\because A+\bar{A}=1,\ 1+\bar{C}=1]$

$\Rightarrow AB(1+\bar{C})$

$\Rightarrow AB$

Ex:- $(A\bar{B}+A\bar{C})(BC+B\bar{C})(ABC)=0$

$\Rightarrow (A\bar{B}+A\bar{C})\cdot B(C+\bar{C})(ABC)$

$\Rightarrow (A\bar{B}+A\bar{C})\cdot B(ABC)$

$\Rightarrow ABC(A\bar{B}+A\bar{C})$ $[\because B\bar{B}=C\bar{C}=0]$

$\Rightarrow ABC\cdot A\bar{B} + ABC\cdot A\bar{C}$

$\Rightarrow 0$

Ex:- $A\bar{B}C + \bar{A}BC + ABC = C(A+B)$

$\Rightarrow C(A\bar{B}+\bar{A}B)+ABC$ $(\because B+\bar{B}=1)$

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

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

$\Rightarrow C[(A+\bar{A})\cdot(A+B)]$

$\Rightarrow C(A+B)$

Ex:- $\overline{ABC}\,(\overline{A+B+C}) = \bar{A}\bar{B}\bar{C}$

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

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

$\Rightarrow \bar{A}\bar{B}\bar{C}.$

Ex: $(A+B)'(A'+B')'$

$A'B'[(A')'\cdot(B')']$

$A'B'(AB) \Rightarrow 0.$

Ex:- Express the following functions in Sum of minterms and Product of maxterms:

(i) $F(x,y,z) = (xy+z)(y+xz)$

(ii) $F(x,y,z) = 1$.

Sol:- (i) $F(x,y,z) = (xy+z)(y+xz)$

$\Rightarrow xyy + xxyz + yz + xzz$

$\Rightarrow xy + xyz + yz + xz$

$\Rightarrow xy(z+z') + xyz + yz(x+\bar{x}) + xz(y+\bar{y})$

$\Rightarrow xyz + xyz' + xyz + \underline{xyz} + \bar{x}yz + \underline{xyz} + x\bar{y}z.$

$\Rightarrow xyz + xyz' + x'yz + xy'z$

$\Rightarrow m_7 + m_6 + m_3 + m_5.$

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

Also using Complementary relation we have;

$F(x,y,z) = \Pi M(0,1,2,4)$

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

(ii) $F(x,y,z) = 1$; it includes all minterms and no maxterms

$\therefore F(x,y,z) = \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z + \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z + xy\bar{z} + xyz$

$= m_0 + m_1 + m_2 + m_3 + m_4 + m_5 + m_6 + m_7$

$= \Sigma m(0,1,2,3,4,5,6,7)$

* DIGITAL LOGIC GATES *

→ The logic gates are the basic elements that make up a digital system.

→ The gate is a digital circuit with one or more input voltages but only one output voltage.

→ The operation of a logic gate can easily understood with the help of "Truth Table."

→ A truthtable is a table that shows all the input-output possibilities of a logic circuit.

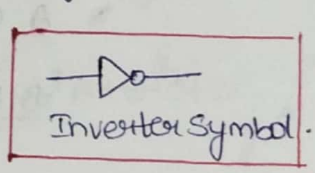
ie TruthTable indicates the outputs for different possibilities of the inputs.

* INVERTER : NOT Gate.

→ The Inverter (NOT circuit) performs a basic logic function called "Inversion" or "Complementation"

→ It changes a logic 1 to a logic 0 and logic 0 to a logic 1.

→ The bubble [o] appearing on the output is the negation (Inversion) indicator.


Inverter Symbol.

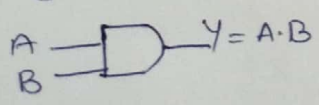
* Truth Table :

Input	output
Low (0)	High (1)
High (1)	Low (0)

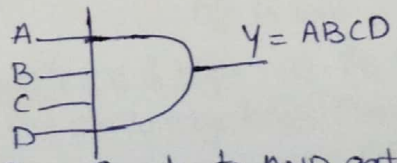
* AND Gate :-

→ The AND gate performs logical multiplication; most commonly known as AND function.

→ It has or it may have two or more inputs and a single output; as indicated by standard logic symbols.



Two inputs to AND gate



Four inputs to AND gate.

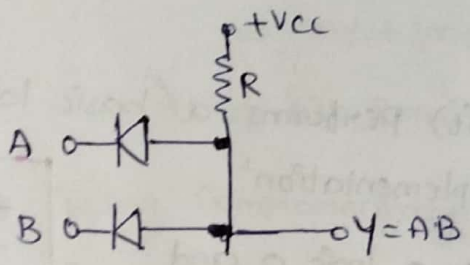
→ The Operation of AND gate is such that the output is High only when all the inputs are High.

→ If any of the inputs is low; the output is low.

* Truth Table for "AND" Gate:-

Inputs		output
A	B	Y=AB
0	0	0
0	1	0
1	0	0
1	1	1

→ Let us Consider 2 input And gate Circuit.



2 Input AND gate.

→ Assume; Supply voltage Vcc of +5V; Inputs A, B and O/P is Y.

→ Assume Input voltages are Either 0V (low) or +5V (High)

Case(i): A is low; B is low

The Cathode of Each diode is grounded.

∴ Positive Supply forward-biases both diodes in Parallel.

→ Due to this, the output Voltage is Ideally zero (Practically 0.7V for Si)

→ This means Y is low.

Case (ii): A is low; B is High

When A is low, upper diode is forward biased (ON) and it pulls the output down to a low voltage; i.e Y=0.

→ with the B Input high; the lower diode goes into reverse bias (OFF).

Case(iii): A is high and B is low.

Because of the symmetry of circuit; it is similar to Case(ii).

→ But in this case; upper diode is reverse biased (OFF); lower diode is forward biased (ON) and Y is low.

Case(iv): A is high and B is high.

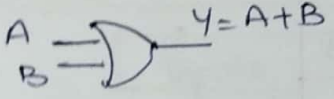
When both inputs are at +5V; both diodes are reverse biased and there is no current through diodes and Resistor R. This pulls up the output Y to the supply voltage.

∴ Y is high.

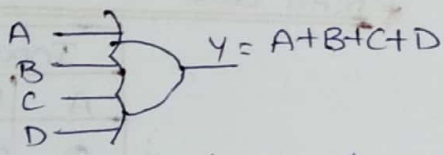
* The OR Gate:

→ The OR gate performs logical addition; most commonly known as the OR function.

→ OR gate has two or more inputs and one output, as indicated by standard logic symbol.



Two inputs to OR gate



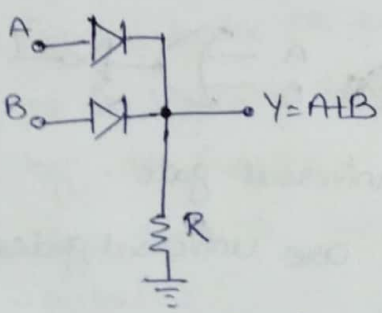
Four inputs to OR gate.

* Truth Table for OR gate.

Inputs		output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

→ Produces a high output when any of the input is high.

→ The output is low only when all of the inputs are low.



Case(i): Both i/p's are low; Anodes of 2 diodes are grounded.

∴ Diodes are Reverse biased

∴ O/p is low.

Case(ii): B is high; It FB lower diode; Producing O/p voltage high. (Upper diode is RB)

Case(iii): Same as above Case.

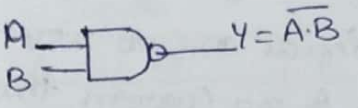
Case (iv): Both diodes are FB. i/p voltages are Hel. O/p voltage is 5V. Y is high.

*NAND Gate:

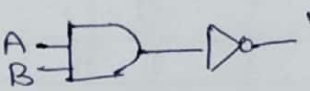
→ The term NAND is a Contraction of NOT-AND.

→ It implies an AND function with Complemented (Inverted) output.

→ A standard logic symbol for two input NAND gate is



→ Its Equivalent to AND gate is;



→ NAND gate is a universal gate; as it can be used to Construct an AND gate; an OR gate; an inverter or any Combination of these functions.

*Truth Table for NAND gate:

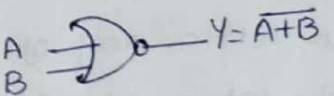
Inputs		Output
A	B	$Y = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

*NOR Gate :-

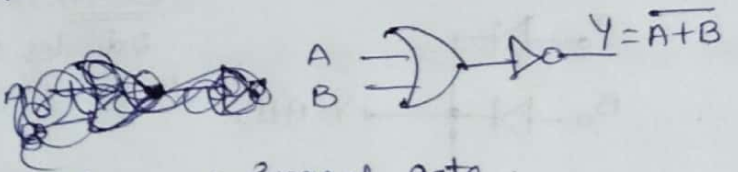
→ The term NOR is a Contraction of NOT-OR.

→ It implies an OR function with an inverted output.

→ A Standard logic Symbol for two input NOR gate is



→ Its Equivalent to NOR gate is;



→ NOR Gate is a universal gate.

∴ NAND and NOR are universal gates.

*Truth Table for NOR Gate :

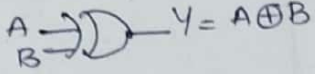
Inputs		output
A	B	$Y = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

*The Exclusive-OR Gate :

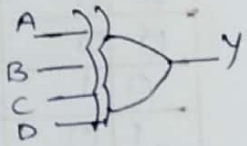
→ The Ex-OR gate is an abbrevation for Exclusive-OR gate.

→ An EX-OR gate has two or more inputs and one output.

→ The standard logic symbol for EX-OR is



Two Input Ex-OR.



Four Input Ex-OR

*Truth Table for Ex-OR :

Inputs		output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

* Truth Table can be expanded for any number of inputs; However; regardless of the number of inputs; the output is "High" only when odd number of inputs are High.

*The Exclusive-NOR Gate :

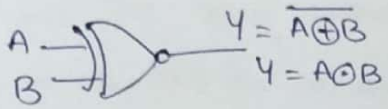
→ The term Ex-NOR is a Contraction of NOT-X-OR.

→ NOT Exclusive OR gate.

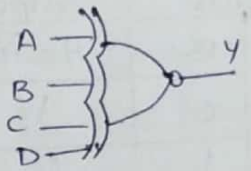
→ It is logically equivalent to an Ex-OR gate followed by an inverter.

→ The Ex-NOR gate has two or more inputs and one-output.

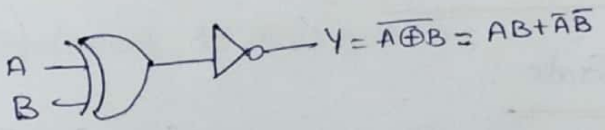
→ The Standard logic symbol for EXNOR gate is.



Two input Ex-NOR



→ Its Equivivalent to Ex-OR gate is;



* Truth Table for Ex-NOR Gate:

Inputs		output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

* Universal Gates:

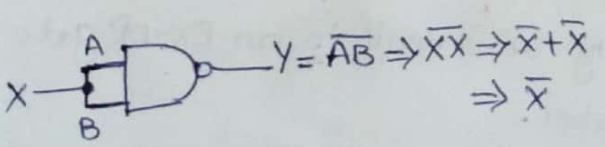
→ NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates.

* NAND Gate:

NAND gate can be used to generate NOT function, AND function, OR function and the NOR function.

(i) NOT function:

An Inverter can be made from a NAND gate by Connecting all of the inputs together and creating, in effect, a single Common Input.



A	B	$\overline{AB}$	
0	0	1	X=0 → Y=1
0	1	1	
1	0	1	
1	1	0	X=1 → Y=0

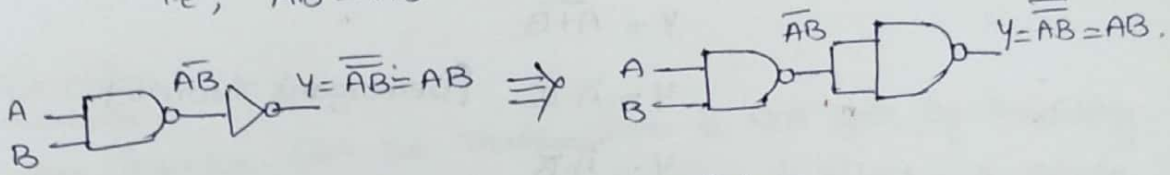
NOT function using NAND gate.

AND function :

→ An AND function can be generated using only NAND gates.

→ It is generated by inverting output of NAND gate.

i.e; $\overline{\overline{AB}} = AB$.



AND function using NAND Gates.

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

⇒

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

OR function :-

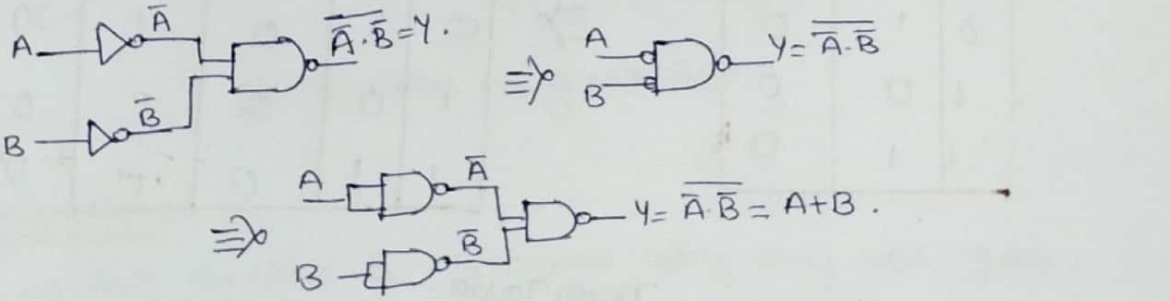
→ OR function is generated using only NAND gates.

→ OR gate Boolean Expression;

$$Y = A + B$$
$$= \overline{\overline{A}} + \overline{\overline{B}} \quad [\because \overline{\overline{A}} = A]$$
$$Y = \overline{\overline{A} \cdot \overline{B}} \quad [\because \text{Demorgans Theorem}]$$

[NOTE:- Bubble at the input of NAND gate indicate inverted input]

→ $Y = \overline{\overline{A} \cdot \overline{B}}$



OR function using only NAND gates.

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

⇒

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

Truth Table.

*NOR function :-

→ NOR function is generated using only NAND gate.

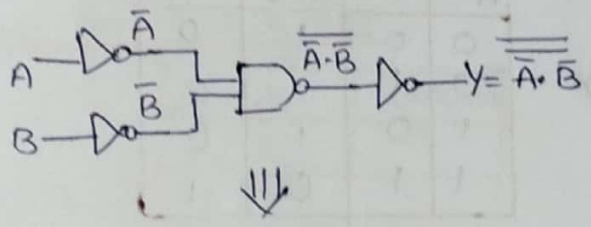
→ Boolean Expression for NOR gate is;

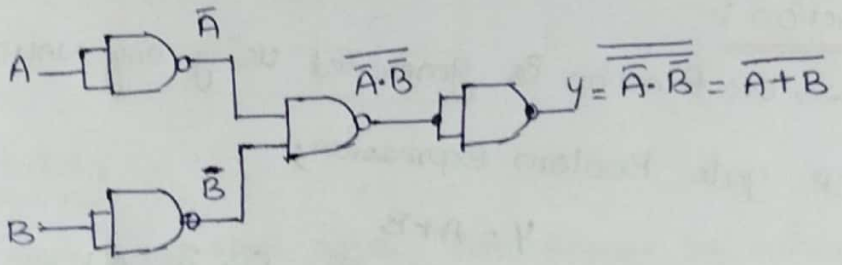
$$Y = \overline{A+B}$$

$$Y = \bar{A} \cdot \bar{B} \quad \text{[Demorgan's Theorem]}$$

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

→ The above Equation is implemented using only NAND gates.





NOR function using only NAND Gates.

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

⇒

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

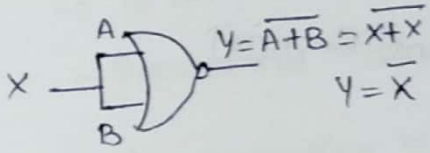
Truth Table.

*NOR Gate :-

NOR gate is also a universal gate; since it can be used to generate the NOT, AND, OR and NAND functions.

*⇒NOT function :-

→ An inverter can be made from a NOR gate by connecting all of the inputs together and creating, in effect, a single common input.



NOT function using NOR gate.

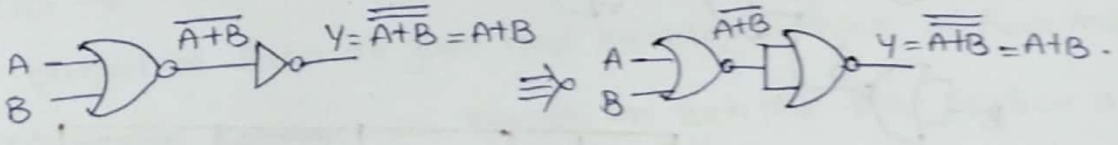
	A	B	$\overline{A+B}$	
X=0	0	0	1	Y=1
	0	1	0	
	1	0	0	
X=1	1	1	0	Y=0

*⇒OR function :-

→ An OR function can be generated using only NOR gates.

→ By inverting the output of NOR gate we get OR gate.

i.e., $\overline{A+B} = A+B$.



OR function using NOR Gate.

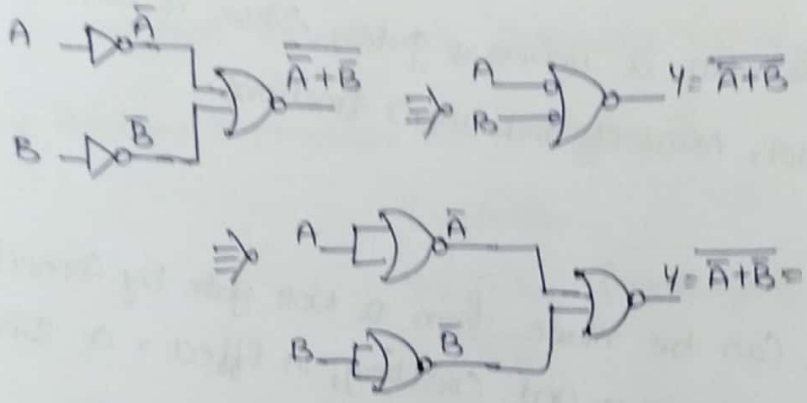
*⇒AND function :-

→ AND function is generated using only NOR gates.

→ Boolean expression for AND gate is,

$$Y = A \cdot B$$
$$= \overline{\overline{A} \cdot \overline{B}} \quad [\because \overline{\overline{A}} = A]$$
$$Y = \overline{A+B} \quad [\because \overline{A} \cdot \overline{B} = \overline{A+B}]$$

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



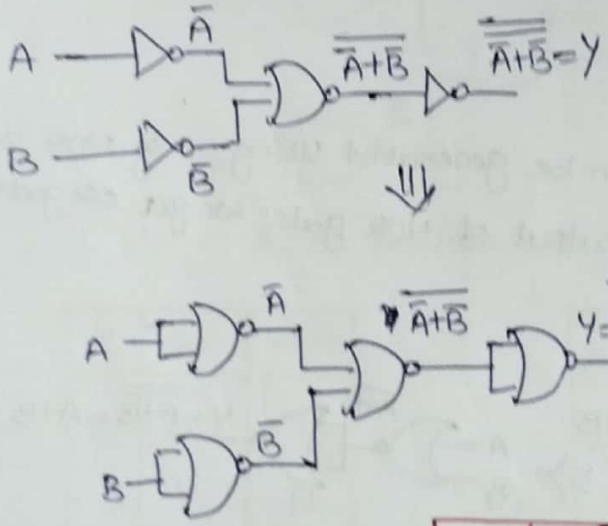
* NAND function :-

Boolean Expression for NAND gate is;

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

$Y = \bar{A}+\bar{B}$ [∵ Demorgans Theorem]

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



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

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

* Conversion of AND/OR/NOT logic to NAND/NOR logic using Graphical procedure.

→ Implementation of given Boolean Expression, one need various Standard ICs. Many times all gates within Standard ICs are not utilized.

→ To improve the utilization of ICs and to reduce number of ICs required; one can use only NAND/NOR gates to implement Boolean Expression.

→ To do this we have to convert given AND/OR/NOT Boolean-Expressions logic to NAND/NOR logic.

* Steps for Converting to NAND/NOR logic using Graphical procedure:

(1) Draw AND/OR logic.

(2) If NAND hardware has been choosen; add bubbles on the output of each AND gate; and bubbles on input side to all OR gates.

(3) If NOR hardware has been choosen add bubbles on output of each OR gate and bubble on input of each AND gate.

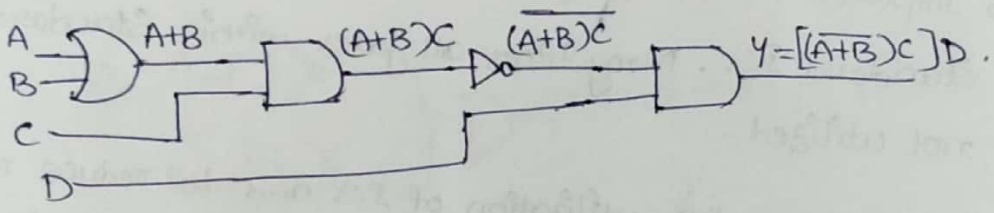
(4) Add or subtract an inverter on each line that received a bubble in step 2 or 3.

(5) Replace bubbled OR by NAND and bubbled AND by NOR.

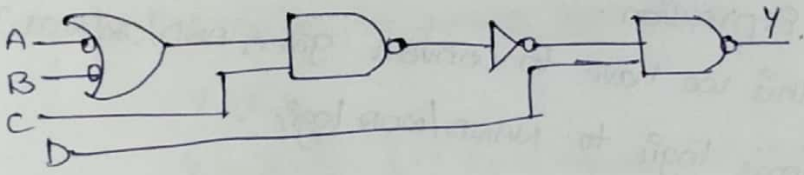
(6) Eliminate double inversions.

Ex:- Boolean expression $[\overline{(A+B)C}]D$

* Original Circuit :-

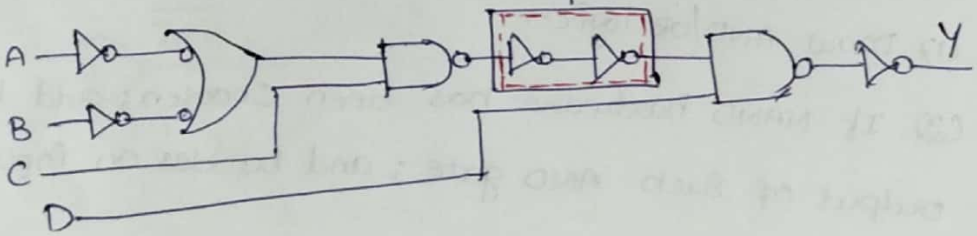


* NAND Circuit :-



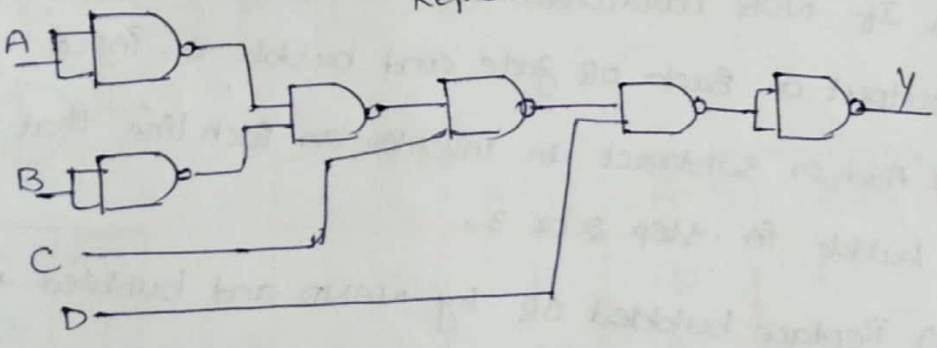
→ Add an inverter

→ eliminate double inversion

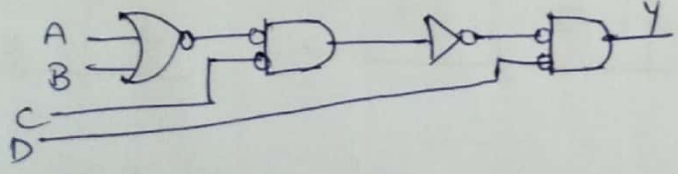


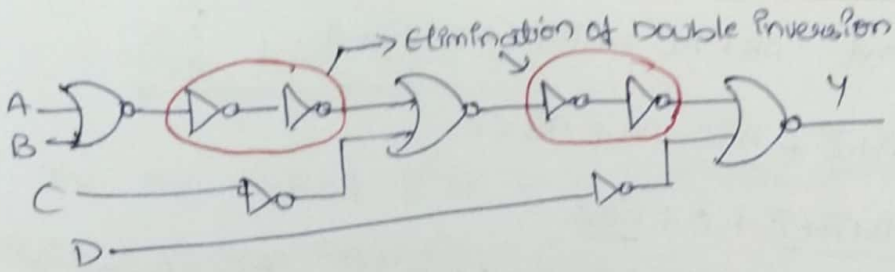
Replace Bubbled 'OR' by 'Nand'.

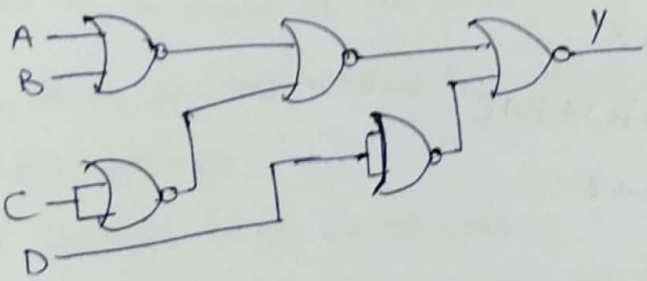
→



* NOR Circuit :-







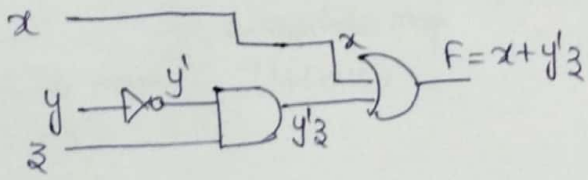
We find that three ICs are required (AND, OR & INVERTER) to implement original circuit.

→ Whereas; only two NAND ICs are required for the circuit with NAND gates. (with 2 NAND ICs we have 8 NAND gates and only 6 NAND's are required to implement the logic circuits.

→ Similar for NOR gate also.

Ex:- Draw the logic circuit diagram for $f = x + y'z$.

$F = x + y'z$.

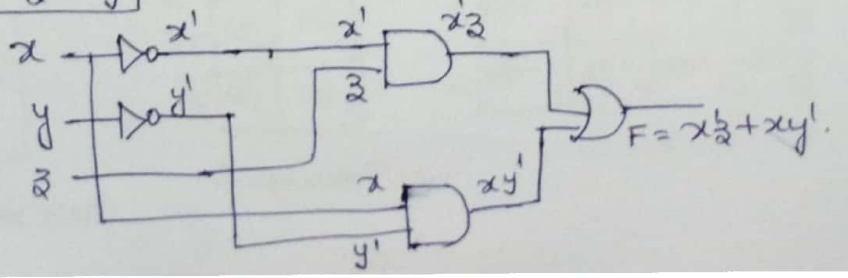


Ex:- Simplify and draw logic circuit for $F = x'y'z + x'yz + xy'$

$F = x'y'z + x'yz + xy'$ [$\because y + y' = 1$]

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

$\boxed{F = x'z + xy'}$



Ex: Prove that $AB + \bar{A}C + A\bar{B}C(AB+C) = 1$.

$AB + \bar{A} + \bar{C} + A\bar{B}C \cdot AB + A\bar{B} \cdot CC$

$\Rightarrow AB + \bar{A} + \bar{C} + 0 + A\bar{B}C$

$(\bar{A}+A)(\bar{A}+B) + (A\bar{B}+\bar{C})(C+\bar{C})$ [$\because B\bar{B}=0$, $CC=C$]

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

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

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

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

$\Rightarrow 1 + \bar{A} + \bar{C}$

$\Rightarrow 1$

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

$\bar{A}\bar{B}C + \bar{A} \cdot \bar{B} \cdot C + \bar{A}\bar{B}\bar{C}D$

$\Rightarrow \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}D$

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

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

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